99SP4020E-1

SSL II USER'S GUIDE (Scientific Subroutine Library)

FUJITSU

PREFACE

This manual describes the functions and use of the Scientific Subroutine Library II (SSL II), SSL II can be used for various systems ranging from personal computers to vector computers. The interface between a user-created program and SSL II is always the same regardless of the system type. Therefore, this manual can be used for all systems that use SSL II. When using SSL II for the first time, the user should read “How to Use This Manual” first.

The contents of SSL II or this manual may be amended to keep up with the latest state of technology, that is, if the revised or added subroutines should functionally include or surpass some of the old subroutines, those old subroutines will be deleted in a time of allowance.

Note:
Some of the SSL II functions may be restricted in certain systems due to hardware restrictions. These functions are in the SSL II Subroutine List in this manual.

First Edition June 1989

ACKNOWLEDGEMENTS

We believe that a scientific subroutine library can be one of convenient vehicles for the dissemination of the fruits of investigation made by experts concerned with numerical analysis, furthermore, through the library a user can make use of these new fruits without ever having to become familiar with the technical details.

This led us to conclude that we had better request as many experts as possible to implement their new methods, and then promote distribution of them to a wide community of users. As a result of our efforts to show our idea to many experts or authorities and to discuss continuously with them, we could fortunately get a common consensus from them. We are pleased to enumerate below experts who have contributed to SSL II until now in acknowledgment of their efforts.

Contributors

Name	Organization
Masatsugu Tanaka	Faculty of Engineering, Yamanashi University, Japan
Ichizo Ninomiya	Faculty of Management Information Chubu University, Japan
Tatsuo Torii	Faculty of Engineering, Nagoya University, Japan
Takemitsu Hasegawa	Faculty of Engineering, Fukui University, Japan
Kazuo Hatano	Faculty of Engineering, Aichi Technical University, Japan
Yasuyo Hatano	Department of Education, Chukyo University, Japan
Toshio Yoshida	Faculty of Management Information Chubu University, Japan
Kaoru Tone	Graduate School for Policy Science, Saitama University, Japan
Takashi Kobayashi	Graduate School for Policy Science, Saitama University, Japan
Toshio Hosono	College of Science and Technology, Nihon University, Japan

(Nov., 1987)

Also, our thanks are due to many universities and research institutes, especially those shown below, for helpful discussions and advises on our project, for providing information on recent algorithms, numerical comparisons, and for suggesting several important improvements in this users' guide.

Organizations
Computation Center, Hokkaido University
Computation Center, Nagoya University
Computation Center, Kyoto University
Computation Center, Kyusyu University
Japan Atomic Energy Research Institute

It would be another pleasure that users would appreciate the efforts by contributors, and on the occasions of publication of users' own technical papers they would advertise on the papers as possible the fact of having used SSL II.

CONTENTS

SSL II SUBROUTINE LIST

The SSL II functions are listed below. Generally, a single-precision routine and a double-precision routine are available for each function. The subroutine name column gives the names of single-precision routines. Double-precision routine names start with a D, followed by the single-precision names. If the use of a function is restricted due to hardware restrictions, it is indicated in the remarks column.

The symbols that appear in the remarks column mean the following:

#: Only the single-precision routine is available in all systems.

A. Linear Algebra

Storage mode conversion of matrices

Subroutine name	Item	Page	Remarks
CGSM	Storage mode conversion of matrices (real symmetric to real general)	264	
CSGM	Storage mode conversion of matrices (real general to real symmetric)	290	
CGSBM	Storage mode conversion of matrices (real general to real symmetric band)	263	
CSBGM	Storage mode conversion of matrices (real symmetric band to real general)	287	
CSSBM	Storage mode conversion of matrices (real symmetric to real symmetric band)	291	
CSBSM	Storage mode conversion of matrices (real symmetric band to real symmetric)	289	

Matrix manipulation

Subroutine name	Item	Page	Remarks
AGGM	Addition of two matrices (real general + real general)	85	
SGGM	Subtraction of two matrices (real general – real general)	563	
MGGM	Multiplication of two matrices (real general by real general)	454	
MGSM	Multiplication of two matrices (real general by real symmetric)	465	
ASSM	Addition of two matrices (real symmetric + real symmetric)	131	
SSSM	Subtraction of two matrices (real symmetric – real symmetric)	582	
MSSM	Multiplication of two matrices (real symmetric by real symmetric)	477	
MSGM	Multiplication of two matrices (real symmetric by real general)	476	
MAV	Multiplication of a real matrix by a real vector	456	
MCV	Multiplication of a complex matrix by a complex vector	460	
MSV	Multiplication of a real symmetric matrix by a real vector	478	
MSBV	Multiplication of a real symmetric band matrix and a real vector	474	
MBV	Multiplication of a real band matrix and a real vector	458	

Linear equations

Subroutine name	Item	Page	Remarks
LAX	A system of linear equations with a real general matrix (Crout' s method)	388	
LCX	A system of linear equations with a complex general matrix (Crout' s method)	407	
LSX	A system of linear equations with a positive-definite symmetric matrix (Modified Cholesky' s method)	445	
LSIX	A system of linear equations with a real indefinite symmetric matrix (Block diagonal pivoting method)	438	
LSBX	A system of linear equations with a positive-definite symmetric band matrix (Modified Cholesky' s method)	433	
LSBIX	A system of linear equations with a real indefinite symmetric band matrix (block diagonal pivoting method)	431	
LBX1	A system of linear equations with a real general band matrix (Gaussian elimination method)	402	
LSTX	A system of linear equations with a positive-definite symmetric tridiagonal matrix (Modified Cholesky' s method)	442	
LTX	A system of linear equations with a real tridiagonal matrix (Gaussian elimination method)	449	
LAXR	Iterative refinement of the solution to a system of linear equations with a real general matrix	399	
LCXR	Iterative refinament of the solution to a system of linear equations with a complex general matrix	409	
LSXR	Iterative refinement of the solution to a system of linear equations with a positive-definite symmetric matrix	447	
LSIXR	Iterative refinament of the solution to a system of linear equations with a real indefinite symmetric matrix	440	
LSBXR	Iterative refinament of the solution to a system of linear equations with a positive-definite symmetric band matrix	435	
LBX1R	Iterative refinement of the solution to a system of linear equations with a real general band matrix	404	

Matrix inversion

Subroutine name	Item	Page	Remarks
LUIV	The inverse of a real general matrix decomposed into the factors L and U	452	
CLUIV	The inverse of a complex general matrix decomposed into the factors L and U	279	
LDIV	The inverse of a positive-definite symmetric matrix decomposed into the factors L, D and L^T	412	

Decomposition of matrices

Subroutine name	Item	Page	Remarks
ALU	LU-decomposition of a real general matrix (Crout' s method)	98	
CLU	LU-decomposition of a complex general matrix (Crout' s method)	277	
SLDL	LDL^T-decomposition of a positive-definite symmetric matrix (Modified Cholesky' s method)	570	
SMDM	MDM^T-decomposition of a real indefinite symmetric matrix (Block diagonal pivoting method)	572	
SBDL	LDL^T-decomposition of a positive-definite symmetric band matrix (Modified Cholesky' s method)	553	
SBMDM	MDM^T-decomposition of a real indefinite symmetric band matrix (block diagonal pivoting method)	555	
BLU1	LU-decomposition of a real general band matrix (Gaussian elimination method)	189	

Solution of decomposed system

Subroutine name	Item	Page	Remarks
LUX	A system of linear equations with a real general matrix decomposed into the factors L and U	454	
CLUX	A system of linear equations with a complex general matrix decomposed into the factors L and U	281	
LDLX	A system of linear equations with a positive-definite symmetric matrix decomposed into the factors L, D and L^T	414	
MDMX	A system of linear equations with a real indefinite symmetric matrix decomposed into the factors M, D and M^T	462	
BDLX	A system of linear equations with a positive-definite symmetric band matrix decomposed into the factors L, D and L^T	136	
BMDMX	A system of linear equations with a real indefinite symmetric band matrix decomposed into factors M, D, and M^T	192	
BLUX1	A system of linear equations with a real general band matrix decomposed into the factors L and U	186	

Least squares solution

Subroutine name	Item	Page	Remarks
LAXL	Least squares solution with a real matrix (Householder transformation)	390	
LAXLR	Iterative refinement of the least squares solution with a real matrix	397	
LAXLM	Least squares minimal norm solution with a real matrix (Singular value decomposition method)	393	
GINV	Generalized Inverse of a real matrix (Singular value decomposition method)	341	
ASVD1	Singular value decomposition of a real matrix (Householder and QR methods)	132	

B. Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors

Subroutine name	Item	Page	Remarks
EIG1	Eigenvalues and corresponding eigenvectors of a real matrix (double QR method)	298	
CEIG2	Eigenvalues and corresponding eigenvectors of a complex matrix (QR method)	242	
SEIG1	Eigenvalues and corresponding eigenvectors of a real symmetric matrix (QL method)	558	
SEIG2	Selected eigenvalues and corresponding eigenvectors of a real symmetric matrix (Bisection method, inverse iteration method)	560	
HEIG2	Eigenvalues and corresponding eigenvectors of an Hermition matrix (Householder method, bisection method, and inverse iteration method)	356	
BSEG	Eigenvalues and eigenvectors of a real symmetric band matrix (Rutishauser-Schwarz method, bisection method and inverse iteration method)	206	
BSEGJ	Eigenvalues and eigenvectors of a real symmetric band matrix (Jennings method)	208	
TEIG1	Eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix (QL method)	583	
TEIG2	Selected eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix (Bisection method, inverse iteration method)	585	
GSEG2	Eigenvalues and corresponding eigenvectors of a real symmetric generalized matrix system $\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$ (Bisection method, inverse iteration method)	347	
GBSEG	Eigenvalues and corresponding eigenvectors of a real symmetric band generalized eigenproblem (Jennings method)	335	

Eigenvalues

Subroutine name	Item	Page	Remarks
HSQR	Eigenvalues of a real Hessenberg matrix (double QR method)	361	
CHSQR	Eigenvalues of a complex Hessenberg matrix (QR method)	270	
TRQL	Eigenvalues of a real symmetric tridiagonal matrix (QL method)	598	
BSCT1	Selected eigenvalues of a real symmetric tridiagonal matrix (Bisection method)	198	

Eigenvectors

Subroutine name	Item	Page	Remarks
HVEC	Eigenvectors of a real Hessenberg matrix (Inverse iteration method)	363	
CHVEC	Eigenvectors of a complex Hessenberg matrix (Inverse iteration method)	272	
BSVEC	Eigenvectors of a real symmetric band matrix (Inverse iteration method)	218	

Others

Subroutine name	Item	Page	Remarks
BLNC	Balancing of a real matrix	184	
CBLNC	Balancing of a complex matrix	239	
HES1	Reduction of a real matrix to a real Hessenberg matrix (Householder method)	358	
CHES2	Reduction of a complex matrix to a complex Hessenberg matrix (Stabilized elementary transformation)	268	
TRID1	Reduction of a real symmetric matrix to a real symmetric tridiagonal matrix (Householder method)	596	
TRIDH	Reduction of an Hermition matrix to a real symmetric tridiagonal matrix (Householder method and diagonal unitary transformation)	593	
BTRID	Reduction of a real symmetric band matrix to a tridiagonal matrix (Rutishauser-Schwarz method)	221	
HBK1	Back transformation and normalization of the eigenvectors of a real Hessenberg matrix	354	
CHBK2	Back transformation of the eigenvectors of a complex Hessenberg matrix to the eigenvectors of a complex matrix	266	
TRBK	Back transformation of the eigenvectors of a tridiagonal matrix to the eigenvectors of a real symmetric matrix	589	
TRBKH	Back transformation of eigenvectors of a tridiagonal matrix to the eigenvectors of an Hermition matrix	591	
NRML	Normalization of eigenvectors	498	
CNRML	Normalization of eigenvectors of a complex matrix	283	
GSCHL	Reduction of a real symmetric matrix system $\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$ to a standard form	345	
GSBK	Back transformation of the eigenvectors of the standard form the eigenvectors of the real symmetric generalized matrix system	343	

C. Nonlinear Equations

Subroutine name	Item	Page	Remarks
RQDR	Zeros of a quadratic with real coefficients	552	
CQDR	Zeros of a quadratic with complex coefficients	286	
LOWP	Zeros of a low degree polynomial with real coefficients (fifth degree or lower)	423	
RJETR	Zeros of a polynomial with real coefficients (Jenkins-Traub method)	546	
CJART	Zeros of a polynomial with complex coefficients (Jarratt method)	275	
TSD1	Zero of a real function which changes sign in a given interval (derivative not required)	604	
TSDM	Zero of a real function (Muller' s method)	601	
CTSDM	Zero of complex function (Muller' s method)	292	
NOLBR	Solution of a system of nonlinear equations (Brent' s method)	487	

D. Extrema

Subroutine name	Item	Page	Remarks
LMINF	Minimization of function with a variable (quadratic interpolation using function values only)	418	
LMING	Minimization of function with a variable (cubic interpolation using function values and its derivatives)	420	
MINF1	Minimization of function with several variables (revised quasi-Newton method, uses function values only)	466	
MING1	Minimization of a function with several variables (Quasi-Newton method, using function values and its derivatives)	470	
NOLF1	Minimization of the sum of squares of functions with several variables (Revised Marquardt method, using function values only)	490	
NOLG1	Minimization of the sum of squares of functions (revised Marquardt method using function values and its derivatives)	494	
LPRS1	Solution of a linear programming problem (Revised simplex method)	425	
NLPG1	Nonlinear programming (Powell' s method using function values and its derivatives)	482	

E. Interpolation and Approximation

Interpolation

Subroutine name	Item	Page	Remarks
AKLAG	Aitken-Lagrange interpolation	89	
AKHER	Aitkan-Hermite interpolation	86	
SPLV	Cubic spline interpolation	579	
BIF1	B-spline interpolation (I)	156	
BIF2	B-spline interpolation (II)	158	
BIF3	B-spline interpolation (III)	160	
BIF4	B-spline interpolation (IV)	162	
BIFD1	B-spline two-dimensional interpolation (I-I)	151	
BIFD3	B-spline two-dimensional interpolation (III-III)	154	
AKMID	Two-dimensional quasi-Hermite Interpolation	91	
INSPL	Cubic spline interpolation coefficient calculation	377	
AKMIN	Quasi-Hermite interpolation coefficient calculation	95	
BIC1	B-spline interpolation coefficient calculation (I)	143	
BIC2	B-spline interpolation coefficient calculation (II)	145	
BIC3	B-spline interpolation coefficient calculation (III)	147	
BIC4	B-spline interpolation coefficient calculation (IV)	149	
BICD1	B-spline two-dimensional interpolation coefficient calculation (I-I)	138	
BICD3	B-spline two-dimensional interpolation coefficient calculation (III-III)	141	

Approximation

Subroutine name	Item	Page	Remarks
LESQ1	Polynomial least squares approximation	416	

Smoothing

Subroutine name	Item	Page	Remarks
SMLE1	Data smoothing by local least squares polynomials (equally spaced data points)	575	
SMLE2	Data smoothing by local least squares polynomials (unequally spaced data points)	577	
BSF1	B-spline smoothing	216	
BSC1	B-spline smoothing coefficient calculation	201	
BSC2	B-spline smoothing coefficient calculation (variable knots)	203	
BSFD1	B-spline two-dimensional smoothing	214	
BSCD2	B-spline two-dimensional smoothing coefficient calculation (variable knots)	194	

Series

Subroutine name	Item	Page	Remarks
FCOSF	Fourier Cosine series expansion of an even function (Function input, fast cosine transform)	312	
ECOSP	Evaluation of a cosine series	296	
FSINF	Fourier sine series expansion of an odd function (Function input, fast sine transform)	324	
ESINP	Evaluation of a sine series	302	
FCHEB	Chabyshev series expansion of a real function (Function input, fast cosine transform)	306	
ECHEB	Evaluation of a Chebyshev series	294	
GCHEB	Differentiation of a Chebyshev series	339	
ICHEB	Indefinite integral of a Chebyshev series	367	

F. Transforms

Subroutine name	Item	Page	Remarks
FCOST	Discrete cosine transform (Trapezoidal rule, radix 2 FFT)	321	
FCOSM	Discrete cosine transform (midpoint rule, radix 2 FFT)	318	
FSINT	Discrete since transform (Trapezoidal rule, radix 2 FFT)	333	
FSINM	Discrete sine transform (midpoint rule, radix 2 FFT)	330	
RFT	Discrete real Fourier transform	543	
CFTM	Multi-variate discrete complex Fourier transform (mixed radix FFT)	250	
CFT	Multi-variate discrete complex Fourier transform (radix 8 and 2 FFT)	247	
CFTN	Discrete complex Fourier transforms (radix 8 and 2 FFT, reverse binary order output)	254	
CFTR	Discrete complex Fourier transforms (radix 8 and 2 FFT, reverse binary order input)	259	
PNR	Permutation of data (reverse binary transformation)	522	
LAPS1	Inversion of Laplace transform of a rational function (regular in the right-half plane)	379	
LAPS2	Inversion of Laplace transform of a general rational function	381	
LAPS3	Inversion of Laplace transform of a general function	383	
HRWIZ	Judgment on Hurwiz polynomials	360	

G. Numerical Differentiation and Quadrature

Numerical Differentiation

Subroutine name	Item	Page	Remarks
SPLV	Cubic spline differentiation	579	
BIF1 BIF2 BIF3 BIF4	Differentiation (Unequally spaced discrete points, B-spline Interpolation)	156 158 160 162	
BSF1	Differentiation by B-spline least squares fit (Fixed knots)	216	
BIFD1 BIFD3	Two-dimensional differentiation (unequally spaced lattice points. B-spline two-dimensional interpolation)	151 154	
BSFD1	Two-dimensional differentiation by B-spline least squares fit	214	
GCHEB	Differentiation of a Chebyshev series	339	

Numerical Quadrature

Subroutine name	Item	Page	Remarks
SIMP1	Integration of a tabulated function by Simpson' s rule (equally spaced)	564	
TRAP	Integration of a tabulated function by trapezoidal rule (unequally spaced)	588	
BIF1 BIF2 BIF3 BIF4	Intergration of a tabulated function by B-spline interpolation (unequally spaced discrete points, B-spline interpolation)	156 158 160 162	
BSF1	Smoothing differentiation and integration of a tabulated function by B-spline least squares fit (fixed knots)	216	
BIFD1 BIFD3	Integration of a two-dimensional tabulated function (unequally spaced lattice points, B-spline two-dimensional interpolation)	151 154	
BSFD1	Two-dimensional integration of a tabulated function by B-spline interpolation	214	
SIMP2	Integration of a function by adaptive Simpson' s rule	565	
AQN9	Integration of a function by adaptive Newton-Cotes 9-point rule	126	
AQC8	Integration of a function by a modified Clenshaw-Curtis rule	100	
AQE	Integration of a function by double exponential formula	106	
AQEH	Integration of a function over the semi-infinite interval by double exponential formula	110	
AQEI	Integration of a function over the infinite interval by double exponential formula	112	
AQMC8	Multiple integration of a function by a modified Clenshaw-Curtis integration rule	115	
AQME	Multiple integration of a function by double exponential formula	121	

H. Differential equations

Subroutine name	Item	Page	Remarks
RKG	A system of first order ordinary differential equations (Runge-Kutta method)	550	
HAMNG	A system of first order ordinary differential equations (Hamming method)	350	
ODRK1	A system of first order ordinary differential equations (Runge-Kutta-Verner method)	518	
ODAM	A system of first order ordinary differential equations (Adams method)	500	
ODGE	A stiff system of first order ordinary differential equations (Gear' s method)	509	

I. Special Functions

Subroutine name	Item	Page	Remarks
CELI1	Complete elliptic Integral of the first kind $K(x)$	244	
CELI2	Complete elliptic integral of the second kind $E(x)$	245	
EXPI	Exponential integral $E_i(x)$, $\overline{E}_i(x)$	304	
SINI	Sine integral $S_i(x)$	569	
COSI	Cosine integral $C_i(x)$	285	
SFRI	Sine Fresnel integral $S(x)$	562	
CFRI	Cosine Fresnel integral $C(x)$	246	
IGAM1	Incomplete Gamma function of the first kind $\gamma(v, x)$	372	
IGAM2	Incomplete Gamma function of the second kind $\Gamma(v, x)$	373	
IERF	Inverse error function $\mathrm{erf}^{-1}(x)$	369	
IERFC	Inverse complimented error function $\mathrm{erfc}^{-1}(x)$	370	
BJ0	Zero-order Bessel function of the first kind $J_0(x)$	173	
BJ1	First-order Bessel function of the first kind $J_1(x)$	175	
BY0	Zero-order Bessel function of the second kind $Y_0(x)$	228	
BY1	First-order Bessel function of the second kind $Y_1(x)$	230	
BI0	Modified Zero-order Bessel function of the first kind $I_0(x)$	167	
BI1	Modified First-order Bessel function of the first kind $I_1(x)$	168	
BK0	Modified Zero-order Bessel function of the second kind $K_0(x)$	182	
BK1	Modified First-order Bessel function of the second kind $K_1(x)$	183	
BJN	Nth-order Bessel function of the first kind $J_n(x)$	169	
BYN	Nth-order Bessel function of the second kind $Y_n(x)$	223	
BIN	Modified Nth-order Bessel function of the first kind $I_n(x)$	164	
BKN	Modified Nth-order Bessel function of the second kind $K_n(x)$	177	
CBIN	Modified Nth-order Bessel function of the first kind $I_n(z)$ with complex variable	232	
CBKN	Modified Nth-order Bessel function of the second kind $K_n(z)$ with complex variable	236	
CBJN	Integer order Bessel function of the first kind with complex variable $J_n(z)$	233	
CBYN	Integer order Bessel function of the second kind with complex variable $Y_n(z)$	241	
BJR	Real-order Bessel function of the first kind $J_v(x)$	171	
BYR	Real-order Bessel function of the second kind $Y_v(x)$	224	
BIR	Modified real-order Bessel function of the first kind $I_v(x)$	166	
BKR	Real order modified Bessel function of the second kind $K_v(x)$	178	
CBJR	Real-order Bessel function of the first kind with a complex variable $J_v(z)$	234	
NDF	Normal distribution function $\phi(x)$	480	
NDFC	Complementary normal distribution function $\psi(x)$	481	
INDF	Inverse normal distribution function $\phi^{-1}(x)$	375	
INDFC	Inverse complementary normal distribution function $\psi^{-1}(x)$	376	

J. Pseudo Random Numbers

Subroutine name	Item	Page	Remarks
RANU2	Uniform (0, 1) pseudo random numbers	533	#
RANU3	Shuffled uniform (0, 1) pseudo random numbers	536	#
RANN1	Fast normal pseudo random numbers	528	#
RANN2	Normal pseudo random numbers	530	#
RANE2	Exponential pseudo random numbers	527	#
RANP2	Poisson pseudo random integers	531	#
RANB2	Binominal pseudo random numbers	525	#
RATF1	Frequency test for uniform (0, 1) pseudo random numbers	538	#
RATR1	Run test of up-and-down for uniform (0, 1) pseudo random numbers	540	#

HOW TO USE THIS MANUAL

This section describes the logical organization of this manual, and the way in which the user can quickly and accurately get informations necessary to him from the manual.

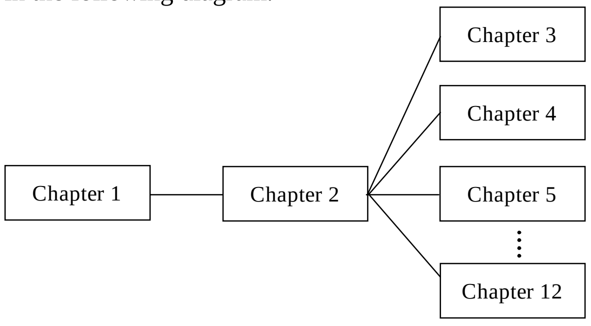
This manual consists of two parts. Part I describes an outline of SSL II. Part II describes usage of SSL II subroutines.

Part I consists of twelve chapters.

Chapter 2 describes the general rules which apply to each SSL II subroutine. It is suggested that the user read this chapter first.

Chapters 3 through 12 are concerned with certain fields of numerical computation, and were edited as independently as possible for easy reference. At the beginning of every chapter, the section "OUTLINE" is given, which describes the classification of available subroutines in the chapter, and how to select subroutines suitable for a certain purpose. The user should read the section at least once.

As mentioned above, there is no confusing or difficult relation between the chapters: it is quite simple as shown in the following diagram.



Each chapter from chapter 3 on has several sections, the first of which is the section "OUTLINE" that, as noted previously, introduces the following sections.

As the diagram shows, if the user wants to obtain eigenvalues, for example, of a certain matrix, he should first read Chapter 2, then jump to Chapter 4, where he can select subroutines suitable for his purposes.

Part II describes how to use SSL II subroutines. The subroutines are listed in alphabetical order.

When describing an individual subroutine, the following contents associated with the subroutine are shown:

- Function
- Parameters
- Comments on use
- Method

and what we intend to describe under each title above are as follows:

Function
Describes explanation of the functions.

Parameters
Describes variables and arrays used for transmitting information into or from the subroutine. Generally, parameter names, which are commonly used in SSL II, are those habitually used so far in many libraries.

Comments on use
This consists of the following three parts.

- Subprograms used
  If other SSL II subroutines are used internally by the subroutine, they are listed under "SSL II". Also, if FORTRAN intrinsic functions or basic external functions are used, they are listed under "FORTRAN basic functions".
- Notes
  Discusses various considerations that the user should be aware of when using the subroutine.
- Example
  An example of the use of the subroutine is shown. For clarity and ease of understanding, any applications to a certain field of engineering or physical science have not been described. In case other subroutines must be used as well as the subroutine to obtain a mathematical solution, the example has been designed to show how other subroutines are involved. This is especially true in the chapters concerning linear equations or eigenvalues etc. Conditions assumed in an example are mentioned at the beginning of the example.

Method
The method used by the subroutine is outlined. Since this is a usage manual, only practical aspects of the algorithm or computational procedures are described. References on which the implementation is based and those which are important in theory, are listed in Appendix D "References", so refer to them for further information or details beyond the scope of the "Method" section.

In this manual, included are the SSL II Subroutine list and four appendices. In the SSL II Subroutine list, SSL II Subroutines are arranged in the order of fields and then in the order of their classification codes. This list can be used for quick reference of subroutines.

Appendix A explains the functions of the auxiliary subroutines and Appendix B contains the three lists, which are concerned respectively with

- General subroutines
- Slave subroutines
- Auxiliary subroutines

General subroutines is an alphabetical listing of all subroutines. In the list, if a certain entry uses other subroutines, they are shown on the right. Slave subroutines is an alphabetical listing of slave subroutines (as for the definition of them, see Section 2.1), and in the list, general subroutines which use the slave subroutine are shown on the right. Auxiliary subroutines is a listing of auxiliary subroutines, and it is also alphabetical. Appendix C explains the subroutine names in order of classification code. This list can be used for quick reference of subroutines by classification code.

Appendix D lists the documents referred for SSL II development and/or logically important theory. Although no preliminary knowledge except the ability to understand FORTRAN is required to use this manual. Mathematical symbols used in this manual are listed below. We expect that the user has same interest in, or some familiarity with numerical analysis.

Mathematical symbol table

Symbol	Example	Meaning	Remarks
$^{\mathrm{T}}$	$\boldsymbol{A}^{\mathrm{T}}$	Transposed matrix of matrix $\boldsymbol{A}$	
	$\boldsymbol{x}^{\mathrm{T}}$	Transposed vector of column vector $\boldsymbol{x}$	
	$\boldsymbol{x} = (x_1, ..., x_n)^{\mathrm{T}}$	Column vector	Refer to the symbol ( ).
$^{-1}$	$\boldsymbol{A}^{-1}$	Inverse of matrix $\boldsymbol{A}$	
*	$\boldsymbol{A}^{*}$	Conjugate transposed matrix of matrix $\boldsymbol{A}$	
	$\boldsymbol{x}^{*}$	Conjugate transposed vector of column vector $\boldsymbol{x}$	
	z^{*}	Conjugate complex number of complex number z	$z = a + ib$ $z^{*} = a - ib$ $\bar{z} = z^{*}$
$\bar{\ }$	$\bar{z}$		
$\begin{bmatrix} & \end{bmatrix}$	$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{bmatrix}$	$\boldsymbol{A}$ is an $n \times n$ matrix whose elements are a_{ij}.	
( )	$\boldsymbol{x} = (x_1, ..., x_n)^{\mathrm{T}}$	$\boldsymbol{x}$ is an n-dimensional column vector whose elements are x_i.	
	$\boldsymbol{A} = (a_{ij})$	Elements of matrix $\boldsymbol{A}$ are defined as a_{ij}.	
	$\boldsymbol{x} = (x_i)$	Elements of column vector $\boldsymbol{x}$ are defined as x_i.	
diag	$\boldsymbol{A} = \mathrm{diag}(a_{ii})$	Matrix $\boldsymbol{A}$ is a diagonal matrix whose elements are a_{ii}.	
$\boldsymbol{I}$		Unit matrix	
det	$\det(\boldsymbol{A})$	Determinant of matrix $\boldsymbol{A}$	
rank	$\mathrm{rank}(\boldsymbol{A})$	Rank of matrix $\boldsymbol{A}$	
$\Vert\ \Vert$	$\Vert\boldsymbol{x}\Vert$	Norm of vector $\boldsymbol{x}$ For n-dimensional vector $\boldsymbol{x}$, $\boldsymbol{x} = (x_j)$: $\Vert \boldsymbol{x} \Vert_1 = \sum_{i=1}^{n} \vert x_i \vert$: Uniform norm $\Vert \boldsymbol{x} \Vert_2 = \sqrt{\sum_{i=1}^{n} \vert x_i \vert^2}$: Euclidean norm $\Vert \boldsymbol{x} \Vert_\infty = \max_i \vert x_i \vert$: Infinity norm Refer to symbols $\sum$, $\vert\ \vert$, max	
	$\Vert\boldsymbol{A}\Vert$	Norm of matrix $\boldsymbol{A}$ For a matrix $\boldsymbol{A} = (a_{ij})$ of order n: $\Vert\boldsymbol{A}\Vert_\infty = \max_i \left(\sum_{j=1}^{n} \vert a_{ij} \vert \right)$: Infinity norm	
(,)	$(\boldsymbol{x}, \boldsymbol{y})$	Inner product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$	When $\boldsymbol{x}$ and $\boldsymbol{y}$ are complex vectors, $(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}} \bar{\boldsymbol{y}}$
	(a, b)	Open interval	
[,]	$[a, b]$	Closed interval	
>>	$a >> b$	a is much greater than b.	
≠	$a \neq b$	a is not equal to b.	
≈	$f(x) \approx P(x)$	$f(x)$ is approximately equal to $P(x)$.	
≡	$f(x) \equiv f'(x) / f(x)$	$f(x)$ is defined as $f'(x) / f(x)$.	
{ }	$\{x_i\}$	Sequence of numbers	

Symbol	Example	Meaning	Remarks
Σ	$\sum_{i=m}^{n} x_i$	Summation (x_m ,..., x_n)	Sum cannot be given if $n < m$ If $\sum_{\substack{i=m \\ i \neq l}}^{n} x_i$ summation excludes x_l.
	$\sum_{i} x_{i+2j}$	Summation with respect to i	
	y', $f'(x)$	$y' = \frac{dy}{dx}$, $f'(x) = \frac{df(x)}{dx}$	For n-order derivative: $f^{(n)}(x) = \frac{d^n f(x)}{dx^n}$
$\vert\ \vert$	$\vert z \vert$	Absolute value of z	If $z = a + ib$ $\vert z \vert = \sqrt{a^2 + b^2}$
max	max(x_1 , ..., x_n)	The maximum value of (x_1, ..., x_n)	
min	$\max_{i} x_i$	The minimum value of (x_1, ..., x_n)	
	min(x_1 , ..., x_n)		
sign	$\min_{i} x_i$	Sign of x	When x is positive, 1. When x is negative, -1.
	sign(x)		
log	log x	Natural logarithm of x	
Re	Re(z)	Real part of complex number z	
Im	Im(z)	Imaginary part of complex number z	
arg	Arg z	Argument of complex number z	
δ_{ij}		Kronecker's delta	
γ		Euler's constant	
π		Ratio of the circumference of the circle to its diameter	
i	$z = a + ib$	Imaginary unit	$i = \sqrt{-1}$
P.V.	P.V. $\int_{-\infty}^{x} \frac{e^t}{t} dt$	Principal value of an integral	
$\frac{\vert}{\vert}$	$b_0 + \frac{a_1 \vert}{\vert b_1} + \frac{a_2 \vert}{\vert b_2} + \cdots$	Continued fraction	
$\in$	$x \in X$	Element x is contained in set X.	
$\{ \vert \}$	$\{x \vert x = \varphi(x)\}$	All elements of set x satisfy the equation.	
C^k	$f(x) \in C^k[a, b]$	$f(x)$ and up to k-th derivatives are continuous in the interval [a , b].	

Note: This table defines how each symbol is used in this guide. A symbol may have a different meaning elsewhere. Commonly used symbols, such as + and – , were not included in the list.

PART I
GENERAL DESCRIPTION

CHAPTER 1
SSL II OUTLINE

1.1 BACKGROUND OF DEVELOPMENT

Many years have passed since SSL (Scientific Subroutine Library) was first developed. Due to advancements in numerical calculation techniques and to increased power of computers, SSL has been updated and new functions have been added on many occasions. However, users have further requested the followings:

- Better balance of the functions and uses of individual subroutines
- That addition of new functions not adversely affect the organization of the system
- Better documentation of various functions and their uses

SSLII was developed with these requirements in mind.

1.2 DEVELOPMENT OBJECTIVES

Systematizing
It is important for a library to enable the user to quickly identify subroutines which will suit his purposes. SSLII is organized with emphasis on the following points:

- We classify numerical computations as follows:
  A Linear algebra
  B Eigenvalues and eigenvectors
  C Nonlinear equations
  D Extrema
  E Interpolations and approximations
  F Transforms
  G Numerical differentiation and quadrature
  H Differential equations
  I Special functions
  J Pseudo random numbers

These categories are further subdivided for each branch. The library is made in a hierarchy organization. The organization allows easier identifying the locations of individual subroutines.

- Some branches have subdivided functions. We present not only general purpose-oriented subroutines but also those which perform as components of the former, so that the user can use the components when he wishes to analize the sequence of the computational procedures.

Performance improvement
Through algorithmic and programming revisions, improvements have been made both in accuracy and speed.

- The algorithmic methods which are stable and precise are newly adopted. Some of the standard methods used in the past are neglected.
- In programming, importance is attached to reducing execution time. Thus, the subroutines, are written in FORTRAN to enjoy the optimization of the compiler.
  SSLII improves the locality of the virtual storage system program, but does not decrease the efficiency of a computer without virtual storage.

Improvement of reliability
In most cases, single and double precision routines are generated from the same source program.

Maintenance of compatibility
Nowadays, developed softwares are easily transferred between different type systems. The SSL II subroutines are structured to maintain compatibility. A few auxiliary subroutines which are dependent of the system are created.

1.3 FEATURES

- SSL II is a collection of subroutines written in FORTRAN and desired subroutines are called in user programs using the CALL statement.
- All subroutines are written without input statements, user data is assumed to be in main storage.
- Data size is defined with subroutine parameters. No restrictions are applied to data size within subroutines.
- To save storage space for linear and eigenvalue-

eigenvector calculus, symmetric and band matrices are stored in a compressed mode (see Section 2.8 “Data Storage Methods”).

- All subroutines have an output parameter which indicates status after execution. A code giving the state of the processing is returned with this parameter (refer to Section 2.6 “Return Conditions of Processing”). Subroutines will always return control to the calling program. The user can check the contents of this parameter to determine proper processing.
- If specified, the condition messages are output (refer to section 2.6 “return conditions of processing”).

1.4 SYSTEM THAT CAN USE SSL II

If the FORTRAN compilers can be used on the user’s system, SSL II can also be used regardless of the system configuration. But, the storage size depends on the number and size of SSL II subroutines, the size of the user program, and the size of the data.
Although, as shown above, SSL II subroutines are usually called by FORTRAN programs, they can also be called by programs written in ALGOL, PL/1, COBOL, etc., if the system permits.
When the user wishes to do that, refer to the section in FORTRAN (or another compiler) User’s Guide which describes the interlanguage linkage.

CHAPTER 2
GENERAL RULES

2.1 TYPES OF SUBROUTINES

There are three types of SSL II subroutines, as shown in Table 2.1.

Table 2.1 Types of subroutines

Subroutine type	Subprogram division	Use
General subroutine	Subroutine subprogram	Used by the user.
Slave subroutine	Subroutine subprogram of function subprogram	Called by general subroutines and cannot be called directly by the user.
Auxiliary subroutine		Support general subroutines and slave subroutines.

The general subroutines in Table 2.1 are further divided, as shown in Table 2.2, into two levels according to the function. This division occurs when, in order to performs a particular function, a routine is internally subdivided into elementary functions.

Table 2.2 Subroutine levels

Level	Function
Standard routine	A single unified function is performed, for instance, when solving a system of linear equations.
Component routine	An elementary function is performed, for instance, for triangular factoring of a coefficient matrix. Several component routines are grouped to make a standard routine.

2.2 CLASSIFICATION CODES

Each of SSL II general subroutines has the eleven character classification codes according to the conventions in Fig. 2.1.

2.3 SUBROUTINE NAMES

Each of SSL II subroutines has the inherent subroutine name according to the subroutine type on following conventions.

General subroutine names
Names begin with S or D depending on the working precision, as shown in Fig. 2.2.

Slave subroutine names
- For subroutine subprograms or real function subprograms
  Names begin with the working precision identifier which is same as in general subroutines followed by a letter 'U' as shown in Fig. 2.3.
- For complex function subprograms
  Names begin with letter 'Z'. The other positions are similar to those shown above. See Fig. 2.4.

Auxiliary subroutines
Auxiliary subroutines are appropriately named according to their functions, see Appendix A for details.

2.4 PARAMETERS

Transmission of data between SSL II subroutines and user programs is performed only through parameters. This section describes the types of SSL II parameters, their order, and precautions.

Usage of parameters
- Input/output parameters
  These parameters are used for supplying values to the subroutine on input, and the resultant values are returned in the same areas on output.

- Input parameters
  These parameters are used for only supplying values to the subroutine on input. The values are unchanged on output. If values are changed by subroutine, it is described as "The contents of

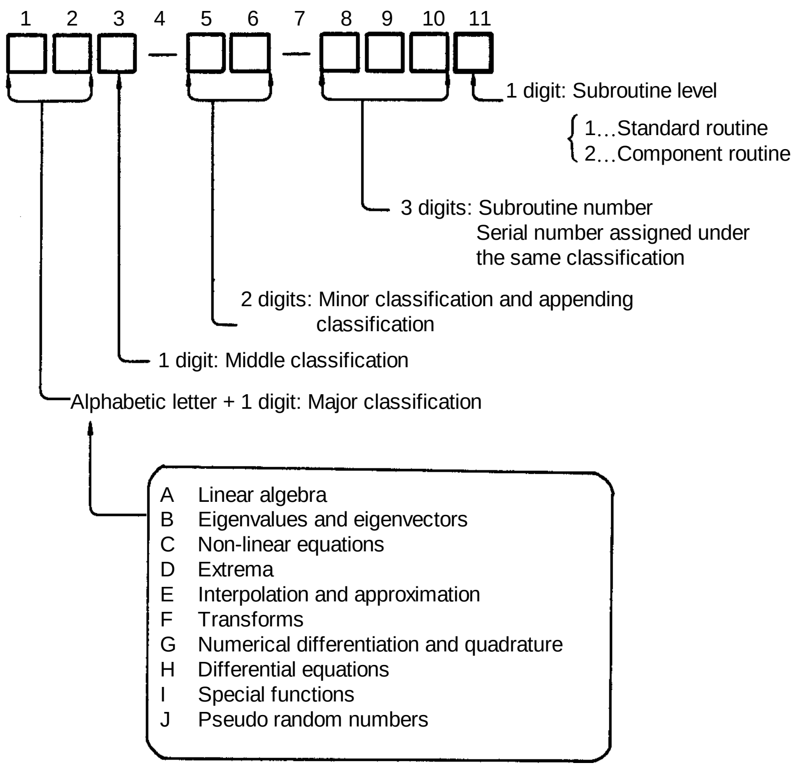

1 2 3 4 5 6 7 8 9 10 11
1 digit: Subroutine level
1...Standard routine
2...Component routine
3 digits: Subroutine number
Serial number assigned under the same classification
2 digits: Minor classification and appending classification
1 digit: Middle classification
Alphabetic letter + 1 digit: Major classification
A Linear algebra
B Eigenvalues and eigenvectors
C Non-linear equations
D Extrema
E Interpolation and approximation
F Transforms
G Numerical differentiation and quadrature
H Differential equations
I Special functions
J Pseudo random numbers

Fig. 2.1 Classification code layout

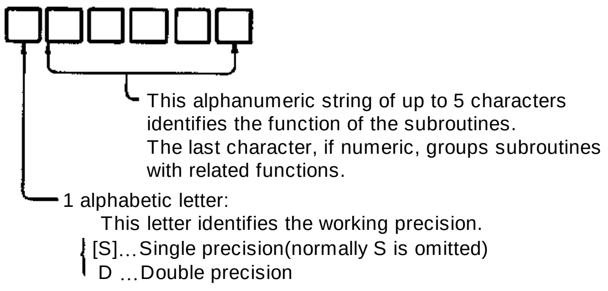

This alphanumeric string of up to 5 characters identifies the function of the subroutines.
The last character, if numeric, groups subroutines with related functions.
1 alphabetic letter:
This letter identifies the working precision.
[S]...Single precision(normally S is omitted)
D ...Double precision

Fig. 2.2 General subroutine names

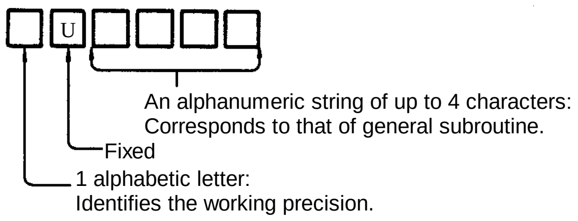

U
An alphanumeric string of up to 4 characters:
Corresponds to that of general subroutine.
Fixed
1 alphabetic letter:
Identifies the working precision.

Fig. 2.3 Subroutine names of slave subroutines (1)

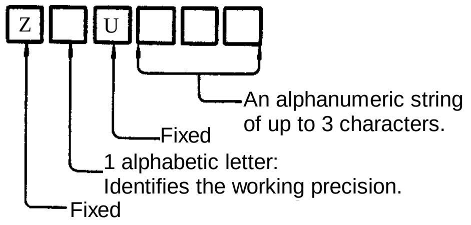

Z
U
An alphanumeric string of up to 3 characters.
Fixed
1 alphabetic letter:
Identifies the working precision.
Fixed

Fig. 2.4 Subroutine names of slave subroutines (2)

parameter are altered on output" in parameter description.

- Output parameters
  The resultant values are returned on output.

- Work parameters
  These are used as work areas. In general, the contents of these parameters are meaningless on output.

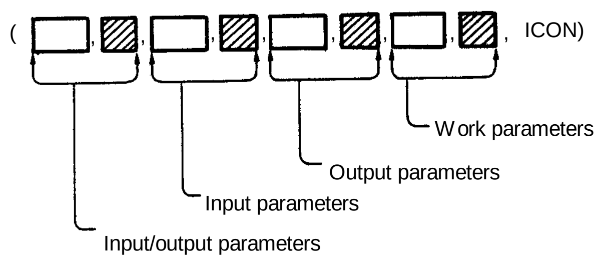
In addition, parameters can be also classified as follows:

- Main parameters
  These parameters contain the data which is used as the object of numerical calculations (for example, the elements of matrices).

- Control parameters
  These parameters contain data which is not used as the object of numerical calculations (for example, the order of matrices, decision values, etc.).

Order of parameter

Generally, parameters are ordered according to their kind as shown in Fig. 2.5.



Note:
The unshaded blocks indicate main parameters: the shaded blocks indicate control parameters. The ICON parameter indicates the return conditions of processing.

Fig. 2.5 Parameter ordering

Some control parameters cannot conform to Fig. 2.5 (for instance, adjustable dimension of array), therefore the explanation of each subroutine gives the actual ordering.

Handling of parameters

- Type of parameter
  Type of parameter conforms to the 'implicit typing' convention of FORTRAN except parameters that begin with the letter 'Z'. For instance, A is a 4 byte real number (8 byte real number in double precision subroutines) and IA is a standard-byte-length integer. When complex data is handled with complex variables, Z is the first letter of the parameter. ZA is an 8-byte complex number (16-byte complex number in double precision subroutines).

- External procedure name
  When external procedure names are specified for parameters, those names must be declared with an EXTERNAL statement in the user program which calls the SSL II subroutines.

- Status after execution
  SSL II subroutines have a parameter named ICON which indicates the return conditions of processing. See Section 2.6.

2.5 DEFINITIONS

Matrix classifications

Matrices handled by SSL II are classified as shown in Table 2.3.

Table 2.3 Matrix classification

Factors	Classifications
Structure	• Dense matrix • Band matrix
Form	• Symmetric matrix • Unsymmetric matrix
Type	• Real matrix • Complex matrix
Character	• Positive definite • Non singular • Singular

Portion names of matrix

In SSL II, the portion names of matrix are defined as shown in Fig. 2.6. The portion names are usually used for collective reference of matrix elements.
Where, the elements of the matrix are referred to as a_{ij}.

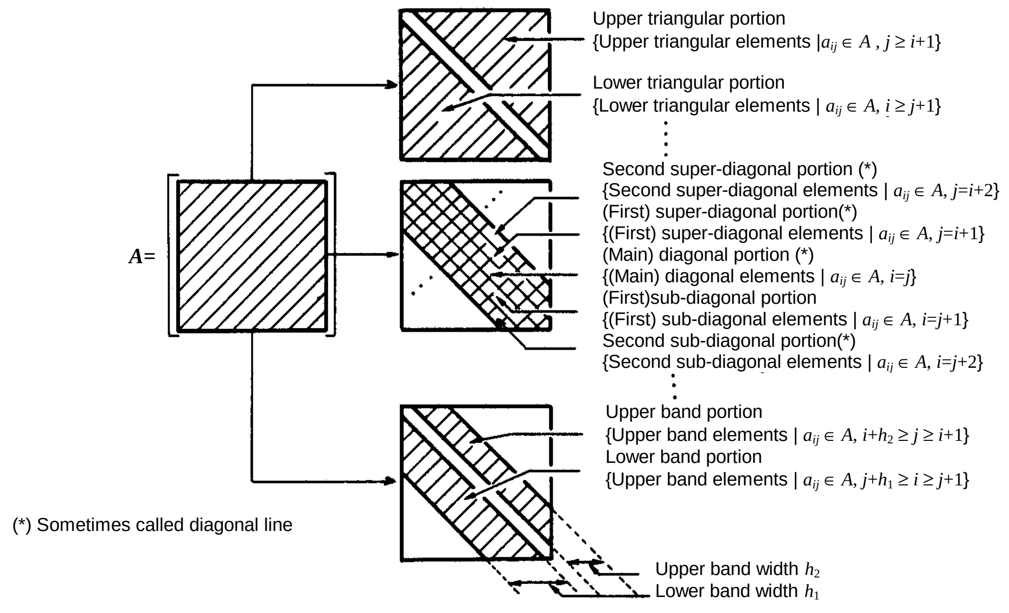

Upper triangular portion
{Upper triangular elements $|a_{ij} \in A, j \geq i+1\}$
Lower triangular portion
{Lower triangular elements $| a_{ij} \in A, i \geq j+1\}$
$A=$
Second super-diagonal portion (*)
{Second super-diagonal elements $| a_{ij} \in A, j=i+2\}$
(First) super-diagonal portion(*)
{(First) super-diagonal elements $| a_{ij} \in A, j=i+1\}$
(Main) diagonal portion (*)
{(Main) diagonal elements $| a_{ij} \in A, i=j\}$
(First)sub-diagonal portion
{(First) sub-diagonal elements $| a_{ij} \in A, i=j+1\}$
Second sub-diagonal portion(*)
{Second sub-diagonal elements $| a_{ij} \in A, i=j+2\}$
Upper band portion
{Upper band elements $| a_{ij} \in A, i+h_2 \geq j \geq i+1\}$
Lower band portion
{Upper band elements $| a_{ij} \in A, j+h_1 \geq i \geq j+1\}$
(*) Sometimes called diagonal line
Upper band width h_2
Lower band width h_1

Fig. 2.6 Portion names of matrix

Matrix definition and naming
Matrices handled by SSL II have special names depending on their construction.

- Upper triangular matrix
  The upper triangular matrix is defined as

$$a_{ij} = 0, \quad j < i \tag{2.1}$$

Namely, the elements of the lower triangular portion are zero.

- Unit upper triangular matrix
  The unit upper triangular matrix is defined as

$$a_{ij} = \begin{cases} 1, & j = i \\ 0, & j < i \end{cases} \tag{2.2}$$

Namely, this matrix is the same as an upper triangular matrix whose diagonal elements are all 1.

- Lower triangular matrix
  The lower triangular matrix is defined as

$$a_{ij} = 0, \quad j > i \tag{2.3}$$

Namely, the elements of the upper triangular portion are zero.

- Unit lower triangular matrix
  The unit lower triangular matrix is defined as

$$a_{ij} = \begin{cases} 1, & j = i \\ 0, & j > i \end{cases} \tag{2.4}$$

This matrix is the same as a lower triangular matrix whose diagonal elements are all 1.

- Diagonal matrix
  The diagonal matrix is defined as

$$a_{ij} = 0, \quad j \neq i \tag{2.5}$$

Namely, all of the elements of the lower and the upper triangular portions are zero.

- Tridiagonal matrix
  The tridiagonal matrix is defined as

$$a_{ij} = \begin{cases} 0, & j < i-1 \\ 0, & j > i+1 \end{cases} \tag{2.6}$$

Namely, all of the elements except for ones of the upper and lower sub-diagonal and main-diagonal portions are zero.

- Block diagonal matrix
  Considering an $n_i \times n_j$ matrix $\boldsymbol{A}_{ij}$ (i.e. a block) within an $n \times n$ matrix $\boldsymbol{A}$, where $n = \sum n_i = \sum n_j$, then the block diagonal matrix is defined as

$$\boldsymbol{A}_{ij} = 0, \quad i \neq j \tag{2.7}$$

In other words, all the blocks are on the diagonal line so that the block diagonal matrix is represented by a direct sum of those blocks.

- Hessenberg matrix
  The Hessenberg matrix is defined as

$$a_{ij} = 0, \quad j < i-1 \tag{2.8}$$

Namely, all of the elements except for ones of the upper triangular, the main-diagonal and the lower sub-diagonal portions are zero.

- Symmetric band matrix
  The symmetric band matrix whose both upper and lower band widths are h is defined as

$$a_{ij} = \begin{cases} 0, & |i-j| > h \\ a_{ji}, & |i-j| \le h \end{cases} \tag{2.9}$$

Namely, all of the elements except for ones of the diagonal, upper and lower band portions are zero.

- Band matrix
  The band matrix whose upper band width is h_1, and lower band width is h_2 is defined as

$$a_{ij} = \begin{cases} 0, & j > i + h_2 \\ 0, & i > j + h_1 \end{cases} \tag{2.10}$$

Namely, all of the elements except for ones of the diagonal, the upper and lower band portions are zero.

- Upper band matrix
  The upper band matrix whose upper band width is h is defined as

$$a_{ij} = \begin{cases} 0, & j > i + h \\ 0, & j < i \end{cases} \tag{2.11}$$

Namely, all of the elements except for ones of the diagonal and upper band portions are zero.

- Unit upper band matrix
The unit upper band matrix whose upper band width is h is defined as

$$a_{ij} = \begin{cases} 1, & j = i \\ 0, & j > i + h \\ 0, & j < i \end{cases} \tag{2.12}$$

This matrix is the same as an upper band matrix whose diagonal elements are all 1.

- Lower band matrix
  The lower band matrix whose lower band width is h is defined as

$$a_{ij} = \begin{cases} 0, & j < i - h \\ 0, & j > i \end{cases} \tag{2.13}$$

Namely, all of the elements except for ones of the diagonal and lower band portions are zero.

- Unit lower ban matrix
  The unit lower band matrix whose lower band width is h is defined as

$$a_{ij} = \begin{cases} 1, & j = i \\ 0, & j < i - h \\ 0, & j > i \end{cases} \tag{2.14}$$

Namely, this matrix is the same as a lower band matrix whose diagonal elements are all 1.

- Hermitian matrix
  The hermitian matrix is defined as

$$a_{ji} = a_{ij}^{*} \tag{2.15}$$

Namely, this matrix equals its conjugate transpose matrix.

2.6 RETURN CONDITIONS OF PROCESSING

The SSL II subroutines have a parameter called ICON which indicates the conditions of processing. The subroutine returns control with a condition code set in ICON. The values of the condition code should be tested before using the results.

Condition codes
The code value is a positive integer values that ranges from 0 to 30000. The code values are classified as shown in Table 2.4.
Each subroutine has appropriate codes that are described in the condition code table given in the section where each subroutine description is.

Table 2.4 Condition codes

Code	Meaning	Integrity of the result	Status
0	Processing has ended normally.	The results are correct.	Normal
1 ~ 9999	Processing has ended normally. Auxiliary information was given.		
10000 ~ 19999	Restrictions were employed during execution in order to complete the processing.	The results are correct on the restrictions.	Caution
20000 ~ 29999	Processing was aborted due to abnormal conditions which had occurred during processing.	The results are not correct.	Abnormal
30000	Processing was aborted due to invalid input parameters.		

Comments about condition codes

- Processing control by code testing
  The condition code had better be tested in the user's program immediately after the statement which calls the SSL II subroutine. Then, the processing should be controlled depending on whether the results are correct or not.

```
     :
CALL LSX (A, N, B, EPSZ, ISW, ICON)
IF (ICON. GE. 20000) STOP
     :
```

- Output of condition messages
  The SSL II subroutines have a function to output condition messages. Normally, these messages are not output. When the user uses message output control routine MGSET (SSL II auxiliary subroutine) messages are automatically output.

2.7 ARRAY

SSL II uses arrays to process vectors or matrices. This section describes the arrays used as parameters and their handling.
Adjustable arrays, whose sizes are declared in an array declaration in the program, are used by SSL II. The user may prepare arrays the size of which are determined corresponding to the size of the data processed in the user program.

One-dimensional arrays
When the user stores n (= N) - dimensional vector $\boldsymbol{b}$ ($=(b_1, ..., b_n)^T$) in a one-dimensional array B of size N,

as shown below, various ways are possible. In examples below, an attention should be paid to the parameter B used in calling the subroutine LAX, which solves systems of linear equations ($n \le 10$ is assumed).

- A general example
  The following describes the case in which a constant vector $\boldsymbol{b}$ is stored in a one-dimensional array B of size 10, as B(1) = $\boldsymbol{b}_1$, B(2) = $\boldsymbol{b}_2$, ...

```
    DIMENSION B(10)
        :
    CALL LAX ( … , N, B, … )
        :
```

- An application example
  The following describes the case in which a constant vector $\boldsymbol{b}$ is stored in the I-th column of a two-dimensional array C (10, 10), such that C (1, I) = b_1, C (2, I) = b_2, ...

```
    DIMENSION C(10, 10)
        :
    CALL LAX ( … , N, C(1, I), … )
        :
```

As shown in the above example, in parameter B, if a leading element (leading address) of one array in which the data is stored consecutively is specified it is not constrained to a one-dimensional array of size N. Therefore, if vector b is stored in I-th row of array C as C(I, 1) = b_1 C (I, 2) = b_2... it is impossible to call as follows.

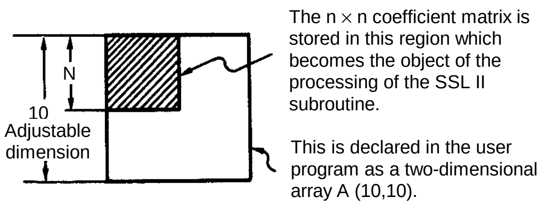
```
        :
    CALL LAX ( … , N, C(I, 1), … )
        :
```

Two-dimensional arrays
Consider an $n \times n$ real matrix A (=(a_{ij})) being stored in a two-dimensional array A (K, N). Note that in handling two dimensional arrays in SSL II subroutines, adjustable dimension K is required as an input parameter in addition to the array A and the order N. The adjustable dimension used by SSL II means the number of rows K of two-dimensional array A declared in the user program. An example using a two-dimensional array along with an adjustable dimension K is shown next. The key points of the example are parameters A and K of the subroutine LAX. (Here $n \le 10$).

The following describes the case in which coefficient matrix is stored in a two-dimensional array A (10, 10) as shown in Fig. 2.7, as A (1, 1) = a_{11}, A (2, 1) = a_{21}, ..., A (1, 2) = a_{12} ....

```
    DIMENSION A (10, 10)
        :
    K = 10
    CALL LAX (A, K, N, … )
        :
```

In this case, regardless of the value of N, the adjustable dimension must be given as K = 10.


The n × n coefficient matrix is stored in this region which becomes the object of the processing of the SSL II subroutine.
N
10
Adjustable dimension
This is declared in the user program as a two-dimensional array A (10,10).
Fig. 2.7 Example of a two-dimensional adjustable array

When equations of different order are to be solved, if the largest order is NMAX, then a two-dimensional array A (NMAX, NMAX) should be declared in the user program. That array can then be used to solve all of the sets of equations. In this case, the value of NMAX must always be specified as the adjustable dimension.

2.8 DATA STORAGE

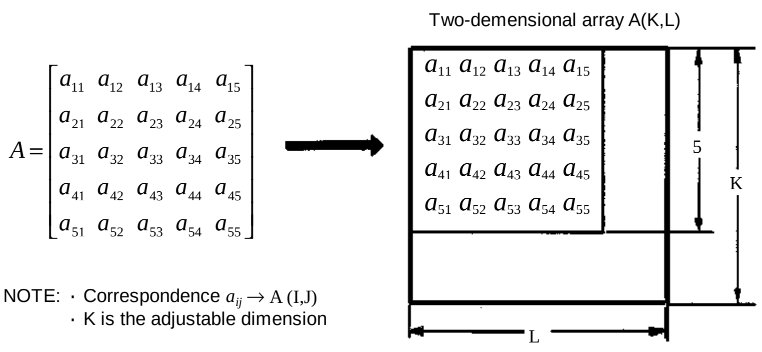
This section describes data storage modes for matrices or vectors.

Matrices
The methods for storing matrices depend on structure and form of the matrices. All elements of unsymmetric dense matrices, called general matrices, are stored in two-dimensional arrays. For all other matrices, only necessary elements can be stored in a one-dimensional array. The former storage method is called “general mode,” the latter “compressed mode.” Detailed definition of storage modes are as follows.

- General mode
  General mode is shown in Fig. 2.8.

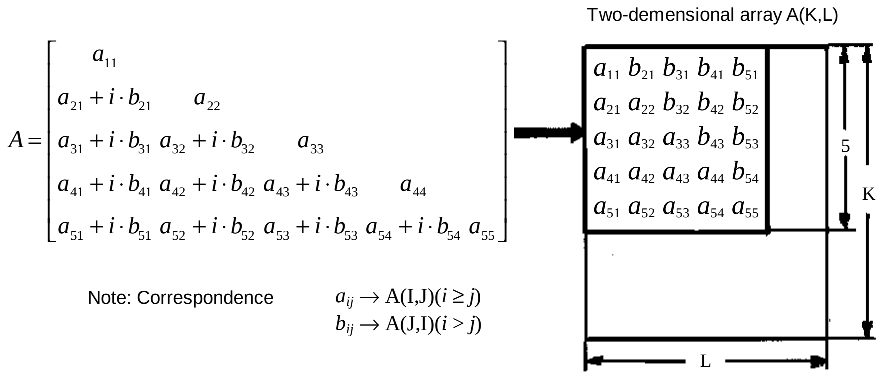
- Compressed mode for symmetric matrix
  As shown in Fig. 2.9, the elements of the diagonal and the lower triangular portions of the symmetric dense matrix $\boldsymbol{A}$ are stored row by row in the one-dimensional array A.

- Compressed mode for Hermitian matrix
  The elements of the diagonal and the lower triangular portions of the Hermitian matrix $\boldsymbol{A}$ are stored in a two-dimensional array A as shown in Fig. 2.10.


Two-demensional array A(K,L)
$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$
5
K
L

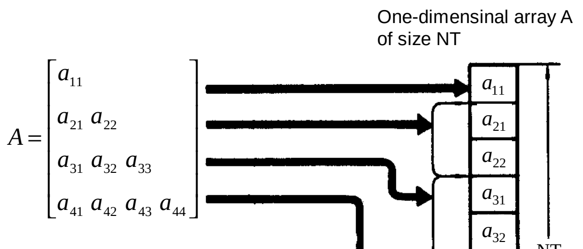
NOTE: · Correspondence $a_{ij} \rightarrow$ A (I,J)
· K is the adjustable dimension

Fig. 2.8 Storage of unsymmetric dense matrices


Two-demensional array A(K,L)
$A=\begin{bmatrix} a_{11} & & & & \\ a_{21}+i\cdot b_{21} & a_{22} & & & \\ a_{31}+i\cdot b_{31} & a_{32}+i\cdot b_{32} & a_{33} & & \\ a_{41}+i\cdot b_{41} & a_{42}+i\cdot b_{42} & a_{43}+i\cdot b_{43} & a_{44} & \\ a_{51}+i\cdot b_{51} & a_{52}+i\cdot b_{52} & a_{53}+i\cdot b_{53} & a_{54}+i\cdot b_{54} & a_{55} \end{bmatrix}$
a_{11} b_{21} b_{31} b_{41} b_{51}
a_{21} a_{22} b_{32} b_{42} b_{52}
a_{31} a_{32} a_{33} b_{43} b_{53}
a_{41} a_{42} a_{43} a_{44} b_{54}
a_{51} a_{52} a_{53} a_{54} a_{55}
5
K
L

Note: Correspondence $a_{ij} \rightarrow$ A(I,J)$(i \geq j)$
$b_{ij} \rightarrow$ A(J,I)$(i > j)$

Fig. 2.10 Storage of Hermitian matrices


One-dimensinal array A of size NT
$A=\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$
a_{11} a_{21} a_{22} a_{31} a_{32} a_{33} a_{41} a_{42} a_{43} a_{44}
NT

Note : Correspondence

$$a_{ij} \rightarrow A\left(\frac{I(I-1)}{2}+J\right)$$

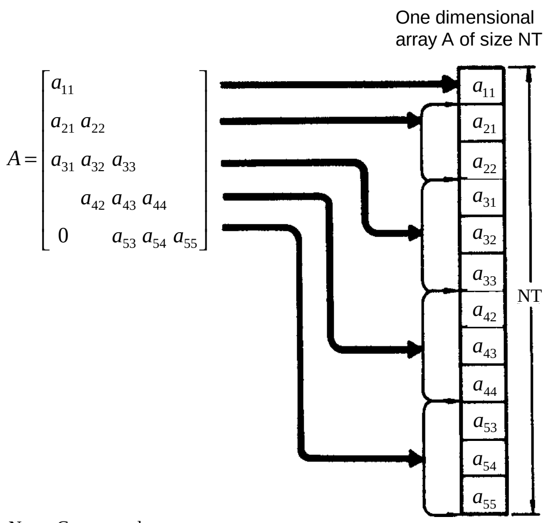
The value of NT is

$$NT=\frac{n(n+1)}{2}$$

Where n : order of matrix

Fig. 2.9 Storage of symmetric dense matrices

- Compressed mode for symmetric band matrix
  The elements of the diagonal and the lower band portions of a symmetric band matrix A are stored row by row in a one-dimensional array *A* as shown in Fig. 2.11.
- Compressed mode for band matrix
  The elements of the diagonal and the upper and lower band portions of an unsymmetric band matrix *A* are stored row by row in a one-dimensional array *A* as shown in Fig. 2.12.


One dimensional array A of size NT
$A=\begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ & a_{42} & a_{43} & a_{44} & \\ 0 & & a_{53} & a_{54} & a_{55} \end{bmatrix}$
a_{11} a_{21} a_{22} a_{31} a_{32} a_{33} a_{42} a_{43} a_{44} a_{53} a_{54} a_{55}
NT

Note : Correspondence

For $i \leq h+1$, $a_{ij} \rightarrow A(I(I-1)/2+J)$

For $j > h+1$, $a_{ij} \rightarrow A(hI+J-h(h+1)/2)$

The value of NT is

NT $= n(h+1)-h(h+1)/2$

Where n = orderof matrix

h = bandwidth

Fig. 2.11 Storage of symmetric band matrices

Vectors

Vector is stored as shown in Fig. 2.13.

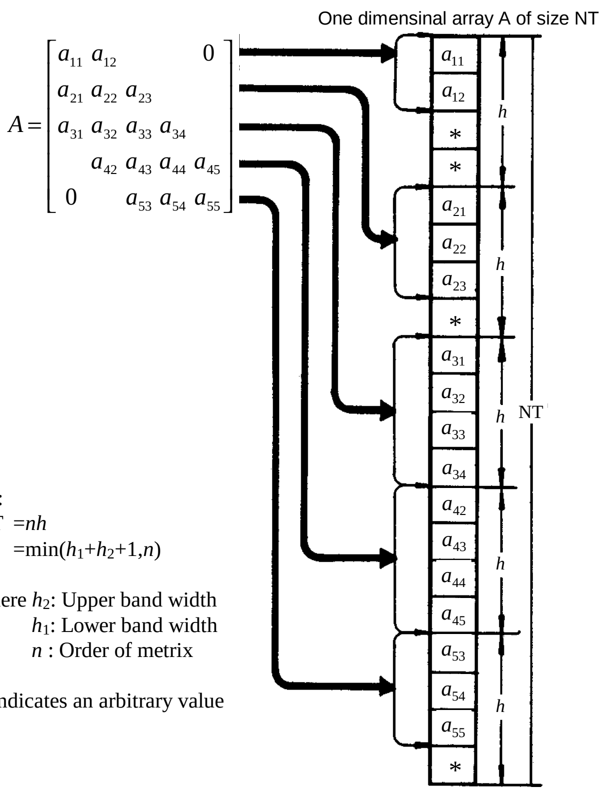

One dimensinal array A of size NT
$A=\begin{bmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ 0 & & a_{53} & a_{54} & a_{55} \end{bmatrix}$
a_{11} a_{12} * * a_{21} a_{22} a_{23} * a_{31} a_{32} a_{33} a_{34} a_{42} a_{43} a_{44} a_{45} a_{53} a_{54} a_{55} *
h
NT
Note:
· NT =nh
h =min(h_1+h_2+1,n)
where h_2: Upper band width
h_1: Lower band width
n : Order of metrix
· * Indicates an arbitrary value

Fig. 2.12 Storage of unsymmetric band matrices

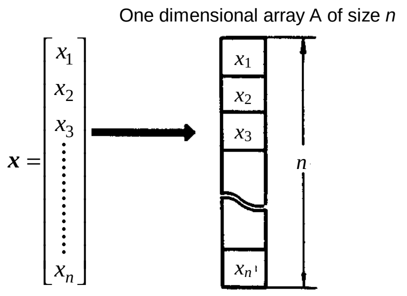

One dimensional array A of size n
$\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$
x_1 x_2 x_3 x_n
n

Fig. 2.13 Storage of vectors

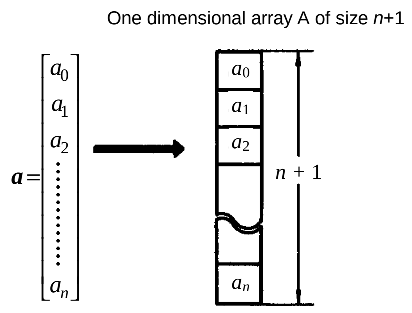

One dimensional array A of size n+1
$\boldsymbol{a}=\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$
a_0 a_1 a_2 a_n
$n + 1$

Fig. 2.14 Storage of the coefficients of polynomial equations

Coefficients of polynomial equations

The general form of polynomial equation is shown in (2.16)

$$a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n = 0 \tag{2.16}$$

Regarding the coefficients as the elements of a vector, the vector is stored as shown in Fig. 2.14.

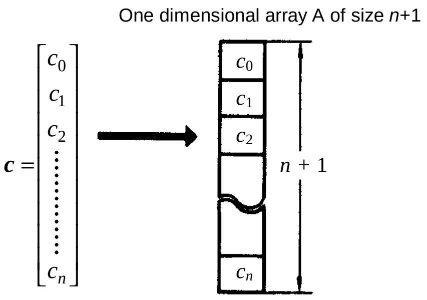

One dimensional array A of size n+1
$\boldsymbol{c}=\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$
c_0 c_1 c_2 c_n
$n + 1$

Fig. 2.15 Storage of the coefficients of approximating polynomials

Coefficients of approximating polynomials

The general form of approximating polynomial is shown in (2.17).

$$P_n(x) = c_0 + c_1x + c_2x^2 + ... + c_nx^n \tag{2.17}$$

Regarding the coefficients as the elements of a vector, the vector is stored as shown in Fig. 2.15.

2.9 UNIT ROUND OFF

SSL II subroutines frequently use the unit round off.
The unit round off is a basic concept in error analysis for floating point arithmetic.

Definition
The unit round off of floating-point arithmetic are defined as follows:

$u = M^{1-L} / 2$, for (correctly) rounded arithmetic
$u = M^{1-L}$, for chopped arithmetic,

where M is the base for a floating-point number system, and L is the number of digits used to hold the mantissa.

In SSL II, the unit round off is used for convergence criterion or testing loss of significant figures.
Error analysis for floating point arithmetic is covered in the following references:

[1] Yamashita, S.
On the Error Estimation in Floating-point Arithmetic
Information Processing in Japan Vol. 15, PP.935-939, 1974

[2] Wilkinson, J.H.
Rounding Errors in Algebraic Process
Her Britannic Majesty's Stationery Office, London 1963

2.10 ACCUMULATION OF SUMS

Accumulation of sums is often used in numerical calculations. For instance, it occurs in solving a system of linear equations as sum of products, and in calculations of various vector operations.
On the theory of error analysis for floating point arithmetic, in order to preserve the significant figures during the operation, it is important that accumulation of sums must be computed as exactly as possible.
As a rule, in SSL II the higher precision accumulation is used to reduce the effect of round off errors.

2.11 Computer Constants

This manual uses symbols to express computer hardware constants. The symbols are defined as follows:

- fl_{max}: Positive maximum value for the floating-point number system
(See AFMAX in Appendix A.)
- fl_{min}: Positive minimum value for the floating-point number system
(See AFMIN in Appendix A.)
- t_{max}: Upper limit of an argument for a trigonometric function (sin and cos)

Upper limit of argument		Application
Single precision	8.23×10^{5}	FACOM M series FACOM S series SX/G 100/200 series FM series
Double precision	3.53×10^{15}	

CHAPTER 3
LINEAR ALGEBRA

3.1 OUTLINE

Operations of linear algebra are classified as in Table 3.1 depending on the structure of the coefficient matrix and related problems.

Table 3.1 Classification of operation for linear equations

Structures	Problem		Item
Conversion of matrix storage mode			3.2
Dense matrix	Matrix manipulation		3.3
	Systems of linear equations Matrix inversion		3.4
	Least squares solution		3.5
Band matrix	Matrix manipulation		3.3
	Systems of linear equations	Direct method	3.4

This classification allows selecting the most suitable solution method according to the structure and form of the matrices when solving system of linear equations. The method of storing band matrix elements in memory is especially important. It is therefore important when performing linear equations to first determine the structure of the matrices.

3.2 MATRIX STORAGE MODE CONVERSION

In mode conversion the storage mode of a matrix is converted as follows:

$$\begin{tikzcd}
& \text{Real general matrix} \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\text{Real symmetric matrix} \arrow[rr, leftrightarrow] & & \text{Real symmetric band matrix}
\end{tikzcd}$$

The method to store the elements of a matrix depends on the structure and form of the matrix.
For example, when storing the elements of a real symmetric matrix, only elements on the diagonal and lower triangle portion are stored. (See Section 2.8).
Therefore, to solve systems of linear equations, SSL II provides various subroutines to handle different matrices. The mode conversion is required when the subroutine assumes a particular storage mode. The mode conversion subroutines shown in Table 3.2 are provided for this purpose.

3.3 MATRIX MANIPULATION

In manipulating matrices, the following basic manipulations are performed.

Table 3.2 Mode conversion subroutines

Before conversion \ After conversion	General mode	Compressed mode for symmetric matrices	Compressed mode for symmetric band matrices
General mode		CGSM (A11-10-0101)	CGSBM (A11-40-0101)
Compressed mode for symmetric matrices	CSGM (A11-10-201)		CSSBM (A11-50-0101)
Compressed mode for symmetric band matrices	CSBGM (A11-40-0201)	CSBSM (A11-50-0201)	

Table 3.3 Subroutines for matrix manipulation

A \ ***B*** or ***x***		Real general matrix	Real symmetric matrix	Vector
Real general matrix	Addition	AGGM (A21-11-0101)		
	Subtraction	SGGM (A21-11-0301)		
	Multiplication	MGGM (A21-11-0301)	MGSM (A21-11-0401)	MAV (A21-13-0101)
Complex general matrix	Multiplication			MCV (A21-15-0101)
Real symmetric matrix	Addition		ASSM (A21-12-0101)	
	Subtraction		SSSM (A21-12-0201)	
	Multiplication	MSGM (A21-12-0401)	MSSM (A21-12-0301)	MSV (A21-14-0101)
Real general band matrix	Multiplication			MBV (A51-11-0101)
Real symmetric band matrix	Multiplication			MSBV (A51-14-0101)

- Addition/Subtraction of two matrices $\boldsymbol{A} \pm \boldsymbol{B}$
- Multiplication of a matrix by a vector $\boldsymbol{Ax}$
- Multiplication of two matrices $\boldsymbol{AB}$

SSL II provides the subroutines for matrix manipulation, as listed in Table 3.3.

Comments on use

These subroutines for multiplication of matrix by vector are designed to obtain the residual vector as well, so that the subroutines can be used for iterative methods for linear equations.

3.4 LINEAR EQUATIONS AND MATRIX INVERSION (DIRECT METHOD)

This section describes the subroutines that is used to solve the following problems.

- Solve systems of linear equations

  $\boldsymbol{Ax} = \boldsymbol{b}$

  $\boldsymbol{A}$ is an $n \times n$ matrix, $\boldsymbol{x}$ and $\boldsymbol{b}$ are n-dimensional vectors.
- Obtain the inverse of a matrix $\boldsymbol{A}$.
- Obtain the determinant of a matrix $\boldsymbol{A}$.

In order to solve the above problems, SSL II provides the following basic subroutines (here we call them component subroutines) for each matrix structure.

(a) Numeric decomposition of a coefficient matrix
(b) Solving based on the decomposed coefficient matrix
(c) Iterative refinement of the initial solution
(d) Matrix inversion based on the decomposed matrix

Combinations of the subroutines ensure that systems of linear equations, inverse matrices, and the determinants can be obtained.

- Linear equations
  The solution of the equations can be obtained by calling the component routines consecutively as follows:

```
    :
CALL Component routine from (a)
CALL Component routine from (b)
    :
```

- Matrix inversion
  The inverse can be obtained by calling the above components routines serially as follows:

```
    :
CALL Component routine from (a)
CALL Component routine from (b)
    :
```

The inverse of band matrices generally result in dense matrices so that to obtain such the inverse is not beneficial. That is why those component routines for the inverse are not prepared in SSL II.

- Determinants
  There is no component routine which computes the values of determinant. However, the values can be obtained from the elements resulting from decomposition (a).

Though any problem can be solved by properly combining these component routines, SSL II also has routines, called standard routines, in which component routines are called internally. It is recommended that the user calls these standard routines.
Table 3.4 lists the standard routines and component routines.

Comments on uses

- Iterative refinement of a solution
  In order to refine the accuracy of solution obtained by standard routines, component routines (c) should be successively called regarding the solution as the initial approximation. In addition, the component routine (c) serves to estimate the accuracy of the initial approximation.

- Matrix inversion
  Usually, it is not advisable to invert a matrix when solving a system of linear equations.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.1}$$

That is, in solving equations (3.1), the solution should not be obtained by calculating the inverse $\boldsymbol{A}^{-1}$ and then multiplying $\boldsymbol{b}$ by $\boldsymbol{A}^{-1}$ from the left side as shown in (3.2).

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{3.2}$$

Instead, it is advisable to compute the LU-decomposition of $\boldsymbol{A}$ and then perform the operations (forward and backward substitutions) shown in (3.3).

$$\left.\begin{array}{l} \boldsymbol{Ly} = \boldsymbol{b} \\ \boldsymbol{Ux} = \boldsymbol{y} \end{array}\right. \tag{3.3}$$

Higher operating speed and accuracy can be attained by using method (3.3). The approximate number of multiplications involved in the two methods (3.2) and (3.3) are compared in (3.4).

$$\begin{array}{ll} \text{For (3.2)} & n^3 + n^2 \\ \text{For (3.3)} & n^3/3 \end{array} \tag{3.4}$$

Therefore, matrix inversion should only be performed when absolutely necessary.

- Equations with identical coefficient matrices
  When solving a number of systems of linear equations as in (3.5) where the coefficient matrices are the identical and the constant vectors are the different,

$$\left.\begin{array}{l} \boldsymbol{Ax}_1 = \boldsymbol{b}_1 \\ \boldsymbol{Ax}_2 = \boldsymbol{b}_2 \\ \quad\vdots \\ \boldsymbol{Ax}_m = \boldsymbol{b}_m \end{array}\right\} \tag{3.5}$$

it is not advisable to decompose the coefficient $\boldsymbol{A}$ for each equation. After decomposing $\boldsymbol{A}$ when solving the first equation, only the forward and backward substitution shown in (3.3) should be performed for solving the other equations. In standard routines, a parameter ISW is provided for the user to control whether or not the equations is the first one to solve with the coefficient matrix.

Notes and internal processing

When using subroutines, the following should be noted from the viewpoint of internal processing.

- Crout's method, modified Cholesky's method
  In SSL II, the Crout's method is used for decomposing a general matrix. Generally the Gaussian elimination and the Crout's method are known for decomposing (as in (3.6)) general matrices.

$$\boldsymbol{A} = \boldsymbol{LU} \tag{3.6}$$

where: $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is an upper triangular matrix.

The two methods require the different calculation order, that is, the former is that intermediate values are calculated during decomposition and all elements are available only after decomposition. The latter is that each element is available during decomposition. Numerically the latter involves inner products for two vecters, so if that calculations can be performed precisely, the final decomposed matrices are more accurate than the former.
Also, the data reference order in the Crout's method is more localized during the decomposition process than in the Gaussian method. Therefore, in SSL II, the Crout's method is used, and the inner products are carried out minimizing the effect of rounding errors.
On the other hand, the modified Cholesky method is used for positive-definite symmetric matrices, that is, the decomposition shown in (3.7) is done.

Table 3.4 Standard and component routines

Problem Basic functions / Types of matrix	Standard routines	Component routines			
	Systems of linear equations	(a)	(b)	(c)	(d)
Real general matrix	LAX (A22-11-0101)	ALU (A22-11-0202)	LUX (A22-11-0302)	LAXR (A32-11-0401)	LUIV (A22-11-0602)
Complex general matrix	LCX (A22-15-0101)	CLU (A22-15-0202)	CLUX (A22-15-0302)	LCXR (A22-15-0401)	CLUIV (A22-15-0602)
Real symmetric matrix	LSIX (A22-21-0101)	SMDM (A22-21-0202)	MDMX (A22-21-0302)	LSIXR (A22-21-0401)	
Real positive-definite symmetric matrix	LSX (A22-51-0101)	SLDL (A22-51-0202)	LDLX (A22-51-0302)	LSXR (A22-51-0401)	LDIV (A22-51-0702)
Real general band matrix [Real tridiagonal matrix]	LBX1 (A52-11-0101) [LTX (A52 - 11 - 0501)]	BLU1 (A52-11-0202)	BLUX1 (A52-11-0302)	LBX1R (A52-11-0401)	
Real symmetric band matrix	LSBIX (A52-21-0101)	SBMDM (A52-21-0202)	BMDMX (A52-21-0302)		
Real positive-definite symmetric band matrix [Positive - definite symmetric tridiagonal matrix]	LSBX (A52-31-0101) LSTX (A52-31-0501)	SBDL (A52-31-0202)	BDLX (A52-31-0302)	LSBXR (A52-31-0401)	

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \tag{3.7}$$

where: $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{D}$ is a diagonal matrix.

Matrices are decomposed as shown in Table 3.5.

Table 3.5 Decomposed matrices

Kinds of matrices	Contents of decomposed matrices
General matrices	$\boldsymbol{PA} = \boldsymbol{LU}$ $\boldsymbol{L}$: Lower triangular matrix $\boldsymbol{U}$: Unit upper triangular matrix $\boldsymbol{P}$ is a permutation matrix.
Positive-definite symmetric matrices	$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}}$ $\boldsymbol{L}$: Unit lower triangular matrix $\boldsymbol{D}$: Diagonal matrix To minimize calculation, the diagonal matrix is given as $\boldsymbol{D}^{-1}$

- Pivoting and scaling
  Let us take a look at decomposition of the non-singular matrix given by (3.6).

$$\boldsymbol{A} = \begin{bmatrix} 0.0 & 1.0 \\ 2.0 & 0.0 \end{bmatrix} \tag{3.8}$$

In this state, LU decomposition is impossible. And also in the case of (3.9)

$$\boldsymbol{A} = \begin{bmatrix} 0.0001 & 1.0 \\ 1.0 & 1.0 \end{bmatrix} \tag{3.9}$$

Decomposing by floating point arithmetic with the only three working digits (in decimal) will cause unstable solutions. These unfavorable conditions can frequently occur when the condition of a matrix is not proper. This can be avoided by pivoting, which selects the element with the maximum absolute value for the pivot.

In the case of (3.9), problems can be avoided by exchanging the first row with the second row.

In order to perform pivoting, the method used to select the maximum absolute value must be unique. By multiplying all of the elements of a row by a large enough constant, any absolute value of non-zero element in the row can be made larger than those of elements in the other rows.

Therefore, it is just as important to equilibrate the rows and columns as it is to validly determine a pivot element of the maximum size in pivoting. SSL II uses partial pivoting with row equilibration.

The row equilibration is performed by scaling so that the maximum absolute value of each row of the matrix becomes 1. However, actually the values of the elements are not changed in scaling; the scaling factor is used when selecting the pivot.

Since row exchanges are performed in pivoting, the history data is stored as the transposition vector. The matrix decomposition which accompanies this partial pivoting can be expressed as;

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{3.10}$$

Where $\boldsymbol{P}$ is the permutation matrix which performs row exchanges.

- Transposition vectors
  As mentioned above, row exchange with pivoting is indicated by the permutation matrix ***P*** of (3.10) in SSL II. This permutation matrix ***P*** is not stored directly, but is handled as a transposition vector. In other words, in the *j*-th stage (j = 1, ... , n) of decomposition, if the *i*-th row ($i \geq j$) is selected as the *j*-th pivotal row, the *i*-th row and the *j*-th row of the matrix in the decomposition process are exchanged and the *j*-th element of the transposition vector is set to *i*.

- How to test the zero or relatively zero pivot
  In the decomposition process, if the zero or relatively zero pivot is detected, the matrix can be considered to be singular. In such a case, proceeding the calculation might fail to obtain the accurate result. In SSL II, parameter EPSZ is used in such a case to determine whether to continue or discontinue processing. In other words, when EPSZ is set to 10^{-s} and, if a loss of over s significant digits occurred, the pivot might be considered to be relatively zero.

- Iterative refinement of a solution
  To solve a system of linear equations numerically means to obtain only an approximate solution to:

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.11}$$

SSL II provides a subroutine to refine the accuracy of the obtained approximate solution.
SSL II repeats the following calculations until a certain convergence criterion is satisfied.

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{3.12}$$
$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{3.13}$$
$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{3.14}$$

where s = 1, 2, ... and $\boldsymbol{x}^{(1)}$ the initial solution obtained from (3.11).
The $\boldsymbol{x}^{(2)}$ could become the exact solution of (3.11) if all the computations from (3.12) through (3.14) were carried out exactly, as can be seen by the following.

$$\boldsymbol{Ax}^{(2)} = \boldsymbol{A}\left(\boldsymbol{x}^{(1)} + \boldsymbol{d}^{(1)}\right) = \boldsymbol{Ax}^{(1)} + \boldsymbol{r}^{(1)} = \boldsymbol{b}$$

Actually, however, rounding errors will be generated in the computation of Eqs. (3.12), (3.13) and (3.14).
If we assume that rounding errors occur only when the correction $\boldsymbol{d}^{(s)}$ is computed in Eq. (3.13), $\boldsymbol{d}^{(s)}$ can be regarded to be the exact solution of the following equation

$$(\boldsymbol{A} + \boldsymbol{E})\boldsymbol{d}^{(s)} = \boldsymbol{r}^{(s)} \tag{3.15}$$

where E is an error matrix.

From Eqs. (3.12), (3.14) and (3.15), the following relationships can be obtained.

$$\boldsymbol{x}^{(s+1)} - \boldsymbol{x} = \left[\boldsymbol{I} - (\boldsymbol{A}+\boldsymbol{E})^{-1}\boldsymbol{A}\right]^s \left(\boldsymbol{x}^{(1)} - \boldsymbol{x}\right) \tag{3.16}$$
$$\boldsymbol{r}^{(s+1)} = \left[\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}+\boldsymbol{E})^{-1}\right]^s \boldsymbol{r}^{(1)} \tag{3.17}$$

As can be seen from Eqs. (3.16) and (3.17), if the following conditions are satisfied, $\boldsymbol{x}^{(s+1)}$ and $\boldsymbol{r}^{(s+1)}$ converge to the solutions ***X*** and 0, respectively, as s→∞.

$$\left\|\boldsymbol{I} - (\boldsymbol{A}+\boldsymbol{E})^{-1}\boldsymbol{A}\right\|_\infty, \left\|\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}+\boldsymbol{E})^{-1}\right\|_\infty < 1$$

In other words, if the following condition is satisfied, an refined solution can be obtained.

$$\|\boldsymbol{E}\|_\infty \cdot \left\|\boldsymbol{A}^{-1}\right\|_\infty < 1/2 \tag{3.18}$$

This method described above is called the iterative refinement of a solution
The SSL II repeats the computation until when the solution can be refined by no more than one binary digit.
This method will not work if the matrix ***A*** conditions are poor. $\left\|\boldsymbol{A}^{-1}\right\|_\infty$ may become large, so that no refined solution will be obtained.

- Accuracy estimation for approximate solution
  Suppose $\boldsymbol{e}^{(1)}$ (= $\boldsymbol{x}^{(1)} - \boldsymbol{x}$) is an error generated when the approximate solution $x^{(1)}$ is computed to solve a linear equations. The relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$.
  From the relationship between $\boldsymbol{d}^{(1)}$ and $\boldsymbol{e}^{(1)}$, and (3.12), (3.15), we obtain Eq.(3.19).

$$\boldsymbol{d}^{(1)} = (\boldsymbol{A}+\boldsymbol{E})^{-1}\left(\boldsymbol{b} - \boldsymbol{Ax}^{(1)}\right) = \left(\boldsymbol{I} + \boldsymbol{A}^{-1}\boldsymbol{E}\right)^{-1}\boldsymbol{e}^{(1)} \tag{3.19}$$

If the iterative method satisfies the condition (3.18) and converges, $\left\|\boldsymbol{d}^{(1)}\right\|_\infty$ is assumed to be almost equal to $\left\|\boldsymbol{e}^{(1)}\right\|_\infty$. Consequently, a relative error for an approximate solution can be estimated by $\left\|\boldsymbol{d}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$.

3.5 LEAST SQUARES SOLUTION

Here, the following problems are handled:
- Least squares solution
- Least squares minimal norm solution
- Generalized inverse
- Singular value decomposition

SSL II provides the subroutines for the above functions as listed in Table 3.6.

Table 3.6 Subroutines for $m \times n$ matrices

Function \ Kinds of matrix	Real matrix
Least squares solution	LAXL (A25-11-0101)
Iterative refinement of the least squares solution	LAXLR (A25-11-0401)
Least squares minimal norm solution	LAXLM (A25-21-0101)
Generalized inverse	GINV (A25-31-0101)
Singular value decomposition	ASVD1 (A25-31-0201)

- Least squares solution

  The least squares solution means the solution $\tilde{\boldsymbol{x}}$ which minimizes $\|\boldsymbol{Ax}-\boldsymbol{b}\|_2$, where $\boldsymbol{A}$ is an $m \times n$ matrix ($m \geq n$, rank ($\boldsymbol{A}$) = n), $\boldsymbol{x}$ is an n-dimensional vector and $\boldsymbol{b}$ is an m-dimensional vector.

  SSL II has two subroutines which perform the following functions.

  – Obtaining the least squares solution

  – Iterative refinement of the least squares solution

- Least squares minimal norm solution

  The least squares minimal norm solution means the solution $\boldsymbol{x}^+$ which minimizes $\|\boldsymbol{x}\|_2$ within the set of $\boldsymbol{x}$ where $\|\boldsymbol{Ax}-\boldsymbol{b}\|_2$ has been minimized, where $\boldsymbol{A}$ is an $m \times n$ matrix, $\boldsymbol{x}$ is an n-dimensional vector and $\boldsymbol{b}$ is an m-dimensional vector.

- Generalized inverse

  When $n \times m$ matrix $\boldsymbol{X}$ satisfies the following equations in (3.20) for a given $m \times n$ matrix $\boldsymbol{A}$, it is called a Moor-Penrose generalized inverse of the matrix $\boldsymbol{A}$.

$$\left.\begin{aligned} \boldsymbol{AXA} &= \boldsymbol{A} \\ \boldsymbol{XAX} &= \boldsymbol{X} \\ (\boldsymbol{AX})^{\mathrm{T}} &= \boldsymbol{AX} \\ (\boldsymbol{XA})^{\mathrm{T}} &= \boldsymbol{XA} \end{aligned}\right\} \qquad (3.20)$$

The generalized inverse always exists uniquely. The generalized inverse $\boldsymbol{X}$ of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^+$

SSL II calculates the generalized inverse as above.

SSL II can handle any $m \times n$ matrices where:

– m is larger than n

– m is equal to n

– m is smaller than n

- Singular value decomposition

  Singular value decomposition is obtained by decomposing a real matrix $\boldsymbol{A}$ of $m \times n$ as shown in (3.21).

$$\boldsymbol{A} = \boldsymbol{U}_0\boldsymbol{\Sigma}_0\boldsymbol{V}^{\mathrm{T}} \qquad (3.21)$$

Here $\boldsymbol{U}_0$ and $\boldsymbol{V}$ are orthogonal matrices of $m \times m$ and $n \times n$ respectively, $\boldsymbol{\Sigma}_0$ is an $m \times n$ diagonal matrix where $\boldsymbol{\Sigma}_0$=diag(σ_i) and $\sigma_i \geq 0$. The σ_i are called singular values of a real matrix $\boldsymbol{A}$. Suppose $m \geq n$ in real matrix $\boldsymbol{A}$ with matrix $m \times n$. Since $\boldsymbol{\Sigma}_0$ is an $m \times n$ diagonal matrix, the first n columns of $\boldsymbol{U}_0$ are used for $\boldsymbol{U}_0\boldsymbol{\Sigma}_0\boldsymbol{V}^{\mathrm{T}}$ in (3.21). That is, $\boldsymbol{U}_0$ may be considered as an $n \times n$ matrix. Let $\boldsymbol{U}$ be this matrix, and let $\boldsymbol{\Sigma}$ be an $n \times n$ matrix consisting of matrix $\boldsymbol{\Sigma}_0$ without the zero part, $(m-n) \times n$, of $\boldsymbol{\Sigma}_0$. When using matrices $\boldsymbol{U}$ and $\boldsymbol{\Sigma}$, if m is far larger than n, the storage space can be reduced. So matrices $\boldsymbol{U}$ and $\boldsymbol{\Sigma}$ are more convenient than $\boldsymbol{U}_0$ and $\boldsymbol{\Sigma}_0$ in practice. The same discussion holds when $m < n$, in which case only the first m raws of $\boldsymbol{V}^{\mathrm{T}}$ are used and therefore $\boldsymbol{V}^{\mathrm{T}}$ can be considered as $m \times n$ matrix.

Considering above ideas SSL II performs the following singular value decomposition.

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{\mathrm{T}} \qquad (3.22)$$

where: l = min (m, n) is assumed and $\boldsymbol{U}$ is an $m \times l$ matrix, $\boldsymbol{\Sigma}$ is an $l \times l$ diagonal matrix where $\boldsymbol{\Sigma}$ = diag (σ_i), and $\sigma_i \geq 0$, and $\boldsymbol{V}$ is an $n \times l$ matrix.

When $l = n$ ($m \geq n$),

$$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{VV}^{\mathrm{T}} = \boldsymbol{I}_n$$

when $l = m$ ($m < n$),

$$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{UU}^{\mathrm{T}} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{I}_m$$

Matrices $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$ which are obtained by computing singular values of matrix $\boldsymbol{A}$ are used and described as follows;

(For details, refer to reference [86],)

– Singular values σ_i, i = 1, 2, ... , l are the positive square roots of the largest l eigenvalues of matrices $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{AA}^{\mathrm{T}}$. The i-th column of matrix $\boldsymbol{V}$ is the eigenvector of matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ corresponding to eigenvalue σ_i^2. The i-th column of matrix $\boldsymbol{U}$ is the eigenvector of matrix $\boldsymbol{AA}^{\mathrm{T}}$ corresponding to eigenvalue σ_i^2. This can be seen by multiplying $\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{V\Sigma U}^{\mathrm{T}}$ from the right and left sides of (3.22) and apply $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{I}_l$ as follows:

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{AV} = \boldsymbol{V\Sigma}^2 \qquad (3.23)$$

$$\boldsymbol{AA}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{U\Sigma}^2 \qquad (3.24)$$

– Condition number of matrix $\boldsymbol{A}$

If $\sigma_i > 0$, i = 1, 2, ..., l, then condition number of matrix $\boldsymbol{A}$ is given as follows;

$$\mathrm{cond}(\boldsymbol{A}) = \sigma_1 / \sigma_l \qquad (3.25)$$

– Rank of matrix $\boldsymbol{A}$

If $\sigma_r > 0$, and $\sigma_{r+1} = ... = \sigma_l = 0$, then the rank of $\boldsymbol{A}$ is given as follows:

$$\text{rank}(\boldsymbol{A}) = r \qquad (3.26)$$

– Basic solution of homogeneous linear equations $\boldsymbol{Ax} = \boldsymbol{0}$ and $\boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{0}$
The set of non-trivial linearly independent solutions of $\boldsymbol{Ax} = \boldsymbol{0}$ and $\boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{0}$ is the set of those columns of matrices $\boldsymbol{V}$ and $\boldsymbol{U}$, respectively, corresponding to singular values $\sigma_i = 0$. This can be easily seen from equations $\boldsymbol{AV} = \boldsymbol{U\Sigma}$ and $\boldsymbol{A}^T\boldsymbol{U} = \boldsymbol{V\Sigma}$.

– Least squares minimal norm solution of $\boldsymbol{Ax} = \boldsymbol{b}$.
The solution $\boldsymbol{x}$ is represented by using singular value decomposition of $\boldsymbol{A}$;

$$\boldsymbol{x} = \boldsymbol{V\Sigma}^{+}\boldsymbol{U}^{T}\boldsymbol{b} \qquad (3.27)$$

where diagonal matrix $\boldsymbol{\Sigma}^{+}$ is defined as follows:

$$\Sigma^{+} = \text{diag}(\sigma_1^{+}, \sigma_2^{+}, ..., \sigma_l^{+}) \qquad (3.28)$$

$$\sigma_i^{+} = \begin{cases} 1/\sigma_i\ , & \sigma_i > 0 \\ 0 & , \sigma_i = 0 \end{cases} \qquad (3.29)$$

For details, refer to “Method” of subroutine LAXLM.

– Generalized inverse of a matrix
Generalized inverse $\boldsymbol{A}^{+}$ of $\boldsymbol{A}$ is given after decomposing $\boldsymbol{A}$ into singular value as follows;

$$\boldsymbol{A}^{+} = \boldsymbol{V\Sigma}^{+}\boldsymbol{U}^{T} \qquad (3.30)$$

Comments on uses

– Systems of linear equations and the rank of coefficient matrices
Solving the systems of linear equations ($\boldsymbol{Ax} = \boldsymbol{b}$) with an $m \times n$ matrix as coefficient, the least squares minimal norm solution can be obtained regardless of the number of columns or rows, or ranks of coefficient matrix $\boldsymbol{A}$. That is, the least squares minimal norm solution can be applied to any type of equations. However, this solution requires a great amount of calculation for each process. If the coefficient matrix is rectangular and number of rows is larger than that of columns and the rank is full (full rank, rank($\boldsymbol{A}$) = n), you should use the subroutine for least squares solution because of less calculation. In this case, the least squares solution is logically as least squares minimal norm solution.

• Least squares minimal norm solution and generalized inverse
The solution of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ with $m \times n$ matrix $\boldsymbol{A}$ ($m \geq n$ or $m < n$, rank($\boldsymbol{A}$) $\neq 0$) is not uniquely obtained. However, the least squares minimal norm solution always exists uniquely.
This solution can be calculated by $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b}$ after generalized inverse $\boldsymbol{A}^{+}$ of coefficient matrix $\boldsymbol{A}$ is obtained. This requires a great amount of calculation. It is advisable to use the subroutine for the least squares minimal norm solution, for the sake of high speed processing. This subroutine provides parameter ISW by which the user can specify to solve the equations with the same coefficient matrix with less calculation or to solve the single equation with efficiency.

• Equations with the identical coefficient matrix
The least squares solution or least squares minimal norm solution of a system of linear equations can be obtained in the following procedures;
– Decomposition of coefficient matrices
For the least squares solution a matrix $\boldsymbol{A}$ is decomposed into triangular matrices , and for the least squares minimal norm solution $\boldsymbol{A}$ is decomposed into singular values
– Obtaining solution
Backward substitution and multiplication of matrices or vectors are performed for the least-squares solution and the least squares minimal norm solution, respectively.

When obtaining the least squares solution or least squares minimal norm solution of a number of systems with the identical coefficient matrices, it is not advisable to repeat the decomposition for each system.

$$\begin{aligned} \boldsymbol{Ax}_1 &= \boldsymbol{b}_1 \\ \boldsymbol{Ax}_2 &= \boldsymbol{b}_2 \\ &: \\ \boldsymbol{Ax}_m &= \boldsymbol{b}_m \end{aligned}$$

In this case, the matrix needs to be decomposed only once for the first equations, and the decomposed form can be used for subsequent systems , thereby reducing the amount of calculation.
SSL II provides parameter ISW which can controls whether matrix $\boldsymbol{A}$ is decomposed or not.

• Obtaining singular values
The singular value will be obtained by singular value decomposition as shown in (3.31):

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{T} \qquad (3.31)$$

As shown in (3.31), matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are involved along with matrix $\boldsymbol{\Sigma}$ which consists of singular values. Since singular value decomposition requires a great amount of calculation, the user need not to calculate $\boldsymbol{U}$ and $\boldsymbol{V}$ when they are no use. SSL II provides parameter ISW to control whether matrices $\boldsymbol{U}$ or $\boldsymbol{V}$ should be obtained. SSL II can handle any type of $m \times n$ matrices ($m > n$, $m = n$, $m < n$).

CHAPTER 4
EIGENVALUES AND EIGENVECTORS

4.1 OUTLINE

Eigenvalue problems can be organized as show in Table 4.1 according to the type of problem ($\boldsymbol{Ax} = \lambda \boldsymbol{x}$, $\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$) and the shape (dense, band), type (real, complex), and form (symmetric, unsymmetric) of the matrices. The reader should refer to the appropriate section specified in the table.

Table 4.1 Organization of eigenvalue problem

Shape of matrix	Type of problem	Matrix type and form	Expla-nation section
Dense matrix	$\boldsymbol{Ax} = \lambda \boldsymbol{x}$	Real matrix Complex matrix Real symmetric matrix Hermitian matrix	4.2 4.3 4.4 4.5
	$\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$	Real symmetric matrix	4.7
Band matrix	$\boldsymbol{Ax} = \lambda \boldsymbol{x}$	Real symmetric band matrix	4.6
	$\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$	Real symmetric band matrix	4.8

Note:
Refer to the section on a real symmetric matrix concerning a real symmetric tridiagonal matrix.

4.2 EIGENVALUES AND EIGENVECTORS OF A REAL MATRIX

A standard sequenses of procedures are shown here when SSL II routines are used to solve the eigenvalue problems.

SSL II provides the following:

– Standard routines by which the entire procedures for obtaining eigenvalues and eigenvectors of real matrices may be performed at one time.

– Component routines performing component functions.

For details, see Table 4.2.

User problems are classified as follows :

- Obtaining all eigenvalues
- Obtaining all eigenvalues and corresponding eigenvectors (or selected eigenvectors)

In the first and second items that follow, the use of component routines and standard routines is explained by describing their procedures. Further comments on processing are in the third item.

When obtaining eigenvalues and eigenvectors of a real matrix, the choice of calling the various component routines or calling the standard routine is up to the user. However, the standard routine, which is easier to use, is recommended to be called.

Table 4.2 Subroutines used for standard eigenproblem of a real matrix

Level	Function	Subroutine name
Standard routines	Eigenvalues and eigenvectors of a real matrix	EIG1 (B21-11-0101)
Component routines	Balancing of a real matrix	BLNC (B21-11-0202)
	Reduction of a real matrix to a Hessenberg matrix	HES1 (B21-11-0302)
	Obtaining the eigenvalues of a real Hessenberg matrix	HSQR (B21-11-0402)
	Obtaining the eigenvectors of a real Hessenberg matrix	HVEC (B21-11-0502)
	Back transformation to and normalization of the eigenvectors of a real matrix	HBK1 (B21-11-0602)
	Normalization of the eigenvectors of a real matrix	NRML (B21-11-0702)

In addition, obtaining the eigenvectors corresponding to specified eigenvalues can only be done by calling a series of component routines.

Obtaining all eigenvalues
In the following programs, all eigenvalues of the real matrix ***A*** are obtained through the use of the component routines shown in steps 1), 2), 3).

```
       :
    CALL BLNC (A, K, N, DV, ICON)      1)
    IF (ICON .EQ. 30000) GO TO 1000
    CALL HES1 (A, K, N, PV, ICON)      2)
    CALL HSQR (A, K, N, ER, EI, M,ICON)
                                       3)
    IF (ICON .EQ. 20000) GO TO 2000
       :
```

1) ***A*** is balanced, if balancing is not necessary, this step is omitted.
2) Using the Householder method, ***A*** is reduced to a Hessenberg matrix.
3) By calculating the eigenvalues of the Hessenberg matrix using the double QR method, the eigenvalues of ***A*** are obtained.

Obtaining all eigenvalues and corresponding eigenvectors (or selected eigenvectors)
All eigenvalues and corresponding eigenvectors of real matrix ***A*** can be obtained by calling the component routines shown in 1) to 5) or by calling the standard routine shown in 6).

```
       :
    CALL BLNC (A, K, N, DV, ICON)      1)
    IF (ICON .EQ. 30000) GO TO 1000
    CALL HES1 (A, K, N, PV, ICON)      2)
    DO 10 I = 1, N
    DO 10 J = 1, N
    AW (J, I) = A (J, I)
 10 CONTINUE
    CALL HSQR (A, K, N, ER, EI, M,ICON)
                                       3)
    IF (ICON .GE. 20000) GO TO 2000
    DO 20 I = 1, M
    IND (I) = 1
 20 CONTINUE
    CALL HVEC (AW, K, N, ER, EI, IND,
   *M, EV, MK, VW, ICON)               4)
    IF (ICON .GE. 20000) GO TO 3000
    CALL HBK1 (EV, K, N, IND, M, A, PV,
   *DV, ICON)                          5)
       :
```

or standard routine.

```
       :
    CALL EIG1 (A, K, N, MODE, ER, EI, EV, VW, ICON)      6)
    IF(ICON .GE. 20000) GO TO 1000
       :
```

1), 2), and 3) were explained in the preceding example "Obtaining all eigenvalues". However since 3) destroys parameter A, A must be stored in array AW after 2) so that contents of A before 3) can be used in 4).

4) The eigenvectors corresponding to the eigenvalues are obtained using the inverse iteration method. The parameter IND is an index vector which indicates the eigenvalues from which eigenvectors are to be obtained. The user can select eigenvectors using this parameter.
5) Back transformation of the eigenvectors obtained in 4) is performed. The transformed eigenvectors are normalized at the same time.
   From the processing of 1) and 2), the eigenvectors of 4) are not the eigenvectors of real matrix ***A***. The eigenvectors of ***A*** are obtained by performing the post-processing corresponding to 1) and 2).
   Each column (i.e., each eigenvector) of parameter EV is normalized such that its Euclidean norm is 1.
6) When obtaining all eigenvalues and corresponding eigenvectors of a real matrix, the functions of 1) to 5) can be performed by calling this standard routine, however, instead of using the inverse iteration method, all eigenvectors are obtained at a time by multiplying all the transformation matrices obtained successively. This calculation will not work if even one eigenvalue is missing.
   However, if eigenvalues are close roots or multiple roots, eigenvectors can be determined more accurately using this method than the inverse iteration method. The standard routine 6) does not always perform the process in step 1). The parameter MODE can be used to specify whether step 1) is performed or not.

Balancing of matrices
Errors on calculating eigenvalues and eigenvectors can be reduced by reducing the norm of real matrix ***A***. Routine 1) is used for this purpose. By diagonal similarity transformation the absolute sum of row *i* and that of column *i* in ***A*** is made equal (this is called balancing).
Symmetric matrices and Hermitian matrices are already balanced. Since this method is especially effective when magnitudes of elements in *A* differ greatly, balancing should be done. Except in certain cases (i.e. when the number of the order of *A* is small), balancing should not take more than 10% of the total processing time.

4.3 EIGENVALUES AND EIGENVECTORS OF A COMPLEX MATRIX

An example will be used to show how the various SSL II subroutines obtain eigenvalues and eigenvectors of a complex matrix.

Table 4.3 Subroutines used for standard eigenproblem of a complex matrix

Level	Function	Subroutine name
Standard routine	Eigenvalues and eigenvectors of a complex matrix	CEIG2 (B21-15-0101)
Component routine	Balancing of a complex matrix	CBLNC (B21-15-0202)
	Reduction of a complex matrix to a complex Hessenberg matrix	CHES2 (B21-15-0302)
	Obtaining the eigenvalues of a complex Hessenberg matrix	CHSQR (B21-15-0402)
	Obtaining the eigenvectors of a complex Hessenberg matrix	CHVEC (B21-15-0502)
	Back transformation to the eigenvectors of a complex matrix	CHBK2 (B21-15-0602)
	Normalization of the eigenvectors of a complex matrix	CNRML (B21-15-0702)

SSL II provides the following:

− Standard routines by which the entire procedures for obtaining eigenvalues and eigenvectors of complex matrices may be performed at one time.

− Component routines performing component functions

For details, see Table 4.3

User problems are classified as follows:

- Obtaining all eigenvalues
- Obtaining all eigenvalues and corresponding eigenvectors (or selected eigenvectors)

In the first and second items that follow, the use of component routines and standard routines are explained by describing their procedures.

When obtaining all eigenvalues and corresponding eigenvectors whether the standard routines only or the various corresponding component routines are used is up to the user. However it is recommended to call the former for its easy handling.

Obtaining all eigenvalues

In the following programs, all eigenvalues of the complex matrix *A* are obtained through the use of the component routines shown in steps 1), 2), and 3).

```
      CALL CBLNC (ZA, K, N, DV, ICON)    1)
      IF (ICON. EQ. 30000) GO TO 1000
      CALL CHES2 (ZA, K, N, IP, ICON)    2)
      CALL CHSQR (ZA, K, N, ZE, M, ICON) 3)
      IF (ICON. GE. 15000) GO TO 2000
```

1) ***A*** is balanced. If balancing is not necessary, this step is omitted.
2) Using the stabilized elementary transformation, ***A*** is reduced to a complex Hessenberg matrix.
3) By calculating the eigenvalues of the complex Hessenberg matrix using the complex QR method, the eigenvalues of ***A*** are obtained.

Obtaining all eigenvalues and the corresponding eigenvectors (or selected eigenvectors)

All eigenvalues and the corresponding eigenvectors of a complex matrix *A* can be obtained by calling the component routines shown in 1) to 6) or by calling the standard routine shown in 7).

```
         :
      CALL CBLNC (ZA, K, N, DV, ICON)    1)
      IF (ICON. EQ. 30000) GO TO 1000
      CALL CHES2 (ZA, K, N, IP, ICON)    2)
      DO 10 J = 1, N
      DO 10 I = 1, N
      ZAW (I, J) = ZA (I, J)
   10 CONTINUE
      CALL CHSQR (ZA, K, N, ZE, M, ICON) 3)
      IF (ICON. GE. 15000) GO TO 2000
      DO 20 I = 1, M
      IND (I) = 1
   20 CONTINUE
      CALL CHVEC (ZAW, K, N, ZE, IND,
     *           M, ZEV, ZW, ICON)       4)
      IF (ICON. GE. 15000) GO TO 3000
      CALL CHBK2 (ZEV, K, N, IND, M, ZP,
     *           IP, DV, ICON)           5)
      CALL CNRML (ZEV, K, N, IND, M, 1,
     *ICON)                              6)
```

or standard routine

```
         :
      CALL CEIG2 (ZA, K, N, MODE, ZE,
     *ZEV, VW, IVW, ICON)                7)
      IF (ICON. GE. 20000) GO TO 1000
         :
```

1), 2), and 3) were explained in the preceding example "Obtaining all eigenvalues". However since 3) destroys parameter ZA, ZA must be stored in array ZAW after 2) so that contents of ZA before 3) can be used in 4) and the subsequent routines.

4) The eigenvectors of the complex Hessenberg matrix corresponding to the eigenvalues are obtained using the inverse iteration method. The parameter IND is an index vector which indicates the eigenvalues from which eigenvectors are to be obtained. The user can select eigenvectors using this parameter.
5) Back transformation of the eigenvectors obtained in 4) is performed to obtain the eigenvectors of ***A***.
6) The eigenvectors transformed in 5) are normalized. When normalizing the eigenvectors (set norm to 1), the user can select whether an Euclidean or infinity norm should be used. Here the former is applied for normalization.
7) When obtaining all eigenvalues and corresponding eigenvectors of a complex matrix, the functions of 1) through 6) can be performed by calling this standard

routine, however, instead of using the inverse iteration method, all eigenvectors are obtained at one time by multiplying all the obtained transformation matrices. This calculation will not work if even one of the eigenvalues is missing. The parameter MODE can be used to specify whether the matrix is to be balanced or not.

4.4 EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC MATRIX

An example will be used here to show how the various SSL II subroutines are used to obtain eigenvalues and eigenvectors of a real symmetric matrix.
SSL II provides the followings:

– Standard routines by which the entire procedures for obtaining eigenvalues and eigenvectors of a real symmetric matrix may be performed at one time.
– Component routines decomposed by functions.

For details, see Table 4.4.

The problems can be classified as follows:

- Obtaining all eigenvalues.
- Obtaining selected eigenvalues.
- Obtaining all eigenvalues and corresponding eigenvectors.
- Obtaining selected eigenvalues and corresponding eigenvectors.

In the above order, the first four items below explain the use of component routines and standard routines. The last two items provide additional information.
The choice of calling individual component routines or of calling a single standard routine for obtaining all eigenvalues and corresponding eigenvectors or selected eigenvalues and corresponding eigenvectors is up to the user. Normally, standard routines which are easier to use, are selected. Component routines TEIG1 or TEIG2 should be used if the real symmetric matrix is originally tridiagonal.
SSL II handles the real symmetric matrix in symmetric matrix compressed mode (For details, refer to Section 2.8).

Obtaining all eigenvalues
All eigenvalues of a real symmetric matrix *A* can be obtained as shown below 1) and 2). *A* is handled in the compressed storage mode, i.e., as a one-dimensional array, for a real symmetric matrix.

```
    :
CALL TRID1 (A, N, D, SD, ICON)     1)
IF (ICON .EQ. 30000) GO TO 1000
CALL TRQL (D, SD, N, E, M, ICON)   2)
    :
```

1) ***A*** is reduced to a tridiagonal matrix using the Householder method. Omit this step if ***A*** is already a tridiagonal matrix.
2) Using the QL method, the eigenvalues of ***A*** i.e. all the eigenvalues of the tridiagonal matrix are obtained.

Table 4.4 Subroutines used for standard eigenproblem of a real symmetric matrix

Level	Function	Subroutine name
Standard routine	Eigenvalues and eigenvectors of a real symmetric matrix	SEIG1 (B21-21-0101)
		SEIG2 (B21-21-0201)
Component routines	Reduction of a real symmetric matrix to a real symmetric tridiagonal matrix	TRID1 (B21-21-0302)
	Obtaining eigenvalues of a real symmetric tridiagonal matrix	TRQL (B21-21-0402)
		BSCT1 (B21-21-0502)
	Obtaining eigenvalues and eigenvectors of a real symmetric tridiagonal matrix	TEIG1 (B21-21-0602)
		TEIG2 (B21-21-0702)
	Back transformation to the eigenvectors of a real symmetric matrix	TRBK (B21-21-0802)

Obtaining selected eigenvalues
Selected eigenvalues of real symmetric matrix *A* can be obtained as shown:

```
      :
   CALL TRID1 (A, N, D, SD, ICON)      1)
   IF (ICON.EQ. 30000) GO TO 1000
   CALL BSCT1 (D, SD, N, M, EPST, E,
 * VW, ICON)                           2)
      :
```

1) Same as step 1) in the first item "Obtaining all eigenvalues".
2) The eigenvalues of the tridiagonal matrix are obtained using the Bisection method. Using the parameter M, the user specifies the number of eigenvalues to be determined by starting from the largest eigenvalue or starting from the smallest eigenvalue.
   If *n*/4 or more eigenvalues are to be determined for ***A***, it is faster to use the procedure explained in item "Obtaining all eigenvalues".

Obtaining all eigenvalues and corresponding eigenvectors
All eigenvalues and corresponding eigenvectors of real symmetric matrix *A* can be obtained by using 1) to 3) or by using step 4).

```
      :
   CALL TRID1(A,N,D,SD,ICON)           1)
   IF(ICON .EQ. 30000)GO TO 1000
   CALL TEIG1(D,SD,N,E,EV,K,M,ICON)    2)
   IF(ICON .GE. 20000)GO TO 2000
   CALL TRBK(EV,K,N,M,A,ICON)          3)
      :
```

or standard routine

```
      :
   CALL SEIG1(A,N,E,EV,K,M,VW,ICON)    4)
   IF(ICON .GE. 20000)GO TO 1000
      :
```

1) Same as step 1) in item "Obtaining all eigenvalues".
2) All eigenvalues and corresponding eigenvectors of the real symmetric tridiagonal matrix can be obtained by the QL method and by multiplying each of the transformation matrices obtained by the QL method. Each eigenvector is normalized such that the Euclidean norm is 1.
3) From step 1), the eigenvectors in 2) are not those of real symmetric matrix ***A***. Therefore the back transformation in 1) is performed to obtain the eigenvectors of real symmetric matrix ***A***.
4) All processing of 1) to 3) is performed.

Obtaining selected eigenvalues and corresponding eigenvectors
Selected eigenvalues and corresponding eigenvectors of a real symmetric matrix *A* can be obtained either by 1) to 3) or by 4).

```
      :
   CALL TRID1 (A,N,D,SD,ICON)          1)
   IF(ICON .EQ. 30000)GO TO 1000
   CALL TEIG2(D,SD,N,M,E,EV,K,VW,ICON)
                                       2)
   IF(ICON .GE. 20000)GO TO 2000
   CALL TRBK(EV,K,N,M,A,ICON)          3)
      :
```

or standard routine

```
      :
   CALL SEIG2(A,N,M,E,EV,K,VW,ICON)    4)
   IF(ICON .GE. 20000)GO TO 1000
      :
```

1) Same as step 1) in item "Obtaining all eigenvalues"
2) Selected eigenvalues and corresponding eigenvectors of a tridiagonal matrix are determined using the Bisection method and the Inverse Iteration method. If the eigenvalues are close, it is not always true that the eigenvectors obtained using the Inverse Iteration are orthogonal. Therefore in 2), the eigenvectors of close eigenvalues are corrected to orthogonalize to those which have already been obtained. The obtained eigenvectors are normalized such that each Euclidean norm is 1.
3) From processing 1), the eigenvectors of 2) are not those of a real symmetric matrix ***A***. Therefore the eigenvectors of a real symmetric matrix ***A*** are obtained by performing back transformation corresponding to 1).
4) This subroutine process steps 1) to 3).

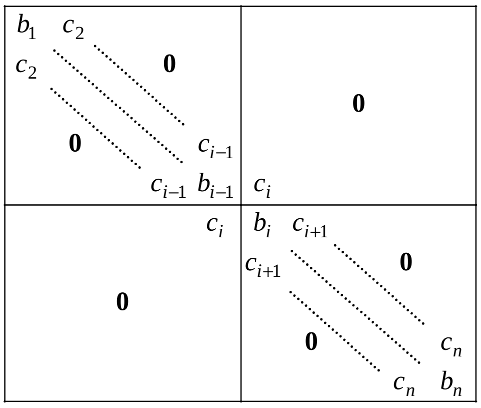
QL method
1) Comparison with the QR method
   The QL method used in 2) of the first and third paragraphs is basically the same as the QR method used to determine the eigenvalues of a real matrix. However, while the QR method determines eigenvalues from the lower right corner of matrices, the QL method determines eigenvalues from the upper left. The choice of these methods is based on how the data in the matrix is organized. The QR method is ideal when the magnitude of the matrix element is "graded decreasingly", i.e., decreases from the upper left to the lower right. If the magnitude of the matrix element is graded increasingly, the QL method is better. Normally, in tridiagonal matrix output by TRID1 in 1) of the first paragraph and 2) of the second paragraph, the magnitude of the element tends to be graded increasingly. For this reason, the QL method is used following TRID1 as in 2) of the first and third paragraphs.
2) Implicit origin shift of the QL method
   There are explicit and implicit QL methods. The difference between the two methods is whether origin shift for improving the rate of convergence when determining eigenvalues is performed explicitly or

implicitly. (See subroutine TRQL.) The explicit QL method is suitable when the magnitude of the matrix element is graded increasingly.
However, when the magnitude is graded decreasing, the precision of the smaller eigenvalues is affected.
For this reason, the implicit QL method is used in 2) of the first and third paragraphs.

Direct sum of submatrices
When a matrix is a direct sum of submatrices, the processing speed and precision in determining eigenvalues and eigenvectors increase if eigenvalues and eigenvectors are obtained from each of the submatrices. In each 2) of the first four paragraphs, a tridiagonal matrix is split into submatrices according to (4.1), and then the eigenvalues and eigenvectors are determined.

$$|c_i| \le u(|b_{i-1}| + |b_i|) \quad (i = 2,3,...,n) \qquad (4.1)$$

u is the unit round off; c_i, b_i are as shown in Fig. 4.1.



Note: Element c_i is regarded to be zero according to (4.1).

Fig. 4.1 Example in which a tridiagonal matrix is the direct sum of two submatrices

4.5 EIGENVALUES AND EIGENVECTORS OF A HERMITIAN MATRIX

The use of SSL II subroutines for obtaining eigenvalues and eigenvectors is briefly explained using standard examples.
The sequence to obtain eigenvalues and eigenvectors of a Hermitian matrix consists of the following four steps:
1) Reduction of a Hermitian matrix to real symmetric tridiagonal matrix.
2) Obtaining eigenvalues of the real symmetric tridiagonal matrix.
3) Obtaining eigenvectors of the real symmetric tridiagonal matrix.
4) Back transformation of the eigenvectors of the real symmetric tridiagonal matrix to form the eigenvectors of the Hermitian matrix.

SSL II provides component routines corresponding to steps 1) through 4), and a standard routine to do all the steps at one time. (See Table 4.5.)
The problems of an Hermite matrix is classified into the following categories:
- Obtaining all eigenvalues
- Obtaining selected eigenvalues
- Obtaining all eigenvalues and corresponding eigenvectors
- Obtaining selected eigenvalues and corresponding eigenvectors

In the following four paragraphs, the use of component routines and standard routine will be described for each objective category.
It is up to the user whether he uses the standard routine or each component routine to obtain all or selected eigenvalues and corresponding eigenvectors. Normally, using the standard routine is recommended since it is easier to use.
SSL II handles the Hermitian matrix in the compressed

Table 4.5 Subroutines used for standard eigenproblems of a Hermitian matrix

Level	Function	Subroutine name
Standard routine	Obtaining eigenvalues and eigenvectors of a Hermitian matrix	HEIG2 (B21-25-0201)
Component routines	Reduction of a Hermitian matrix to a real symmetric tridiagonal matrix	TRIDH (B21-25-0302)
	Obtaining eigenvalues of a real symmetric tridiagonal matrix	TRQL (B21-21-0402)
		BSCT1 (B21-21-0502)
	Obtaining eigenvalues and eigenvectors of a real symmetric tridiagonal matrix	TEIG1 (B21-21-0602)
		TEIG2 (B21-21-0702)
	Back transformation to the eigenvectors of a Hermitian matrix	TRBKH (B21-25-0402)

storage mode. (For details, see Section 2.8)

Obtaining all eigenvalues
All eigenvalues of a Hermitian matrix *A* can be obtained in steps 1) and 2) below.

```
      :
CALL TRIDH(A,K,N,D,SD,V,ICON)      1)
IF(ICON .EQ. 30000)GO TO 1000
CALL TRQL(D,SD,N,E,M,ICON)         2)
      :
```

1) A Hermitian matrix ***A*** is reduced to a real symmetric tridiagonal matrix using the Householder method.
2) All eigenvalues of the real symmetric tridiagonal matrix are obtained using the QL method.

Obtaining selected eigenvalues
By using steps 1) and 2) below, the largest (or smallest) *m* eigenvalues of a matrix ***A*** can be obtained.

```
      :
CALL TRIDH(A,K,N,D,SD,V,ICON)      1)
IF(ICON .EQ. 30000)GO TO 1000
CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON) 2)
      :
```

1) A Hermitian metrics ***A*** is reduced to a real symmetric tridiagonal matrix by the Householder method.
2) The largest (or smallest) *m* eigenvalues of the real symmetric tridiagonal matrix are obtained using the bisection method.

When obtaining more than *n*/4 eigenvalues of ***A*** of order *n*, it is generally faster to use subroutines TRIDH and TRQL as described in “Obtaining all eigenvalues”.

Obtaining all eigenvalues and corresponding eigenvectors
All eigenvalues and corresponding eigenvectors can be obtained either by using steps 1) through 3)or by using step 4), (see below).

```
      :
CALL TRIDH(A,K,N,D,SD,V,ICON)      1)
IF(ICON .EQ. 30000)GO TO 1000
CALL TEIG1(D,SD,N,E,EV,K,M,ICON)   2)
IF(ICON .GE. 20000)GO TO 2000
CALL TRBKH(EV,EVI,K,N,M,P,V,ICON)  3)
IF(ICON .EQ. 30000)GO TO 3000
      :
```

or standard routine

```
      :
 CALL HEIG2(A,K,N,N,E,EVR,EVI,VW,
*ICON)                             4)
 IF(ICON .GE. 20000)GO TO 1000
      :
```

1) A Hermitian matrix ***A*** is reduced to a real symmetric tridiagonal matrix
2) Eigenvalues of the real symmetric tridiagonal matrix (i.e., eigenvalues of ***A***) and corresponding eigenvectors are obtained using the QL method.
3) The Eigenvectors obtained in 2) are transformed to the eigenvectors of ***A***.
4) The standard routine HEIG2 can perform all the above steps 1) through 3). In this case, the fourth parameter N of HEIG2 indicates to obtain the largest *n* eigenvalues.

Obtaining selected eigenvalues and corresponding eigenvectors
A selected number of eigenvalues (*m*) and corresponding eigenvectors of a Hermitian matrix can be obtained either by using steps 1) through 3)or by using step 4), (see below).

```
      :
CALL TRIDH(A,K,N,D,SD,V,ICON)      1)
IF(ICON .EQ. 30000)GO TO 1000
CALL TEIG2(D,SD,N,M,E,EV,K,VW,ICON) 2)
IF(ICON .GE. 20000)GO TO 2000
CALL TRBKH(EV,EVI,K,N,M,P,V,ICON)  3)
IF(ICON .EQ. 30000)GO TO 3000
      :
```

or standard routine

```
      :
CALL HEIG2(A,K,N,M,E,EVR,EVI,VW,ICON)      4)
IF(ICON .GE. 20000)GO TO 1000
      :
```

1) A Hermitian matrix ***A*** is reduced to a real symmetric tridiagonal matrix.
2) The largest (or smallest) *m* eigenvalues and corresponding eigenvectors of the real symmetric tridiagonal matrix are obtained using the bisection method and the inverse iteration method.
3) Back transformation of the eigenvectors obtained in 2) are performed.
4) The standard routine HEIG2 can perform all the above steps 1) through 3).

4.6 EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC BAND MATRIX

Subroutines BSEG, BTRID, BSVEC and BSEGJ are provided for obtaining eigenvalues and eigenvectors of a real symmetric band matrix.

These subroutines are suitable for large matrices, for example, matrices of the order $n > 100$ and $h/n < 1/6$,

where *h* is the band-width. Subroutine BSEGJ, which uses the Jennings method, is effective for obtaining less than *n*/10 eigenvalues. Obtaining of all eigenvalues and corresponding eigenvectors of a real symmetric band matrix is not required in most cases and therefore only standard routines for some eigenvalues and corresponding eigenvectors are provided.
Subroutines BSEG and BSEGJ are standard routines and BTRID and BSVEC are component routines of BSEG (see Table 4.6). Examples of the use of these subroutines are given below. SSL II handles the real symmetric band matrix in compressed mode (see Section 2.8).

Obtaining selected eigenvalues

- Using standard routines

```
     :
  CALL BSEG(A,N,NH,M,0,EPST,E,EV,K,
* VW,ICON)                             1)
  IF(ICON.GE.20000)GO TO 1000
     :
```

1) The largest (or smallest) *m* eigenvalues of a real symmetric band matrix ***A*** of order *n* and bandwidth *h* are obtained. The fifth parameter, 0 indicates that no eigenvectors are required.

- Using component routines

```
     :
  CALL BTRID(A,N,NH,D,SD,ICON)1)
  IF(ICON.EQ.30000) GO TO 1000
  CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)  2)

  IF(ICON.EQ.30000) GO TO 2000
     :
```

1) Real symmetric band matrix ***A*** or order *n* and bandwidth *h* is reduced to the real symmetric tridiagonal matrix ***T*** by using the Rutishauser-Schwarz method.
2) The largest (or smallest) *m* eigenvalues of ***T*** are obtained.

Obtaining all eigenvalues

- Using the standard routine
  All the eigenvalues can be obtained by specifying N as the fourth parameter in the example of BSEG used to obtain some eigenvalues. However, the following component routines are recommended instead.

- Using component routines

```
     :
  CALL BTRID(A,N,NH,D,SD,ICON)        1)
  IF(ICON.EQ.30000) GO TO 1000
  CALL TRQL(D,SD,N,E,M,ICON)          2)
     :
```

1) Real symmetric band matrix ***A*** of order *n* and bandwidth *h* is reduced to the real symmetric tridiagonal matrix ***T*** by using the Rutishauser-Schwarz method.
2) All eigenvalues of ***T*** are obtained by using the QL method.

Obtaining selected eigenvalues and corresponding eigenvectors

- Using standard routines
  The following two standard routines are provided.

```
     :
  CALL BSEG(A,N,NH,M,NV,EPST,E,EV,K,
* VW,ICON)                             1)
  IF(ICON.GE.20000)GO TO 1000
     :
  CALL BSEGJ(A,N,NH,M,EPST,LM,E,EV,K,
* IT,VW,ICON)                         1)'
  IF(ICON.GE.20000)GO TO 1000
     :
```

The subroutine indicated by 1) obtains eigenvalues by using the Rutishauser-Schwarz method, the bisection method and the inverse iteration method consecutively. The subroutine indicated by 1)' obtains both eigenvalues and eigenvectors by using the Jennings method based on a simultaneous iteration.

Table 4.6 Subroutines used for standard eigenproblem of a real symmetric band matrix

Level	Function	Subroutine name
Standard routine	Obtaining eigenvalues and eigenvectors of a real symmetric band matrix	BSEG (B51-21-0201)
		BSEGJ (B51-21-1001)
Component routine	Reduction of a real symmetric band matrix to a tridiagonal matrix	BTRID (B51-21-0302)
	Obtaining eigenvalues of a real symmetric tridiagonal matrix	TRQL (B21-21-0402)
		BSCT1 (B21-21-0502)
	Obtaining eigenvectors of a real symmetric band matrix	BSVEC (B51-21-0402)

Further, 1) obtains the largest (or smallest) eigenvalue and 1)' obtains the largest (or smallest) absolute value of eigenvalues. The subroutine indicated by 1)' is only recommended where a very small number of eigenvalues and eigenvectors (no more than $n/10$) compared to the matrix order n are to be obtained.

1) The m eigenvalues and n_v, number of eigenvectors of a real symmetric band matrix $\boldsymbol{A}$ of order n and bandwidth h are obtained.

1)' Eigenvectors of $\boldsymbol{A}$ as described above are obtained based on the m initial eigenvectors given. At the same time, the corresponding eigenvalues can be also obtained. Care needs to be taken when giving initial eigenvectors to EV and upper limit for the number of iterations to LM.

- Using component routines

  The following subroutines are called consecutively to achieve the same effect as executing subroutine BSEG.

```
        :
     NN = (NH + 1)*(N + N -NH)/2
     DO 10 I = 1, NN
  10 AW (I) = A (I)
     CALL BTRID(A,N,NH,D,SD,ICON)1)
     IF(ICON.EQ.30000) GOTO 1000
     CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)
                                          2)
     IF(ICON.EQ.30000) GO TO 2000
     CALL BSVEC(AW,N,NH,NV,E,EV,K,VW,
    *ICON)                                3)
     IF(ICON.GE.20000) GO TO 3000
        :
```

1) 2) These are the same paths taken when obtaining selected eigenvalues. Since BTRID destroys the contents of parameter A, they need to be stored in array AW for step 3).

3) The eigenvectors corresponding to the first n_v eigenvalues of the m eigenvalues given by 2) are obtained by using the inverse iteration method.

Obtaining all eigenvalues and corresponding eigenvectors

- Using standard routines

  By specifying N for the fourth and fifth parameters of the subroutine BSEG shown earlier in the example on obtaining selected eigenvalues and their corresponding eigenvectors, all eigenvalues and corresponding eigenvectors can be obtained. If a solution is desired more quickly, the following path using component routines is recommended.

- Using component routines

```
        :
     NN=(N+1)*(N+N-NH)/2
     DO 10 I=1,NN
  10 AW(I)=A(I)
     N1 = NN+1
     N2 = N1 + N
     CALL BTRID(A,N,NH,VW(N1),VW(N2),
    *ICON)                                1)
     IF(ICON.EQ.30000) GO TO 1000
     CALL TRQL(VW(N1),VW(N2),N,E,M,ICON)
                                          2)
     IF(ICON.GE.15000) GO TO 2000
     CALL BSVEC(AW,N,NH,N,E,EV,K,VW,
    *ICON)                                3)
     IF(ICON.GE.20000) GO TO 3000
        :
```

1) 2) These are the same as the component routines used when obtaining all of the eigenvalues. Since subroutine BTRID destroys the contents of parameter A, they need to stored in array AW for step 3).

3) The eigenvectors corresponding to all eigenvalues given in step 2) are obtained by using the inverse iteration method.

4.7 EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC GENERALIZED EIGENPROBLEM

When obtaining eigenvalues and eigenvectors of $\boldsymbol{Ax}=\lambda\boldsymbol{Bx}$ ($\boldsymbol{A}$: symmetric matrix and $\boldsymbol{B}$**:** positive definite symmetric matrix), how each SSL II subroutine is used is briefly explained using standard examples.

The sequence to obtain eigenvalues and eigenvectors of a real symmetric matrix consists of the following six steps:

1) Reduction of the generalized eigenvalue problem ($\boldsymbol{Ax}=\lambda\boldsymbol{Bx}$) to the standard eigenvalue problem of a real symmetric matrix ($\boldsymbol{Sy}=\lambda\boldsymbol{y}$)
2) Reduction of the real symmetric matrix $\boldsymbol{S}$ to a real symmetric tridiagonal matrix $\boldsymbol{T}$($\boldsymbol{Sy}=\lambda\boldsymbol{y} \rightarrow \boldsymbol{Ty}'=\lambda\boldsymbol{y}'$).
3) Obtaining eigenvalue λ of the real symmetric tridiagonal matrix $\boldsymbol{T}$.
4) Obtaining eigenvector $\boldsymbol{y}'$ of the real symmetric tridiagonal matrix $\boldsymbol{T}$.
5) Back transformation of eigenvector $\boldsymbol{y}'$ of the real symmetric tridiagonal matrix $\boldsymbol{T}$ to eigenvector $\boldsymbol{y}$ of the real symmetric matrix $\boldsymbol{S}$.
6) Back transformation of eigenvector $\boldsymbol{y}$ of the real symmetric matrix $\boldsymbol{S}$ to eigenvector $\boldsymbol{x}$ of the generalized eigenproblem.

SSL II provides component routines corresponding to these steps and a standard routine to do all the steps at one time. (See Table 4.7.)

Practically, in this section, user's generalized eigenproblems of a real symmetric matrix are classified into the following categories:

- Obtaining all eigenvalues
- Obtaining selected eigenvalues
- Obtaining all eigenvalues and corresponding eigenvectors
- Obtaining selected eigenvalues and corresponding eigenvectors

Table 4.7 Subroutines used for generalized eigenproblems of a real symmetric matrix

Level	Function	Subroutine name
Standard routine	Obtaining general eigenvalue and eigenvector of a real symmetric matrix	GSEG2 (B22-21-0201)
Component routine	Reduction of the generalized eigenproblem to the standard eigenproblem	GSCHL (B22-21-0302)
	Reduction of a real symmetric matrix to a real symmetric tridiagonal matrix	TRID1 (B21-21-0302)
	Obtaining eigenvalues of a real symmetric tridiagonal matrix	TRQL (B21-21-0402)
		BSCT1 (B21-21-0502)
	Obtaining eigenvalues and eigenvectors of a real symmetric tridiagonal matrix	TEIG1 (B21-21-0602)
		TEIG2 (B21-21-0702)
	Back transformation to eigenvectors of a real symmetric matrix	TRBK (B21-21-0802)
	Back transformation to generalized eigenvectors	GSBK (B22-21-0402)

In the following paragraphs, the use of component routines and standard routine will be described. It is up to the user whether he successively calls the component routines one after another or uses the standard routine to obtain all or selected eigenvalues and eigenvectors. Normally, using the latter routine is recommended since that method is easier to use.

SSL II handles the real symmetric matrix in the compressed storage mode (see Section 2.8).

Obtaining all eigenvalues

All the eigenvalues can be obtained from the steps 1), 2), and 3) below.

```
    :
CALL GSCHL(A,B,N,EPSZ,ICON)        1)
IF(ICON.GE.20000) GO TO 1000
CALL TRID1(A,N,D,SD,ICON)          2)
CALL TRQL(D,SD,N,E,M,ICON)         3)
    :
```

1) The generalized eigenproblem ($\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$) is reduced to the standard eigenproblem ($\boldsymbol{Sy} = \lambda \boldsymbol{y}$)
2) The real symmetric matrix $\boldsymbol{S}$ is reduced to a real symmetric tridiagonal matrix using the Householder method.
3) All the eigenvalues of the real symmetric tridiagonal matrix are obtained using the QL method.

Obtaining selected eigenvalues

From the following steps 1), 2), and 3), the largest (or smallest) *m* eigenvalues can be obtained.

```
    :
CALL GSCHL(A,B,N,EPSZ,ICON)        1)
IF(ICON.GE.20000) GO TO 1000
CALL TRID1(A,N,D,SD,ICON)          2)
CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)
                                   3)
    :
```

1), 2) Same as step 1) and 2) in "Obtaining all eigenvalues".
3) The largest (or smallest) *m* eigenvalues of the real symmetric tridiagonal matrix are obtained using the bisection method.

When obtaining more than *n*/4 eigenvalues of $\boldsymbol{A}$, it is generally faster to use the example shown in "Obtaining all eigenvalues".

Obtaining all eigenvalues and corresponding eigenvectors

From either steps 1) through 5), or from step 6), all of the eigenvalues and their corresponding eigenvectors can be obtained.

```
    :
CALL GSCHL(A,B,N,EPSZ,ICON)        1)
IF(ICON.GE.20000) GO TO 1000
CALL TRID1(A,N,D,SD,ICON)          2)
CALL TEIG1(D,SD,N,E,EV,K,M,ICON)   3)
IF(ICON.GE.20000)GO TO 3000
CALL TRBK(EV,K,N,M,A,ICON)         4)
CALL GSBK(EV,K,N,M,B,ICON)         5)
    :
```

or standard routine

```
       :
  CALL GSEG2(A,B,N,N,EPSZ,EPST,E,EV,K,
* VW,ICON)                              6)
  IF(ICON.GE.20000) GO TO 1000
       :
```

1), 2) Same as step 1), 2) in “Obtaining all eigenvalues”.
3) All engenvalues and corresponding eigenvectors of the real symmetric tridiagonal matrix are obtained by the QL method.
4) The eigenvectors of the real symmetric tridiagonal matrix are back-transformed to the eigenvectors of the real symmetric matrix ***S***
5) The eigenvectors of the real symmetric matrix ***S*** are back-transformed to the eigenvectors of $\boldsymbol{Ax}=\lambda\boldsymbol{Bx}$.
6) The standard routine GSEG2 can perform steps 1) through 5) at a time. In this case, the fourth parameter N of GSEG2 indicates to obtain the largest *n* number of eigenvalues.

Obtaining selected eigenvalues and corresponding eigenvectors
From either steps 1) through 5), or from step 6), *m* number of eigenvalues and their corresponding eigenvectors can be obtained.

```
       :
  CALL GSCHL(A,B,N,EPSZ,ICON)           1)
  IF(ICON.GE.20000) GO TO 1000
  CALL TRID1(A,N,D,SD,ICON)             2)
  IF(ICON.EQ.3000) GO TO 2000
  CALL TEIG2(D,SD,N,M,E,EV,K,VW,ICON)   3)
  IF(ICON.GE.20000) GO TO 3000
  CALL TRBK(EV,K,N,M,A,ICON)            4)
  CALL GSBK(EV,K,N,M,B,ICON)            5)
       :
```

or standard routine

```
       :
  CALL GSEG2(A,B,N,M,EPSZ,EPST,E,EV,K,
* VW,ICON)                              6)
  IF(ICON.GE.20000) GO TO 1000
       :
```

1), 2) Same as step 1), 2) in “Obtaining all eigenvalues”.
3) The largest (or smallest) *m* eigenvalues of the real symmetric tridiagonal matrix and the corresponding eigenvectors are obtained using the bisection and inverse iteration methods.
4), 5) Same as step 4), 5) in above paragraphs.
6) The standard routine GSEG2 can perform steps 1) through 5) at a time.

4.8 EIGENVALUES AND EIGENVECTORS OF A REAL SYMMETRIC BAND GENERALIZED EIGENPROBLEM

SSL II provides subroutines as shown in Fig. 4.8 to obtain eigenvalues and eigenvectors of $\boldsymbol{Ax}=\lambda\boldsymbol{Bx}$(***A***: symmetric band matrix and ***B***: positive definite symmetric band matrix). These are used for a large matrix of order *n* with $h/n < 1/6$, where *h* is the bandwidth. Subroutine GBSEG, which uses the Jennings method, is effective when obtaining less than *n* /10 eigenvalues and eigenvectors. Since subroutine GBSEG obtains the specified *m* eigenvalues and eigenvectors at one time, if it terminates abnormally, no eigenvalues and eigenvectors will be obtained.
An example of the use of this routine is shown below.
SSL II handles the real symmetric band matrix in compressed storage mode (see Section 2.8).

Obtaining selected eigenvalues and eigenvectors

```
       :
  CALL GBSEG(A,B,N,NH,M,EPSZ,EPST,
*           LM,E,EV,K,IT,VW,ICON)   1)
  IF(ICON.GE.20000) GO TO 1000
       :
```

The eigenvalues and eigenvectors are obtained by using the Jennings method of simultaneous iteration method. Parameter M is used to specify the largest (or smallest) *m* eigenvalues and eigenvectors to be obtained.

Table 4.8 Subroutines used for generalized eigenproblem of a real symmetric band matrix

Level	Function	Subroutine name
Standard routine	Obtaining eigenvalues and eigenvectors of a real band generalized eigenproblem	GBSEG (B52-11-0101)

CHAPTER 5
NONLINEAR EQUATIONS

5.1 OUTLINE

This chapter is concerned with the following types of problems.

- Roots of non-linear equations: Determining the roots of polynomial equation, transcendental equations, and systems of nonlinear equations (simultaneous nonlinear equations).

5.2 POLYNOMIAL EQUATIONS

The subroutines shown in Table 5.1 are used for these types of problems.

When solving real polynomial equations of fifth degree or lower, LOWP can be used. When solving only quadratic equations, RQDR should be used.

General conventions and comments concerning polynomial equations

The general form for polynomial equations is

$$a_0x^n + a_1x^{n-1} + ... + a_n = 0, \quad |a_0| \neq 0 \qquad (5.1)$$

where a_i (i = 0, 1, ... , n) is real or complex.

If a_i is real, (5.1) is called a real polynomial equation. If a_i is complex, (5.1) is called a complex polynomial equation, and z is used in place of x.

Unless specified otherwise, subroutines which solve polynomial equations try to obtain all of the roots. Methods and their use are covered in this section. Algebraic and iterative methods are available for solving polynomial equations. Algebraic methods use the formulas to obtain the roots of equations whose degree is four or less. Iterative methods may be used for equations of any degree. In iterative methods, an approximate solution has been obtained. For most iterative methods, roots are determined one at a time; after a particular root has been obtained, it is eliminated from the equation to create a lower degree equation, and the next root is determined.

Neither algebraic methods nor iterative methods are "better" since each has merits and demerits.

- Demerits of algebraic methods
  Underflow or overflow situations can develop during the calculations process when there are extremely large variations in size among the coefficients of (5.1).

- Demerits of iterative methods
  Choosing an appropriate initial approximation presents problems. If initial values are incorrectly chosen, convergence may not occur no matter how many iterations are done, so if there is no

Table 5.1 Polynomial equation subroutines

Objective	Subroutine name	Method	Notes
Real quadratic equations	RQDR (C21-11-0101)	Root formula	
Complex quadratic equations	CQDR (C21-15-0101)	Root formula	
Real low degree equations	LOWP (C21-41-0101)	Algebraic method and iterative method are used together.	Fifth degree or lower
Real high degree polynomial equations	RJETR (C22-11-0111)	Jenkins-Traub method	
Complex high degree polynomial equations	CJART (C22-15-0101)	Jaratt method	

convergence, it is assumed that the wrong initial value was chosen. It is possible that some roots can be determined while others can not.
Convergence must be checked for with each iteration, resulting in calculation problem.

In order to avoid the demerits of algebraic methods, SSL II uses iterative methods except when solving quadratic equations. The convergence criterion method in SSL II is described in this section.

When iteratively solving an polynomial equation:

$$f(x) \equiv \sum_{k=0}^{n} a_k x^{n-k} = 0$$

if the calculated value of $f(x)$ is within the range of rounding error, it is meaningless to make the value any smaller. Let the upper limit for rounding errors for solving $f(x)$ be $\varepsilon(x)$, then

$$|\varepsilon(x)| = u \sum_{k=0}^{n} |a_k x^{n-k}| \tag{5.2}$$

where u is the round-off unit.
Thus, when x satisfies

$$|f(x)| \le |\varepsilon(x)| \tag{5.3}$$

there is no way to determine if x is the exact root. Therefore, when

$$\left| \sum_{k=0}^{n} a_k x^{n-k} \right| \le u \sum_{k=0}^{n} |a_k x^{n-k}| \tag{5.4}$$

is satisfied, convergence is judged to have occurred, and the solution is used as one of the roots. (For further information on equation (5.2), see reference [23].).

As for the precision of roots, with both algebraic and iterative methods, when calculating with a fixed number of digits, it is possible for certain roots to be determined with higher precision than others.

Generally, multiple roots and neighboring roots tend to be less precise than the other roots. If neighboring roots are among the solutions of an algebraic equation, the user can assume that those roots are not as precise as the rest.

5.3 TRANSCENDENTAL EQUATIONS

Transcendental equation can be represented as

$$f(x) = 0 \tag{5.5}$$

If $f(x)$ is a real function, the equation is called a real transcendental equation. If $f(x)$ is a complex function, the equation is called a complex transcendental equation, and z is used in place of x.

The objective of subroutines which solve transcendental equations is to obtain only one root of $f(x)$ within a specified range or near a specified point.

Table 5.2 lists subroutines used for transcendental equations.

Iterative methods are used to solve transcendental equations. The speed of convergence in these methods depends mainly on how narrow the specified range is or how close a root is to the specified point. Since the method used for determining convergence differs among the various subroutines, the descriptions of each should be studied.

5.4 NONLINEAR SIMULTANEOUS EQUATIONS

Nonlinear simultaneous equations are given as:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \tag{5.6}$$

where $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}),..., f_n(\boldsymbol{x}))^T$ and $\boldsymbol{0}$ is an n-dimensional zero vector. Nonlinear simultaneous equations are solved by iterative methods in which the user must gives an initial vector $\boldsymbol{x}_0$ and it is improved repeatedly until the final solution for (3.1) is obtained within a required accuracy. Table 5.3 lists subroutines used for nonlinear simultaneous equations. The best known method among iterative methods is Newton method, expressed as:

Table 5.2 Transcendental equation subroutines

Objective	Subroutine name	Method	Notes
Real transcendental equation	TSD1 (C23-11-0101)	Bisection method, linear interpolation method and inverse second order interpolation method are all used.	Derivatives are not needed.
	TSDM (C23-11-0111)	Muller' s method	No derivatives needed. Initial values specified.
Zeros of a complex function	CTSDM (C23-15-0101)	Muller' s method	No derivatives needed. Initial values specified.

Table 5.3 Nonlinear simultaneous equation subroutines

Objective	Subroutine name	Method	Notes
Non-linear simultaneous equations	NOLBR (C24-11-0101)	Brent’ s method	Derivatives are not needed.

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{J}_i^{-1}\boldsymbol{f}(\boldsymbol{x}_i),\ i = 0, 1, \ldots \tag{5.7}$$

where $\boldsymbol{J}_i$ is the Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$ for $\boldsymbol{x} = \boldsymbol{x}_i$, which means:

$$\boldsymbol{J}_i = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\boldsymbol{x}=\boldsymbol{x}_i} \tag{5.8}$$

The Newton method is theoretically ideal, that is, its order of convergence is quadratic and calculations are simple. However, this method develops several calculation problems when it manipulates complex (or larger) systems of nonlinear equations. The major reasons are:

- It is often difficult to obtain the coefficients $\partial f_i / \partial x_j$ in (5.8), (i.e., partial derivatives cannot be calculated because of the complexity of equations).
- The number of calculations for all elements in (5.8) are too large.
- Since a system of linear equations with coefficient matrix $\boldsymbol{J}_i$ must be solved for each iteration, calculation time is long.

If the above problems are solved and the order of convergence is kept quadratic, this method provides short processing time as well as ease of handling.

The following are examples of the above problems and their solutions. The first problem $\partial f_i / \partial x_j$ can be approximated with the difference, i.e., by selecting an appropriate value for h, we can obtain

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1,\ldots,x_j+h,\ldots,x_n) - f_i(x_1,\ldots,x_n)}{h} \tag{5.9}$$

For the second and third problems, instead of directly calculating the Jacobian matrix, a pseudo Jacobian matrix (which need not calculate all the elements) is used to solve the simultaneous equations. All of the above means are adopted in SSL II. Several notes on the use of subroutines for nonlinear simultaneous equations follow.

The user must provide the function subprograms to calculate a series of functions which define equations. These function subprograms should be provided taking the following points into consideration in order to use subroutines effectively and to obtain precise solution.

- Loss of digit should be avoided in calculating functions. This is especially important because values of functions are used in subroutines to evaluate derivatives.
- The magnitude of elements such as those of variable vector $\boldsymbol{x}$ or of function vector $\boldsymbol{f}(\boldsymbol{x})$ should be balanced. Since, if unbalanced the larger elements often mask the smaller elements during calculations. SSL II routines have the function of checking variance in the largest element to detect convergence. In addition, the accuracy of a solution vector depends upon the tolerance given by the user. Generally, the smaller the tolerance for convergence, the higher the accuracy for the solution vector.

  However, because of the round-off errors, there is a limit to the accuracy improvement. The next problem is how to select initial value $\boldsymbol{x}_0$. It should be selected by the user depending upon the characteristics of the problem to be solved with the equations. If such information is not available, the user may apply the cut-and-try method by arbitrarily selecting the initial value and repeating calculations until a final solution is obtained.

  Finally, due to the characteristics of equations, some equations can not be solved by single precision subroutines, but may be solved by double precision subroutines. Because double precision subroutines are more versatile, they are recommended for the user’s needs.

CHAPTER 6 EXTREMA

6.1 OUTLINE

The following problems are considered in this chapter:

- Unconstrained minimization of single variable function
- Unconstrained minimization of multivariable function
- Unconstrained minimization of sum of squares of functions (Nonlinear least squares solution).
- Linear programming
- Nonlinear programming (Constrained minimization of multivariable function)

6.2 MINIMIZATION OF FUNCTION WITH A SINGLE VARIABLE

Given a single variable function $f(x)$, the local minimum point x^* and the function value $f(x^*)$ are obtained in interval $[a, b]$.

Subroutines
Table 6.1-A gives subroutines applicable depending on whether the user can define a derivative $g(x)$ analytically in addition to function $f(x)$.

Table 6.1-A Subroutines for unconstrained minimization of a function with single variable

Analytical definition	Subroutine name	Notes
$f(x)$	LMINF (D11-30-0101)	Quadratic interpolation
$f(x)$, $g(x)$	LMING (D11-40-0101)	Cubic interpolation

Comments on use
Interval $[a, b]$
In the SSL II, only one minimum point of $f(x)$ is obtained within the error tolerance assuming that the $-f(x)$ is unimodal in interval $[a, b]$. If there are several minimum points in interval $[a, b]$, it is not guaranteed to which minimum point the resultant value is converged.
This means that it is desirable to specify values for end points a and b of an interval including minimum point x^* to be close to x^*.

6.3 UNCONSTRAINED MINIMIZATION OF MULTIVARIABLE FUNCTION

Given a real function $f(\boldsymbol{x})$ of n variables and an initial vector $\boldsymbol{x}_0$, the vector (local minimum) $\boldsymbol{x}^*$ which minize the function $f(\boldsymbol{x})$ is obtained together with its function value $f(\boldsymbol{x}^*)$, where $\boldsymbol{x} = (x_1, x_2, \dots , x_n)^{\mathrm{T}}$.
Minimizing a function means to obtain a minimum point $\boldsymbol{x}^*$, starting with an arbitrary initial vector $\boldsymbol{x}_0$, and continuing iteration under the following relationship,

$$f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k),\ k = 0, 1, \dots \tag{6.1}$$

$\boldsymbol{x}_k$: Iteration vector

The iteration vector is modified based on the direction in which the function $f(\boldsymbol{x})$ decreases in the region of $\boldsymbol{x}_k$ by using not only the value of $f(\boldsymbol{x})$ but also the gradient vector $\boldsymbol{g}$ and the Hessian matrix $\boldsymbol{B}$ normally.

$$\begin{aligned} \boldsymbol{g} &= \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots , \frac{\partial f}{\partial x_n}\right)^{\mathrm{T}} \\ \boldsymbol{B} &= (b_{ij})\ , \quad b_{ij} = \partial^2 f \big/ \partial x_i \partial x_j \end{aligned} \tag{6.2}$$

Formula based on Newton method
If the function $f(\boldsymbol{x})$ is quadratic and is concave, the global minimum point $\boldsymbol{x}^*$ should be able to be obtained theoretically within at most n iterations by using iterative formula of the Newton method.
A function can be expressed approximately as quadratic in the region of the local minimum point $\boldsymbol{x}^*$. That is,

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}^*) \qquad (6.3)$$

Therefore, if the Hessian matrix $\boldsymbol{B}$ is positive definite, the iterative formula based on the Newton method which is applied to the quadratic function will be a good iterative formula for any function in general as shown in Eq. (6.3). Now let $\boldsymbol{g}_k$ be a gradient vector at an arbitrary point $\boldsymbol{x}_k$ in the region of the local minimum point $\boldsymbol{x}^*$, then the basic iterative formula of Newton method is obtained by Eq. (6.3) as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{B}^{-1}\boldsymbol{g}_k \qquad (6.4)$$

The SSL II, based on Eq. (6.4), introduces two types of iterative formulae.

Revised quasi-Newton method
Iterative formula

$$\begin{aligned} \boldsymbol{B}_k \boldsymbol{p}_k &= -\boldsymbol{g}_k \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\ \boldsymbol{B}_{k+1} &= \boldsymbol{B}_k + \boldsymbol{E}_k \end{aligned} \qquad (6.5)$$

, where $\boldsymbol{B}_k$ is an approximate matrix to the Hessian matrix and is improved by the matrix $\boldsymbol{E}_k$ of rank two during the iteration process.

$\boldsymbol{p}_k$ is a searching vector which defines the direction in which the function value decreases locally.α_k is a constant by which $f(\boldsymbol{x}_{k+1})$ is locally minimized (linear search).

These formulae can be used when the user cannot define the Hessian matrix analytically.

Quasi-Newton method
Iterative formula

$$\begin{aligned} \boldsymbol{p}_k &= -\boldsymbol{H}_k \boldsymbol{g}_k \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\ \boldsymbol{H}_{k+1} &= \boldsymbol{H}_k + \boldsymbol{F}_k \end{aligned}$$

, where $\boldsymbol{H}_k$ is an approximate matrix to inverse matrix $\boldsymbol{B}^{-1}$ of the Hessian matrix and is improved by the matrix $\boldsymbol{F}_k$ of rank 2 during the iterative process.

$\boldsymbol{p}_k$ is a searching vector which defines the direction in which the function value decrease locally. α_k is a constant by which $f(\boldsymbol{x}_{k+1})$ is locally minimized (linear search).

Subroutines
The subroutines are provided as shown in Table 6.1 depending on whether or not the user can analytically define a gradient vector $\boldsymbol{g}$ in addition to the function $f(\boldsymbol{x})$.

Table 6.1 Subroutines for unconstrained minimization of a function with several variables

Analytical definition	Subroutine name	Notes
$f(\boldsymbol{x})$	MINF1 (D11-10-0101)	Revised quasi-Newton method
$f(\boldsymbol{x})$, $g(\boldsymbol{x})$	MING1 (D11-20-0101)	Quasi-Newton method

Comments on use

- Giving an initial vector $\boldsymbol{x}_0$
  Choose the initial vector $\boldsymbol{x}_0$ as close to the expected local minimum $\boldsymbol{x}^*$ as possible.
  When the function $f(\boldsymbol{x})$ has more than one local minimum point, if the initial vector is not given appropriately, the method used may not converge to the expected minimum point $\boldsymbol{x}^*$. Normally, $\boldsymbol{x}_0$ should be set according to the physical information of the function $f(\boldsymbol{x})$.

- Function calculation program
  Efficient coding of the function programs to calculate the function $f(\boldsymbol{x})$, the gradient vector $\boldsymbol{g}$ is desirable. The number of evaluations for each function made by the SSL II subroutine depends on the method used or its initial vector. In general, it takes a majority in the total processing and takes effect on efficiency.
  In case that the subroutine is given only the function $f(\boldsymbol{x})$, the gradient vector $\boldsymbol{g}$ is usually approximated by using difference. Therefore an efficient coding to reduce the effect of round-off errors should be considered, also.
  When defining function $f(\boldsymbol{x})$, it should be scaled well so as to balance the variable $\boldsymbol{x}$ as much as possible.

- Convergence criterion and accuracy of minimum value $f(\boldsymbol{x}^*)$
  In an algorithm for minimization, the gradient vector $g(\boldsymbol{x}^*)$ of the function $f(\boldsymbol{x})$ at the local minimum point $\boldsymbol{x}^*$ is assumed to satisfy

$$\boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0} \qquad (6.6)$$

, that is, the iterative formula approximates the function $f(\boldsymbol{x})$ as a quadratic function in the region of the local minimum point $\boldsymbol{x}^*$ as follows.

$$f(\boldsymbol{x}^* + \delta\boldsymbol{x}) \approx f(\boldsymbol{x}^*) + \frac{1}{2}\delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{B}\delta\boldsymbol{x} \qquad (6.7)$$

Eq. (6.7) indicates that when $f(\boldsymbol{x})$ is scaled appropriately, if $\boldsymbol{x}$ is changed by ε, function $f(\boldsymbol{x})$ changes by ε^2.
In SSL II, if

$$\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \le \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot \varepsilon \qquad (6.8)$$

, where ε is a convergence criterion, is satisfied for the iteration vector $\boldsymbol{x}_k$, then $\boldsymbol{x}_{k+1}$ is taken as the local minimum point $\boldsymbol{x}^*$. Therefore, if the minimum value $f(\boldsymbol{x}^*)$ is to be obtained as accurate as the rounding error, the convergence criterion ε should be given as follows:

$\varepsilon = \sqrt{u}$ where u is the unit round off.

The SSL II uses $2 \cdot \sqrt{u}$ for a standard convergence criterion.

6.4 UNCONSTRAINED MINIMIZATION OF SUM OF SQUARES OF FUNCTIONS (NONLINEAR LEAST SQUARES SOLUTION)

Given m real functions $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$ of n variables and an initial vector $\boldsymbol{x}_0$, the vector (local minimum) $\boldsymbol{x}^*$ which minimize the following functions is obtained together with its function value $F(\boldsymbol{x}^*)$, where, $\boldsymbol{x}$ is the vector of $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ and $m \geq n$.

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} \{f_i(\boldsymbol{x})\}^2 \tag{6.9}$$

If all the functions $f_i(\boldsymbol{x})$ are linear, it is a linear least squares solution problem. For detailed information on its solution, refer to Section 3.5. (For example subroutine LAXL). If all the functions $f_i(\boldsymbol{x})$ are nonlinear, the subroutines explained in this section may be used. When the approximate vector $\boldsymbol{x}_k$ of $\boldsymbol{x}^*$ is varied by $\boldsymbol{\Delta x}$, $F(\boldsymbol{x}_k + \boldsymbol{\Delta x})$ is approximated as shown in (6.10).

$$\begin{aligned} F(\boldsymbol{x}_k + \boldsymbol{\Delta x}_k) &= \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k + \boldsymbol{\Delta x}_k)\boldsymbol{f}(\boldsymbol{x}_k + \boldsymbol{\Delta x}_k) \\ &\approx \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k) + 2\boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{J}_k\boldsymbol{\Delta x}_k \\ &\quad + \boldsymbol{\Delta x}_k^{\mathrm{T}}\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\boldsymbol{\Delta x}_k \end{aligned} \tag{6.10}$$

, where $|F(\boldsymbol{x}_k)|$ is assumed to be sufficiently small. And, $\boldsymbol{f}(\boldsymbol{x})$ equals to $(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^{\mathrm{T}}$ and $\boldsymbol{J}_k$ is a Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$ at vector $\boldsymbol{x}_k$.

$\boldsymbol{\Delta x}_k$ which minimize this $F(\boldsymbol{x}_k + \boldsymbol{\Delta x}_k)$ can be obtained as the solution of the system of linear equations (6.11) derived by differentiating the right side of (6.10).

$$\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\boldsymbol{\Delta x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \tag{6.11}$$

The equation shown in (6.11) is called a normal equation. The iterative method based on the $\boldsymbol{\Delta x}_k$ is called the Newton-Gauss method. In the Newton-Gauss method function value $F(\boldsymbol{x})$ decrease along direction $\boldsymbol{\Delta x}_k$, however, $\boldsymbol{\Delta x}_k$ itself may diverge.

The gradient vector $\boldsymbol{\nabla}F(\boldsymbol{x}_k)$ at $\boldsymbol{x}_k$ of $F(\boldsymbol{x})$ is given by

$$\boldsymbol{\nabla}F(\boldsymbol{x}_k) = 2\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \tag{6.12}$$

$-\boldsymbol{\nabla}F(\boldsymbol{x}_k)$ is the steepest descent direction of $F(\boldsymbol{x})$ at $\boldsymbol{x}_k$. The following is the method of steepest descent.

$$\Delta\boldsymbol{x}_k = -\boldsymbol{\nabla}F(\boldsymbol{x}_k) \tag{6.13}$$

$\boldsymbol{\Delta x}_k$ guarantees the reduction of $F(\boldsymbol{x})$. However if iteration is repeated, it proceeds in a zigzag fashion.

Formula based on the Levenberg-Marquardt method

Levenberg, Marquardt, and Morrison proposed to determine $\boldsymbol{\Delta x}_k$ by the following equations combining the ideas of the methods of Newton-Gauss and steepest descendent.

$$\{\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k + v_k^2\boldsymbol{I}\}\boldsymbol{\Delta x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \tag{6.14}$$

where v_k is a positive integer (called Marquardt number).

$\boldsymbol{\Delta x}_k$ obtained in (6.14) depends on the value of v_k that is, the direction of $\boldsymbol{\Delta x}_k$ is that of the Newton-gauss method if $v_k \to 0$: if $v_k \to \infty$, it is that of steepest descendent.

SSL II uses iterative formula based on (6.14). It does not directly solve the equation in (6.14) but it obtains the solution of the following equation, which is equivalent to (6.14), by the least squares method (Householder method) to maintain numerical stability.

$$\begin{bmatrix} \boldsymbol{J}_k \\ \cdots\cdots \\ v_k\boldsymbol{I} \end{bmatrix} \boldsymbol{\Delta x}_k = -\begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_k) \\ \cdots\cdots \\ \boldsymbol{0} \end{bmatrix} \tag{6.15}$$

The value v_k is determined adaptively during iteration.

Subroutines

The subroutines are provided as shown in Table 6.2 depending on whether or not the user can analytically define a Jacobian matrix $\boldsymbol{J}$ in addition to the function $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$.

Table 6.2 Subroutines for unconstrained minimization of sum of squares of functions

Analytical definition	Subroutine name	Notes
$f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$	NOLF1 (D15-10-0101)	Revised Marquardt Method
$f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}), \boldsymbol{J}$	NOLG1 (D15-20-0101)	Revised Marquardt Method

Comments on use

- Giving an initial vector $\boldsymbol{x}_0$
  Choose the initial vector $\boldsymbol{x}_0$ as close to the expected local minimum point $\boldsymbol{x}^*$ as possible. When the function $F(\boldsymbol{x})$ has more than one local minimum point, if the initial vector is not given appropriately, the method used may not converge to the expected minimum point $\boldsymbol{x}^*$.
  The user should set $\boldsymbol{x}_0$ after checking the feature of the problem for which this subroutine is to be used.

- Function calculation program
  Efficient coding of the function programs to calculate the function $\{f_i(\boldsymbol{x})\}$ value of Jacobian matrix $\boldsymbol{J}$ is desirable. The number of evaluations for each function made by the SSL II subroutine depends on the method used or its initial vector. In general, it takes a majority of the total processing and has an effect on the efficiency.
  When that subroutine is given only the function $\{f_i(\boldsymbol{x})\}$ Jacobian matrix $\boldsymbol{J}$ is usually approximated by using differences. Therefore, an efficient coding to reduce the effect of round-off errors should also be considered.

- Convergence criterion and accuracy of minimum value $F(\boldsymbol{x}^*)$
  In an algorithm for minimization, $F(\boldsymbol{x})$ at the local minimum point $\boldsymbol{x}^*$ is assumed to satisfy

$$\nabla F(\boldsymbol{x}^*) = 2\boldsymbol{J}^{\mathrm{T}} \boldsymbol{f}(\boldsymbol{x}^*) = 0 \tag{6.16}$$

  that is, the iterative formula approximates the function $F(\boldsymbol{x})$ as a quadratic function in the region of the local minimum point $\boldsymbol{x}^*$ as follows:

$$F(\boldsymbol{x}^* + \delta\boldsymbol{x}) \approx F(\boldsymbol{x}^*) + \delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}\delta\boldsymbol{x} \tag{6.17}$$

Eq. (6.17) indicates that when $F(\boldsymbol{x})$ is scaled appropriately, if $\boldsymbol{x}$ is changed by ε, function $F(\boldsymbol{x})$ changes by ε^2.
In SSL II, if

$$\begin{aligned} &F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k) \\ &\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 \le \max(1.0, \|\boldsymbol{x}_k\|_2) \cdot \varepsilon \end{aligned} \tag{6.18}$$

, where ε is a convergence criterion,
is satisfied for the iteration vector $\boldsymbol{x}_k$, then $\boldsymbol{x}_{k+1}$ is taken as the local minimum point $\boldsymbol{x}^*$. Therefore, if the minimum value $F(\boldsymbol{x})$ is to be obtained as accurately as the rounding-error, the convergence criterion should be given as follows:

$$\varepsilon \approx \sqrt{u}$$

, where u is the unit round-off.
The SSL II uses $2 \cdot \sqrt{u}$ for a standard convergence criterion.

6.5 LINEAR PROGRAMMING

Linear programming is defined as the problem where a linear function with multiple variables is minimized (or maximized) under those constraints on the variables which are represented as some linear equations and inequality relations.
The following is a standard linear programming problem:
"Minimize the following linear objective function":

$$z = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + c_0$$

subject to

$$\boldsymbol{Ax} = \boldsymbol{d} \tag{6.19}$$
$$\boldsymbol{x} \ge \boldsymbol{0} \tag{6.20}$$

, where $\boldsymbol{A}$ is an $m \times n$ coefficient matrix and the rank of $\boldsymbol{A}$ is:

$$\mathrm{rank}(\boldsymbol{A}) = m \le n$$

where, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ is a variable vector,
$\boldsymbol{d} = (d_1, d_2, \ldots, d_m)^{\mathrm{T}}$ is a constant vector,
$\boldsymbol{c} = (c_1, c_2, \ldots, c_n)^{\mathrm{T}}$ is a coefficient vector,
and
c_0 is a constant term.

Let $\boldsymbol{a}_j$ denote the j-th column of $\boldsymbol{A}$. If m columns of $\boldsymbol{A}$, $\boldsymbol{a}_{k_1}$, $\boldsymbol{a}_{k_2}, \ldots, \boldsymbol{a}_{k_m}$, are linearly independent, a group of the corresponding variables ($x_{k_1}, x_{k_2}, \ldots, x_{k_m}$) are called the base. x_{k_i} is called a (i-th) base variable. If a basic solution satisfies (6.20) as well , it is called a feasible basic solution. It is proved that if there exist feasible basic solutions and there exist the optimal solution to minimize the objective function, then the optimal solution exist in the set of feasible basic solutions (principle of Linear Programming).

- Simplex method
  The optimal solution is calculated by starting from a basic feasible solution, exchanging base variables one by one and obtaining a next basic feasible solution consecutively, making z smaller and smaller.

- Revised simplex method
  Using iterative calculation of the simplex method, coefficients and constant terms required for determining the basic variables to be changed are calculated from the matrix inversion of the basic matrix, $\boldsymbol{B} = [\boldsymbol{a}_{k_1}, \boldsymbol{a}_{k_2}, \ldots, \boldsymbol{a}_{k_m}]$, the original coefficient

A, ***c***, and constant term ***d***.

SSL II provides subroutine LPRS1 using this revised simplex method. If the constrained condition contains inequalities, new variables are introduced to change into equalities.

For example,

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq d_1$$

is changed into

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_{n+1} = d_1,\ x_{n+1} \geq 0$$

and,

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq d_2$$

is changed into

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n - x_{n+2} = d_2,\ x_{n+2} \geq 0$$

Non-negative variables such as x_{n+1} or x_{n+2} which are added to change an inequality into an equality are called slack variables.

Maximization is performed by multiplying the objective function by -1 instead.

Subroutine LPRS1 performs these processes and thus enables the solution of problems containing inequalities or to maximize the problem.

The algorithm is divided into two stages as follows.

- At the first stage, obtain the basic feasible solution
- At the second stage, obtain the optimal solution

This is called the two-stage method. At the first stage the optimal solution is obtained for the following problem.

"Minimize the following equation".

$$z_1 = \sum_{i=1}^{m} x_i^{(a)}$$

subject to

$$\boldsymbol{Ax} + \boldsymbol{A}^{(a)}\boldsymbol{x}^{(a)} = \boldsymbol{d},$$
$$\boldsymbol{x} \geq \boldsymbol{0},\ \boldsymbol{x}^{(a)} \geq \boldsymbol{0}$$

where $\boldsymbol{x}^{(a)}$ is an m-dimensional vector of $\boldsymbol{x}^{(a)} = (x_1^{(a)}, x_2^{(a)}, \ldots, x_m^{(a)})^T$

$\boldsymbol{A}^{(a)}$ is an m-order diagonal matrix of $\boldsymbol{A}^{(a)} = (a_{ij}^{(a)})$

where, when $d_i \geq 0$, $a_{ii}^{(a)} = 1$

when $d_i < 0$, $a_{ii}^{(a)} = -1$

$x_i^{(a)}$ is called an artificial variable. When the optimal solution is obtained, if z_1 is larger than zero ($z_1 > 0$), no $\boldsymbol{x}$ will satisfy the conditions in (6.19) and (6.20).

If z_1 is equivalent to zero ($z_1 = 0$), that is $\boldsymbol{x}^{(a)} = \boldsymbol{0}$, the basic feasible solution of the given problem has been obtained, and so we proceed to the second stage. But if

$$\mathrm{rank}(\boldsymbol{A}) < m$$

but when there exists x satisfying (6.19), we can put

$$r = \mathrm{rank}(\boldsymbol{A})$$

, then $(m - r)$ equations in (6.19) turn out to be useless.

The optimal solution obtained at the first stage results in the basic feasible solution excepting the useless conditional equations. There remain $(m - r)$ number of artificial variables in basic variables. The conditional equations corresponding to these artificial variables (i-th basic variable corresponds to the i-th conditional equation) are useless.

Comments on use

- Coefficient relative zero criterion

  Subroutine LPRS1 performs the relative zero criterion in which if the absolute value of the elements during iterative process becomes small, LPRS1 assumes it as zero. Parameter EPSZ is used to specify the value of relative zero criterion.

  Suppose the following extended coefficient matrix consisting of coefficient matrix ***A***, constant vector ***d***, coefficient vector ***c*** and constant term c_0.

$$\left[\begin{array}{c:c} \boldsymbol{A} & \boldsymbol{d} \\ \hdashline \boldsymbol{c} & c_0 \end{array}\right]$$

  Let the absolute maximum element of this matrix be $a_{\max}$. Then if the absolute value of a coefficient vector and constant term obtained by iteration is smaller than $a_{\max} \cdot$ EPSZ, it is assumed to be zero.

  If EPSZ does not give an appropriate value, although the feasible solution is obtained z_1 may be larger than zero when the optimal solution is obtained at the first stage. Furthermore, the optimal solution at the first stage may have to be obtained before iteration.

  To keep accurate precision, you should multiply an appropriate constant to each row or column to enable the ratio of the maximum and minimum absolutes of the extended coefficient matrix elements.

- Number of iterations

  LPRS1 exchanges basic variables so that the same basic feasible solution does not appear twice. It can check whether the optimal solution can be obtained in a certain number of iterations or not. It also can terminate in the middle of processing. Parameter IMAX is used to specify the number of iterations.

  If the iteration terminates as specified by parameter IMAX, LPRS1 can continue calculation when the feasible basic solution has been obtained (that is, at the second stage).

6.6 NONLINEAR PROGRAMMING (CONSTRAINED MINIMIZATION OF MULTIVARIABLE FUNCTION)

Given an n-variable real function $f(\boldsymbol{x})$ and the initial vector $\boldsymbol{x}_0$, the local minimum point and the function value $f(\boldsymbol{x}^*)$ are obtained subject to following constraints:

$$c_i(\boldsymbol{x}) = 0, \quad i = 1, 2, \dots, m_1 \tag{6.21}$$
$$c_i(\boldsymbol{x}) \geq 0, \quad i = m_1 + 1, \dots, m_1 + m_2 \tag{6.22}$$

Where $\boldsymbol{x}$ is vector as $(x_1, x_2, \dots, x_n)^T$ and m_1 and m_2 are the numbers of equality constraints and unequality constraints respectively.

The algorithm for this problem is derived from that for unconstrained minimization explained in section 6.3 by adding certain procedures for constraints of (6.21), (6.22). That is, the algorithm minimizes $f(\boldsymbol{x})$ by using the quatratic approximation for $f(\boldsymbol{x})$ at approximate point $\boldsymbol{x}_k$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{g}_k + \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{B}\boldsymbol{y} \tag{6.23}$$

where $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{x}_k$ and $\boldsymbol{B}$ is a Hessian matrix, under the constraints of (6.21), (6.22) at the same point $\boldsymbol{x}_k$ as follows:

$$c_i(\boldsymbol{x}_k) + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\nabla} c_i(\boldsymbol{x}_k) = 0, \quad i = 1,2,\dots,m_1 \tag{6.24}$$

$$\begin{aligned} &c_i(\boldsymbol{x}_k) + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\nabla} c_i(\boldsymbol{x}_k) \geq 0, \\ &i = m_1 + 1, \dots, m_1 + m_2 \end{aligned} \tag{6.25}$$

Where $\boldsymbol{\nabla} c_i$ is a gradient vector of c_i
This is a quadratic programming with respect to $\boldsymbol{y}$.

The SSL II supplies the NLPG1 that gives minimum point by solving quadratic programming successively during iteration.

CHAPTER 7
INTERPOLATION AND APPROXIMATION

7.1 OUTLINE

This chapter is concerned with the following types of problems.

- Interpolation
  Given discrete points $x_1 < x_2 < ...< x_n$ and their corresponding function values $y_i = f(x_i)$, $i = 1, ... , n$ (in some cases $y'_i = f'(x_i)$ is given), an approximation to $f(x)$ (hereafter called interpolating function) is determined such that it passes through the given points; or, that the interpolating function is used to determine an approximate value (hereafter called interpolated value) to $f(x)$ at a point $x = v$ other than x_i.

- Least-squares approximation
  Given discrete points $x_1 < x_2 < ... < x_n$ and their corresponding observed values y_i, $i = 1, ... , n$ the approximation $\bar{y}_m(x)$ that minimizes

  $$\sum_{i=1}^{n} w(x_i)\{y_i - \bar{y}_m(x_i)\}^2, w(x_i) \geq 0$$

  is determined; $w(x)$ is a weight function, and $\bar{y}_m(x)$ is a polynomial of degree m.
  In this type of problem y_i is observed data. This method is used when the observation error varies among the data.

- Smoothing
  Given discrete points $x_1, x_2, ... , x_n$ and their corresponding observed values y_i, $i = 1, 2, ... , n$ a new series of points $\{ \tilde{y}_i \}$ which approximates the real function is obtained by smoothing out the observation errors contained in the observed value $\{y_i\}$. Hereafter, this processing is referred to as smoothing. $\tilde{y}_i$ (or $\{ \tilde{y}_i \}$)is called the smoothed value for y_i (or $\{y_i\}$), $|y_i - \tilde{y}_i|$ shows the extent of smoothing, and the polynomial used for smoothing is called the smoothing polynomial.

- Series
  When a smooth function $f(x)$ defined on a finite interval is expensive to evaluate, or its derivatives or integrals can not be obtained analytically, it is suggested $f(x)$ be expanded to the Chebyshev series.
  The features of Chebyshev series expansion are described below.
  – Good convergence
  – Easy to differentiate and integrate term by term
  – Effective evaluation owing to fast Fourier transformation, leading to numerical stability.
    Obtain the item number of n and the coefficient number of Chebyshev expansion depending upon the required precision. Then obtain the derivative and indefinite integral of $f(x)$ by differentiating and integrating each item of the obtained series in forms of series. The derivative value, differential coefficient and definite integral can be obtained by summing these series. If the function $f(x)$ is a smooth periodic function, it can be expanded to triangular series. Here the even function and odd function is expanded to the cosine series and the sine series depending upon the required precision.

In the field of interpolation or smoothing in this chapter, and also in that of numerical differentiation or quadrature of a tabulated function, very powerful functions, what is called spline functions, are used. So, the definition and the representations of the functions are described below.

Spline function

(1) Definition

Suppose that discrete points $x_0 ..., x_n$ divide the range $[a, b]$ into intervals such that

$$a = x_0 < x_1 < ...< x_n = b \qquad (7.1)$$

Then, a function $S(x)$ which satisfies the following conditions:

a. $D^k S(x) = 0$ for each interval (x_i, x_{i+1})
b. $S(x) \in C^{k-2}[a,b]$ (7.2)

, where $D \equiv d / dx$

is defined as the spline function of degree $(k-1)$ and the discrete points are called knots.

As shown in (7.2), $S(x)$ is a polynomial of degree $(k-1)$ which is separately defined for each interval (x_i, x_{i+1}) and whose derivatives of up to degree $(k-2)$ are continuous over the range $[a, b]$.

(2) Representation-1 of spline functions

Let a_j, $j = 0, 1, \ldots, k-1$ and b_i, $i = 1, 2, \ldots, n-1$ be arbitrary constants, then a spline function is expressed as

$$\begin{cases} S(x) = p(x) + \sum_{i=1}^{n-1} b_i (x-x_i)_+^{k-1} \\ \text{where,} \\ p(x) = \sum_{j=0}^{k-1} a_j (x-x_0)^j \end{cases} \tag{7.3}$$

The function $(x-x_i)_+^{k-1}$ is defined as

$$(x-x_i)_+^{k-1} = \begin{cases} (x-x_i)^{k-1}, x \ge x_i \\ 0, x < x_i \end{cases} \tag{7.4}$$

The following illustration proves that (7.3) satisfies (7.2). Suppose that x is moved from x_0 to the right in (7.3).

For $x_0 \le x < x_1$, $S(x) = p(x)$, so $S(x)$ is a polynomial of degree $(k$-$1)$.

For $x_1 \le x < x_2$, $S(x) = p(x) + b_1(x - x_1)^{k-1}$, so $S(x)$ is a polynomial of degree $(k-1)$.

In general, for $x_i \le x < x_{i+1}$

$$S(x) = p(x) + \sum_{r=1}^{i} b_r (x-x_r)^{k-1}$$

So, it is found that $S(x)$ is a polynomial of degree $(k$-$1)$ which is separately defined for each interval.

From equation (7.3) we obtain

$$\begin{aligned} &\frac{d^l}{dx^l} S(x) = S^{(l)}(x) \\ &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1)a_j(x-x_0)^{j-l} \\ &+ \sum_{j=1}^{n-1} (k-1)(k-2)\cdots(k-l)b_i(x-x_i)_+^{k-1-l} \end{aligned}$$

The l-th derivatives from the left and the right of $S(x)$ at x_i, are

$$\begin{aligned} \lim_{\varepsilon\to 0} S^{(l)}(x_i-\varepsilon) &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1)a_j(x_i-x_0)^{j-l} \\ &+ \sum_{r=1}^{i-1} (k-1)(k-2)\cdots(k-l)b_r(x_i-x_r)^{k-1-l} \\ \lim_{\varepsilon\to 0} S^{(l)}(x_i+\varepsilon) &= \sum_{j=1}^{k-1} j(j-1)\cdots(j-l+1)a_j(x_i-x_0)^{j-l} \\ &+ \sum_{r=1}^{i-1} (k-1)(k-2)\cdots(k-l)b_r(x_i-x_r)^{k-1-l} \\ &+ \lim_{\varepsilon\to 0} (k-1)(k-2)\cdots(k-l)b_i(x_i+\varepsilon-x_i)^{k-1-l} \end{aligned}$$

Thus,

$$\begin{aligned} &\lim_{\varepsilon\to 0} S^{(l)}(x_i+\varepsilon) - \lim_{\varepsilon\to 0} S^{(l)}(x_i-\varepsilon) \\ &\qquad = \lim_{\varepsilon\to 0} (k-1)(k-2)\cdots(k-l)b_i\varepsilon^{k-1-l} \end{aligned} \tag{7.5}$$

For $l = 0, 1, \ldots, k-2$, the righthand side is zero, so that

$$\lim_{\varepsilon\to 0} S^{(l)}(x_i+\varepsilon) = \lim_{\varepsilon\to 0} S^{(l)}(x_i-\varepsilon) \tag{7.6}$$

(7.5) shows the $S^{(l)}(x)$ is continuous at $x = x_i$.

When $l = k-1$ the righthand side becomes $(k-1)(k-2)\ldots1 \cdot b_i$.

Since generally $b_i \ne 0$

$$\lim_{\varepsilon\to 0} S^{(k-1)}(x_i+\varepsilon) \ne \lim_{\varepsilon\to 0} S^{(k-1)}(x_i-\varepsilon) \tag{7.7}$$

(7.7) shows that the $(k-1)$th derivative of $S(x)$ becomes discontinuous at $x = x_i$. Even in this case, if b_i, $i = 1, 2, \ldots, n-1$ are all zero, the $(k-1)$th derivative of $S(x)$ becomes continuous. Then, from (7.3), it can be found that $S(x) = p(x)$ over the range $[a, b]$. This means that $S(x)$ is virtually equal to the power series expanded at $x = x_0$. Therefore, it can be said that an arbitrary polynomial of degree $(k-1)$ defined on $[a, b]$ is a special form of the spline function. Equation (7.3) is referred to as the expression of spline function by the truncated power function, it is in general numerically unstable because $(x - x_i)^{k-1}$ tends to assume a large absolute value.

(3) **Representation-2 of spline functions (introduction of B-splines)**

In contrast with the representation(7.3), the representation by B-splines, which are defined below, can avoid numerical difficulties.

Let a series of points $\{t_r\}$ shown in Fig. 7.1 be defined as

$$\begin{aligned} &t_{-k+1} \le t_{-k+2} \le \cdots \le t_{-1} \le t_0 = x_0 < t_1 = x_1 < \cdots \\ &< t_n = x_n \le t_{n+1} \le t_{n+2} \le \cdots \le t_{n+k-1} \end{aligned} \tag{7.8}$$

And define $g_k(t; x)$ as a function of t with parameter x.

$$g_k(t;x) = (t-x)_+^{k-1} = \begin{cases} (t-x)^{k-1}, t \ge x \\ 0, t < x \end{cases} \tag{7.9}$$

See Fig. 7.2.

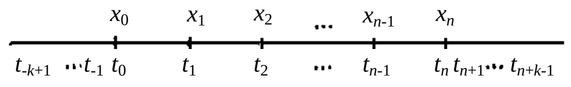

x_0
x_1
x_2
...
x_{n-1}
x_n
t_{-k+1} ... t_{-1} t_0
t_1
t_2
...
t_{n-1}
t_n t_{n+1} ... t_{n+k-1}

Fig. 7.1 A series of points

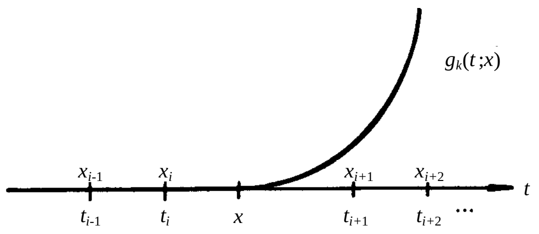


Fig. 7.2 g_k (t; x)

Then, the k th order divided difference of g_k (t ; x) with respect to $t = t_j, t_{j+1}, \ldots, t_{j+k}$ multiplied by a constant:

$$N_{j,k}(x) = (t_{j+k} - t_j)g_k[t_j, t_{j+1}, \cdots, t_{j+k}; x] \qquad (7.10)$$

is called the normalized B-spline (or simply B-spline) of degree (k -1).

The characteristics of B-spline $N_{j,k}$ (x) are as follows. Now, suppose that the position of x is moved with t_j, t_{j+1}, ..., t_{j+k} fixed. When $x \le t_j$ since $N_{j,k}(x)$ includes the k th order divided difference of a polynomial of degree (k-1) with respect to t, it becomes zero. When $t_{j+k} \le x$, $N_{jk}(x)$ is zero because it includes the k th order divided difference of a function which is identically zero. When $t_j < x < t_{j+k}$, $N_{j,k}(x) \neq 0$. In short,

$$N_{j,k}(x)\begin{cases} \neq 0, & t_j < x < t_{j+k} \\ = 0, & x \le t_j \text{ or } t_{j+k} \le x \end{cases} \qquad (7.11)$$

(actually, when $t_j < x < t_{j+k}$, $0 < N_{j,k}(x) \le 1$)
Next, suppose that j is moved with x fixed. Here, let $t_i = x_i < x < x_{i+1} = t_{i+1}$.
Then, in the same way as above, we can obtain

$$N_{j,k}(x)\begin{cases} \neq 0, & i-k+1 \le j \le i \\ = 0, & j \le i-k \text{ or } i+1 \le j \end{cases} \qquad (7.12)$$

The characteristics (7.11) and (7.12) are referred to as the locality of B-spline functions.
From (7.10), B-spline $N_{j,k}(x)$ can be written as

$$N_{j,k}(x) = (t_{j+k} - t_j)\sum_{r=j}^{j+k} \frac{(t_r - x)_+^{k-1}}{(t_r - t_j)\cdot\cdot(t_r - t_{r-1})(t_r - t_{r+1})\cdot\cdot(t_r - t_{j+k})} \qquad (7.13)$$

Therefore, $N_{j,k}(x)$ is a polynomial of degree (k-1) defined separately for each interval (x_i, x_{i+1}) and its derivatives of up to degree k-2 are continuous. Based on this characteristic of $N_{j,k}(x)$, it is proved that an arbitrary spline function $S(x)$ satisfying equation (7.2) can be represented as

$$S(x) = \sum_{j=-k+1}^{n-1} c_j N_{j,k}(x) \qquad (7.14)$$

where c_j, j = -k + 1, -k + 2, ... , n -1 are constants

(4) Calculating spline functions

Given a (k – 1)-th degree spline function,

$$S(x) = \sum_{j=-k+1}^{n-1} c_j N_{j,k}(x) \qquad (7.15)$$

the method of calculating its function value, derivatives and integral

$$\int_{x_0}^{x} S(y)dy$$

at the point $x \in [x_i, x_{i+1})$ is described hereafter.

– Calculating the function value
The value of S(x) at $x \in [x_i, x_{i+1})$ can be obtained by calculating $N_{j,k}(x)$. In fact, because of locality (7.12) of $N_{j,k}(x)$, only non-zero elements have to be calculated.
$N_{j,k}(x)$ is calculated based on the following recurrence equation

$$N_{r,s}(x) = \frac{x - t_r}{t_{r+s-1} - t_r} N_{r,s-1}(x) + \frac{t_{r+s} - x}{t_{r+s} - t_{r+1}} N_{r+1,s-1}(x) \qquad (7.16)$$

where,

$$\begin{aligned} N_{r,1}(x) &= (t_{r+1} - t_r)g_1[t_r, t_{r+1}; x] \\ &= (t_{r+1} - t_r)\frac{g_1(t_{r+1}; x) - g_1(t_r; x)}{t_{r+1} - t_r} \\ &= (t_{r+1} - x)_+^0 - (t_r - x)_+^0 \\ &= \begin{cases} 1, r = i \\ 0, r \neq i \end{cases} \end{aligned} \qquad (7.17)$$

By applying s = 2, 3, ..., k, $r = i - s + 1, i - s + 2, \ldots, i$ to Eqs. (7.16) and (7.17), all of the $N_{r,s}(x)$ given in Fig. 7.3 can be calculated, and the values in the rightmost column are used for calculating the $S(x)$.

$$\begin{tikzcd}
 & & & & N_{i-k+1,k} \arrow[d, dash, dashed] & \arrow[dd, "r"] \\
 & & 0 \arrow[urr, dash, dashed] & & & \\
 & 0 \arrow[r] & N_{i-2,3} & & & \\
0 \arrow[r] & N_{i-1,2} \arrow[r] & N_{i-1,3} & & N_{i-1,k} & \\
N_{i,1} \arrow[r] & N_{i,2} \arrow[r] & N_{i,3} \arrow[rr, dash, dashed] & & N_{i,k} & \\
0 & 0 & 0 & & & \\
 & & \arrow[rr, "s"'] & & &
\end{tikzcd}$$

Fig. 7.3 Calculating $N_{r,s}(x)$ at $x \in [x_i, x_{i+1})$

– Calculating derivatives and integral
From

$$\frac{d^l}{dx^l}S(x) = S^{(l)}(x) = \sum_{j=-k+1}^{n-1} c_j N_{j,k}^{(l)}(x) \tag{7.18}$$

$S^{(l)}(x)$ can be obtained by calculating $N_{j,k}^{(l)}(x)$.
From Eq. (7.9)

$$\frac{\partial^l}{\partial x^l} g_k(t;x) = \frac{(-1)^l (k-1)!}{(k-1-l)!}(t-x)_+^{k-1-l} \tag{7.19}$$

so $N_{j,k}^{(l)}(x)$ is the divided difference of order k at $t = t_j$, t_{j+1}, ..., t_{j+k} of Eq. (7.19).
Now let

$$d_k(t;x) = (t-x)_+^{k-1-l} = \begin{cases} (t-x)^{k-1-l} & ,t \ge x \\ 0 & ,t < x \end{cases}$$

and let $D_{j,k}(x)$ be the divided difference of order k at $t = t_j, t_{j+1}, \cdots, t_{j+k,}$ i.e.,

$$D_{j,k}(x) = d_k[t_j, t_{j+1}, \cdots, t_{j+k}; x] \tag{7.20}$$

This $D_{j,k}(x)$ can be calculated by the following recurrence equations. For $x \in [x_i, x_{i+1})$,

$$D_{r,1}(x) = \begin{cases} 1/(x_{i+1} - x_i) & ,r = i \\ 0 & ,r \ne i \end{cases}$$

$$D_{r,s}(x) = \frac{D_{r+1,s-1}(x) - D_{r,s-1}(x)}{t_{r+s} - t_s}, 2 \le s \le l+1$$

$$D_{r,s}(x) = \frac{(x - t_r)D_{r,s-1}(x) + (t_{r+s} - x)D_{r+1,s-1}(x)}{t_{r+s} - t_r}$$

$$,l+2 \le s \le k \tag{7.21}$$

and if $s = 2, 3, \ldots, k$, and $r = i - s + 1, i - s + 2, \ldots, i$ are applied, $D_{j,k}$ for $i - k + 1 \le j \le i$, can be obtained. The objective $N_{j,k}^{(l)}(x)$ can be obtained as follows:

$$N_{j,k}^{(l)}(x) = (t_{j+k} - t_j)\frac{(-1)^l (k-1)!}{(k-1-l)!} D_{j,k}(x)$$

and $S^{(l)}(x)$ can then be obtained by using this equation. Next, the integral is expressed as

$$I = \int_{x_0}^{x} S(y)dy = \sum_{j=-k+1}^{n-1} c_j \int_{x_0}^{x} N_{j,k}(y)dy \tag{7.22}$$

so it can be obtained by calculating $\int_{x_0}^{x} N_{j,k}(y)dy$

Integration of. $N_{j,k}(x)$ can be carried out by exchanging the sequence of the integration calculation with the calculation of divided difference included in $N_{j,k}(x)$.

First, from Eq. (7.9), the indefinite integral of $g_k(t\,;x)$ can be expressed by

$$\int g_k(t;x)dx = -\frac{1}{k}(t-x)_+^k$$

where an integration constant is omitted. Letting $e_k(t;x) = (t-x)_+^k$ and its divided difference of order k represent

$$I_{j,k}(x) = e_k[t_j, t_{j+1}, \ldots, t_{j+k}\,; x] \tag{7.23}$$

then the $I_{j,k}(x)$ satisfies the following recurrence equation.

$$I_{r,1}(x) = \begin{cases} 0 & ,r \le i-1 \\ (x_{i+1} - x)/(x_{i+1} - x_i) & ,r = i \\ 1 & ,r \ge i+1 \end{cases}$$

$$I_{r,s}(x) = \frac{(x - t_r)I_{r,s-1}(x) + (t_{r+s} - x)I_{r+1,s-1}(x)}{t_{r+s} - t_r} \tag{7.24}$$

where $x \in [x_i, x_{i+1})$.
If equation (7.24) is applied for $s = 2, 3, \ldots, k$ and $r = i - s + 1, i - s + 2, \ldots, i$ then a series of $I_{j,k}(x)$ are obtained as shown in the rightmost column in Fig. 7.4.

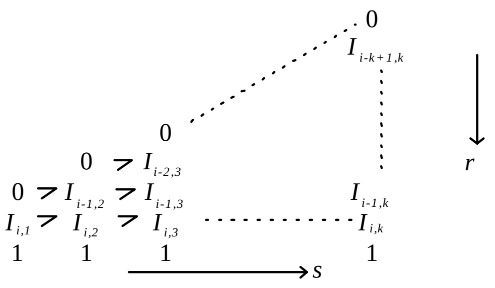


Fig. 7.4 Calculation $I_{r,s}(x)$ at $x \in [x_i, x_{i+1})$

The integration of $N_{j,k}(y)$ is represented by

$$\int_{x_0}^{x} N_{j,k}(y)dy = -\frac{(t_{i+k} - t_j)}{k}\left[I_{j,k}(x) - I_{j,k}(x_0)\right]$$
$$= \frac{(t_{j+k} - t_j)}{k}\left[I_{j,k}(x_0) - I_{j,k}(x)\right]$$

Therefore from Eq. (7.22),

$$I = \int_{x_0}^{x} S(y)dy = \frac{1}{k}\sum_{j=-k+1}^{n-1} c_j (t_{j+k} - t_j)\left[I_{j,k}(x_0) - I_{j,k}(x)\right]$$
$$= \frac{1}{k}\left\{\sum_{j=-k+1}^{0} c_j (t_{j+k} - t_j) I_{j,k}(x_0) - \sum_{j=i-k+1}^{i} c_j (t_{j+k} - t_j) I_{j,k}(x) + \sum_{j=1}^{i} c_j (t_{j+k} - t_j)\right\} \tag{7.25}$$

The coefficients c_j in Equation (7.15) has been so far assumed to be known in the calculation procedures for function values, derivatives, and integral values of the spline function $S(x)$. c_j can be determined from the interpolation condition if $S(x)$ is an interpolation function, or from least squares approximation if $S(x)$ is a smoothing function. In the case of interpolation, for example, since $n + k - 1$ coefficients c_j $(- k + 1 \le j \le n - 1)$ are involved in equation (7.15), c_j will be determined by assigning $n + k - 1$ interpolation conditions to Equation (7.15). If function values are given at $n + 1$ points $(x = x_0, x_1, ..., x_n)$ in Fig. 7.1 function values must be assigned at additional $(n + k - 1) - (n + 1) = k - 2$ points or $k - 2$ other conditions (such as those on the derivatives) of $S(x)$ must be provided in order to determine $n + k - 1$ coefficients c_j. Further information is available in 7.2 "Interpolation."

The SSL II applies the spline function of Eq. (7.15) to smoothing, interpolation, numerical differentiation, quadrature, and least squares approximation.

(5) Definition, representation and calculation method of bivariate spline function

The bivariate spline function can be defined as an extension of the one with a single variable described earlier.

Consider a closed region $R = \{(x,y) \mid a \le x \le b,\ c \le y \le d\}$ on the $x - y$ plane and points (x_i, y_j), where $0 \le i \le m$ and $0 \le j \le n$ according to the division(7.26)

$$a = x_0 < x_1 < \cdots < x_m = b$$
$$c = y_0 < y_1 < \cdots < y_n = d \qquad (7.26)$$

Denoting $D_x = \partial/\partial x$ and $D_y = \partial/\partial y$, the function $S(x, y)$ which satisfies

a. $D_x^k S(x,y) = D_y^k S(x,y) = 0$ for each of the open regions

$$R_{i,j} = \{(x,y) \mid x_i < x < x_{i+1}, y_j < y < y_{j+1}\} \qquad (7.27)$$

b. $S(x,y) \in C^{k-2,k-2}[R]$

is called a bivariate spline function of fual degree $k - 1$. (7.27) a. shows that $S(x,y)$ is a polynomial in x and y on each of R_{ij} and is at most $(k - 1)$-th degree with repeat to both of x and y. Further, b. shows that on the entire R

$$\frac{\partial^{\lambda+\mu}}{\partial x^{\lambda} \partial y^{\mu}} S(x,y)$$

exists and is continuous when $\lambda = 0, 1, .., k{-}2$ and $\mu = 0, 1, ..., k{-}2$.

If a series of points are taken as follows

$$s_{-k+1} \le s_{-k+2} \le \cdots \le s_{-1} \le s_0 = x_0 < s_1 = x_1 < \cdots < s_m = x_m \le s_{m+1} \le \cdots \le s_{m+k-1}$$

$$t_{-k+1} \le t_{-k+2} \le \cdots \le t_{-1} \le t_0 = y_0 < t_1 = y_1 < \cdots < t_n = y_n \le t_{n+1} \le \cdots \le t_{n+k-1}$$

the B-splines of in x and y directions are defined in the same way as the B-spline with a single variable.

$$N_{\alpha,k}(x) = (s_{\alpha+k} - s_\alpha)\, g_k[s_\alpha, s_{\alpha+1}, ..., s_{\alpha+k}\,; x]$$
$$N_{\beta,k}(y) = (t_{\beta+k} - t_\beta)\, g_k[t_\beta, t_{\beta+1}, ..., t_{\beta+k}\,; y]$$

Then the bivariate spline function of dual degree $k - 1$ defined above can be represented in the form

$$S(x,y) = \sum_{\beta=-k+1}^{n-1} \sum_{\alpha=-k+1}^{m-1} c_{\alpha,\beta} N_{\alpha,k}(x) N_{\beta,k}(y) \qquad (7.28)$$

where, $c_{\alpha,\beta}$ sare an arbitrary constants.

The calculation of function values, partial derivatives and indefinite integral of $S\,(x,y)$ can be done by simply applying to it the calculation for a single variable, if using the expression (7.28). First of all, for $\lambda \ge 0$ and $\mu \ge 0$,

$$S^{(\lambda,\mu)}(x,y) = \frac{\partial^{\lambda+\mu}}{\partial x^{\lambda} \partial y^{\mu}} S(x,y)$$
$$= \sum_{\beta=-k+1}^{n-1} \sum_{\alpha=-k+1}^{m-1} c_{\alpha,\beta} N_{\alpha,k}^{(\lambda)}(x) N_{\beta,k}^{(\mu)}(y) \qquad (7.29)$$

Therefore, the calculation of the function values and partial derivatives are accomplished by separately calculating $N_{\alpha,k}^{(\lambda)}(x)$, and $N_{\beta,k}^{(\mu)}(y)$ which can be done by applying the previously described method for a single variable.

Next, consider the value which is obtained by differentiating $S(x,y)$ μ times with respect to y and then by integrating with respect to x, namely

$$S^{(-1,\mu)}(x,y) = \int_{x_0}^{x} \frac{\partial^{\mu} S(x,y)}{\partial y^{\mu}} dx \qquad (7.30)$$

This value is unchanged even when the order of differentiation and integration is reversed. Rewriting the right-hand side of Eq. (7.30) by using Eq. (7.28), we obtain

$$\sum_{\alpha=-k+1}^{m-1} \left\{ \sum_{\beta=-k+1}^{n-1} c_{\alpha,\beta} N_{\beta,k}^{(\mu)}(y) \right\} \cdot \int_{x_0}^{x} N_{\alpha,k}(x) dx$$
$$= \sum_{\alpha=-k+1}^{m-1} c_\alpha \int_{x_0}^{x} N_{\alpha,k}(x) dx$$
$$\text{, where} \quad c_\alpha = \sum_{\beta=-k+1}^{n-1} c_{\alpha,\beta} N_{\beta,k}^{(\mu)}(y) \qquad (7.31)$$

This is similar to Eq. (7.23) given previously. Therefore, calculation of Eq. (7.31) is performed first by calculating c_α and then by calculating the integral by using the method for a single variable.

In addition $S^{(-1,\mu)}(x,y)$,

$$S^{(\lambda,-1)}(x,y)=\int_{y_0}^{y}\frac{\partial^{\lambda}S(x,y)}{\partial x^{\lambda}}dy$$

$$S^{(-1,-1)}(x,y)=\int_{y_0}^{y}dy\int_{x_0}^{x}S(x,y)dx$$

can be calculated by applying the method for calculating derivatives and integrals for a single variable each for x and y separately.

7.2 INTERPOLATION

The general procedure of interpolation is first to obtain an approximate function; ex., polynomial, piecewise polynomial, etc. which fits given sample points (x_i,y_i), then to evaluate that function.

When polynomials are used for approximation, they are called Lagrange interpolating polynomials or Hermite interpolating polynomials (using derivatives as well as function values). The Aitken-Lagrange interpolation and Aitken-Hermite interpolation methods used in SSL II belong to this. As a characteristic, they find the most suitable interpolated values by increasing iteratively the degree of interpolating polynomials.

While, piecewise polynomials are used for the approximate function when a single polynomial are difficult to apply. SSL II provides quasi-Hermite interpolation and spline interpolation methods.

Interpolating splines are defined as functions which satisfies the interpolating condition; i.e fits the given points. Interpolating splines are not uniquely determined: they can vary with some additional conditions. In SSL II, four types of spline interpolation are available. As for the representation of splines, we mainly use B-spline representain because of its numerical stability.

Interpolation by B-spline

Subroutines using B-spline are divided into two types according to their objectives.

(1) Subroutines by which interpolated values (or derivatives, integrals) are obtained

(2) Subroutines by which interpolating splines are obtained.

Since subroutines in (1) use interpolating splines, subroutines in (2) must be called first.

SSL II provides various interpolating splines using B-spline. Let discrete points be x_i, i = 1, 2, ..., n, then four types of B-spline interpolating function of degree m ($=2l-1$, $l\geq 2$) are available depending on the presence/absence or the contents of boundary conditions.

Type I .... $S^{(j)}(x_1)$, $S^{(j)}(x_n)$, $j = 1, 2, ..., l-1$ are specified by the user.

Type II .... $S^{(j)}(x_1)$, $S^{(j)}(x_n)$, $j = l, l+1, ..., 2l-2$ are specified by the user.

Type III .... No boundary conditions.

Type IV .... $S^{(j)}(x_1) = S^{(j)}(x_n)$, $j = 0, 1, ..., 2l-2$ are satisfied. This type is suitable to interpolate periodic functions.

Selection of the above four types depends upon the quantity of information on the original function available to the user.

Typically, subroutines of type III (No boundary conditions) can be used.

Bivariate spline function $\boldsymbol{S}(x,y)$ shown in (7.28) is used as an interpolation for a two-dimensional interpolation. In this case, different types could be used independently for each direction x and y. However, SSL II provides the interpolation using only type I or III in both directions of x and y.

It will be a problem how the degree of spline should be selected. Usually m is selected as 3 to 5, but when using double precision subroutines, if the original function does not change abruptly, m may take a higher value.

However, m should not exceed 15 because it may cause another problem.

Table 7.1 lists interpolation subroutines.

Quasi-Hermite interpolation

This is an interpolation by using piecewise polynomials similar to the spline interpolation. The only difference between the two is that quais-Hermite interpolation does not so strictly require continuity of higher degree derivatives as the spline interpolation does.

A characteristic of quasi-Hermite interpolation is that no wiggle appear between discrete points. Therefore it is suitable for curve fitting or surface fitting to the accuracy of a hand-drawn curve by a trained draftsman.

However, if very accurate interpolated values, derivatives or integrals are to be obtained, the B-spline interpolation should be used.

7.3 APPROXIMATION

This includes least-squares approximation polynomials as listed in Table 7.2. The least squares approximation using B-spline is treated in “Smoothing”.

7.4 SMOOTHING

Table 7.3 lists subroutines used for smoothing.

Subroutines SMLE1 and SMLE2 apply local least-squares approximation for each discrete point instead

Table 7.1 Interpolation subroutines

Objective	Subroutine name	Method	Notes
Interpolated value	AKLAG (E11-11-0101)	Aitken-Lagrange interpolation	Derivatives not needed.
	AKHER (E11-11-0201)	Aitken-Hermite interpolation	Derivatives needed
	SPLV (E11-21-0101)	Cubic spline interpolation	
	BIF1 (E11-31-0101)	B-spline interpolation (I)	Type I
	BIF2 (E11-31-0201	B-spline interpolation (II)	Type II
	BIF3 (E11-31-0301	B-spline interpolation (III)	Type III
	BIF4 (E11-31-0401	B-spline interpolation (IV)	Type IV
	BIFD1 (E11-32-1101)	B-spline two-dimensional interpolation(I-I)	Type I-I
	BIFD3 (E11-32-3301)	B-spline two-dimensional interpolation (III-III)	Type III-III
	AKMID (E11-42-0101)	Two-dimensional quasi-Hermite interpolation	
Interpolating function	INSPL (E12-21-0101)	Cubic spline interpolation	Two derivatives of the second order at both ends are needed
	AKMIN (E12-21-0201)	Quasi-Hermite interpolation	
	BIC1 (E12-31-0102)	B-spline interpolation (I)	Type I
	BIC2 (E12-31-0202)	B-spline interpolation (II)	Type II
	BIC3 (E12-31-0302)	B-spline interpolation (III)	Type III
	BIC4 (E12-31-0402)	B-spline interpolation (IV)	Type IV
	BICD1 (E11-32-1102)	B-spline two-dimensional interpolation (I-I)	Type I-I
	BICD3 (E12-32-3302)	B-spline two-dimensional interpolation (III-III)	Type III-III

Table 7.2 Approximation subroutine

Objective	Subroutine name	Method	Notes
Least squares approxi-mation polynomials	LESQ1 (E21-20-0101)	Discrete point polynomial	The degree of the polynomial is determined within the subroutine.

of applying the identical least-squares approximation over the observed values. However, it is advisable for the user to use B-spline subroutines for general purpose. In B-spline smoothing, spline functions shown in (7.14) and (7.28) are used for the one-dimensional smoothing and two-dimensional smoothing respectively. Coefficients c_j or $c_{\alpha,\beta}$ are determined by the least squares method. The smoothed value is obtained by evaluating the obtained smoothing function. SSL II provides subroutines for evaluating the smoothing functions.

There are two types of subroutines to obtain B-spline smoothing functions depending upon how to determine knots.

They are:

- The user specifies knots (fixed knots)
- Subroutines determine knots adaptively (variable knots)

The former requires experience on how to specify knots. Usually the latter subroutines are recommendable.

7.5 SERIES

SSL II provides subroutines shown in Table 7.4 for Chebyshev series expansion, evaluation of it, derivatives and indefinite integral.

Table 7.5 lists subroutines used for cosine series expansion, sine series expansion and their evaluation, which are for periodic functions.

Table 7.3 Smoothing subroutines

Objective	Subroutine name	Method	Notes
Smoothed value	SMLE1 (E31-11-0101)	Local least-squares approximation polynomials	Equally spaced discrete points
	SMLE2 (E31-21-0101)	Local least-squares approximation polynomials	Unequally spaced discrete points
	BSF1 (E31-31-0101)	B-spline smoothing	Unequally spaced discrete points
	BSFD1 (E31-32-0101)	B-spline two-dimensional smoothing	Unequally spaced lattice points
Smoothing function	BSC1 (E32-31-0102)	B-spline smoothing (fixed nodes)	Unequally spaced discrete points
	BSC2 (E32-31-0202)	B-spline smoothing (added nodes)	
	BSCD2 (E32-32-0202)	B-spline two-dimensional smoothing (added nodes)	Unequally spaced lattice points

Table 7.4 Chebyshev series subroutines

Objective	Subroutine name	Method	Notes
Series expansion	FCHEB (E51-30-0101)	Fast cosine transformation	Number of terms is (Power of 2) + 1.
Evaluation of series	ECHEB (E51-30-0201)	Backward recurrence equation	
Derivatives of series	GCHEB (E51-30-0301)	Differention formula for Chebyshev polynomials	
Indefinite inte-gral of series	ICHEB (E51-30-0401)	Integral formula for Chebyshev polynomials	

Table 7.5 Cosine or sine series subroutines

Objective	Subroutine name	Method	Notes
cosine series expansion	FCOSF (E51-10-0101)	Fast cosine transformation	Even functions
cosine series evaluation	ECOSP (E51-10-0201)	Backward recurrence equation	Even functions
sine series expansion	FSINF (E51-20-0101)	Fast sine transformation	Odd functions
sine series evaluation	ESINP (E51-20-0201)	Backward recurrence equation	Odd functions

CHAPTER 8 TRANSFORMS

8.1 OUTLINE

This chapter is concerned with discrete Fourier transforms and Laplace transforms.

For a discrete Fourier transform, subroutines are provided for each of the characteristics of data types.

- Real or complex data, and
- For real data, even or odd function

8.2 DISCRETE REAL FOURIER TRANSFORMS

When handling real data, subroutines are provided to perform the transform (8.1) and the inverse transform(8.2)

$$\left.\begin{aligned} a_k &= \frac{2}{n}\sum_{j=0}^{n-1} x_j \cos\frac{2\pi kj}{n}, k = 0,1,...,\frac{n}{2} \\ b_k &= \frac{2}{n}\sum_{j=0}^{n-1} x_j \sin\frac{2\pi kj}{n}, k = 1,2,...,\frac{n}{2}-1 \end{aligned}\right\} \qquad (8.1)$$

$$\begin{aligned} x_j = \frac{1}{2}a_0 + \sum_{k=1}^{n/2-1}\left(a_k \cos\frac{2\pi kj}{n} + b_k \sin\frac{2\pi kj}{n}\right) \\ + \frac{1}{2}a_{n/2}\cos\pi j, j = 0,1,...,n-1 \end{aligned} \qquad (8.2)$$

where a_k and b_k are called discrete Fourier coefficients.

If we consider the integrals,

$$\left.\begin{aligned} &\frac{1}{\pi}\int_0^{2\pi} x(t)\cos kt\,dt \\ &\frac{1}{\pi}\int_0^{2\pi} x(t)\sin kt\,dt \end{aligned}\right\} \qquad (8.3)$$

which define Fourier coefficients of a real valued function $x(t)$ with period 2π, the transfomrs (8.1) can be derived by representing the function $x(t)$ by n points $x_j = x\left(\frac{2\pi}{n}j\right), j = 0,1,...,n-1$, in the closed interval $[0,2\pi]$ and by applying the trapezoidal rule. Particularly, if $x(t)$ is the $(n/2 - 1)$th order trigonometric polynomial, the transforms (8.1) are the exact numerical integral formula of the integrals (8.3). In other words, the discrete Fourier coefficients are identical to the analytical Fourier coefficients.

If $x(t)$ is an even or odd function, the discrete cosine and sine transforms are provided by using their characteristics.

8.3 DISCRETE COSINE TRANSFORMS

For an even function $x(t)$, subroutines are provided to perform the two types of transform. One of the transforms uses the points including end points of the closed interval$[0,\pi]$, and the other transform does not include the end points.

- Discrete cosine transform (Trapezoidal rule)

  Representing an even function $x(t)$ by $x_j = x\left(\frac{\pi}{n}j\right)$, j=0, 1, ..., n in the closed interval $[0,\pi]$ the transform (8.4) and the inverse transform (8.5) are performed.

$$a_k = \frac{2}{n}\sum_{k=0}^{n}{}'' x_j \cos\frac{\pi}{n}kj, k = 0,1,...,n \qquad (8.4)$$

$$x_j = \sum_{k=0}^{n-1}{}'' a_k \cos\frac{\pi}{n}kj, j = 0,1,...,n \qquad (8.5)$$

where Σ'' denotes both the first and the last terms of the sum are taken with factor 1/2. The transform (8.4) can be derived by representing an even function $x(t)$ by $x_j = x\left(\frac{\pi}{n}j\right)$, $j = 0,1,..., n$ in the closed interval $[0,\pi]$ and by applying the trapezoidal rule to

$$\frac{2}{\pi}\int_0^{\pi} x(t)\cos kt\,dt \qquad (8.6)$$

which defining the Fourier coefficient of $x(t)$.

- Discrete cosine transform (midpoint rule)
  Representing an even function $x(t)$ by $x_{j+1/2} = x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)$, $j=0,1,...,n-1$ in the open interval $(0,\pi)$, the transform (8.7) and the inverse transform (8.8) are performed.

$$a_k = \frac{2}{n}\sum_{j=0}^{n-1} x_{j+\frac{1}{2}} \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right),\ k=0,1,...,n-1 \tag{8.7}$$

$$x_{j+\frac{1}{2}} = \sum_{k=0}^{n-1}{}' a_k \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right),\ j=0,1,...,n-1 \tag{8.8}$$

where Σ′ denotes the sum of terms except for the first term which is multiplied by 1/2.
The transform (8.7) can be derived by applying a midpoint rule with n terms to the integral (8.6).

8.4 DISCRETE SINE TRANSFORMS

If the function $x(t)$ is an odd function, subroutines are provided to perform the two types of transforms. Similar to the discrete cosine transform, one of the transforms is performed based on the trapezoidal rule, and the other on the midpoint rule.

- Discrete sine transform (Trapezoidal rule)
  Representing an odd function $x(t)$ by $x_j = x\left(\frac{\pi}{n}j\right)$, j=1, 2, ..., n-1, in the closed interval $[0,\pi]$ the transform (8.9) and the inverse transform (8.10) are performed.

$$b_k = \frac{2}{n}\sum_{j=1}^{n-1} x_j \sin\frac{\pi}{n}kj,\ k=1,2,...,n-1 \tag{8.9}$$

$$x_j = \sum_{k=1}^{n-1} b_k \sin\frac{\pi}{n}kj,\ j=1,2,...,n-1 \tag{8.10}$$

The transform (8.9) can be derived by representing the odd function $x(t)$ by $x_j = x\left(\frac{\pi}{n}j\right)$, j=1, 2, ..., n-1, in the closed interval $[0,\pi]$ and by applying the trapezoidal rule to the integral

$$\frac{2}{\pi}\int_0^{\pi} x(t)\sin kt\,dt \tag{8.11}$$

which defining Fourier coefficients of $x(t)$.

- Discrete sine transform (midpoint rule)
  Representing an odd function $x(t)$ by $x_{j+1/2} = x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)$, $j=0,1,...,n-1$ in the open interval $(0,\pi)$ the transform (8.12) and the inverse transform (8.13) are performed.

$$b_k = \frac{2}{n}\sum_{j=0}^{n-1} x_{j+\frac{1}{2}} \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right),\ k=1,2,...,n \tag{8.12}$$

$$x_{j+\frac{1}{2}} = \sum_{k=1}^{n-1} b_k \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right) + \frac{1}{2}b_n \sin\pi\left(j+\frac{1}{2}\right)$$
$$j=0,1,...,n-1 \tag{8.13}$$

The transform (8.12) can be derived by applying the midpoint rule with n terms to the integral (8.11).

8.5 DISCRETE COMPLEX FOURIER TRANSFORMS

For complex data, subroutines are provided to perform the transforms corresponding to the transform (8.14) and the inverse transform (8.15)

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \exp\left(-2\pi i\frac{jk}{n}\right), k=0,1,...,n-1 \tag{8.14}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \exp\left(2\pi i\frac{jk}{n}\right), j=0,1,...,n-1 \tag{8.15}$$

Transform (8.14) can be derived by representing the complex valued function $x(t)$ with period 2π by $x_j = x\left(\frac{2\pi}{n}j\right), j=0,1,...,n,$ in the closed interval $[0,2\pi]$ and by applying the trapezoidal rule to the integral

$$\frac{1}{2\pi}\int_0^{2\pi} x(t)\exp(-ikt)dt \tag{8.16}$$

which defines Fourier coefficients of $x(t)$.
The discrete type Fourier transforms described above are all performed by using the Fast Fourier Transform (FFT).
When transforms are performed by using the FFT, the internal processings are divided as follows:
(a) Transforms are performed by repeating elementary transforms of the small dimension in place.
(b) Arranging the data in the normal order.

Subroutines (component routines) are provided for each of the above processings.

The Fourier transform can be performed by using both subroutines for (a) and (b) above. Another subroutine (standard subroutine) which combines these subroutines is also provided and should usually be used.

The amount of data should be expressed by number to the power of 2 in taking the processing speed into consideration. However, for the complex Fourier transform, the following points are also considered:

- The amount of data can be expressed by either power of 2 or product of the powers of the prime numbers.
- Multi-variate transform can also be accomplished.

Table 8.1 lists the subroutines for each data characteristic.

Comments on use

- Number of Sample points
  The number of sample points, *n*, of transformed data is defined differently depending on the (data) characteristic of the function *x*(*t*). That is, *n* is
  - the number of sample points taken in the half period interval, (0,π), or [0,π], for the cosine and sine transforms, or
  - the number of sample points taken in the full period interval, [0,2π], for the real and complex trans-forms.
- Real transform against cosine and sine transforms
  If the function *x*(*t*) is an ordinary real function, the subroutine for a real transform can be used, but if it is known in advance that the *x*(*t*) is either an even or odd function, the subroutine for cosine and sine transforms should be used. (The processing speed is about half as fast as for a real transform.)
- Fourier coefficients in real and complex transforms
  The following relationships exist between the Fourier coefficients $\{a_k\}$ and $\{b_k\}$ used in a real transform (including cosine and sine transforms) and the Fourier coefficient $\{\alpha_k\}$ used in a complex transform.

$$\left.\begin{aligned} a_0 &= 2\alpha_0 \qquad\qquad , \; a_{n/2} = 2\alpha_{n/2} \\ a_k &= (\alpha_k + \alpha_{n-k}) \;, \; k = 1,2,...,n/2-1 \\ b_k &= i(\alpha_k - \alpha_{n-k}) \;, \; k = 1,2,...n/2-1 \end{aligned}\right\} \qquad (8.17)$$

where *n* denotes equally spaced points in a period [0,2π]. Based on the above relationships, users can use both subroutines for real and complex transforms when necessary. In this case, however, attention must be paid to scaling and data sequence.

- Trigonometric functions
  For cosine and sine transforms, the necessary trigonometric function table for transforms is provided in the subroutine for better processing efficiency. The function table is output in the parameter TAB which can be used again for successive transforms.
  For each transform, two subroutines are provided based on the trapezoidal rule and the midpoint rule. For the former, the size of the trigonometric function table is smaller and therefore more efficient.
- Scaling
  Scaling of the resultant values is left to the user.

Table 8.1 Subroutines for discrete Fourier transform

Type of transform		Amount of data	Subroutine name		
			Standard routine	Component routine	
				(a)	(b)
Cosine	Trapezoidal rule	(Power of 2) + 1	FCOST (F11-11-0101)		
	Midpoint rule	Power of 2	FCOSM (F11-11-0201)		
Sine	Trapezoidal rule		FSINT (F11-21-0101)		
	Midpoint rule		FSINM (F11-21-0201)		
Real transform			RFT (F11-31-0101)		
Complex transform			CFT (F12-15-0101)	CFTN (F12-15-0202)	PNR (F12-15-0402)
				CFTR (F12-15-0302)	
		Product of power of prime numbers	CFTM (F12-11-0101)		

Note:
(a) and (b) given in the table are described in Section 8.5.

8.6 LAPLACE TRANSFORM

The Laplace transform of $f(t)$ and its inverse are defined respectively as:

$$F(s) = \int_0^\infty f(t)e^{-st}dt \tag{8.18}$$

$$f(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \tag{8.19}$$

where $\gamma > \gamma_0$, γ_0 (: abscissa of convergence) and $i = \sqrt{-1}$.

In these transforms, $f(t)$ is called the original function and $F(s)$ the image function. Assume the following conditions about $F(s)$.

$$\left.\begin{array}{l} \text{1) F(s) is nonsingular for } \operatorname{Re}(s) > \gamma_0 \\ \text{2) } \lim\limits_{|s|\to\infty} F(s) = 0 \text{ for } \operatorname{Re}(s) > \gamma_0 \\ \text{3) } F^*(s) = F(s^*) \text{ for } \operatorname{Re}(s) > \gamma_0 \end{array}\right\} \tag{8.20}$$

where Re [•] denotes the real part of [•] and $F^*(s)$ the conjugate of $F(s)$. Condition 1) is always satisfied, condition 2) is satisfied unless $f(t)$ is a distribution and condition 3) is satisfied when $f(t)$ is a real function. The subroutines prepared perform the numerical transformation of expression (8.19). The outline of the method is described below.

Formula for numerical transformation

Assume $\gamma_0 \le 0$ for simplicity, that is $F(s)$ is regular in the domain of $\operatorname{Re}(s) > 0$, and the integral (8.19) exists for an arbitrary real value γ greater than 0. Since

$$e^s = \lim_{\sigma_0\to\infty} e^{\sigma_0}/[2\cosh(\sigma_0 - s)]$$

e^{st} in (8.19) is approximated as follows using an appropriate value for σ_0:

$$E_{ec}(st,\sigma_0) \equiv e^{\sigma_0}/[2\cosh(\sigma_0 - st)]$$

Function $E_{ec}(st,\sigma_0)$ is characterized as follows: There are an infinite number of poles on the line expressed by $\operatorname{Re}(s)=\sigma_0/t$. Figure 8.1 shows locations of the poles. This can be explicitly represented as:

$$E_{ec}(st,\sigma_0) = \frac{e^{\sigma_0}}{2t}\sum_{n=-\infty}^{\infty}\frac{(-1)^n i}{s - [\sigma_0 + i(n-0.5)\pi]/t}$$

Then, $f(t,\sigma_0)$ which denotes an approximation of the original function $f(t)$ is:

$$f(t,\sigma_0) \equiv \frac{1}{2\pi i}\int_{r-i\infty}^{r+i\infty} F(s)E_{ec}(st,\sigma_0)ds \tag{8.21}$$

where $\gamma_0 < \gamma < \sigma_0/t$ is assumed.

It follows that the integral of the right-hand side can be expanded in terms of integrals around the poles of $E_{ec}(st,\sigma_0)$.

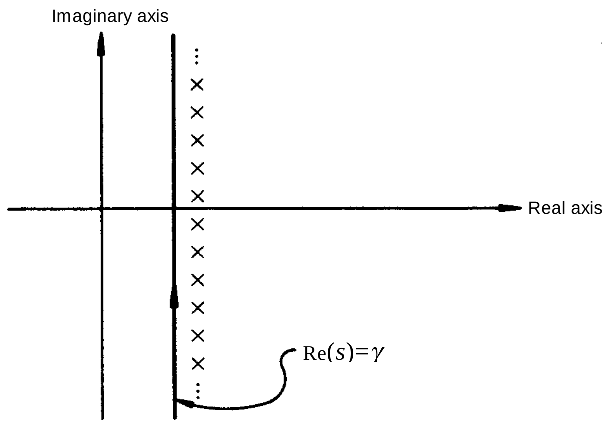


Fig. 8.1 Poles of $E_{ec}(st,\sigma_0)$

Since $F(s)$ is regular in the domain of $\operatorname{Re}(s) > 0$, the following is obtained according to Cauchy's integral formula:

$$\begin{aligned} f(t,\sigma_0) &= \frac{e^{\sigma_0}}{2t}\sum_{n=-\infty}^{n}(-1)^{n+1} iF\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right) \\ &= \frac{e^{\sigma_0}}{t}\sum_{n=1}^{\infty}(-1)^n \operatorname{Im}\left[F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right)\right] \end{aligned} \tag{8.22}$$

where Im [•] denotes the imaginary part of [•]. If $\gamma_0 > 0$ the condition $\gamma_0 < \gamma < \sigma_0/t$ cannot be satisfied for a certain value of $t (0 < t < \infty)$. This means $\gamma_0 \le 0$ is necessary for (8.22) to be used for $0 < t < \infty$.

Function $f(t,\sigma_0)$ gives an approximation to function $f(t)$ and is expressed as follows according to the error analysis in reference [98]:

$$f(t,\sigma_0) = f(t) - e^{-2\sigma_0}\cdot f(3t) + e^{-4\sigma_0}\cdot f(5t) - \cdots \tag{8.23}$$

This means that function $f(t,\sigma_0)$ gives a good approximation to $f(t)$ when $\sigma_0 \gg 1$. Moreover, (8.23) can be used for estimating the approximation error.

For numerical calculation, the approximation can be obtained principally by truncating (8.22) up to an appropriate term; however, the direct summation is often not practical. Euler transformation that can be generally applied in this case is incorporated in the subroutines. Define function F_n as follows:

$$F_n \equiv (-1)^n \operatorname{Im}\left[F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right)\right] \tag{8.24}$$

Then, Euler transformation is applicable when the following condition is satisfied (reference [100]):

1) For an integer $k \geq 1$, the sign of F_n alternates when $n \geq k$ (8.25)
2) $1/2 \leq | F_{n+1}/F_n | < 1$ when $n \geq k$

When F_n satisfies these conditions, the series represented by (8.22) can be transformed as:

$$\sum_{n=1}^{\infty} F_n = \sum_{n=1}^{k-1} F_n + \sum_{q=0}^{p} \frac{1}{2^{q+1}} D^q F_k + R_{p+1}(k) \tag{8.26}$$

where $R_p(k)$ is defined as:

$R_p(k) \equiv 2^{-P}(D^pF_k + D^pF_{k+1} + D^pF_{k+2} + \cdots)$

D^pF_k is the pth difference defined as

$$D^0F_k = F_k, D^{p+1}F_k = D^pF_k + D^pF_{k+1} \tag{8.27}$$

In the subroutines, the following expression is employed:

$$f_N(t,\sigma_0) = \frac{e^{\sigma_0}}{t} \sum_{n=1}^{N} F_n = \frac{e^{\sigma_0}}{t} \left\{ \sum_{n=1}^{k-1} F_n + \sum_{q=0}^{p} \frac{D^q F_k}{2^{q+1}} \right\} \tag{8.28}$$

where $N = k + p$,

$$\left. \begin{aligned} &\sum_{q=0}^{p} \frac{D^q F_k}{2^{q+1}} = \frac{1}{2^{p+1}} \sum_{r=0}^{p} A_{p,r} F_{k+r} \\ &A_{p,p} = 1, A_{p,r-1} = A_{p,r} + \binom{p+1}{r} \end{aligned} \right\} \tag{8.29}$$

Determination of values for σ_0, k, and p is explained in each subroutine description.

The following has been proved for the truncation error of $f_N(t,\sigma_0)$. Suppose $\phi(n) \equiv F_n$. If the p th derivative of $\phi(x)$, $\phi^{(p)}(x)$, is of constant sign for positive x and monotonously decreases with increase of x (for example, if $F(s)$ is a rational function), the following will be satisfied:

$$\begin{aligned} | f(t,\sigma_0) - f_N(t,\sigma_0) | &= \frac{e^{\sigma_0}}{t} | R_{p+1}(k) | \\ \leq | f_{N+1}(t,\sigma_0) - f_N(t,\sigma_0) | &= \frac{e^{\sigma_0}}{t} \left| \frac{1}{2^{p+1}} D^{p+1} F_k \right| \end{aligned} \tag{8.30}$$

where $f_{N+1}(t,\sigma_0)$ stands for (8.28) with $k + 1$ instead of k. To calculate $D^{p+1}F_k$ in the above formula, F_{k+p+1} is required, in addition to the set $\{F_n;\ n = k, k+1, ..., k+p\}$ to be used for calculation of $f_N(t,\sigma_0)$; hence, one more evaluation of the function is needed. To avoid that, the following expression is substituted for the truncation error of $f_N(t,\sigma_0)$ in the subroutines;

$$| f_N(t,\sigma_0) - f_{N-1}(t,\sigma_0) | = \frac{e^{\sigma_0}}{t} \left| \frac{1}{2^{p+1}} D^{p+1} F_{k-1} \right|$$

In the subroutines, the truncation error is output in the form of the following relation error:

$$\left| \frac{f_N(t,\sigma_0) - f_{N-1}(t,\sigma_0)}{f_N(t,\sigma_0)} \right| = \left| \frac{\frac{1}{2^{p+1}} D^{p+1} F_{k-1}}{\sum_{n=1}^{k-1} F_n + \frac{1}{2^{p+1}} \sum_{r=0}^{p} A_{p,r} F_{k+r}} \right|$$

$D^{p+1}F_{k-1}$ is a linear combination of F_{k-1}, F_k, ..., F_{k+p}, and the coefficients are equal to the binomial coefficients. $A_{p,r}$ can be calculated as the cumulative sum, as shown in (8.29). Thus, these coefficients can easily be calculated by using Pascal's triangle. Figure 8.2 shows this calculation techniques (for $p = 4$)

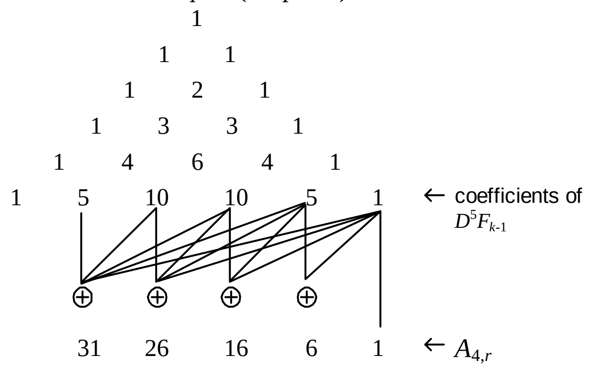


Fig. 8.2 Pascal's triangle (for p=4)

Next, in the case of $\gamma_0 > 0$, since $F(s)$ is not regular in the domain of $\mathrm{Re}(s) > 0$; the above technique cannot be directly applied. Note, however, that the integral in (8.19) can be expressed as:

$$\begin{aligned} f(t) &= \frac{1}{2\pi i} \int_{r-i\infty}^{r+i\infty} F(s+\gamma_0)\, e^{(s+\gamma_0)t} ds \\ &= \frac{e^{\gamma_0 t}}{2\pi i} \int_{r-i\infty}^{r+i\infty} G(s)\, e^{st} ds \\ &= e^{\gamma_0 t} g(t) \end{aligned}$$

where $r>0$, $G(s)=F(s+r_0)$ (8.31)

$$g(t) = \frac{1}{2\pi i} \int_{r-i\infty}^{r+i\infty} G(s)\, e^{st} ds$$

Since $G(s)$ is regular in the domain of $\mathrm{Re}(s) > 0$, $g(t)$ can be calculated as explained above; then $f(t)$ is obtained by multiplying $g(t)$ by $e^{\gamma_0 t}$

Transformation of rational functions
A rational function $F(s)$ can be expressed as follows using polynomials $Q(s)$ and $P(s)$ each having real coefficients:

$$F(s) = Q(s) / P(s) \tag{8.32}$$

To determine whether $\gamma_0 \le 0$ or $\gamma_0 > 0$, it is only necessary to check whether $P(s)$ is a Hurwitz polynomial (that is, all zeros are on the left-half plane $\{s \mid \mathrm{Re}(s)<0\}$. The procedure used for the check is described below (reference [94]):

A polynomial $P(s)$ of degree n with real coefficients is expressed as follows:

$$\begin{aligned} P(s) &= a_1 s^n + a_2 s^{n-1} + \cdots + a_n s + a_{n+1} \\ &= m(s) + n(s) \end{aligned}$$

where

$$\begin{aligned} m(s) &= a_1 s^n + a_3 s^{n-2} + \cdots, a_1 \ne 0 \\ n(s) &= a_2 s^{n-1} + a_4 s^{n-3} + \cdots \end{aligned}$$

The ratio of $n(s)$ to $m(s)$ is defined as:

$$W(s) \equiv m(s)/n(s)$$

Then, $W(s)$ is expanded into continued fraction as:

$$W(s) = h_1 s + \frac{1|}{|h_2 s} + \frac{1|}{|h_3 s} + \frac{1|}{|h_4 s} + \cdots \tag{8.33}$$

If all of $h_1, h_2, \ldots$, are positive, $P(s)$ is a Hurwitz polynomial. If $F(s)$ has singularities in the domain of $\mathrm{Re}(s) > 0$, the above procedure can be repeated increasing $\alpha(>0)$ so that $G(s)=F(s+\alpha)$ is regular in the domain of $\mathrm{Re}(s) > 0$. The value of $f_N(t,\sigma_0)$ is calculated by multiplying $e^{\alpha t}$ by $g_N(t,\sigma_0)$, the inverse of $G(s)$.

When $F(s)$ is an irrational function or a distribution, there is no practical method that tests if $F(s)$ is regular in the domain of $\mathrm{Re}(s) > 0$, therefore, the abscissa of convergence of a general function $F(s)$ must be specified by the user.

Choice of subroutines
Table 8.2 shows subroutines for the inversion of Laplace transforms. LAPS1 and LAPS2 are used for rational functions where LAPS1 for $\gamma_0 \le 0$ and LAPS2 otherwise. HRWIZ judges the condition $P(s)$, that is, examines if $\gamma_0 > 0$ in (8.32) is a Hurwitz polynomial; and if $\gamma_0 > 0$ is detected, the approximated value of γ_0 is calculated. The condition $\gamma_0 > 0$ means that the original function $f(t)$ increases exponentially as $t \to \infty$. So, HRWIZ can be used for examining such a behavior. Figure 8.3 shows a flowchart for choosing subroutines.

Table 8.2 Laplace transform subroutines

Function type	Subroutine name	Remarks
Rational functions	LAPS1 (F20-01-0101)	Rational functions regular in the right-half plane.
	LAPS2 (F20-02-0101)	General rational functions.
	HAWIZ (F20-02-0201)	Judgment on Hurwitz polynomials.
General functions	LAPS3 (F20-03-0101)	Convergence coordinate γ_0 must be input.

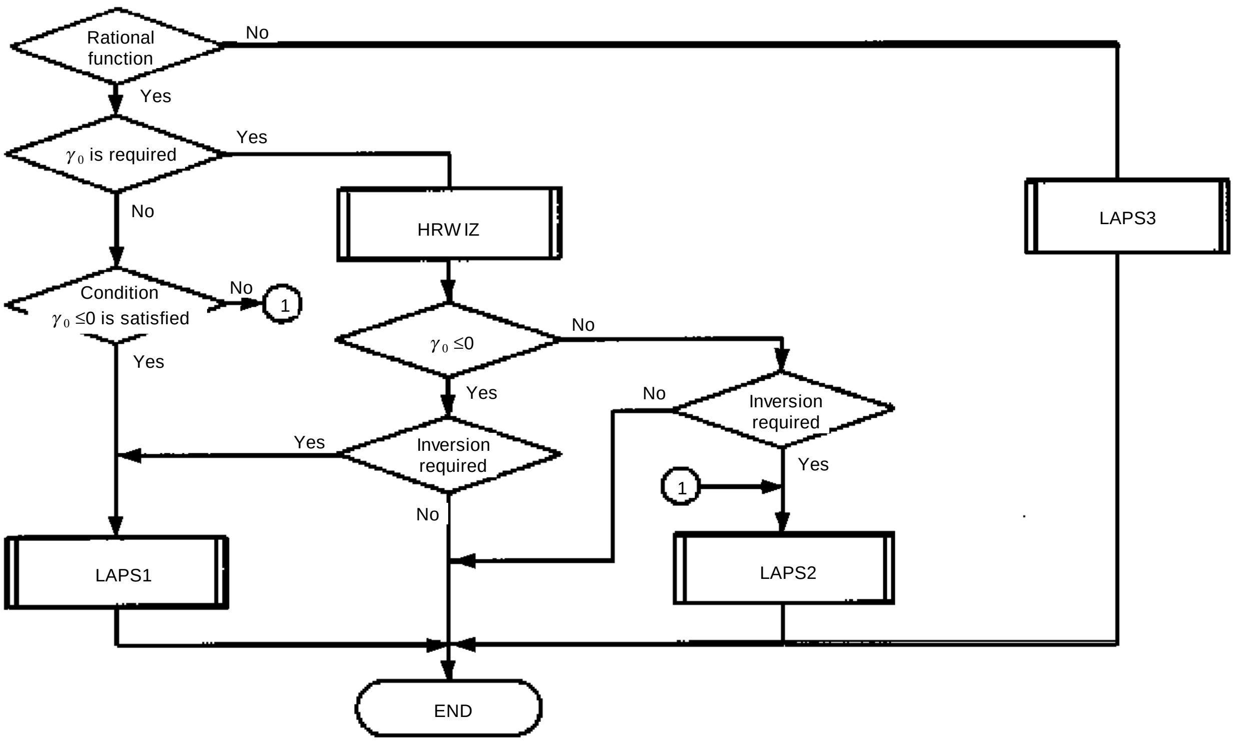


Fig. 8.3 Flowchart for choosing Laplace transform subroutines.

CHAPTER 9 NUMERICAL DIFFERENTIATION AND QUADRATURE

9.1 OUTLINE

This chapter describes the following types of problems.

• Numerical differentiation:
Given function values $y_i = f(x_i)$, $i = 1, \ldots n$ at discrete points $x_1, x_2, \ldots, x_n$ $(x_1 < x_2 < \ldots < x_n)$, the l - th order derivative $f^{(l)}(v)$, at $x = v$ in the interval $[x_1, x_n]$ is determined, where $l \geq 1$.
In addition two-dimensional differentiation is included. Also, given function $f(x)$, the following derivative is expanded to Chebyshev series.

$$f^{(l)}(x) = d^l f(x) / dx^l\ , \quad l \geq 1$$

• Numerical quadrature:
Given function values $y_i = f(x_i)$, $i = 1, \ldots, n$ at discrete points $x_1, x_2, \ldots, x_n$, the integral of $f(x)$ over the interval $[x_1, x_n]$ is determined. Also, given function $f(x)$, the integral

$$S = \int_a^b f(x)dx$$

is determined within a required accuracy. Multiple integrals are also included.

9.2 NUMERICAL DIFFERENTIATION

When performing numerical differentiation, SSL II divides problems into the following two types:

Discrete point input
In numerical differentiation, an appropriate interpolation function is first obtained to fit the given sample point (x_i, y_i) where $i = 1, 2, \ldots, n$, then it is differentiated.
Among several functions available, SSL II exclusively uses the spline function and we preferably use its B-spline representations.
See Chapter 7 as for the spline function and B-spline representations.

Function input
Given function $f(x)$ and domain $[a, b]$, $f(x)$ is expanded to Chebyshev series within a required accuracy. That is, it is approximated by the following functions:

$$f(x) \approx \sum_{k=0}^{n-1} c_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

Then by differentiating term by term.

$$f^{(l)}(x) \approx \sum_{k=0}^{n-l-1} c^l_k T_k \left(\frac{2x - (b+a)}{b-a} \right)$$

the derivatives are expanded to Chebyshev series. The derivative values are obtained by evaluating $f^{(l)}(v)$ at point $x = v$ in the interval in which the function is defined, that is, by summing Chebyshev series.
Table 9.1 lists subroutines used for numerical differentiation.

9.3 NUMERICAL QUADRATURE

Numerical Quadrature is divided into the following two types.

Integration of a tabulated function
Given function values $y_i = f(x_i)$, $i = 1, \ldots, n$ at discrete points $x_1 < x_2 < \ldots. < x_n$, the definite integral:

$$S = \int_{x_1}^{x_n} f(x)dx \qquad (9.1)$$

is approximated using only the given function values y_i. The bounds of error of the approximated value can not be calculated. Different subroutines are used depending on whether or not the discrete points are equally spaced.

Integration of a function
Given a function $f(x)$ and the interval of integration

Table 9.1 Subroutine used for numerical differentiation

Objective	Subroutine name	Method	Remarks
Derivative value	SPLV (E11-21-0101)	Cubic spline interpolation	Discrete point input
	BIF1 (E11-31-0101)	B-spline interpolation (I)	
	BIF2 (E11-31-0201)	B-spline interpolation (II)	
	BIF3 (E11-31-0301)	B-spline interpolation (III)	
	BIF4 (E11-31-0401)	B-spline interpolation (IV)	
	BSF1 (E31-31-0101)	B-spline smoothing	
	BIFD1 (E11-32-1101)	B-spline 2-dimensional interpolation (I-I)	Discrete point input 2-dimensional
	BIFD3 (E11-32-3301)	B-spline 2-dimensional interpolation (III-III)	
	BSFD1 (E31-32-0101)	B-spline 2-dimensional smoothing	
Derivative function and derivative value	FCHEB (E51-30-0101)	Fast cosine transformation	Function input, Chebyshev series expansion
	GCHEB (E51-30-0301)	Backward recurrence equation	Chebyshev series derivative
	ECHEB (E51-30-0201)	Backward recurrence equation	Summing Chebyshev series

[a, b], the definite integral:

$$S = \int_a^b f\ (x)\,dx \tag{9.2}$$

is calculated within a required accuracy. Different subroutines are used according to the form, characteristics, and the interval of integration of $f(x)$.

Besides (1. 2), the following types of integrals are calculated.

$$\int_0^\infty f(x)dx,$$

$$\int_{-\infty}^\infty f(x)dx,$$

$$\int_a^b dx\int_c^d f(x,y)dy$$

Subroutines used for numerical quadrature are shown in Table 9.2.

General conventions and comments on numerical quadrature

The subroutines used for numerical quadrature are classified primarily by the following characteristics.

- Dimensions of the variable of integration: 1 dimension or 2
- Interval of integration: finite interval, infinite interval, or semi-infinite interval.

Titles of the subroutines are based on that classification, so we say, for example:

- 1-dimensional finite interval integration
- 1-dimensional infinite interval integration

If a subroutine is characterized by other aspects or by its method, they are included in parentheses:

- 1-dimensional finite interval integration (unequally spaced discrete point input, trapezoidal rule)
- 1-dimensional finite interval integration (function input, adaptive Simpson's rule)

Numerical integration methods differ depending on whether a tabulated function or a continuous function is given. For a tabulated function, since integration is performed using just the function values $y_i = f(x_i)$, $i = 1, ...n$ it is difficult to obtain an approximation with high accuracy. On the other hand, if a function is given, function values in general can be calculated anywhere (except for singular cases), thus the integral can be obtained to a desired precision by calculating a sufficient number of function values. Also, the bounds of error can be estimated.

Integrals of one-dimensional functions over finite interval

The following notes apply to the subroutines which compute $\int_a^b f(x)dx$

Table 9.2 Numerical quadrature subroutines

Objective	Subroutine name	Method	Remarks
1-dimensional finite interval (equally spaced)	SIMP1 (G21-11-0101)	Simpson' s rule	Discrete point input
1-dimensional finite interval (unequally spaced)	TRAP (G21-21-0101)	Trapezoidal rule	
	BIF1 (E11-31-0101)	B-spline interpolation (I)	
	BIF2 (E11-31-0201)	B-spline interpolation (II)	
	BIF3 (E11-31-0301)	B-spline interpolation (III)	
	BIF4 (E11-31-0401)	B-spline interpolation (IV)	
	BSF1 (E31-31-0101)	B-spline smoothing	
2-dimensional finite interval	BIFD1 (E11-32-1101)	B-spline 2-dimensional interpolation (I-I)	Discrete point input 2-dimensional
	BIFD3 (E11-32-3301)	B-spline 2-dimensional interpolation (III-III)	
	BSFD1 (E31-32-0101)	B-spline 2-dimensional smoothing	
1-dimensional finite interval	SIMP2 (G23-11-0101)	Adaptive Simpson' s rule	Integration of a function
	AQN9 (G23-11-0201)	Adaptive Newton-Cotes 9 point rule	
	AQC8 (G23-11-0301)	Clenshaw-Curtis integration	
	AQE (G23-11-0401)	Double exponential formula	
1-dimensional semi-infinite interval	AQEH (G23-21-0101)	Double exponential formula	
1-dimensional infinite interval	AQEI (G23-31-0101)	Double exponential formula	
Multi-dimensional finite region	AQMC8 (G24-13-0101)	Clenshaw-Curtis quadrature	Multi-variate function input
Multi-dimensional region	AQME (G24-13-0201)	Double exponential formula	

- Automatic quadrature routines
  Four quadrature subroutines, SIMP2, AQN9, AQC8, and AQE are provided for the integration $\int_a^b f(x)dx$, as shown in Table 9.2. All these subroutines are automatic quadrature routines. An automatic quadrature routine is a routine which calculates the integral to satisfy the desired accuracy when integrand $f(x)$, integration interval $[a, b]$, and a desired accuracy for the integral are given. Automatic quadrature is the algorithm designed for this purpose.
  Generally in automatic quadrature subroutines, an integral calculation starts with only several abscissas (where the integrand is evaluated), and next improves the integral by increasing the number of abscissas gradually until the desired accuracy is satisfied. Then the calculation stops and the integral is output.
  In recent years, many automatic quadrature subroutines have been developed all over the world. These subroutines have been tested and compared with each other many times for reliability (i.e., ability to satisfy the desired accuracy) and economy (i.e., less calculation) by many persons. These efforts are well reflected in the SSL II subroutines.

- Adaptive method
  This is most (popularly) used for integral calculation as a typical method of automatic integration. This is not a specific integration formula (for example, Simpson’s rule, Newton-Cotes 9 point rule, or Gauss’ rule, etc.), but a method which controls the number of abscissas and their positions automatically in response to the behavior of integrand. That is, it locates

abscissas densely where integrand changes rapidly, or sparsely where it changes gradually. Subroutines SIMP2 and AQN9 use this method.

- Subroutine selection
  As a preliminary for subroutine selection, Table 9.3 shows several types of integrands from the viewpoint of actual use.
  It is necessary in subroutine selection to know which subroutine is suitable for the integrand. The types of routines and functions are described below in conjunction with Table 9.3.

Table 9.3 Integrand type

Code	Meaning	Example
Smooth	Function with good convergent power series.	$\int_0^{\pi} \sin x dx$, $\int_0^1 e^{-x} dx$
Peak	Function with some high peaks and wiggles in the integration interval.	$\int_{-1}^1 dx/(x^2+10^{-6})$
Oscilla-tory	Function with severe, short length wave oscillations.	$\int_0^1 \sin 100\pi x dx$
Singu-lar	Function with algebraic singularity (x^{α}, $-1 < \alpha$) or logarithmic singularity (log x).	$\int_0^1 dx/\sqrt{x}$, $\int_0^1 \log x dx$
Discon-tinuous	Function with discontinuities in the function value or its derivatives	$\int_0^1 [2x]\, dx$, $\int_0^{\pi} \lvert\cos x\rvert dx$

SIMP2 .... Uses adaptive method based on Simpson's rule. This is the first adaptive method used in the SSL II, and is the oldest in the history of adaptive methods. More useful adaptive methods are now available. That is, SIMP2 is inferior to the adaptive routine AQN9 in many respects.

AQN9 .... Adaptive method based on Newton-Cotes' 9-point rule. This is the most superior adaptive method in the sense of reliability or economy. Since this subroutine is good at detecting local actions of integrand, it can be used for functions which have singular points such as a algebraic singularity, logarithmic singularity, or discontinuities in the integration interval, and in addition, peaks.

AQC8 ... Since this routine is based on Chebyshev series expansion of a function, the better convergence property the integrand has the more effectively the routine can perform integration. For example, it can be used for smooth functions and oscillatory functions but is not suitable for singular functions and peak type functions.

AQE .....Method which extends the integration interval [*a*, *b*] to ($-\infty,\infty$) by variable transformation and uses the trapezoidal rule. In this processing, the transformation is selected so that the integrand after conversion will decay in a manner of a double exponential function (exp ($-a\cdot\exp|x|$), where a>0) when $x\rightarrow\infty$. Due to this operation, the processing is still effective even if the function change rapidly near the end points of the original interval [*a*, *b*]. Especially for functions which have algebraic singularity or logarithmic singularity only at the end points, processing is more successful than any other subroutine, but not so successful for functions with interior singularities.

Table 9.4 summarizes these descriptions. The subroutine marked by '○' is most suitable for corresponding type of function, and the subroutine marked by '×' should not be used for the type. No mark indicates that the subroutine is not always suitable but can be used. All these subroutines can satisfy the desired accuracy for the integral of smooth type. However, AQC8 is best in the sense of economy, that is, the amount of calculation is the least among the three.

SSL II provides subroutines AQMC8 and AQME for up to three-dimensional integration. They are automatic quadrature routines as shown below.

AQMC8 .... Uses Clenshaw-Curtis quadrature for each dimension. It can be used for a smooth and oscillatory functions. However, it is not applicable to functions having singular points or peaked functions.

AQME ..... Uses double exponential formula for each dimension. Since this subroutines has all formulas used in AQE, AQEH and AQEI, it can be used for any type of intervals (finite, semifinite or infinite interval)

Table 9.4 Subroutine selection

Function type / Subroutine	Smooth	Peak	Oscillatory	Singular		Discon-tinuous	Unknown*
				End point	Interior		
AQN9		O			O	O	O
AQC8	O	×	O	×	×	×	
AQE				O	×	×	

* Functions with unknown characteristics

and those combining these types.

AQME can be used efficiently if the function has singular points on the boundary of a region. However, it is not applicable to the function which has singular points in the region.

CHAPTER 10
DIFFERENTIAL EQUATIONS

10.1 OUTLINE

This chapter describes the following types of problems.

- Ordinary differential equations (initial value problems)
  Initial value problems of systems of first order oridnary differential equations are solved.

$$\left.\begin{array}{ll} y_1' = f_1(x, y_1, ..., y_n), & y_{10} = y_1(x_0) \\ y_2' = f_2(x, y_1, ..., y_n), & y_{20} = y_2(x_0) \\ \quad : & \quad : \\ y_n' = f_n(x, y_1, ..., y_n), & y_{n0} = y_n(x_0) \end{array}\right\} \quad (10.1)$$

Initial value problems of high order ordinary differential equations can be reduced to the form shown in (10.1). Namely, letting

$$y^{(k)} = f\left(x, y, y', y'', ..., y^{(k-1)}\right),$$
$$y_{10} = y(x_0), y_{20} = y'(x_0), ..., y_{k0} = y^{(k-1)}(x_0)$$

and

$$y_1 = y(x), y_2 = y'(x), ..., y_k = y^{(k-1)}(x)$$

Then, the high order equations can be reduced to and expressed as:

$$\left.\begin{array}{lll} y_1' = y_2 & , y_{10} & = y_1(x_0) \\ y_2' = y_3 & , y_{20} & = y_2(x_0) \\ \quad : & , & \quad : \\ y_{k-1}' = y_k & , y_{k-10} & = y_{k-1}(x_0) \\ y_k' = f(x, y_1, y_2, ..., y_k) & , y_{k0} & = y_k(x_0) \end{array}\right\} \quad (10.2)$$

10.2 ORDINARY DIFFERENTIAL EQUATIONS

To solve the initial value problem $y' = f(x, y)$, $y(x_0) = y_0$ on the interval $[x_0, x_e]$ means to obtain approximate solutions at discrete points

$x_0 < x_1 < x_2 < ... < x_e$

step by step as shown in Fig. 10.1.

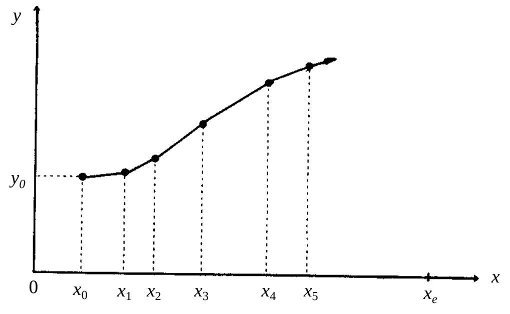


Fig. 10.1 Approximate solutions of $y' = f(x, y)$, $y(x_0) = y_0$

Solution output
In Fig. 10.1, solution output points x_1, x_2, x_3, ... are either specified by the user or selected as a result of step size control by the subroutine. The purpose of solving the differential equations is to obtain:
(a) the solution $y(x_e)$ only at x_e
(b) the solutions at the points selected as a result of step-size control by the subroutine. In this case, the purpose is to know the behavior of solutions, and no restriction is necessary to the solution output points because the behaviour of the solutions is all that is needed
(c) the solution at user-specified points $\{\xi_j\}$ or at equally spaced points.

The SSL II ordinary differential equation subroutines provide two output methods (timing to return to the user program from the subroutine) corresponding to the purposes described above, as follows:

- Final value output
  When the solution $y(x_e)$ is obtained, return to the user program. For the purpose of (c), set x_e to ξ_i sequentially, where i = 1, 2, ..., and call the subroutine repeatedly.

- Step output
  Under step-size control, return to the user program after one step integration. The user program can call this subroutine repeatedly to accomplish (b) described above.

SSL II provides subroutines ODRK1, ODAM and ODGE which incorporate final value output and step output. The user can select the manner of output by specifying a parameter.

Stiff differential equations

This section describes stiff differential equations, which appear in many applications, and presents definitions and examples.

The equations shown in (10.1) are expressed in the from of vectors as shown below.

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}), \boldsymbol{y}(x_0) = \boldsymbol{y}_0 \tag{10.3}$$

where

$$\boldsymbol{y} = (y_1, y_2, ..., y_N)^{\mathrm{T}},$$
$$\boldsymbol{f}(x, \boldsymbol{y}) = (f_1(x, \boldsymbol{y}), f_2(x, \boldsymbol{y}), ..., f_N(x, \boldsymbol{y}))^{\mathrm{T}},$$
$$f_i(x, \boldsymbol{y}) = f_i(x, y_1, y_2, ..., y_N)$$

Suppose $\boldsymbol{f}(x, \boldsymbol{y})$ is linear, that is

$$\boldsymbol{f}(x, \boldsymbol{y}) = \boldsymbol{Ay} + \boldsymbol{\Phi}(x) \tag{10.4}$$

where, $\boldsymbol{A}$ is a constant coefficient matrix and $\boldsymbol{\Phi}(x)$ is an appropriate function vector. Then, the solution for (10.3) can be expressed by using eigenvalues of $\boldsymbol{A}$ and the corresponding eigenvectors as follows:

$$\boldsymbol{y}(x) = \sum_{i=1}^{N} k_i e^{\lambda_i x} \boldsymbol{u}_i + \boldsymbol{\Psi}(x) \tag{10.5}$$

k_i : constant

Let us assume the following conditions for λ_i and $\boldsymbol{\Psi}(x)$ in (10.5):

(a) $\mathrm{Re}(\lambda_i)<0$, for i=1, 2, ..., N

(b) $\boldsymbol{\Psi}(x)$ is smoother than any $e^{\lambda_i x}$ (that is, it has good convergent power expansion).

Under these conditions, as x tends to infinity, the following can be seen.

$$\sum_{i=1}^{N} k_i e^{\lambda_i x} \boldsymbol{u}_i \to 0 \tag{10.6}$$

So, solution $\boldsymbol{y}(x)$ tends to $\boldsymbol{\Psi}(x)$. After $\boldsymbol{\Psi}(x)$ has become dominant, the solution can be obtained by the approximate solution for $\boldsymbol{\Psi}(x)$. The step sizes can be spaced rather roughly.

However, attempts to use methods such as Euler and classical Runge-Kutta encounter a phenomenon that errors introduced at a certain step increase from step to step. Therefore, when using these methods, the step sizes are substantially restricted. The larger the value of max ($|\mathrm{Re}(\lambda_i)|$) is, the smaller the step size must be.

Although solution $\boldsymbol{y}(x)$ can be approximated numerically by the smoothing function $\boldsymbol{\Psi}(x)$, the step sizes must be small for integration. This causes an imbalance between two step sizes, one of which is enough to approximate the solution numerically, and the other is required for error protection.

If $\boldsymbol{\Phi}(x)$ = 0, that is, $\boldsymbol{\Psi}(x)$ = 0 in (10.3), solution $\boldsymbol{y}(x)$ becomes smaller. Therefore, it is actually approximated by the term $k_i\ e^{\lambda_i x}\ u_i$ corresponding to the smallest $|\mathrm{Re}(\lambda_i)|$. In this case, if max $|\mathrm{Re}(\lambda_i)|$ is large, the above mentioned difficulty occurs.

The stiff differential equation is defined as follows:

- Definition 1
  When the following linear differential equation

  $$\boldsymbol{y}' = \boldsymbol{Ay} + \boldsymbol{\Phi}(x) \tag{10.7}$$

  satisfies the following (10.8) and (10.9),

  $$\mathrm{Re}(\lambda_i)<0, \quad i=1, 2, ..., N \tag{10.8}$$

  $$\frac{\max(|\mathrm{Re}(\lambda_i)|)}{\min(|\mathrm{Re}(\lambda_i)|)} \gg 1 \tag{10.9}$$

  they are called stiff differential equations. The left side of the equation in (10.9) is called stiff ratio. If this value is large, it is strongly stiff: otherwise, it is mildly stiff. Actually, strong stiffness with stiff ratio of magnitude 10^6 is quite common.

  An example of stiff linear differential equations is shown in (10.10). Its solution is shown in (10.11) (See Fig. 10.2).

  $$\boldsymbol{y}' = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix} \boldsymbol{y}, \boldsymbol{y}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{10.10}$$

  $$\boldsymbol{y} = e^{-x} \begin{pmatrix} 2 \\ -1 \end{pmatrix} + e^{-1000x} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \tag{10.11}$$

  Obviously, the following holds: when $x \to \infty$
  $y_1 \to 2e^{-x}$, $y_2 \to -e^{-x}$

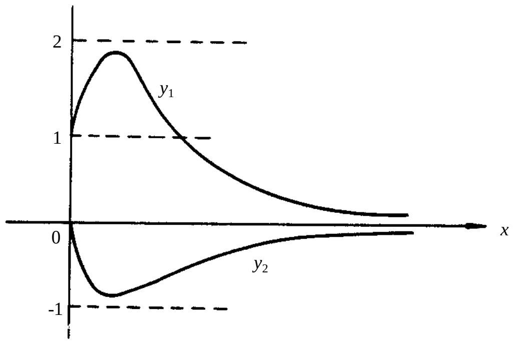


Fig. 10.2 Graph for the solution in (10.11)

Suppose $\boldsymbol{f}(x, \boldsymbol{y})$ is nonlinear.
The eigenvalue of the following Jacobian matrix determines stiffness.

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}(x, \boldsymbol{y})}{\partial \boldsymbol{y}}$$

where, the eigenvalues vary with x. Then, definition 1 is extended for nonlinear equations as follows.

- Definition 2
  When the following nonlinear differential equation

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}) \qquad (10.12)$$

satisfies the following (10.13) and (10.14) in a certain interval, it is said to be stiff in that interval.

$$\mathrm{Re}(\lambda_i(x)) < 0, i = 1,2,\ldots,N \qquad (10.13)$$
$$x \in I$$

$$\frac{\max(|\,\mathrm{Re}(\lambda_i(x))\,|)}{\min(|\,\mathrm{Re}(\lambda_i(x))\,|)} >> 1 \quad , x \in I \qquad (10.14)$$

where $\lambda_i(x)$ are the eigenvalues of $\boldsymbol{J}$.
Whether the given equation is stiff or not can be checked to some extent in the following way:

- When the equation is linear as shown in (10.7), the
- stiffness can be checked directly by calculating the eigenvalues of $\boldsymbol{A}$.
- When the equation is nonlinear, subroutine ODAM can be used to check stiffness. ODAM uses the Adams method by which non-stiff equations can be solved. ODAM notifies of stiffness via parameter ICON if the equation is stiff.
  Subroutine ODGE can be used to solve stiff equations.

Subroutine selection

Table 10.1 lists subroutines used for differential equations.

- ODGE for stiff equations
- ODRK1 or ODAM for non-stiff equations
  ODRK1 is effective when the following conditions are satisfied:
  - The accuracy required for solution is not high.
  - When requesting output of the solution at specific points of independent variable x, the interval of points is wide enough.
    The user should use ODAM when any of these conditions is not satisfied.
- Use ODAM at first when the equation is not recognized stiff.
  If ODAM indicated stiffness, then the user can shift to ODGE.

Table 10.1 Ordinary differential equation subroutines

Objective	Subroutine name	Method	Comments
Initial value problem	RKG (H11-20-0111)	Runge-Kutta Gill method	Fixed step size
	HAMNG (H11-20-0121)	Hamming method	Variable step size
	ODRK1 (H11-20-0131)	Rung-Kutta-Verner method	Variable step size
	ODAM (H11-20-0141)	Adams method	Variable step size, variable order
	ODGE (H11-20-0151)	Gear method	Variable step size, variable order (Stiff equations)

CHAPTER 11
SPECIAL FUNCTIONS

11.1 OUTLINE

The special functions of SSL II are functions not included in FORTRAN basic functions. The special functions are basically classified depending upon whether the variables and functions are real or complex.

Special functions
- Real type (variable and function are both real)
- Complex type (variable and function are both complex)

The following properties are common in special function subroutines.

Accuracy
The balance between accuracy and speed is important and therefore taken into account when selecting calculation formulas. In SSL II, calculation formulas have been selected such that the theoretical accuracies (accuracies in approximation) are guaranteed to be within about 8 correct decimal digits for single precision versions and 18 digits for double precision versions. To insure the accuracy, in some single precision versions, the internal calculations are performed in double precision. However, since the accuracy of function values depends on the number of working digits available for calculation in the computer, the theoretical accuracy cannot always be assured.

The accuracy of the single precision subroutines has been sufficiently checked by comparing their results with those of double precision subroutines, and for double precision subroutines by comparing their results with those of extended precision subroutines which have much higher precision than double precision subroutines.

Speed
Special functions are designed with emphasis on accuracy first and speed second. Though real type functions may be calculated with complex type function subroutines, separate subroutines are available with greater speed for real type calculations. Separate subroutines, single precision and double precision subroutines have been prepared also for interrelated special functions. For frequently used functions, both general and exclusive subroutines are available.

ICON
Special functions use FORTRAN basic functions, such as exponential functions and trigonometric functions. If errors occur in these basic functions, such as overflow or underflow, detection of the real cause of problems will be delayed. Therefore, to notice such troubles as early as possible, the detection is made before using basic functions in special function subroutines, and if detected, informations about them are returned in parameter ICON.

Calling method
Since various difficulties may occur in calculating special functions, subroutines for these functions have a parameter ICON to indicate how computations have finished. Accordingly,special functions are implemented in SUBROUTINE form which are called by using the CALL statements, while it is said that these functions should be implemented in FUNCTION form as basic functions.

11.2 ELLIPTIC INTEGRALS

Elliptic integrals are classified into the types shown in Table 11.1.

A second order iteration method can be used to calculate complete elliptic integrals, however, it has the disadvantage that the speed changes according to the magnitude of variable. In SSL II subroutines, an approximation formula is used so that a constant speed is maintained.

Table 11.1 Elliptic integral subroutines

	Item	Mathematical symbol	Subroutine name
Complete	Complete elliptic integral of the first kind	$K(k)$	CELI1 (I11-11-0101)
	Complete elliptic integral of the second kind	$E(k)$	CELI2 (I11-11-0201)

11.3 EXPONENTIAL INTEGRAL

Exponential integral is as shown in Table 11.2.

Table 11.2 Subroutines for exponential integral

Item	Mathematical symbol	Subroutine name
Exponential integral	$E_i(-x)$, $x > 0$	EXPI (I11-31-0101)
	$\overline{E}_i(x)$, $x > 0$	

Since exponential integral is rather difficult to compute, various formulas are used for various range of variable.

11.4 SINE AND COSINE INTEGRALS

Sine and cosine integrals are listed in Table 11.3.

Table 11.3 Subroutines for sine and cosine integrals

Item	Mathematical symbol	Subroutine name
Sine integral	$S_i(x)$	SINI (I11-41-0101)
Cosine integral	$C_i(x)$	COSI (I11-41-0201)

11.5 FRESNEL INTEGRALS

Fresnel integrals are shown in Table 11.4.

Table 11.4 Subroutines for Fresnel integrals

Item	Mathematical symbol	Subroutine name
Sine Fresnel integral	$S(x)$	SFRI (I11-51-0101)
Cosine Fresnel integral	$C(x)$	CFRI (I11-51-0201)

11.6 GAMMA FUNCTIONS

Gamma functions are provided as shown in Table 11.5.

Table 11.5 Subroutines for gamma functions

Item	Mathematical symbol	Subroutine name
Incomplete gamma function of first kind	$\gamma(\nu,x)$	IGAM1 (I11-61-0101)
Incomplete gamma function of second kind	$\Gamma(\nu,x)$	IGAM2 (I11-61-0201)

Between the complete Gamma function $\Gamma(\nu)$ and the first and the second kind incomplete Gamma functions the relationship

$$\Gamma(v) = \gamma(v, x) + \Gamma(v, x)$$

holds.

As for $\Gamma(v)$, the corresponding FORTRAN basic external function should be used.

11.7 ERROR FUNCTIONS

Error functions are provided as shown in Table 11.6.

Table 11.6 Subroutines for error functions

Item	Mathematical symbol	Subroutine name
Inverse error function	$\mathrm{erf}^{-1}(x)$	IERF (I11-71-0301)
Inverse complementary error function	$\mathrm{erfc}^{-1}(x)$	IERFC (I11-71-0401)

Between inverse error function and inverse complementary error function, the relationship

$$\mathrm{erf}^{-1}(x) = \mathrm{erfc}^{-1}(1 - x)$$

holds. Each is evaluated by using either function which is appropriate for that range of x.

As for $\mathrm{erf}(x)$ and $\mathrm{erfc}(x)$, the corresponding FORTRAN basic external function used.

11.8 BESSEL FUNCTIONS

Bessel functions are classified into various types as shown in Table 11.7, and they are frequently used by the user. Since zero-order and first-order Bessel functions are used quite often, exclusive subroutines used for them which are quite fast, are provided.

Table 11.7 Subroutines for Bessel functions with real variable

Item		Mathematical symbol	Subroutine name
First kind	Zero-order Bessel function	$J_0(x)$	BJ0 (I11-81-0201)
	First-order Bessel function	$J_1(x)$	BJ1 (I11-81-0301)
	Integer order Bessel function	$J_n(x)$	BJN (I11-81-1001)
	Real-order Bessel function	$J_v(x)$ $(v \geq 0.0)$	BJR (I11-83-0101)
	Zero order modified Bessel function	$I_0(x)$	BI0 (I11-81-0601)
	First order modified Bessel function	$I_1(x)$	BI1 (I11-81-0701)
	Integer order modified Bessel function	$I_n(x)$	BIN (I11-81-0701)
	Real order modified Bessel function	$I_v(x)$ $(v \geq 0.0)$	BIR (I11-83-0301)
Second kind	Zero-order Bessel function	$Y_0(x)$	BY0 (I11-81-0401)
	First-order Bessel function	$Y_1(x)$	BY1 (I11-81-0501)
	Integer order Bessel function	$Y_n(x)$	BYN (I11-81-1101)
	Real order Bessel function	$Y_v(x)$ $(v \geq 0.0)$	BYR (I11-83-0201)
	Zero order modified Bessel function	$K_0(x)$	BK0 (I11-81-0801)
	First order modified Bessel function	$K_1(x)$	BK1 (I11-81-0901)
	Integer order modified Bessel function	$K_n(x)$	BKN (I11-81-1301)
	Real order modified Bessel function	$K_v(x)$	BKR (I11-83-0401)

Table 11.8 Bessel function subroutines for complex variables

Item		Mathematical symbol	Subroutine name
First kind	Integer order Bessel function	$J_n(z)$	CBJN (I11-82-1301)
	Real order Bessel function	$J_v(z)$ $(v \geq 0.0)$	CBJR (I11-84-0101)
	Integer order modified Bessel function	$I_n(z)$	CBIN (I11-82-1101)
Second kind	Integer order Bessel function	$Y_n(z)$	CBYN (I11-82-1401)
	Integer order modified Bessel function	$K_n(z)$	CBKN (I11-82-1201)

11.9 NORMAL DISTRIBUTION FUNCTIONS

Table 11.9 lists subroutines used for normal distribution functions.

Table 11.9 Normal distribution function subroutines

Item	Mathematical symbol	Subroutine name
Normal distribution function	$\phi(x)$	NDF (I11-91-0101)
Complementary normal distribution function	$\psi(x)$	NDFC (I11-91-0201)
Inverse normal distribution function	$\phi^{-1}(x)$	INDF (I11-91-0301)
Inverse complementary normal distribution	$\psi^{-1}(x)$	INDFC (I11-91-0401)

CHAPTER 12
PSEUDO RANDOM NUMBERS

12.1 OUTLINE

This chapter deals with generation of pseudo-random (real or integer) numbers with various probability distribution functions, and with the test of random numbers.

12.2 PSEUDO RANDOM GENERATION

Random numbers with any given probability distribution can be obtained by transformation of the uniform (0, 1) pseudo-random numbers. That is, in generation of required pseudo random numbers let $g(x)$ be the probability density function of the distribution. Then, the pseudo-random numbers y are obtained by the inverse function (12.1) of $F(y)=\int_0^y g(x)dx$

$$y = F^{-1}(u) \tag{12.1}$$

where:
y is the required pseudo random number, $F(y)$ is the cumulative distribution function of $g(x)$ and u is uniform pseudo random number.

Pseudo-random numbers with discrete distribution are slightly more complicated by intermediate calculations. For example subroutine RANP2 first generates a table of cumulative Poisson distribution and a reference table which refers efficiently for a generated uniform (0, 1) number and then produces Poisson pseudo-random integers.

Table 12.1 shows a list of subroutines prepared for SSL II. These subroutines provide a parameter to be used as a starting value to control random number generation. (Usually, only one setting of the parameter will suffice to yield a sequence of random numbers.)

Table 12.1 List of subroutines for pseudo random number generation

Type	Subroutine name
Uniform (0, 1) pseudo random numbers	RANU2 (J11-10-0101)
Shuffled uniform (0, 1) pseudo random numbers	RANU3 (J11-10-0201)
Exponential pseudo random numbers	RANE2 (J11-30-0101)
Fast normal pseudo random numbers	RANN1 (J11-20-0301)
Normal pseudo random numbers	RANN2 (J11-20-0101)
Poisson pseudo random integers	RANP2 (J12-10-0101)
Binomial pseudo random numbers	RANB2 (J12-20-0101)

12.3 PSEUDO RANDOM TESTING

When using pseudo random numbers, the features of the random numbers must be fully recognized. That is, the random numbers generated arithmetically by a computer must be tested whether or not they can be assumed as "realized values of the sequence of each probability variable depending upon specific probability distribution".

SSL II generates pseudo random numbers with various probability distribution by giving appropriate transformation to the uniform (0, 1) pseudo random number. SSL II provides parameter IX to give the starting value for pseudo random number generation. This enables easier generation of some different random numbers. Generally to give the starting value, parameter IX is specified as zero. The results of testing the pseudo random numbers are described in comment on use for subroutines RANU2. The features of pseudo random numbers depend on the value of parameter IX. SSL II provides subroutines which are used to test the generated pseudo random numbers as shown in Table 12.2.

Table 12.2 Pseudo random number testing subroutines

Item	subroutine name	Notes
Frequency test	RATF1 (J21-10-0101)	Testing of probability unity
Runs test of up-and-down	RATR1 (J21-10-0201)	Testing of randomness

Test of statistical hypothesis
The test of statistical hypothesis is used to determine whether a certain hypothesis is accepted or rejected by the realized value of the amount of statistics obtained by random sampling.

Whether or not one of the dice is normal can be checked by throwing it for several times and checking the result.

Say, for example, that the same face came up five times in a row. If the die is normal, that is, if the ratio of obtaining a particular roll is identical for each face, the probability that the same face will come up five times in a row is $(1/6)^5$ which is 1/7776. This implies that if such testing is performed 7776 times repeatedly, the same combination is expected to come up once as an average. Therefore, if only 5 throws result in obtaining the same five numbers, the hypothesis that the die is normal, is assumed to be doubtful.

Suppose an event is tested under a certain hypothesis and occurs at less than the probability of α percent (generally 5 or 1 percent). In this state, if the event occurs by one testing, it is rejected because it is doubtful; otherwise it is received. This hypothesis has been tested whether or not it is accepted. It is called null hypothesis, where the region in which the probability is less than α percent is called critical region. The region is rejected or accepted according to a significance level.

When using this testing method, the hypothesis may be rejected despite the fact that it is true. This is expressed by α percent which is the level of significance.

Chi-square (χ^2) testing

Suppose the significance level at α percent. Also suppose the population is classified into l number of exclusive class $c_1, c_2, \ldots , c_l$, where when selecting n number of them, the actually corresponding frequencies are $f_1, f_2, \ldots , f_l$ and the expected frequencies are based on null hypothesis $F_1, F_2, \ldots , F_l$ respectively.

Class	$C_1, C_2, \ldots, C_l$	Total
Actual frequency	$f_1, f_2, \ldots, f_l$	n
Expected frequency	$F_1, F_2, \ldots, F_l$	n

Then the ratio of the actual frequency for the expected frequency is expressed as follows:

$$\chi_0^2 = \sum_{i=1}^{l} \frac{(f_i - F_i)^2}{F_i} \tag{12.2}$$

The larger the difference between the actual frequency and the expected frequency is, the larger the value of χ_0^2 becomes. Whether or not the hypothesis is accepted or rejected depends upon the ratio. The frequency varying for each n-size sampling is expressed by $\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_l$ – probability variables.
The statistic is expressed as

$$\chi^2 = \sum_{i=1}^{l} \frac{(\tilde{f}_i - F_i)^2}{F_i} \tag{12.3}$$

When expected frequency F_i is large enough, χ^2 is approximately distributed depending upon chi-square distribution of freedom l-1. Obtain point χ_α^2 equivalent to significance level percent in chi-square distribution of freedom l-1 for the following testing.

When $\chi_\alpha^2 < \chi_0^2$, the hypothesis is rejected.

When $\chi_\alpha^2 \geq \chi_0^2$, the hypothesis is accepted.

This is called chi-square (χ^2) testing. The value of actual frequency and expected frequency depend upon the contents (frequency testing, run testing) of testing.

Comments on use

- Sample size
  The size of a sample must be large enough. That is, the statistic in (12.3) is approximated to chi-square distribution of freedom l-1 for large n. If n is small, the statistic cannot be sufficiently approximated and the test results may not be reliable. The expected frequency should be

  $$F_i > 10 \quad , \; i=1, 2, \ldots , l \tag{12.4}$$

  If the conditions in (12.4) are not satisfied, freedom must be lower by combining several classes.

PART II
USAGE OF SSL II SUBROUTINES

A21-11-0101 AGGM, DAGGM

Addition of two real matrices
CALL AGGM (A, KA, B, KB, C, KC, M, N, ICON)

Function

These subroutines perform addition of two $m \times n$ real general matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$$

where $\boldsymbol{C}$ is an $m \times n$ real general matrix. $m, n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, two-dimensional array, A (KA, N).
KA	Input. The adjustable dimension of array A, (≥ M).
B	Input. Matrix $\boldsymbol{B}$, two-dimensional array B (KB, N).
KB	Input. The adjustable dimension of array B, (≥M).
C	Output. Matrix $\boldsymbol{C}$, two-dimensional array C (KC, N). (Refer to "Comment.")
KC	Input. The adjustable dimension of array C, (≥ M).
M	Input. The number of rows m of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$
N	Input. The number of columns n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.
ICON.	Input. Condition codes. Refer to Table AGGM-1.

Table AGGM-1 Condition code

Code	Meaning	Processing
0	No error	
30000	M<1, N<1, KA<M, KB<M or KC<M	Bypassed

Comments on use

- Subprograms used
  SSL II .....MGSSL
  FORTRAN basic function ... None

- Notes
  Saving the storage area:
  If there is no need to keep the contents on the array A or B, more storage area can be saved by specifing parameters C and KC as follows;
  When the contents of array A are not needed:

  CALL AGGM (A, KA, B, KB, A, KA, M, N, ICON)

  When the contents of array B are not needed:

  CALL AGGM (A, KA, B, KB, B, KB, M, N, ICON)

  In this case, matrix $\boldsymbol{C}$ is stored in array A or B.

- Example
  The following shows an example of obtaining the addition of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Here, $m, n \leq 50$.

```
C     **EXAMPLE**
      DIMENSION A(50,50),B(60,60),C(100,100)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
      DATA KA/50/,KB/60/,KC/100/
   10 READ(5,100) M,N
      IF(M.EQ.0) STOP
      WRITE(6,150)
      READ(5,200) ((A(I,J),I=1,M),J=1,N)
      READ(5,200) ((B(I,J),I=1,M),J=1,N)
      CALL AGGM(A,KA,B,KB,C,KC,M,N,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PGM(IA,1,A,KA,M,N)
      CALL PGM(IB,1,B,KB,M,N)
      CALL PGM(IC,1,C,KC,M,N)
      GOTO 10
  100 FORMAT(2I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX ADDITION **')
      END
```

The subroutine PGM in the example is for printing a real matrix. this program is shown in the example for subroutine MGSM.

E11-11-0201 AKHER, DAKHER

Aitken-Hermite interpolation
CALL AKHER (X, Y, DY, N, V, M, EPS, F, VW, ICON)

Function

Given discrete points $x_1 < x_2 < ... < x_n$, function values $y_i = f(x_i)$, and first derivatives $y_i = f(x_i)$, $i = 1, ..., n$ this subroutine interpolates at a given point $x = v$ using the Aitken-Hermite interpolation.
$n \geq 1$.

Parameters

X	Input. Discrete points x_i. X is a one-dimensional array of size n.
Y	Input. Function value y_i. Y is a one-dimensional array of size n.
DY	Input. First order derivatives y'_i. DY is a one-dimensional array of size n.
N	Input. Number (n) of discrete points.
V	Input. The point to be interpolated.
M	Input. Number of discrete points to be used in the interpolation ($\leq n$). Output. Number of discrete points actually used. (See the comments)
EPS	Input. Threshold value. Output. Absolute error of the interpolated value. (See the comments)
F	Output. Interpolated value.
VW	Work area. One-dimensional array of size $5n$
ICON ..	Output. Condition code. Refer to Table AKHER-1.

Table AKHER-1 Condition codes.

Code	Meaning	Processing
0	No error	
10000	υ is equal to one of the discrete points x_i.	F is set to y_i.
30000	$n < 1$, M = 0, or $x_i \geq x_{i+1}$	F is set to 0.0.

Comments

- Subprogram used
  SSL II ... AFMAX, MGSSL
  FORTRAN basic functions ... ABS, and IABS

- Notes
  Stopping criterion:
  Let's consider the effect of the degree of interpolation on numerical behavior first. Here, Z_j denotes the interpolated value obtained by using j discrete points near $x = v$. Discrete points are ordered according to their closeness to $x = v$. The difference D_j is

$$D_j \equiv Z_j - Z_{j-1}, \quad j = 2,...,m$$

where m is the maximum number of discrete points to be used. Generally, as the order of an interpolation polynomial increases, $|D_j|$ behaves as shown in Fig. AKHER-1.

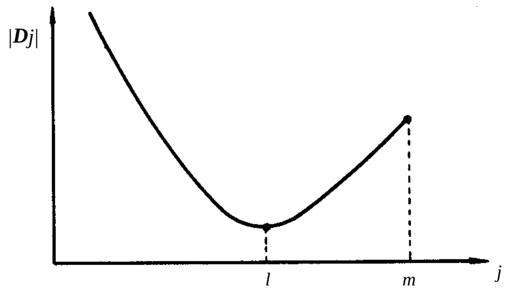


Fig. AKHER-1

In Fig. AKHER-1, l indicates that the truncation error and the calculation error of the approximation polynomial are both at the same level. Usually, Z_l is considered as numerically the optimum interpolated value.

How to specify EPS:
The following conditions are considered. Convergence is tested as described in "Stopping criterion", but D_j exhibits various types of behavior depending on the tabulated function. As shown in Fig. AKHER-2 in some cases vacillation can occur.

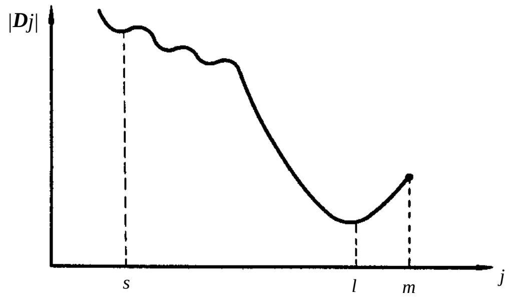


Fig. AKHER-2

In this case, Z_l instead of Z_s should be used for the interpolated value. Based on this philosophy the interpolated value to be output is determined as shown below. When calculating $D_2, D_3, ..., D_m$,

– If $|D_j|>|\mathrm{EPS}|, j=2, 3, ... , m$
  l is determined such that

$$|D_l| = \min_j(|D_j|) \qquad (3.1)$$

and the parameters F, M, and EPS are set to the value of Z_l, l, $|D_l|$ and

– if $|D_j| \leq$ EPS occurs for a certain j, from then on, l is determined such that

$$|D_l| \leq |D_{l+1}| \tag{3.2}$$

If (3.2) does not occur, l is set to m, and Z_m, m and $|D_m|$ are output. If the user specifies EPS as 0.0, Z_j corresponding the minimum $|D_j|$ is output as the interpolated value.

How to specify M:

a) If it is known that in the neighbourhood of $x = v$ the original function can be well approximated by polynomials of degree $2k-1$ or less, it is natural to use a polynomial of the degree $2k-1$ or less. In this parameter M should be se specified equal to k.

b) If the condition in a) is unknown, parameter N should be entered in parameter M.

c) It is possible that the user wants an interpolated value which is obtained by using exactly m points without applying the stopping criterion. In this case, the user can specify M equal to $-m$.

- Example
  The values of the input parameters are read and the interpolated value F is determined $n \leq 30$.

```
C      **EXAMPLE**
       DIMENSION X(30),Y(30),DY(30),VW(150)
       READ(5,500) N,(X(I),Y(I),DY(I),I=1,N)
       WRITE(6,600) (I,X(I),Y(I),DY(I),I=1,N)
    10 READ(5,510) M,EPS,V
       IF(M.GT.30) STOP
       CALL AKHER(X,Y,DY,N,V,M,EPS,F,VW,ICON)
       WRITE(6,610) ICON,M,V,F
       IF(ICON.EQ.30000) STOP
       GO TO 10
   500 FORMAT(I2/(3F10.0))
   510 FORMAT(I2,2F10.0)
   600 FORMAT(15X,I2,5X,3E20.8)
   610 FORMAT(20X,'ICON =',I5,10X,'M =',I2/
      *      20X,'V           =',E20.8/
      *      20X,'COMPUTED VALUES =',E20.8)
       END
```

Method

Let discrete point x_i be rearranged as $v_1, v_2, \ldots, v_n$ according to their distance from v with the closest value being selected first, and correspondingly $y_i = f(v_i)$ and $y'_i = f(v_i)$

The condition $y_i = f(v_i)$ at point (v_i) is symbolized here as $(i, 1)$. Now, let's consider how to obtain the interpolated value which is based on the (2m-1) th degree interpolating polynomial that satisfies the 2m conditions (1,0), (1,1), (2,0), (2,1), ... , $(m,0)$, $(m,1)$. (Hereafter, this value will be referred as the interpolated value which satisfies the conditions $(i, 0)$, $(i, 1)$, $i = 1, \ldots, m$

Before discussing general cases, an example in the case m = 2 is shown that determines an interpolated value which satisfies the four conditions (1,0), (1,1), (2,0), (2,1).

- Procedure 1
  An interpolated value

$$P_{A_1}(v) \equiv y_{1,1'} \equiv y_1 + y'_1(v - v_1)$$

which satisfies (1,0), (1,1) is determined. An interpolated value

$$P_{A_3}(v) \equiv y_{2,2'} \equiv y_2 + y'_2(v - v_2)$$

which satisfies(2,0), (2,1) is determined.

- Procedure 2
  An interpolated value

$$P_{A2}(v) \equiv y_{1,2} \equiv y_1 + \frac{y_1 - y_2}{v_1 - v_2}(v - v_1)$$

which satisfies conditions (1,0), (2,0) is determined.

- Procedure 3
  An interpolated value

$$P_{A4}(v) \equiv y_{1,1',2} \equiv P_{A1}(v) + \frac{P_{A1}(v) - P_{A2}(v)}{v_1 - v_2}(v - v_1)$$

which satisfies conditions (1,0), (1,1), (2.0) is determined.
An interpolated value

$$P_{A5}(v) \equiv y_{1,2,2'} \equiv P_{A2}(v) + \frac{P_{A2}(v) - P_{A3}(v)}{v_1 - v_2}(v - v_1)$$

Which satisfies conditions (1,0), (2,0), (2,1) is determined.

- Procedure 4
  An interpolated value

$$P_{A6}(v) \equiv y_{1,1',2,2'} \equiv P_{A4}(v) + \frac{P_{A4}(v) - P_{A5}(v)}{v_1 - v_2}(v - v_1)$$

which satisfies condition (1,0), (1,1), (2,0), (2,1) is determined.

Then $P_{A6}(v)$ is the objective interpolated value. For general cases, based on the following formulas

$$y_{i,i'} \equiv y_i + y_i'(v - v_i) \qquad (i = 1, \ldots, m) \tag{4.1}$$

$$y_{i,i+1} \equiv y_i + \frac{y_i - y_{i+1}}{v_i - v_{i+1}}(v - v_i) \qquad (i = 1, \ldots, m) \tag{4.2}$$

the same procedure as with m = 2 is followed. See Fig. AKHER-3.

$$\begin{array}{ccccccc}
y_{1,1'} & y_{1,1',2} & y_{1,1',2,2'} & y_{1,1',2,2',3} & y_{1,1',2,2',3,3'} & \cdot & y_{1,1',2,2',\ldots,m,m'} \\
y_{1,2} & y_{1,2,2'} & y_{1,2,2',3} & y_{1,2,2',3,3'} & \cdot & \cdot & \\
y_{2,2'} & y_{2,2',3} & y_{2,2',3,3'} & \cdot & \cdot & & \\
y_{2,3} & y_{2,3,3'} & \cdot & \cdot & & & \\
y_{3,3'} & \cdot & \cdot & & & & \\
\cdot & \cdot & & & & & \\
y_{m,m'} & & & & & &
\end{array}$$

Fig. AKHER-3 For general cases

For further information, see Reference [47].

E11-11-0101 AKLAG, DAKLAG

Aitken-Lagrange interpolation
CALL AKLAG (X,Y, N, V, M, EPS, F, VW, ICON)

Function

Given discrete points $x_1 < x_2 < ... < x_n$ and their corresponding function values $y_i = f(x_i)$, $i = 1, ..., n$, this subroutine interpolates at a given point $x = v$ using the Aitken-Lagrange interpolation. $n \geq 1$

Parameters

X ...	Input. Discrete points x_i. X is a one-dimensional array of size n.
Y	Input. Function values y_i. Y is a one-dimensional array of size n.
N	Input. Number of discrete points n.
V	Input. The point to be interpolated.
M	Input. Number of discrete points to be used in the interpolation ($\leq n$). Output. Number of discrete points actually used. (See the comments)
EPS ...	Input. Threshold value. Output. Absolute error of the interpolated value. (See the comments)
F	Output. Interpolated value.
VW	Work area. A one-dimensional array of size $4n$.
ICON	Output. Condition code. Refer to Table AKLAG-1.

Comments on use

- Subprograms used
  SSL II ... AFMAX, MGSSL
  FORTRAN basic functions ... ABS, and IABS

Table AKLAG-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	v matched a discrete point x_i.	F is set to y_i.
30000	$n < 1$, M = 0 or $x_i \geq x_{i+1}$	F is set to 0.0.

- Notes
  Stopping criterion:
  Let's consider the effect of the degree of interpolation on numerical behavior first. Here, Z_j denotes the interpolated value obtained by using j discrete points near $x = v$. (Discrete points are selected such that the points closest to $x = v$ are selected first.)
  The difference D_j is defined:

$$D_j \equiv Z_j - Z_{j-1}, \quad j = 2,...,m$$

where m is the maximum number of discrete points to be used. Generally, as the degree of an interpolation polynomial increases, $|D_j|$ behaves as shown in Fig. AKLAG-1

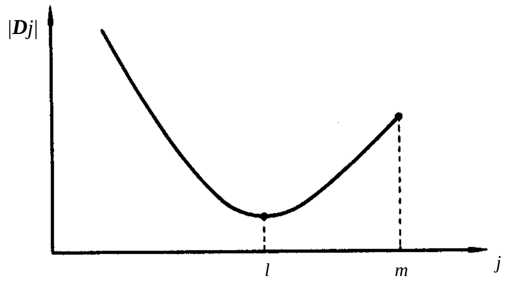


Fig. AKLAG-1

In Fig. AKLAG-1,l indicates that the truncation error and the calculation error of the approximation polynomial are both at the same level. Z_l is usually consided as the numerically optimum interpolated value.

How to specify EPS:
The following conditions are considered. Convergence is tested as described in "Stopping criterion", but D_j exhibits various types of behavior depending on the tabulated function. As shown in Fig. AKLAG-2, vacillation can occur in some cases.

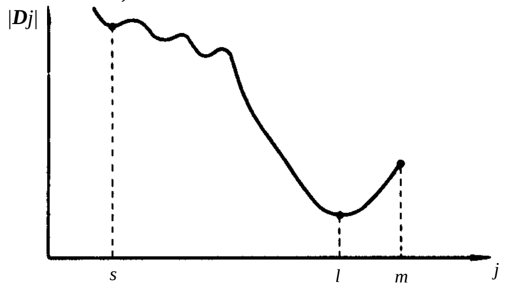


Fig. AKLAG-2

In this case, Z_l instead of Z_s should be used for the interpolated value. Based on this, the interpolated value to be output is determined as shown below.
When calculating $D_2, D_3, ..., D_m$,
– If $|D_j| > |\text{EPS}|$, j=2, 3, ..., m
  l is determined such that

$$|D_l| = \min_j \left(|D_j| \right) \tag{3.1}$$

– if $|D_j| \leq |\text{EPS}|$ occurs for a certain j, from then on l is determined such that

$$|D_l| \le |D_{l+1}| \qquad (3.2)$$

and Z_l, l, D_i are output.
If (3.2) does not occur, l is set to m, and Z_m, m, and $|D_m|$ are output.
If the user specifies EPS as 0.0 Z_j corresponding the minimum $|D_j|$ is output as the interpolated value.

How to specify M:

a) If it is known that in the neighbourhood of $x = v$ the original function can be well approximated by polynomials of degree k or less, it is natural to use interpolating polynomials of degree k or less. In this case parameter M should be specified equal to $k + 1$.
b) If the condition in a) is unknown, parameter M should be the same as parameter N.
c) It is possible that the user wants a interpolated value which is obtained by using exactly m points without applying the stopping criterion. In this case, the user can specify M equal to $-m$.

- Example

The input parameters are read, and the interpolated value F is determined. $n \le 30$

```
C      **EXAMPLE**
       DIMENSION X(30),Y(30),VW(120)
       READ(5,500) N,(X(I),Y(I),I=1,N)
       WRITE(6,600) (I,X(I),Y(I),I=1,N)
    10 READ(5,510) M,EPS,V
       IF(M.GT.30) STOP
       CALL AKLAG(X,Y,N,V,M,EPS,F,VW,ICON)
       WRITE(6,610) ICON,M,V,F
       IF(ICON.EQ.30000) STOP
       GO TO 10
   500 FORMAT(I2/(2F10.0))
   510 FORMAT(I2,2F10.0)
   600 FORMAT(23X,'ARGUMENT VALUES',15X,
      *'FUNCTION VALUES'/(15X,I2,5X,
      *E15.7,15X,E15.7))
   610 FORMAT(20X,'ICON =',I5,10X,'M =',I2/
      *       20X,'V          =',E15.7/
      *       20X,'COMPUTED VALUES =',E15.7)
       END
```

Method

Let discrete points x_i be rearranged as $v_1, v_2, ..., v_n$ according to their distance from v with the closest value being selected first, and corresponding $y_i = f(v_i)$. Usually, a subset of the sample points $(v_i, y_i = f(v_i);\ i = 1, ..., m)$ is used for interpolation as shown below. The interpolated values of the Lagrangian interpolation polynomial of degree i which passes through the discrete points (v_1, v_1), (v_2, v_2), ..., (v_i, y_i), (v_j, y_j) are expressed here as $y_{1,2...,i,j}$ (where $j > i$).
The Aitken-Lagrange interpolation method is based on:

$$y_{1,2,...,i,j} = \frac{1}{v_j - v_i} \begin{vmatrix} y_{1,2,...,i} & v_i - v \\ y_{1,2,...,i-1,j} & v_j - v \end{vmatrix}$$

$$= y_{1,2,...,i} + \frac{y_{1,2,...,i} - y_{1,2,...,i-1,j}}{v_j - v_i}(v_i - v)$$

As shown in Fig. AKLAG-3, calculation proceeds from the top to bottom of each column, starting at the first and ending at the mth column, row to the bottom row and from the left column to the right, such that $y_{1.2}, y_{1.3}, y_{1.2.3}, y_{1.4}, y_{1.2.4}, ...$

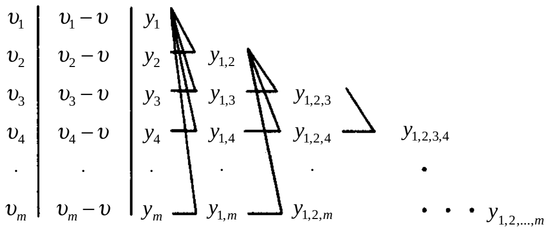


Fig. AKLAG-3

The values $y_{1,2,......,k}$ on the diagonal line are interpolated values based on the Lagrangian interpolation formula which passes through the discrete points $(v_i, y_i,\ i = 1, ..., k)$. Then, $y_{1,2,......,m}$ is the final value.
For details, see Reference [46] pp.57-59.

E11-42-0101 AKMID, DAKMID

Two-dimensional quasi-Hermite interpolation
CALL AKMID (X, NX, Y, NY, FXY, K, ISW, VX, IX, VY, IY, F, VW, ICON)

Function

Given function values $f_{ij} = f(x_i, y_j)$ at the node points (x_i, y_j), $i = 1, 2, ..., n_x, j = 1, 2, ..., n_y (x_1 < x_2 < ... < x_{nx}, y_1 < y_2 < ... < x_{ny}$, an interpolated value at the point $(P(v_x, v_y)$, is obtained by using the piecewise two-dimensional quasi-Hermite interpolating function of dually degree 3. See Fig. AKMID-1.

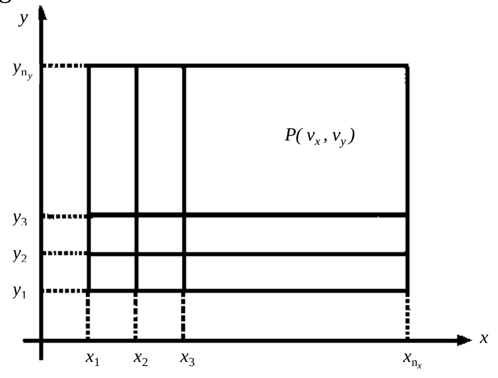


Fig. AKMID-1 Point P in the area $R=\{(x,y) \mid x_1 \le x_{nx}, y_1 \le y \le y_{ny}\}$

Parameters

X .... Input. Discrete points x_j' s in the x-direction. One-dimensional array of size n_x.

NX .... Input. Number of x_j' s, n_x

Y …. Input. Discrete points y_j's in the y-direction. One-dimensional array of size n_y.

NY .... Input. Number of y_j' s, n_y

FXY .... Input. Function value f_{ij}. Two-dimensional array as FXY (K, NY). f_{ij} needs to be assigned to the element FXY(I,J).

K .... Input. Adjustable dimension for array FXY (K $\ge$ NX).

ISW .... Input. ISW = 0 (INTEGER *4) must be assigned the first time the subroutine is called with the input data (x_i, y_j, f_{ij}) given. When a series of interpolated values need to be obtained by calling the subroutine repeatedly, the ISW value must not be changed from the second time on.
Output. Information on (i,j) which satisfies $x_i \le \boldsymbol{v}_x < x_{i+1}$ and $y_j \le v_y < y_{j+1}$.
When the user starts interpolation for the newly given data (x_i, y_j, f_{ij}), he needs to set the ISW to zero again.

VX .... Input. x-coordinate at the point $P(v_x, v_y)$.

IX .... Input. The i which satisfies $x_i \le v_x < x_{i+1}$. When $v_x = x_{nx}$, IX = $n_x - 1$.
Output. The i which satisfies $x_i \le v_x < x_{i+1}$. See Note.

VY .... Input. y-coordinate at the point $P(v_x, v_y)$.

IY .... Input. The j which satisfies $y_j \le v_y < y_{j+1}$. When $v_y = y_{ny}$, IY = $n_y - 1$.
Output. The j which satisfies $y_j \le v_y < y_{j+1}$ See Note.

F .... Output. Interpolated value.

VW .... Work area. One-dimensional array of size 50. While the subroutine is called repeatedly with identical input data (x_i, y_j, f_{ij}), the contents of VW must not be altered.

ICON ... Output. Condition code. See Table AKMID-1.

Table AKMID-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Either X(IX)≤VX<X(IX+1) or Y(IY)≤VY<Y(IY+1) is not satisfied.	IX or IY satisfying the relationship on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: 1 X(I) which satisfies X(I)≥X(I+1) exists 2 Y(J) which satisfies Y(J) ≥ Y (J+1) exists 3 NX<3 or NY < 3 4 K< NX 5 VX < X(1) or VX > X(NX) 6 VY < Y(1) or VY > Y(NY) 7 ISW specification is wrong.	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... IABS, ABS, and MOD

- Notes
  The interpolating function used in the subroutine and its first order derivative are continuous in the area $R=\{(x,y) \mid x_1 \le x \le x_{nx}, y_1 \le y \le y_{ny}\}$, but its second order and the higher order derivative of the function may not be continuous. On the other hand, this interpolating function has a characteristic that irregular points or planes do not appear.
  To obtain an interpolated value, derivative and integral value for a bivariate function, with accuracy

subroutines BIFD3 or BIFD1, which use an interpolation method by the spline function, should be used. When obtaining more than one interpolated value with the identical input data (x_i, y_j, f_{ij}), the subroutine is more effective if it is called with its input points continuous in the same grid area. (See "Example".) In this case parameters ISW and VW must not be altered.

The parameters IX and IY should satisfy X(IX) ≤ VX < X(IX +1) and Y(IY) ≤ VY < Y(IY+1), respectively. If not, IX and IY which satisfy these relationships are searched for to continue the processing.

The parameter error conditions accompanied with ICON = 30000 are listed in Table AKMID-1. Of the error conditions, 1 to 4 are checked only when ISW = 0 is specified, i.e., when the subroutine is called the first time with the input data (x_i, y_j, f_{ij}) given.

- Example

By inputting points (x_i, y_j) and their function values f_{ij}: $i = 1, 2, ..., n_x$, $j = 1, 2, ..., n_y$, interpolated values at points (v_{il}, v_{jk}), shown below are obtained. $n_x \le 121$ and $n_y \le 101$.

$v_{il} = x_i + (x_{i+1} - x_i) \times (l/4)$
$\quad i = 1, 2, ..., n_x - 1,\ l = 0, 1, 2, 3$
$v_{jk} = y_j + (y_{j+1} - y_j) \times (k/2)$
$\quad j = 1, 2, ..., n_y - 1, k = 0, 1$

```
C      **EXAMPLE**
       DIMENSION X(121),Y(101),FXY(121,101),
      *          VW(50),XV(4),YV(2),FV(4,2)
       READ(5,500) NX,NY
       READ(5,510) (X(I),I=1,NX)
       READ(5,510) (Y(J),J=1,NY)
       READ(5,510) ((FXY(I,J),I=1,NX),J=1,NY)
       WRITE(6,600) NX,NY
       WRITE(6,610) (I,X(I),I=1,NX)
       WRITE(6,620) (J,Y(J),J=1,NY)
       WRITE(6,630) ((I,J,FXY(I,J),I=1,NX)
      *             ,J=1,NY)
       ISW=0
       NX1=NX-1
       NY1=NY-1
       DO 40 I=1,NX1
       HX=(X(I+1)-X(I))*0.25
       DO 10 IV=1,4
    10 XV(IV)=X(I)+HX*FLOAT(IV-1)
       DO 40 J=1,NY1
       HY=(Y(J+1)-Y(J))*0.5
       DO 20 JV=1,2
    20 YV(JV)=Y(J)+HY*FLOAT(JV-1)
       DO 30 IV=1,4
       DO 30 JV=1,2
    30 CALL AKMID(X,NX,Y,NY,FXY,121,ISW,
      *          XV(IV),I,YV(JV),J,
      *          FV(IV,JV),VW,ICON)
    40 WRITE(6,640) I,J,((IV,JV,FV(IV,JV),
      *             IV=1,4),JV=1,2)
       STOP
   500 FORMAT(2I6)
   510 FORMAT(6F12.0)
   600 FORMAT('1'//10X,'INPUT DATA',3X,
      *'NX=',I3,3X,'NY=',I3/)
   610 FORMAT('0','X'/(6X,6(I6,E15.7)))
   620 FORMAT('0','Y'/(6X,6(I6,E15.7)))
   630 FORMAT('0','FXY'/(6X,5('(',2I4,
      *E15.7,')')))
   640 FORMAT('0','APP.VALUE',2I5/(6X,
      *5('(',2I4,E15.7,')')))
       END
```

Method

The subroutine obtains interpolated values based on the dual third degree two-dimensional quasi-Hermite interpolating function which is a direct extension of the one-dimensional quasi-Hermite interpolating function obtained by subroutine AKMIN.

- Dual third degree two-dimensional quasi-Hermite interpolating function.

  The interpolation function $S(x, y)$ described hereafter is defined in the area $R = \{(x,y) | x_1 \le x \le x_{nx}, y_1 \le y \le y_{ny}\}$ and satisfies the following condition:

  (a) $S(x, y)$ is polynomial at most of dually degree three within each partial region $R_{i,j} = \{(x,y) | x_i \le x < x_{i+1}, y_j \le y < y_{j+1}\}$.

  (b) $S(x,y) \in C^{1,1}[R]$, that is, the following values exist and are all continuous on R:

$$S^{(\alpha,\beta)}(x,y) = \frac{\partial^{\alpha+\beta}}{\partial x^{\alpha} \partial y^{\beta}} S(x,y),$$
$$\alpha = 0,1,\ \beta = 0,1$$

  (c) $S^{(\alpha,\beta)}(x_i, y_j) = f^{(\alpha,\beta)}(x_i, y_j),$
$$\alpha = 0,1,\ \beta = 0,1,\ i = 1,2,...,n_x,\ j = 1,2,...,n_y$$

It has been proved that the function $S(x, y)$ exists uniquely and that it can be expressed, in partial region R_{ij}, as follows:

$$\begin{aligned} S(x,y) &= S_{i,j}(s,t) \\ &= \sum_{\alpha=0}^{1} \sum_{\beta=0}^{1} a_i^{\alpha} b_j^{\beta} \{ f_{ij}^{(\alpha,\beta)} p_{\alpha}(s) q_{\beta}(t) \\ &+ f_{i+1,j}^{(\alpha,\beta)} q_{\alpha}(s) p_{\beta}(t) + f_{i,j+1}^{(\alpha,\beta)} p_{\alpha}(s) q_{\beta}(t) \\ &+ f_{i+1,j+1}^{(\alpha,\beta)} q_{\alpha}(s) q_{\beta}(t) \end{aligned} \quad (4.1)$$

where

$$a_i = x_{i+1} - x_i,\ b_j = y_{j+1} - y_j$$
$$s = \frac{x - x_i}{a_i},\ t = \frac{y - y_j}{b_j},\ 0 \le s, t < 1$$

$$
\begin{aligned}
p_0(t) &= 1 - 3t^2 + 2t^3 \\
q_0(t) &= 3t^2 - 2t^3 \\
p_1(t) &= t(t-1)^2 \\
q_1(t) &= t^2(t-1) \\
f_{ij}^{(\alpha,\beta)} &= f^{(\alpha,\beta)}(x_i, y_j) = \frac{\partial^{\alpha+\beta}}{\partial x^\alpha \, \partial y^\beta} f(x_i, y_j)
\end{aligned}
$$

Eq. (4.1) requires function values and derivatives at the four points (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1}) and (x_{i+1}, y_{j+1}). Therefore, if the derivatives can be obtained (or approximated in some way) an interpolated value in the area R_{ij} can be obtained by using Eq. (4.1). The $S(x, y)$ given in Eq. (4.1) which can be obtained by using approximated derivatives is called the dual thrid degree piecewise two-dimensional quasi-Hermite interpolating function.

- Determination of derivatives $f_{ij}^{(1,0)}$, $f_{ij}^{(0,1)}$ and $f_{ij}^{(1,1)}$ at a node point
  This subroutine uses the Akima's geometrical method to obtain these derivatives.
  It applies the method used by the one-dimensional quasi-Hermite interpolating function (in subroutine AKMIN) to that two-dimensional as follows:
  As a preparatory step the following quatities should be defined.

$$
\begin{aligned}
&a_i = x_{i+1} - x_i \ , \ b_j = y_{j+1} - y_j \\
&c_{ij} = (f_{i+1,j} - f_{ij}) / a_i \\
&d_{ij} = (f_{i,j+1} - f_{ij}) / b_j \\
&e_{ij} = (c_{i,j+1} - c_{ij}) / b_j \\
&\quad = (d_{i+1,j} - d_{ij}) / a_i \qquad (4.2)
\end{aligned}
$$

For simplicity, consider a sequence of five points, x_1, x_2, x_3, x_4 and x_5 in the x-direction and y_1, y_2, y_3, y_4, and y_5 in the y-direction to obtain partial derivatives at the point (x_3, y_3). Fig. AKMID-2 illustrates the needed *c* 's, *d* 's and *e* 's.

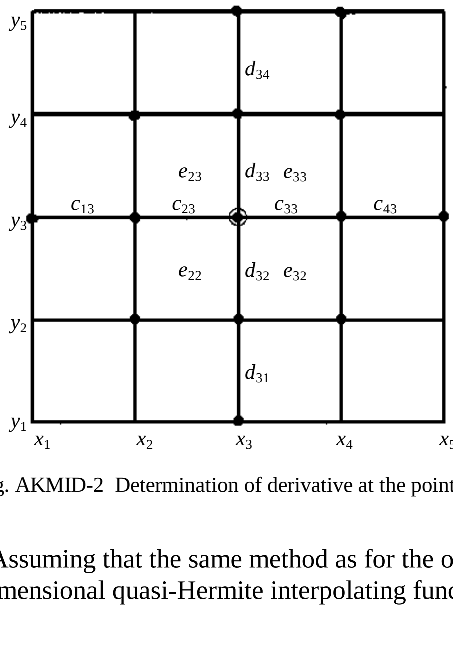


Fig. AKMID-2 Determination of derivative at the point (x_3, y_3)

Assuming that the same method as for the one-dimensional quasi-Hermite interpolating function is used, the first order partial derivatives both in x- and y-directions are as follows:

$$
\begin{aligned}
f_{33}^{(1,0)} &= (w_{x2}c_{23} + w_{x3}c_{33})/(w_{x2} + w_{x3}) \\
f_{33}^{(0,1)} &= (w_{y2}d_{32} + w_{y3}d_{33})/(w_{y2} + w_{y3})
\end{aligned} \qquad (4.3)
$$

where

$$
\begin{aligned}
w_{x2} &= |c_{43} - c_{33}|, w_{x3} = |c_{23} - c_{13}| \\
w_{y2} &= |d_{34} - d_{33}|, w_{y3} = |d_{32} - d_{31}|
\end{aligned}
$$

$f_{33}^{(1,0)}$ and $f_{33}^{(0,1)}$ are expressed as weighted means about c and d, respectively.
Based on the similar assumption, $f_{33}^{(1,1)}$ is determined as doubly weighted means of e in the directions of x and y as follows.

$$
\begin{aligned}
f_{33}^{(1,1)} = &\{w_{x2}(w_{y2}e_{22} + w_{y3}e_{23}) + w_{x3}(w_{y2}e_{32} + w_{y2}e_{33})\} \\
&/\{(w_{x2} + w_{x3})(w_{y2} + w_{y3})\}
\end{aligned} \qquad (4.4)
$$

- Determination of derivatives on the boundary
  Assuming that this is similar to the one-dimensional quasi-Hermite interpolating function, the partial derivatives, $f_{ij}^{(0,1)}$, $f_{ij}^{(1,0)}$ and $f_{ij}^{(1.1)}$; $i = 1, 2, n_x - 1, n_x$, $j = 1, 2, n_y - 1, n_y$, on the boundary are obtained by calculating c_{ij}, d_{ij} and e_{ij} outside the area and after that by applying Eqs. (4.3) and (4.4) Fig. AKMID-3 illustrates the situation for $i = j = 1$. The points marked by "o" are those given by assuming the same method as for the one-dimensional quasi-Hermite interpolating function.

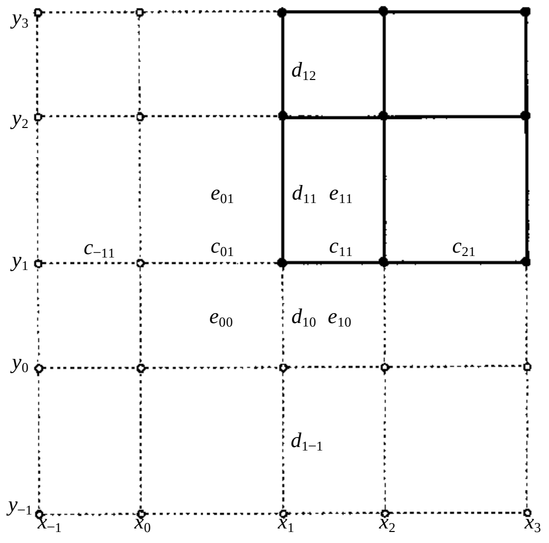


Fig. AKMID-3 Determination of derivatives on the boundary

The quatities, c_{ij} 's, d_{ij} 's and e_{ij} 's outside the area can be obtained as follows:

$$
\begin{aligned}
&c_{-11} = 3c_{11} - 2c_{21}\\
&c_{01} = 2c_{11} - c_{21}\\
&d_{1-1} = 3d_{11} - 2d_{12}\\
&d_{10} = 2d_{11} - d_{12}\\
&e_{01} = 2e_{11} - e_{21}\\
&e_{00} = 2e_{01} - e_{02} \quad (e_{02} = 2e_{12} - e_{22})\\
&e_{10} = 2e_{11} - e_{12}
\end{aligned} \tag{4.5}
$$

Thus, the partial derivatives at the point (x_1, y_1) can be obtained by applying Eqs. (4.3) and (4.4) after calculating the necessary *c*, *d* and *e*.

- Calculation of interpolated values
  The interpolated value at the point (v_x, v_y) can be obtained by evaluating $S_{ix,iy}(s, t)$ in Eq. (4.1) which is constructed by using the coordinate area numbers (i_x, i_y) to which the poit (x_x, v_y) belongs. The subroutine uses available partial derivatives, if any, obtained when called previously, and calculates the other needed derivatives, which are also stored in the work area and kept for later use.

For further details, see Reference [54].

E12-21-0201 AKMIN, DAKMIN

Quasi-Hermite interpolation coefficient calculation
CALL AKMIN (X, Y, N, C, D, E, ICON)

Function

Given function values $y_i = f(x)$, i = 1, ..., n for discrete points $x_1, x_2, ..., x_n$ ($x_1 < x_2 < ... < x_n$), this subroutine obtains the quasi-Hermite interpolating polynomial of degree 3, represented as (1.1) below. $n \geq 3$

$$S(x) = y_i + c_i(x - x_i) + d_i(x - x_i)^2 + e_i(x - x_i)^3 \quad (1.1)$$
$$x_i \leq x \leq x_{i+1}, i = 1, 2, ..., n - 1$$

Parameters

X	Input. Discrete points x_i. One-dimensional array of size n.
Y	Input. Function values y_i. One-dimensional array of size n.
N	Input. Number n of discrete points.
C	Output. Coefficient c_i, in (1.1). One-dimensional array of size n – 1.
D	Output. Coefficient d_i in (1.1). One-dimensional array of size n –1.
E	Output. Coefficient e_i in (1.1). One-dimensional array of size n – 1.
ICON	Output. Condition code. See Table AKMIN-1.

Table AKMIN-1

Code	Meaning	Processing
0	No error	
30000	$n < 3$ or $x_i \geq x_{i+1}$	Bypassed

Comments on use

- Subprograms used
  SSL II .... MGSSL
  FORTRAN basic function ... ABS

- Notes
  The interpolating function obtained by this subroutine is characterized by the absence of unnatural deviation, and thus produces curves close to those manually drawn. However, the derivatives of this function in interval (x_1, x_n) are continuous up to the first degree, but discontinuous above the second and higher degrees.
  If $f(x)$ is a quadratic polynomial and x_i, i = 1, ..., n are given at equal intervals, then the resultant interpolating function represents $f(x)$ itself, provided there are no calculation errors.
  If interpolation should be required outside the interval ($n < x_1$ or $x > x_n$), the polynomials corresponding to i = 1 or $i = n$ – 1 in (1.1) may be employed, though they does not yield good precision.

- Example
  A quasi-Hermite interpolating polynomial is determined by inputting the number n of discrete points, discrete points x_i and function values y_i, i = 1, ..., n, so that the interpolated value at a certain point $x = v$ in a certain interval [x_k, x_{k+1}] is determined. $n \leq 10$.

```
C      **EXAMPLE**
       DIMENSION X(10),Y(10),C(9),D(9),E(9)
       READ(5,500) N
       READ(5,510) (X(I),Y(I),I=1,N)
       CALL AKMIN(X,Y,N,C,D,E,ICON)
       WRITE(6,600) ICON
       IF(ICON.NE.0) STOP
       READ(5,500) K,V
       XX=V-X(K)
       YY=Y(K)+(C(K)+(D(K)+E(K)*XX)*XX)*XX
       N1=N-1
       WRITE(6,610)
       WRITE(6,620) (I,C(I),I,D(I),I,E(I)
      *             ,I=1,N1)
       WRITE(6,630) K,V,YY
       STOP
   500 FORMAT(I5,F10.0)
   510 FORMAT(2F10.0)
   600 FORMAT('0',10X,
      *'RESULTANT CONDITION',' CODE=',I5//)
   610 FORMAT('0',10X,
      *'RESULTANT COEFFICIENTS'//)
   620 FORMAT(' ',15X,'C(',I2,')=',E15.7,
      *5X,'D(',I2,')=',E15.7,
      *5X,'E(',I2,')=',E15.7)
   630 FORMAT('0',10X,2('*'),'RANGE',2('*'),
      *5X,2('*'),'DESIRED POINT',2('*'),
      *5X,2('*'),'INTERPOLATED VALUE',
      *2('*')//13X,I2,2X,2(8X,E15.7))
       END
```

Method

Given function values $y_i = f(x_i)$, i = 1, ..., n, for discrete points $x_1, x_2, ..., x_n$ ($x_1 < x_2 < .... < x_n$), let's consider the determination of the interpolating function represented in (1.1). (1.1) represents a different cubic polynomial for each interval [x_i, x_{i+1}]. $S(x)$ represented in (1.1) is piecewisely expressed as (4.1).

$$\left.\begin{aligned} S(x) &= S_i(x) \\ &= y_i + c_i(x - x_i) + d_i(x - x_i)^2 + e_i(x - x_i)^3 \\ & x_i \leq x \leq x_{i+1}, i = 1, ..., n - 1 \end{aligned}\right\} \quad (4.1)$$

Each $S_i(x)$ is determined by the following procedure:

(a) The first order derivatives at two points x_i and x_{i+1} are approximated. (They are taken as t_i and t_{i+1}).

(b) $S_i(x)$ is determined under the following four conditions:

$$\left.\begin{aligned} S_i'(x_i) &= t_i \\ S_i'(x_{i+1}) &= t_{i+1} \\ S_i(x_i) &= y_i \\ S_i(x_{i+1}) &= y_{i+1} \end{aligned}\right\} \qquad (4.2)$$

The interpolating function thus obtained is called a quasi-Hermite interpolating polynomial.
This subroutine features the geometric approximation method for first order derivatives in (a).
In that sense, the interpolating function is hereinafter called a "curve" and the first order derivative "the slope of the curve".

- Determination of the slope of a curve at each discrete point
  The slope of a curve at each discrete point is locally determined using five points; the discrete point itself and two on each side.
  Now let us determine the slope of the curve at point 3 from the five consecutive points 1 through 5. (See Fig. AKMIN-1)

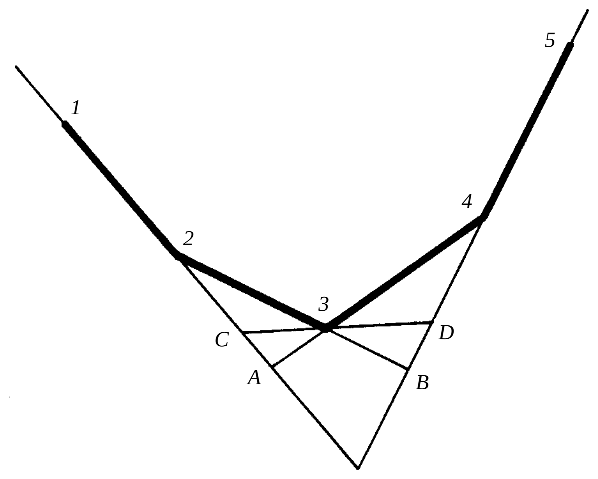


Fig. AKMIN-1

In Fig. AKMIN-1, the intersection of the extensions of segments 1 2 and 3 4 are taken as A and that the intersection of the extensions of segments 2 3 and 4 5 as B.
Also, the intersections of the tangent line at point 3 of the curve and segments 2 A and 4 B are taken as C and D, respectively.
The slopes of segments 1 2, 2 3, 3 4 and 4 5 are respectively designated as m_1, m_2, m_3, and m_4. If the slope of the curve at point 3 is t, t is determined so that it will approach m_2 when m_1 approaches m_2, or will approach m_3 when m_4 approaches m_3. One sufficient condition for satisfying this is given in (4.3.).

$$\left|\frac{\overline{2C}}{\overline{CA}}\right| = \left|\frac{\overline{4D}}{\overline{DB}}\right| \qquad (4.3)$$

(4.4) is derived from the relationship in (4.3)

$$\left.\begin{aligned} t &= \frac{w_2 m_2 + w_3 m_3}{w_2 + w_3} \\ w_2 &= \left|(m_3 - m_1)(m_4 - m_3)\right|^{\frac{1}{2}} \\ w_3 &= \left|(m_2 - m_1)(m_4 - m_2)\right|^{\frac{1}{2}} \end{aligned}\right\} \qquad (4.4)$$

t obtained from (4.4) has the following preferable characteristics:

$$\left.\begin{aligned} &\text{(a)When } m_1 = m_2, m_3 \neq m_4, m_2 \neq m_3, t = m_1 = m_2 \\ &\text{(b)When } m_3 = m_4, m_1 \neq m_2, m_4 \neq m_2, t = m_3 = m_4 \\ &\text{(c)When } m_2 = m_3, m_1 \neq m_4, m_3 \neq m_4, t = m_2 = m_3 \end{aligned}\right\} \qquad (4.5)$$

However, (4.4) has the following drawbacks:

(d) when $w_2 = w_3 = 0$, t is undifinite
(e) when $m_2 = m_4, m_3 \neq m_1, m_4 \neq m_3, t = m_2$
or (4.6)
when $m_1 = m_3, m_2 \neq m_4, m_3 \neq m_2, t = m_3$

This subroutine determines t based on (4.7), not on (4.4), to avoid these drawbacks.

when $m_1 \neq m_2$ or $m_3 \neq m_4$

$$t = \frac{|m_4 - m_3|m_2 + |m_2 - m_1|m_3}{|m_4 - m_3| + |m_2 - m_1|} \qquad (4.7)$$

when $m_1 = m_2$ and $m_3 = m_4$

(4.7) satisfies the requirements of the characteristics (4.5)

- Determination of slope at both end points
  At points located at both ends (x_1, y_1), (x_2, y_2), (x_{n-1}, y_{n-1}), and (x_n, y_n), the following virtual discrete points are adopted to determine their slope.
  Take the left end for example, the five points shown in Fig. AKMIN-2 are set so that the slope t_1 of the curve at (x_1, y_1) can be determined.

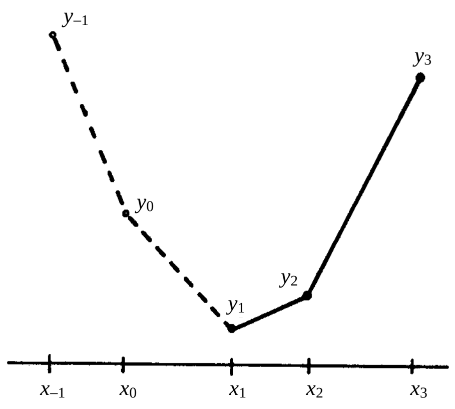


Fig. AKMIN-2

The two points (x_{-1}, y_{-1}) and (x_0, y_0) are virtual points. x_{-1} and x_0 are determined from (4.8).

$$x_3 - x_1 = x_2 - x_0 = x_1 - x_{-1} \tag{4.8}$$

y_{-1} and y_0 are assumed to be the values obtained by evaluating a quadratic polynomial passing (x_1, x_1), (x_2, y_2) and (x_3, y_3) at x_{-1} and x_0. The five points satisfy the conditions of (4.9).

$$\left.\begin{aligned}&\frac{y_3-y_2}{x_3-x_2}-\frac{y_2-y_1}{x_2-x_1}=\frac{y_2-y_1}{x_2-x_1}-\frac{y_1-y_0}{x_1-x_0}\\&=\frac{y_1-y_0}{x_1-x_0}-\frac{y_0-y_{-1}}{x_0-x_{-1}}\end{aligned}\right\} \tag{4.9}$$

If the slope of segments (x_{-1}, y_{-1}) and (x_0,y_0) is m_1 and the slopes of the segments extending to the right are m_2, m_3, and m_4, respectively, the following equations are obtained from (4.9):

$$m_2 = 2\,m_3 - m_4,\ m_1 = 3\,m_3 - 2\,m_4 \tag{4.10}$$

The slope t_1 at (x_1, y_1) is determined by applying these m_1, m_2, m_3 and m_4 to the method in a. In the determination of t_2 at point (x_2, y_2), the five points (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, x_4) are used.

The slope t_{n-1}, t_n at right-end points (x_{n-1}, y_{n-1}) and (x_n, y_n) are similarly determined by assuming (x_{n+1}, y_{n+1}) and (x_{n+2}, y_{n+2}).

- Determination of curves

  The coefficient of $S_i(x)$ in (4.1), as determined by the conditions of (4.2), is represented in (4.11).

$$\left.\begin{aligned}&c_i = t_i\\&d_i = \{3(y_{i+1}-y_i)/(x_{i+1}-x_i)-2t_i-t_{i+1}\}\\&\quad /(x_{i+1}-x_i)\\&e_i = \{t_i+t_{i+1}-2(y_{i+1}-y_i)/(x_{i+1}-x_i)\}\\&\quad /(x_{i+1}-x_i)^2\end{aligned}\right\} \tag{4.11}$$

For further information, see Reference [52].

A22-11-0202 ALU, DALU

LU-decomposition of a real general matrix (Crout's method)
CALL ALU (A, K, N, EPSZ, IP, IS, VW, ICON)

Function

An $n \times n$ nonsingular real matrix $\boldsymbol{A}$ is LU-decomposed using the Crout's method.

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{1.1}$$

$\boldsymbol{P}$ is the permutation matrix which performs the row exchanges required in partial pivoting, $\boldsymbol{L}$ is a lower triangular matrix, and $\boldsymbol{U}$ is a unit upper triangular matrix. $n \geq 1$.

Parameters

A .... Input. Matrix $\boldsymbol{A}$
Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.
Refer to Fig. ALU-1,
A is a two-dimensional array, A (K, N).

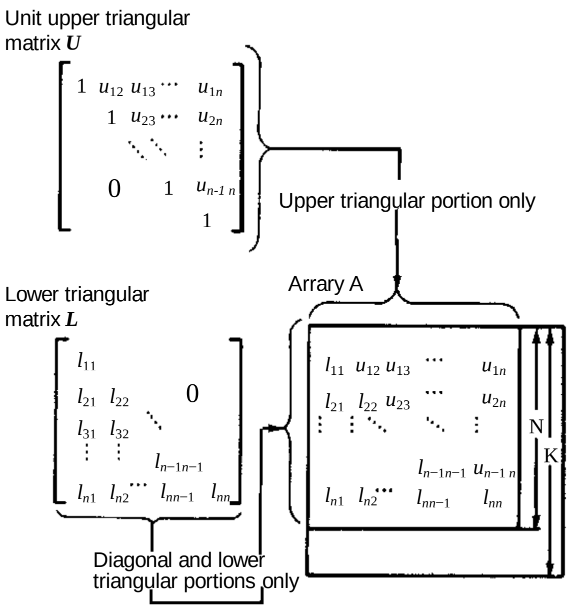

Unit upper triangular matrix U
Upper triangular portion only
Arrary A
Lower triangular matrix L
Diagonal and lower triangular portions only
N
K

Fig. ALU-1 Storage of the elements of L and U in array A

K .... Input. Adjustable dimension of array A (≥N)
N .... Input. Order n of matrix $\boldsymbol{A}$
EPSZ .. Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ (≥ 0.0) When EPSZ is 0.0, a standard value is used. (Refer to Notes.)
IP .... Output. The transposition vector which indicates the history of row exchanging that occurred in partial pivoting.
IP is a one-dimensional array of size n. (Refer to Notes.)
IS ... Output. Information for obtaining the determinant of matrix $\boldsymbol{A}$. If the n elements of the calculated diagonal of array A are multiplied by IS, the determinant is obtained.
VW .... Work area. VW is one-dimensional array of size n.
ICON .... Output. Condition code. Refer to Table ALU-1.

Table ALU-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Either all of the elements of some row were zero or the pivot became relatively zero. It is highly probable that the matrix is singular.	Discontinued
30000	K<N, N<1 or EPSZ<0.0	Bypassed

Comments on use

- Subprograms used
  SSL II .... AMACH, MGSSL
  FORTRAN basic functions ... ABS

- Notes
  If EPSZ is set to 10^{-s}, this value has the following meaning. In LU-decomposition, if the loss of over s significant digits occurred for the pivot, the LU-decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero. Let u be the unit round-off, and the standard value of EPSZ is 16 u. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value, but the result is not always guaranteed.
  The transposition vector corresponds to the permutation matrix $\boldsymbol{P}$ of LU decomposition in partial pivoting. In this subroutine, the elements of the array A are actually exchanged in partial pivoting. In the J th stage (J = 1, ..., n) of decomposition, if the Ith row (I ≥ J) has been selected as the pivotal row the elements of the Ith row and the elements of the Jth row are exchanged. Then, in order to record the history of this exchange, I is stored in IP (J).
  A system of linear equations can be solved by calling subroutine LUX following this subroutine. However, instead of these subroutines, subroutine LAX can be normally called to solve such equations in one step.

- Example
  An $n \times n$ matrix is input and LU-decomposition is computed. $n \leq 100$.

```
C      **EXAMPLE**
       DIMENSION A(100,100),VW(100),IP(100)
    10 READ(5,500) N
       IF(N.EQ.0) STOP
       READ(5,510) ((A(I,J),I=1,N),J=1,N)
       WRITE(6,600) N,((I,J,A(I,J),J=1,N),
      *             I=1,N)
       CALL ALU(A,100,N,0.0,IP,IS,VW,ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) GOTO 10
       DET=IS
       DO 20 I=1,N
       DET=DET*A(I,I)
    20 CONTINUE
       WRITE(6,620) (I,IP(I),I=1,N)
       WRITE(6,630) ((I,J,A(I,J),J=1,N),
      *             I=1,N)
       WRITE(6,640) DET
       GOTO 10
   500 FORMAT(I5)
   510 FORMAT(4E15.7)
   600 FORMAT(///10X,'** INPUT MATRIX **'
      * /12X,'ORDER=',I5//(10X,4('(',I3,',',
      * I3,')',E16.8)))
   610 FORMAT('0',10X,'CONDITION CODE =',I5)
   620 FORMAT('0',10X,'TRANSPOSITION VECTOR'
      * /(10X,10('(',I3,')',I5)))
   630 FORMAT('0',10X,'OUTPUT MATRICES'
      * /(10X,4('(',I3,',',I3,')',E16.8)))
   640 FORMAT('0',10X,
      * 'DETERMINANT OF THE MATRIX =',E16.8)
       END
```

Method

- Crout’s method

  Generally, in exchanging rows using partial pivoting, an $n \times n$ regular real matrix $\boldsymbol{A}$ can be decomposed into the product of a lower triangular matrix $\boldsymbol{L}$ and a unit upper triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{4.1}$$

$\boldsymbol{P}$ is the permutation matrix which performs the row exchanging required in partial pivoting. The Crout’s method is one method to obtain the elements of $\boldsymbol{L}$ and $\boldsymbol{U}$. This subroutine obtains values in the jth column of $\boldsymbol{L}$ and jth column of $\boldsymbol{U}$ in the order (j = 1, ..., n) using the following equations.

$$u_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right) \Big/ l_{ii}, i = 1,..., j-1 \tag{4.2}$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}, i = j,...,n \tag{4.3}$$

where, $\boldsymbol{A} = (a_{ij})$, $\boldsymbol{L} = (l_{ij})$ and $\boldsymbol{U} = (u_{ij})$. Actually using partial pivoting, rows are exchanged.

The Crount’s method is a variation of the Gaussian elimination method.
Both perform the same calculation, but the calculation sequence is different. With the Crout’s method, elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ are calculated at the same time using equations (4.2) and (4.3). By increasing the precision of the inner products in this step, the effects of rounding errors are minimized.

- Partial pivoting

  When matrix $\boldsymbol{A}$ is given as

$$\boldsymbol{A} = \begin{bmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \end{bmatrix}$$

Though the matrix is numerically stable, it can not be LU decomposed. In this state, even if a matrix is numerically stable large errors would occur if LU decomposition were directly computed. So in this subroutine, to avoid such errors partial pivoting with row equilibration is adopted for decomposition.

For more information, see References [1],[3], and [4].

G23-11-0301 AQC8, DAQC8

Integration of a function by a modified Clenshaw-Curtis rule
CALL AQC8 (A, B, FUN, EPSA, EPSR, NMIN, NMAX, S, ERR, N, ICON)

Function

Given a function $f(x)$ and constants a, b, ε_a and ε_r this subroutine obtains an approximation S which satisfies

$$\left| S - \int_a^b f(x)dx \right| \le \max\left(\varepsilon_a , \varepsilon_r \cdot \left| \int_a^b f(x)dx \right| \right) \qquad (1.1)$$

by a modified Clenshaw-Curtis rule which increases a fixed number of abscissas at a time.

Parameters

A	Input. Lower limit a of the interval.
B	Input. Upper limit b of the interval.
FUN ..	Input. The name of the function subprogram which evaluates the integrand $f(x)$ (see the example).
EPSA ..	Input. The absolute error tolerance ε_a (≥ 0.0) for the integral.
EPSR ..	Input. The relative error tolerance ε_r (≥ 0.0) for the integral.
NMIN ..	Input. Lower limit on the number of function evaluation (≥ 0). A proper value is 15.
NMAX ..	Input. Upper limit on the number of function evaluations (NMAX $\ge$ NMIN) A proper value is 511. (A higher value, if specified, is interpreted as 511.) (See "Comments on use".)
S ...	Output. An approximation (see "Comments on use").
ERR	Output. An estimate of the absolute error in the approximation.
N	Output. The number of function evaluations actually performed.
ICON	Output. Condition code. See Table AQC8-1.

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH
  FORTRAN basic functions ... ABS, AMAX1, FLOAT, MAX0, MIN0, SQRT

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable.
  Its function name must be declared as EXTERNAL in the calling program. If the integrand includes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with the main program. (See the example.)

Table AQC8-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The desired accuracy was not attained due to rounding-off errors.	Approximation obtained so far is output in S. The accuracy is the maximum attainable.
20000	The desired accuracy was not attained though the number of integrand evaluations has reached the upper limit.	Processing stops. S is the approximation obtained so far, but is not accurate.
30000	One of the followings occurred. 1 EPSA < 0.0 2 EPSR < 0.0 3 NMIN < 0 4 NMAX < NMIN	Processing stops.

When this subroutine is called many times, 511 constants (Table of abscissas, weights for the integration formula) are determined only on the first call, and this computation is bypassed on subsequent calls. Thus, the computation time is shortened.

This subroutine works most successfully when the integrand $f(x)$ is a oscillatory type function. For a smooth function, it is best in that it requires less evaluations of $f(x)$ than subroutines AQN9 and AQE.

For a function which contains singularity points, subroutine AQE is suitable if the singularity points are only on the end point of the integration interval and subroutine AQN9 for a function whose singularity points are between end points, or for a peak type function.

Parameters NMIN and NMAX must be specified considering that this subroutine limits the number of evaluations of integrand $f(x)$ as NMIN $\le$ Number of evaluation times $\le$ NMAX

This means that $f(x)$ is evaluated at least NMIN times and not more than NMAX times regardless of the result of the convergence test. When a value of S that satisfies the expression (1.1) within NMAX evaluations cannot be obtained, processing stops with ICON code 20000. If the value of NMAX is less than 15, a default of 15 is used.

Accuracy of the approximation
S is obtained as follows. This subroutine obtains S to satisfy the expression (1.1) when constants ε_a and ε_r are given. Thus ε_r=0 means to obtain the approximation with its absolute error within ε_a, Similarly, ε_a=0 means to obtain it with its relative error within ε_r.

This purpose is sometimes obstructed by unexpected characteristics of the function or unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small in comparison with arithmetic precision in function evaluation, the effect of round-off error becomes greater, so it is no use to continue the computation, even though the number of integrand evaluation has not reached the upper limit. In this case, processing stops with the code 10000 in ICON. At this time, the accuracy of S becomes the attainable limit for computer used. The approximation sometimes does not converge within NMAX evaluations. In this case, S is an approximation obtained so far, and is not accurate and indicated by ICON code 20000.

To determine the accuracy of integration, this subroutine always puts out an estimate of absolute error in parameter ERR, as well as the approximation S.

- Example
  Increasing the value of auxiliary variable p from 0.1 to 0.9 with increment 0.1, this example computes the integral

```
C     **EXAMPLE**
      COMMON P
      EXTERNAL FUN
      A=-1.0
      B=1.0
      EPSA=1.0E-5
      EPSR=1.0E-5
      NMIN=15
      NMAX=511
      DO 10 I=1,10
      P=FLOAT(I)
      CALL AQC8(A,B,FUN,EPSA,EPSR,NMIN,
     *     NMAX,S,ERR,N,ICON)
   10 WRITE(6,600) P,ICON,S,ERR,N
      STOP
  600 FORMAT(' ',30X,'P=',F6.1,5X,
     *'ICON=',I5,5X,'S=',E15.7,5X,
     *'ERR=',E15.7,5X,'N=',I5)
      END
      FUNCTION FUN(X)
      COMMON P
      FUN=COS(P*X)
      RETURN
      END
```

Method

This subroutine uses an extended Clenshaw-Curtis integration method which increases a fixed number of abscissas (8 points) at a time. The original Clenshaw-Curtis rule sometimes wastes abscissas because it increases them doubly even when the desired accuracy could be attained by adding only a few abscissas.

For the purpose of avoiding this as much as possible, this subroutine increases 8 points at a time. Moreover, the costs of computations is reduced by using the Fast Fourier Transform algorithm (FFT).

- Clenshaw-Curtis integration which increases a fixed number of points at a time
  The given integral $\int_a^b f(x)\,dx$ may be transformed by linear transformation:

$$x = \frac{b-a}{2}t + \frac{a+b}{2}$$

to

$$\frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)dt$$

For simplicity, let's consider the integration(4.1) over the interval [-1,1] in what follows.

$$I = \int_{-1}^{1} f(x)dx \qquad (4.1)$$

The original Clenshaw-Curtis rule is as follows. By an interpolation polynomial (i.e., Chebyshev interpolation polynomial), whose interpolating points are the series of points (Fig. AQC8-1) made by projecting the series of points equally-sectioned on the unit half circle over the interval [-1,1], $f(x)$ is approximated, and this is integrated term by term to obtain the integral approximation. The number of data points will increase doubly to meet the required accuracy. This method sometimes wasted data points.

The method which increases a fixed number of data points at a time is explained next. First, based on Van der Corput series uniformly distributed on (0,1), the series of points $\{\alpha_j\}$(j=1,2,3,...) are made by the recurrence relation

$$\alpha_1 = 1/4, \alpha_{2j} = \alpha_j/2, \alpha_{2j+1} = \alpha_{2j} + 1/2 \qquad (j = 1,2,3,...)$$

The series of points on a unit circle $\{\exp(2\pi i\alpha_j)\}$ is symmetric to the origin and unsymmetric to the real axis (Fig. AQC8-2). Since those points $\{x_j=\cos 2\pi\alpha_j\}$ for j=1,2,3,... form the Chebyshev Distribution on (-1,1), they are used as data points (Fig. AQC8-2).

As shown in Fig. AQC8-2, seven points are used first as abscissas. After that, eight points are added at a time repeatedly. When the total amount of abscissas reaches 2^n-1, their locations match those of points which are made by projecting biseetional points placed on the unit half circle to the open interval(-1,1). Thus, they are regarded as a series of data points as used by the Clenshaw-Curtis rule on the open interval.

The descriptions for forming the interpolation

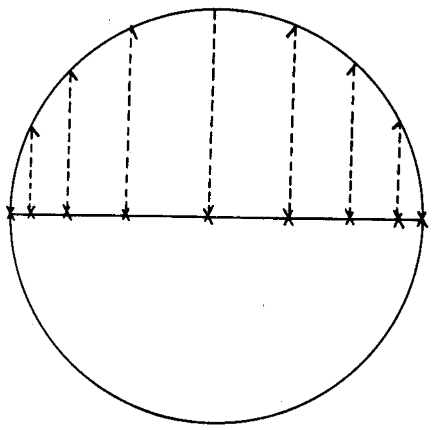


Fig. AQC8-1 Data points used by the original Clenshaw-Curtis rule

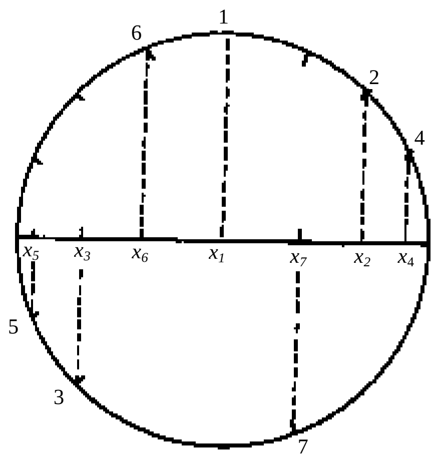


Fig. AQC8-2 Series of data points $\{x_j\}$ used by this subroutine

polynomials are given next. From the characteristics of α_j, the sequence $\{x_j\}(j=1,2,...,7)$ matches $\{\cos \pi_j / 8\}$. When N-th degree Chebyshev polynomial is expressed as $T_N(x)$ $x_{8k+j}(j=0, 1, ..., 7)$ corresponds to eight roots of $T_8(x)-x_k=0$ ($k=1,2,\ldots$). Using these characteristics, seven points at firrst, eight points at each subsequent time, are added to make the series of interpolation polynomial $P_l(x)$. In the expressions below, $P_0(x)$ is the interpolation which uses the first 7 points, and $P_l(x)$ is the one which uses $8l+7$ points as the result of adding 8 points l times.

With a change of variable $x=\cos\theta$, the expression

$$I=\int_{-1}^{1} f(x)dx=\int_{0}^{\pi} f(\cos\theta)\sin\theta \quad d\theta$$

is derived, and $f(\cos\theta)$ is approximated by $P_l(\cos\theta)$, i.e.

$$f(\cos\theta)\approx P_l(\cos\theta)$$

Where,

$$P_0(\cos\theta)=\sum_{k=1}^{7} A_{-1,k}\sin k\theta / \sin\theta$$

$$P_{l+1}(\cos\theta)=P_l(\cos\theta)+\frac{\sin 8\theta}{\sin\theta}\omega_l(\cos 8\theta)\sum_{k=0}^{7}{}' A_{l,k}\cos k\theta$$

$$(l=0,1,2,\ldots) \qquad (4.2)$$

subject to

$$\omega_0(\cos 8\theta)=1$$

$$\omega_l(\cos 8\theta)=2^l\prod_{k=1}^{l}(\cos 8\theta-\cos 2\pi\alpha_k)$$

$$=2^l\prod_{k=1}^{l}(T_8(x)-x_k),(l\geq 1)$$

The notation Σ' on the right-hand side of (4.2) means to sum up the subsequent terms with the first term multiplied by 1/2. Coefficients. $A_{-1,k}$ and $A_{i,k}$ of polynomial $P_l(x)$ are determined by the interpolating conditions. First, using the first seven points of $\{\cos\pi_j / 8\}(j=1,2,...,7)$, $A_{-1,k}(k=1,2,...,7)$ is given as

$$A_{-1,k}=\frac{2}{8}\sum_{j=1}^{7}\left\{f\left(\cos\frac{\pi}{8}j\right)\sin\frac{\pi}{8}j\right\}\sin\frac{\pi}{8}kj,$$

$$(k=1,2,\ldots,7)$$

Next, using $A_{i,k}$ ($-1\leq i\leq l-1$) which is known, $A_{l,k}$ ($l\geq 0$) which appears in $P_{l+1}(\cos\theta)$ are obtained. At this stage, added data points are roots of $T_8(x)-x_{l+1}=0$, that is, $\cos\theta_j^{l+1}$, $(\theta_j^{l+1}=2\pi/8\cdot(j+\alpha_{l+1})$ $(j=0,1,2,...,7)$. So, the interpolating conditions at this stage,

$$f(\cos\theta_j^{l+1})\sin\theta_j^{l+1}=\sum_{k=1}^{7}A_{-1,k}\sin k\theta_j^{l+1}$$

$$+\sin 2\pi\quad\alpha_{l+1}\sum_{i=0}^{l}\omega_i(\cos 2\pi\quad\alpha_{l+1})\sum_{k=0}^{7}{}' A_{i,k}\cos k\theta_j^{l+1}$$

$$(0\leq j\leq 7)$$

are used to determine $A_{l,k}$. To the left-hand side of (4.3) a cosine transformation including parameter α_{l+1} is applied.

$$f(\cos\theta_j^{l+1})\sin\theta_j^{l+1}=\sum_{k=0}^{7}a_k\cos k\theta_j^{l+1}\quad,\ (0\leq j\leq 7)$$

By regarding this as a system of linear equations with a_k's being unknowns and by solving them we have

$$a_k = \frac{2}{8}\sum_{j=0}^{7} f\left(\cos\theta_j^{l+1}\right)\sin\theta_j^{l+1}\cos k\theta_j^{l+1}$$

In actual computations real FFT algorithm is used. Then, from (4.3),

$$a_k = \frac{1}{\sin 2\pi \quad \alpha_{l+1}}\left(A_{-1,8-k} - \cos 2\pi \quad \alpha_{l+1}\cdot A_{-1,k}\right)$$
$$+\sin 2\pi \quad \alpha_{l+1}\sum_{i=0}^{l}\omega_i\left(\cos 2\pi \quad \alpha_{l+1}\right)A_{i,k},$$
$$(0 \le k \le 7)$$

where, $A_{-1,0}=A_{-1,8}=0$.

From $\omega_0(\cos 2\pi\alpha_1)=1$ when $l=0$, the value $A_{0,k}$ is obtained easily. For $l \ge 1$, letting

$l=m+2^n \qquad (0 \le m < 2^n)$

and using the relation

$\sin 2\pi\alpha_{l+1}\cdot\omega_{2^n-1}(\cos 2\pi\alpha_{l+1})=\sin(2^{n+1}\pi\alpha_{l+1})=1$

$A_{l,k}$ can be computed as shown below.

Letting first:

$$B_{m+1} = a_k - \left(A_{-1,8-k} - \cos 2\pi\alpha_{l+1}\cdot A_{-1,k}\right)/\sin 2\pi\alpha_{l+1}$$
$$-\sin 2\pi\alpha_{l+1}\sum_{i=0}^{2^n-2}\omega_i\left(\cos 2\pi\alpha_{l+1}\right)A_{i,k} \tag{4.4}$$

then computing each B_{m-i} - 1 sequentially by

$$B_{m-i} = (B_{m+1-i} - A_{2^n+i-1}) / (\cos 2\pi\alpha_{l+1} - \cos 2\pi\alpha_{2^n+i}) \qquad i=0,1,2,...,m \tag{4.5}$$

And $A_{l,k}$'s are computed as

$A_{l,k}=B_0$

Comparing this to that of Newton difference quotient formula, calculation in (4.4) and (4.5) are more stable because the number of divisions are reduced from l+2 to m+2. Using interpolation polynominal $P_{l+1}(\cos\theta)$ computed so far, the integral approximation I_{l+1} is obtained by termwise integration.

$$I_{l+1} = \int_0^{\pi} P_{l+1}(\cos\theta)\sin\theta\, d\theta$$
$$= \sum_{k=1}^{7} A_{-1,k}\frac{2}{k} + \sum_{i=1}^{l}\sum_{k=0}^{7}{}' A_{i,k}W_{i,k}$$

(Only terms with odd value of k are summed.)

where, the weight coefficient $W_{i,k}$ is defined as

$$W_{i,k} = \int_0^{\pi}\omega_i\left(\cos 8\theta\right)\sin 8\theta\cos k\theta\, d\theta \tag{4.6}$$

and computed as follows. The weight coefficient $W_{2^n-1,k}$ is obtained by:

$$W_{2^n-1,k} = \int_0^{\pi}\sin 2^{n+3}\theta\cos k\theta \quad d\theta = \frac{2^{n+4}}{4^{n+3}-k^2}$$

Using this $W_{2^n-1,k}$ as a starting value, the required $N(=2^m\times 4)$ members of $W_{i,k}(0 \le i \le 2^m-1, k=1,3,5,7)$ are computed by the following recurrence formula:

$$W_{2^n-1+2^{n-j+1}\cdot s+2^{n-j},k}=W_{2^n-1+2^{n-j+1}\cdot s,\, 2^{n-j}\cdot 8+k}$$
$$+W_{2^n-1+2^{n-j+1}\cdot s,2^{n-j}\cdot 8-k}$$
$$-2\cos 2\pi\alpha^{j}_{2+2s}\cdot W_{2^n-1+2^{n-j+1}\cdot s,k}$$
$$, n=1,2,...,m-1 \ , \ 1 \le j \le n, \ 0 \le s < 2^{j-1}-1,$$
$$0 \le k \le 2^{n-j}\cdot 8 - 1 (k \text{ is an odd number}). \tag{4.7}$$

To obtain these weight coefficients, $N/2(\log_2 N - 2)+4$ multiplications and divisions are required.

- Computing procedures

Procedure 1 ... Initialization

This procedure computes $m=(a+b)/2$, $r=(b-a)/2$, which are required to transform the integration interval $[a,b]$ to $[-1,1]$. All the initializations required are performed here.

Procedure 2 ... Determination of abscissas and weights

In this subroutine the number of abscissas is limited to 511 ($=2^9-1$). Since points $\{x_j\}=\{\cos 2\pi\alpha_j\ (j=1,2,...,511)$ are distributed symmetrically to the origin, a table of only 256($=2^8$)$\{\cos\pi\alpha_j\}(j=1,2,...,256)$ is needed. The table is generated by the recurrence formula. Also the weights $\{W_{i,2k+1}\}(0 \le i \le 63, 0 \le k \le 3)$ are computed by the recurrence formula (4.7) and stored in a vector. This procedure is performed only on the first call to the subroutine, but bypassed on the subsequent calls.

Procedure 3 ... Integral approximation based on initial 7 points Integral approximation I_0 is obtained Integral approximation I_0 is obtained by multiplying the weights determined in procedure 2 and $A_{-1,2k+1}(0 \le k \le 3)$, which are obtained by using real FFT algorithm to data points $\cos\pi j/8(j=1,2,...,7)$.

Procedure 4 ... Trigonometric function evaluations

Using data point table $\{\cos\pi\alpha_j\}$, values of trigonometric functions $\cos 2\pi\alpha_{l+1}$, $\sin 2\pi\alpha_{l+1}$, $\sin\pi\alpha_{l+1}$ to be required in procedure 5 are evaluated.

Procedure 5 ... a_k and $A_{l,k}$
After evaluating function values at the added 8 data points, $a_{2k+1}(0 \le k \le 3)$ are obtained by using real FFT to 8 terms. And $A_{l,2k+1}(0 \le k \le 3)$ are obtained based on (4.4) - (4.6).

Procedure 6 ... Integration and convergence test
Previous integral approximation $I_l(l \ge 0)$ is added with $\sum_{k=0}^{3} A_{l,2k+1}W_{l,2k+1}$ to obtain the updated integral approximation I_{l+1}. Next, an estimate e_{l+1} of truncation error in I_{l+1} is computed, and the convergence test is done to e_{l+1} as well as to e_l.
After the convergence test for e_l the computation stops if both tests are successful, otherwise goes back to procedure 4 with l increased by 1.

- Error estimation
  Letting $R_l(x)$ denote the error when $f(x)$ is approximated by an interpolation polynomial $P_l(x)$ mentioned above it can be seen that

$$\begin{aligned} f(x) &= P_l(x) + R_l(x) \\ &= \sum_{k=1}^{7} A_{-1,k}U_{k-1}(x) + U_7(x)\sum_{i=0}^{l-1}\omega_i(T_8(x)) \\ &\times \sum_{k=0}^{7}{}' A_{i,k}T_k(x) + U_7(x)\omega_l(T_8(x)) \\ &\times 2^{-(8l+7)} f[x, x_1, x_2, \ldots, x_{8l+7}] \end{aligned}$$

The coefficient $2^{-(8l+7)}$ is used in order to match the conventional error term expression with the divided difference.
$U_k(x)$ is the k-th degree Chebyshev polynomial of the second kind defined as follows:

$U_k(x)=\sin(k+1)\theta / \sin\theta$, $x=\cos\theta$

Truncation error E_l for the approximation I_l is expressed as

$$\begin{aligned} E_i &= \int_{-1}^{1} R_l(x)dx = \int_{-1}^{1} U_7(x)\omega_l(T_8(x))2^{-(8l+7)} \\ &\quad f[x, x_1, \ldots, x_{8l+7}]dx \end{aligned}$$

The divided difference can be expressed by the form of integration and also expanded to a Chebyshev series as follows:

$$\begin{aligned} 2^{-(8l+7)} f[x, x_1, \ldots, x_{8l+7}] &= \frac{1}{2\pi i}\oint_C \frac{f(z)dz}{(z-x)U_7(z)\omega_l(T_8(z))} \\ &= \sum_{k=0}^{\infty}{}' C_{l,k}T_k(x) \end{aligned}$$

Figure AQC8-3 shows the integration path c, which is a simple closed curve.

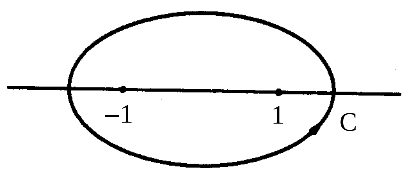


Fig. AQC8-3

Coefficient $C_{l,k}$ is as follows:

$$C_{l,k} = \frac{1}{2\pi i}\frac{2}{\pi}\oint_C \frac{U_k^*(z) f(z)dz}{U_7(z)\omega_l(T_8(z))}$$

$U_k^*(z)$ is the Chebyshev function of the second kind defined as follows:

$$U_k^*(z) = \int_{-1}^{1} \frac{T_k(x)dx}{(z-x)\sqrt{1-x^2}} = \frac{\pi}{\left(z+\sqrt{z^2-1}\right)^k \sqrt{z^2-1}}$$

Therefore, E_l will be obtained by

$$E_l = \sum_{k=0}^{\infty}{}' C_{l,k}W_{l,k} \qquad (k = \text{odd number})$$

If $f(z)$ is a rational function having a poles at point z_m where m=1,2,...M), the following is established:

$$C_{l,k} = -\frac{2}{\pi}\sum_{m=1}^{M}\frac{U_k^*(z_m)\mathrm{Res}f(z_m)}{U_7(z_m)\omega_l(T_8(z_m))}$$

Here, Res$f(z_m)$ is a residue at a point z_m.
Under the following conditions,

$$U_k^*(z_m) \propto r_m^{-k}, r_m = \left|z_m + \sqrt{z_m^2 - 1}\right| > 1$$

as far as z_m is not too close to the interval [– 1,1] on a real axis,

$$|C_{l,1}| > |C_{l,k}| \quad , \ (k \ge 3)$$

holds.
The truncation error can be estimated by

$$|E_l| \approx |C_{l,1}| \cdot |W_{l,1}| \le \left(|A_{l-1,7}| + |A_{l-1,5}|\right) \cdot |W_{l,1}| \equiv e_l$$

$|A_{l-1,7}|$ and $|A_{l-1,5}|$ are used instead of $|C_{l,1}|$, the value of which cannot be really evaluated.
If z_m is very close to [-1,1] or $f^{(p)}(x)$, derivative of order p(where p≥1) becomes discontinuous on [– 1,1], the error estimation above is no longer valid. To cope with this situation, take the following procedures. If $A_{i,k}$ decreases rapidly, the error for l_{2^n-1} can be estimated well by

$$\left|I_{2^n-1} - I_{2^{n-1}-1}\right|$$

Therefore, if the following holds,

$$e_{2^n-1} > \left| I_{2^n-1} - I_{2^{n-1}-1} \right|$$

e_l is used as an error estimation in 2^n-1 ≤ l ≤ 2^{n+1} – 1; otherwise the following is used instead.

$$e'_l \equiv e_l \left| I_{2^n-1} - I_{2^{n-1}-1} \right| / e_{2^{n-1}-1}$$

- Convergence criterion
  As well as a truncation error, the integral approximation has a computation error. This subroutine estimates the upper bound ρ of the computation error as

$$\rho = u(l+1)\|f\|_\infty$$

where u is the round-off unit, and $\|f\|_\infty = \max_j \left| f(x_j) \right|$.

This assumption is reasonable in actual use because abscissas form the Chebysev distribution and FFT is used.

Setting a tolerance for convergence test as

$$\tau = \max\left(\varepsilon_a, \varepsilon_r \left| \int_{-1}^{1} f(x)dx \right|, \rho \right)$$

If the following condition is satisfied,

e_{l+1}(or e'_{l+1}) < τ

I_{l+1} is output in parameter ***S*** as an approximation to the integral.

In parameter ERR, e_{l+1}(ICON = 0) is put out if $e_{l+1} \geq \rho$, or ρ(ICON=10000) if $e_{l+1} < \rho$.

$|I_l|$ substitutes for $\left| \int_{-1}^{1} f(x)dx \right|$ which is used in τ.

For the detailed description, see References [65] and [66].

G23-11-0401 AQE,DAQE

Integration of a function by double exponential formula
CALL AQE (A,B,FUN,EPSA,EPSR,NMIN,NMAX,S, ERR,N,ICON)

Function

Given a function $f(x)$ and constants $a,b,\varepsilon_a,\varepsilon_r$ this subroutine obtains an approximation S that satisfies

$$\left|S - \int_a^b f(x)dx\right| \le \max\left(\varepsilon_a, \varepsilon_r \cdot \left|\int_a^b f(x)dx\right|\right) \qquad (1.1)$$

by Takahashi-Mori's double exponential formula.

Parameters

A	Input. Lower limit a of the interval.
B	Input. Upper limit b of the interval
FUN	Input. The name of the function subprogram which evaluates the integrand $f(x)$ (see the example).
EPSA ..	Input. The absolute error tolerance $\varepsilon_a(\ge 0.0)$ for the integral.
EPSR ..	Input. The relative error tolerance $\varepsilon_r(\ge 0.0)$ for the integral.
NMIN ..	Input. Lower limit on the number of function evaluations. A proper value is 20.
NMAX ..	Input. Upper limit on the number of function evaluations. (NMAX≥NMIN) A proper value is 641 (a higher value, if specified, is interpreted as 641).
S	Output. An approximation to the integral. (See "Comments on use" and "Notes".)
ERR ..	An estimate of the absolute error in the approximation.
N	Output. The number of function evaluations actually performed.
ICON ..	Output. Condition code. See Table AQE-1.

Table AQE-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The desired accuracy is not attained due to rounding-off errors.	Approximation obtained so far is output in S. The accuracy has reached the attainable limit.
11000 12000 13000	1,2,3 at the place of 1000 mean that the function value increases steeply near the upper, lower, or both limits of the interval respectively	Processing continues with a relaxed tolerance.
20000	The desired accuracy is not attained though the number of integrand evaluation has reached the upper limit	Processing stops. S is the approximation obtained so far, but is not accurate.
21000 22000 23000	After the occurrence of any even of code 11000 - 13000, the number of integrand evaluations has reached the upper limit.	
25000	The table for abscissas(i.e. work area) has been exhausted.	Processing stops. S is an approximation by using the smallest stepsize allowed in this subroutine.
30000	One of the followings occurred: 1 EPSA < 0.0 2 EPSR < 0.0 3 NMIN < 0 4 NMAX < NMIN	Processing stops.

Comments on use

- Subprograms used
  SSL II... MGSSL, AMACH, AFMIN
  FORTRAN basic functions... MAX0, AMAX1,AMIN1, ABS, FLOAT, EXP, COSH, SINH

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable. Its function name must be declared as EXTERNAL in a calling program. If the integrand includes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with the calling program.
  When this subroutine is called many times, 641 constants (table of abscissas and weights for the integration formula) are determined only on the first call, and this computation is bypassed on subsequent calls. Thus, the computation time is shortened.
  This subroutine works most successfully when the integrand $f(x)$ changes rapidly in the neighborhood of endpoints of the interval. Therefore, when $f(x)$ has an algebraic or logarithmic singularity only at endpoint(s), the subroutine should be used with first priority.
  When $f(x)$ has interior singularities, the user can also use the subroutine provided that the subroutine is applied to each of subintervals into which the original interval is divided at the singularity points, or he can use subroutine AQN9 directly for the original interval.
  Subroutine AQN9 is suitable also for peak type functions, and subroutine AQC8 for smooth functions or oscillatory functions.
  This subroutine does not evaluate the function at both endpoints. A function value ($f(x) \to \pm\infty$) of infinity is allowed at the end points, but not allowed between them. Parameters NMIN and NMAX must be specified considering that this subroutine limits the number of evaluations of integrand $f(x)$ as

NMIN ≤ Number of evaluations ≤ NMAX

This means that $f(x)$ is evaluated at least NMIN times, but less than NMAX times, regardless of the result of the convergence test. When a value S that satisfies expression (1.1) is not obtained within NMAX evaluations, processing stops with ICON code 20000 - 23000.

Accuracy of the integral approximation S. The subroutine tries to obtain an approximation S which hopely satisfies (1.1) when ε_a and ε_r are given. $\varepsilon_a = 0$ means to obtain the approximation with its absolute error within ε_r. Sumilarly, ε_a=0 means to obtain it with its relative error within ε_r. This purpose is sometimes obstructed by unexpected characteristics of the function, or an unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small in comparison with arithmetic precision in function evaluations, the effect of rounding-off errors becomes greater, so it is no use to continue the computation even though the number of integrand evaluation has not reached the upper limit.
In this case, the accuracy of S becomes the attainable limit for the computer used. The approximation sometimes does not converge within NMAX evaluations. In this case, S is an approximation obtained so far, and is not accurate. This is indicated by ICON within the code range 20000 - 23000. In addition, ICON is set to 25000 when the approximation does not converge though the smallest step-size defined in this subroutine is used.
To determine the accuracy of integration, this subroutine always puts out an estimate of its absolute error in parameter ERR, as well as the integral approximation S.

An alternative definition of function for avoiding numerical cancellation.

For example, the integrand in the following integral has singularities at end points of x = 1,3,

$$I = \int_1^3 \frac{dx}{x(3-x)^{1/4}(x-1)^{3/4}}$$

and the function value diverges at that points. There, the integrand makes a great contribution to the integral. So, the function values near the end points must be accurately computed. Unfortunately, the function values cannot be accurately computed there since cancellation occurs in computing (3.0 - X) and (X-1.0).

This subroutine allows the user to describe the integrand in another form by variable transformation so that cancellation can be avoided. Parameters in subprograms are specified as follows:

FUNCTION FUN(X)

where,

X... Input. One dimensional array of size 2. X(1) corresponds to integration variable x and X(2) is defined depending upon the value of integration variable x as follows:

Letting AA = min(a,b), and BB = max(a,b),

- when AA ≤ x <(AA+BB) / 2, X(2)=AA − x
- when (AA+BB) / 2 ≤ x ≤ BB, X(2)=BB − x

In other words X(2) denotes the distance from either of the end points. The user can write his function using X(2) as follows:

$$f(x) = \begin{cases} f(\mathrm{AA} - \mathrm{X}(2)) & \text{when} \quad \mathrm{X}(2) < 0.0 \\ f(\mathrm{BB} - \mathrm{X}(2)) & \text{when} \quad \mathrm{X}(2) \geq 0.0 \end{cases}$$

The user can select either of X(1) or X(2) (See example).

- Example
  Two integrals

$$I_1 = \int_0^1 \frac{dx}{\sqrt{x}}, I_2 = \int_1^3 \frac{dx}{x(3-x)^{1/4}(x-1)^{3/4}}$$

are computed. The integrand in I_1 is defined in the function subprogram FUN1 and that in I_3 is defined in FUN2 respectively. FUN2 uses the technique described in Note.

```
C     **EXAMPLE**
      EXTERNAL FUN1,FUN2
      A=0.0
      B=1.0
      EPSA=1.0E-5
      EPSR=0.0
      NMIN=20
      NMAX=641
      CALL AQE(A,B,FUN1,EPSA,EPSR,NMIN,
     *     NMAX,S1,ERR1,N1,ICON1)
      A=1.0
      B=3.0
      CALL AQE(A,B,FUN2,EPSA,EPSR,NMIN,
     *     NMAX,S2,ERR2,N2,ICON2)
      WRITE(6,600) ICON1,S1,ERR1,N1,
     *             ICON2,S2,ERR2,N2
      STOP
  600 FORMAT(' ',30X,'ICON1=',I5,5X,
     *'S1=',E15.7,5X,'ERR1=',E15.7,
     *5X,'N1=',I5//
     *       ' ',30X,'ICON2=',I5,5X,
     *'S2=',E15.7,5X,'ERR2=',E15.7,
     *5X,'N2=',I5)
      END

      FUNCTION FUN1(X)
      FUN1=0.0
      IF(X.GT.0.0) FUN1=1.0/SQRT(X)
      RETURN
      END

      FUNCTION FUN2(X)
      DIMENSION X(2)
      T=X(2)
      IF(T.GE.0.0) GO TO 10
      P=(1.0-T)*(2.0+T)**0.25*(-T)**0.75
      GO TO 20
   10 P=(3.0-T)*T**0.25*(2.0-T)**0.75
   20 FUN2=0.0
      IF(P.GT.0.0) FUN2=1.0/P
      RETURN
      END
```

Method

This subroutine uses the automatic integration method based on Takahashi-Mori's double exponential formula. The principle of this method is given first, and next the computing procedures in this subroutine.

- Double exponential formula

  The given integral $\int_a^b f(x)dx$ may be transformed by using the linear transformation

$$x = \frac{b-a}{2}t + \frac{a+b}{2}$$

into

$$\frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)dt$$

For simplicity, let's consider the integration (4.1) over the finite interval [-1,1].

$$I = \int_{-1}^{1} f(x)dx \qquad (4.1)$$

On condition that $f(x)$ is analytical in the open interval (-1,1), it is allowed to have singularities at $x=\pm 1$ such as

$(1-x)^{\alpha}(1+x)^{\beta}$, $-1 < \alpha, \beta$

By a variable transformation,

$$x=\phi(t) \qquad (4.2)$$

the interval [-1,1] is transformed to $(-\infty, \infty)$ and consequently the integral (4.1) is transformed to

$$I = \int_{-\infty}^{\infty} f(\phi(t))\phi'(t)dt \qquad (4.3)$$

Remembering that the trapezoidal rule is best for integrals over the infinite interval, the following integration formula is obtained by applying that rule with step-size h to (4.3).

$$I_h = h\sum_{n=-\infty}^{\infty} f(\phi(nh))\phi'(nh) \qquad (4.4)$$

Based on an analysis on errors which arise in approximating the infinite sum above by the finite one, Takahashi and Mori showed the optimal transformation $\phi(t)$ with which the integrand in (4.3) will decay in a manner shown by (4.5) below as $|t|$ increase (see Fig.AQE-1).

$$\left|f(\phi(t))\phi'(t)\right| \approx \exp(-a\exp|t|), a > 0 \qquad (4.5)$$

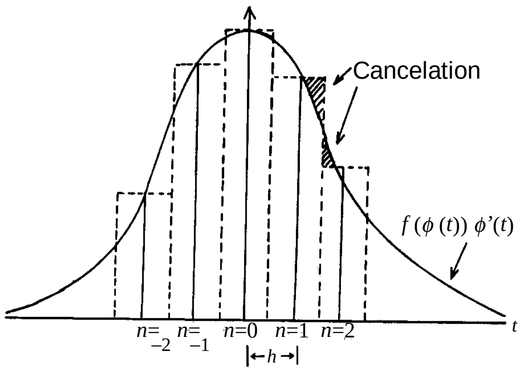


Fig. AQE-1 Trapezoid rule applied to f

As the transformation which enable the double exponential decay such as (4.5) to happen, this subroutine takes the following one.

$$\begin{aligned} x &= \phi(t) = \tanh\left(\frac{3}{2}\sinh(t)\right) \\ \phi'(t) &= \cosh(t) \Big/ \cosh^2\left(\frac{3}{2}\sinh(t)\right) \end{aligned} \qquad (4.6)$$

- Computing procedure

Procedure 1 ... Initialization
To transform the finite interval $[a, b]$ to [-1,1], determine the constant.

$r=(b-a)/2$

All the initialization required are done in this procedure.

Procedure 2 ... Determines the upper and lower limits of the infinite summation used to approximate the infinite summation (4.4). A very approximate integral S' is obtained.

Procedure 3 ... By the summation of the finite number of products, approximations $S(h)$, $S(h/2)$, $S(h/4)$... for $\boldsymbol{I}_h$, $\boldsymbol{I}_{h/2}$, $\boldsymbol{I}_{h/4}$,..., are computed by bisectioning the step-size, until it converges.

Procedure 4 ... Sets values in S, ERR and ICON.

- Convergence criterion

  If a step-size h used in the equally spaced trapezoidal rule is sufficiently small, it is proved analytically that the error $\Delta\boldsymbol{I}_h = \boldsymbol{I} - \boldsymbol{I}_h$ can be expressed

$$\left|\Delta I_{h/2}\right| \approx \left|\Delta I_h\right|^2$$

From this, letting ε denote the desired accuracy, $\left|\Delta I_{h/2}\right| \le \varepsilon$ requires $\left|\Delta I_h\right| \le \varepsilon^{1/2}$.
Since $|\,\Delta\boldsymbol{I}_{h/2}\,| \ll \|\,\Delta\boldsymbol{I}_h\,\|$ numerically,

$$\left|S(h) - S(h/2)\right| \approx \left|I_h - I_{h/2}\right| = \left|\Delta I_h - \Delta I_{h/2}\right| \approx \left|\Delta I_h\right|$$

holds takes the form. Thus, if the convergence criterion takes the form

$$|S(h) - S(h/2)| \leq \eta = \varepsilon^{\alpha}$$

α=1/2 is allowed theoretically. From experience, this subroutine uses the following values of α for security, where $\varepsilon' = \max(\varepsilon_a / |S'|, \varepsilon_r)$ (or $\varepsilon' = \varepsilon_r$ when S' = 0)

When	$10^{-4} \leq \varepsilon'$	α=1.0
When	$10^{-5} \leq \varepsilon' < 10^{-4}$	α=0.9
When	$10^{-10} \leq \varepsilon' < 10^{-5}$	α=0.8
	$\varepsilon' < 10^{-10}$	α=0.75

As the desired accuracy ε, $\max(\varepsilon_a, \varepsilon_r \cdot |S(h/2)|,)$ is used.

- Determination of initial step-size and threshold
  The step-size is initialized to 0.5 such an integer n (n= ±1, ..., ±10) as satisfies the condition.

$$F = \left| f(\phi(nh))\phi'(nh) \right| \leq \eta'$$

continuously twice is searched for, where

$$\eta' = \min\left(\max\left(\varepsilon_a, \varepsilon_r \|F\|_{\infty}\right) 10^{-4} \|F\|_{\infty}\right)$$

When $F > \eta'$ even at n = ±10, the convergence tolerance ε' is replaced by F and procedure 3 is executed.

- Detection of round-off error effect
  Letting $e_h = |S(h) - S(h/2)|$, then if $\varepsilon_h \leq \eta$ is satisfied, $S(h/2)$ is put out in parameter S as an approximation to the integral and $e_h^{1/\alpha}$ is put out in parameter ERR. However, when $e_h \leq \eta$ but also any of the phenomena below occurs, it is regarded that the round-off error effect dominates over the truncation error effect, and processing stops with ICON=10000.
  Phenomenon1 $e_{h/2} \geq e_h \geq e_{2h}$
  Phenomenon 2 $e_h \leq u^{\alpha} |S(h/2)|$, where u is round-off unit.

In this case, e_h or $u|S(h/2)|$ put out in parameter ERR correspondingly to Phenomenon 1 or 2 respectively.

For details, see References [67] and [68].

G23-21-10101 AQEH, DAQEH

Integration of a function over the semi-infinite interval by double exponential formula
CALL AQEH(FUN,EPSA,EPSR,NMIN,NMAX,S,ERR, N,ICON)

Function

Given a function $f(x)$ and constants ε_a ε_r this subroutine obrains an approximation that satisfies (1.1) by using the Takahashi-Mori's double exponential formula

$$\left|S - \int_{-\infty}^{\infty} f(x)dx\right| \le \max\left(\varepsilon_a, \varepsilon_r \cdot \left|\int_{-\infty}^{\infty} f(x)dx\right|\right) \tag{1.1}$$

Parameter

FUN ...	Input. The name of the function subprogram which evaluates the integrand $f(x)$. (See the example.)
EPSA ..	Input. The absolute error tolerance $\varepsilon_a(\ge 0.0)$ for the integral.
EPSR ..	Input. The relative error tolerance $\varepsilon_r(\ge 0.0)$ for the integral.
NMIN ..	Input. Lower limit on the number of function evaluations (≥0.0). A proper value is 20.
NMAX ..	Input. Upper limit on the number of function evaluations (≥0). A proper value is 689 (a higher value, if specified, is interpreted to 689). (See "Comments on use" and "Notes".)
S	Output. An approximation to the integral. (See "Comments on use" and "Notes".)
ERR ...	Output. An estimate of the absolute error in the approximation.
N	Output. The number of function evaluations actually performed.
ICON ..	Output. Condition code. See Table AQEH-1.

Table AQEH-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The desired accuracy is not attained due to rounding-off errors.	The approximation obtained so far is output in S. The accuracy has reached the attainable limit.
11000 12000 13000	1, 2, 3 at the place of 1000 means respectively: 1 Function value increases steeply near at x=0. 2 Function value converges too late to 0 when X→∞ 3 Both 1 and 2 occur.	Processing continues with a relaxed tolerance.
20000	The desired accuracy is not attained though the number of integrand evaluations has reached the upper limit.	Processing stops. S is the approximation obtained so far, but is not accurate.
21000 22000 23000	After the occurrence of any event of code 11000 - 13000, the number of integrand evaluations reaches the upper limit.	
25000	The table for abscissas(i.e. work area) has been exhausted.	Processing stops. S is an approximation by using the smallest step-size allowed in this subroutine.
30000	One of the followings occurred: 1 EPSA < 0.0 2 EPSR < 0.0 3 NMIN < 0 4 NMAX < NMIN	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH, AFMAX
  FORTRAN Basic Functions ... AMIN1, ABS, AMAX1, FLOAT, SINH, COSH, EXP

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable. Its function name must be declared as EXTERNAL in a calling program. If the integrand in cludes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with the calling program.
  When this subroutine is called many times, 689 constants (table of abscissas and weights for the integration formula) are determined only on the first call and this computation is bypassed on subsequent calls. Thus the computation time is shortened.
  This subroutine works most successfully even for the integrand $f(x)$ which converges relatively slowly to zero when $x \to +\infty$, or $f(x)$ to which Gauss-Laguerre's rule cannot be applied.
  When the integrand $f(x)$ severely oscillates, highly accurate integral value may not be obtained.
  This subroutine does not evaluate the function at the lower limit (origin). A function value is allowed to be infinite ($f(x) \to +\infty$) at the lower limit. Since function values at large values of x will be required, the function subprogram FUN needs to have a defence against overflows and underflows if the high accuracy is desired.
  Parameters NMIN and NMAX must be specified considering that this subroutine limits the number of evaluations of integrand $f(x)$ as
  NMIN≤Number of evaluations≤NMAX
  This means that $f(x)$ is evaluated at least NMIN times, but less than NMAX times, regardless of the result of the convergence test. When a value ***S*** that satisfies expression (1.1) is not obtained within NMAX evaluations, processing stops with ICON code 20000 - 23000. When an extremely small NMAX is given, for

example NMAX=2, NMAX is automatically increased to a certain value which depends upon the behavior of *f*(*x*).

Accuracy of the integral approximation ***S***

The subroutine tries to obtain an approximation ***S*** which satisfies (1.1) when ε_a and ε_r are given. When $\varepsilon_r = 0$, the approximation is obtained with its relative error within ε_r. This is sometimes obstructed by unexpected characteristics of the function or an unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small compared to the arithmetic precision in function evaluations, the effect of the rounding-off errors becomes greater, so there is no use in continuing the computation even though the number of integrand evaluations has not reached the upper limit (NMAX). In this case, the accuracy of ***S*** becomes the attainable limit for the computer used. The approximation sometimes does not converge within NMAX evaluations. In this case, ***S*** is an approximation obtained up to that time and so is not accurate. This is indicated by ICON within the code range 20000 - 23000. In addition, ICON is set to 25000 when the approximation does not converge though the *x* smallest step-size defined in this subroutine is used.

To determine the accuracy of integration, this subroutine always puts out an estimate of its absolute error in parameter ERR, as well as the integral approximation ***S***.

- Example

  The integral

$$\int_0^\infty e^{-x}\sin x\,dx$$

is computed in the program below.

```
C      **EXAMPLE**
       EXTERNAL FUN
       EPSA=1.0E-5
       EPSR=0.0
       NMIN=20
       NMAX=689
       CALL AQEH(FUN,EPSA,EPSR,NMIN,NMAX,
      *S,ERR,N,ICON)
       WRITE(6,600) ICON,S,ERR,N
       STOP
   600 FORMAT(' ',30X,'ICON=',I5,5X,
      *'S=',E15.7,5X,'ERR=',E15.7,
      *5X,'N=',I5)
       END
       FUNCTION FUN(X)
       IF(X.GT.176.0)GO TO 10
       FUN=EXP(-X)*SIN(X)
       RETURN
    10 FUN=0.0
       RETURN
       END
```

Method

This subroutine uses an automatic integration method based on Takahashi-Mori's double exponential formula.

For detailed information on this method, refer to the method of subroutine AQE. Here, the variable transformation applied to the integration variable is

$$x = \phi(t) = \exp\left(\frac{3}{2}\sinh t\right)$$

thus, to a weight function $\phi\,(t)$.

$$\phi'(t) = \frac{3}{2}\cosh t \cdot \exp\left(\frac{3}{2}\sinh t\right)$$

is used.

G23-31-0101 AQEI, DAQEI

Integration of a function over the infinite interval by double exponential formula
CALL AQEI(FUN, EPSA, EPSR, NMIN, NMAX, S, ERR, N, ICON)

Function

Given a function $f(x)$ and constants ε_a, ε_r this subroutine obtains an approximation that satisfies

$$\left|S - \int_{-\infty}^{\infty} f(x)dx\right| \leq \max\left(\varepsilon_a, \varepsilon_r \cdot \left|\int_{-\infty}^{\infty} f(x)dx\right|\right) \qquad (1.1)$$

by using Takahashi-Mori's double exponential formula.

Parameters

FUN ...	Input. The name of the function subprogram which evaluates the integrand $f(x)$ (see the example).
EPSA ..	Input. The absolute error tolerance $\varepsilon_a(\geq 0.0)$ for the integral.
EPSR ..	Input. The relative error tolerance $\varepsilon_r(\geq 0.0)$ for the integral.
NMIN ..	Input. Lower limit on the number of function evaluations (≥0). A proper value is 20.
NMAX ..	Input. Upper limit on the number of function evaluations (NMAX ≥ NMIN). A proper value is 645 (a higher value, if specified, is interpreted as 645). (See "Comments on use" and Notes.)
S	Output. An approximation to the integral. (See "Comments on use" and "Notes".)
ERR ...	Output. An estimate of the absolute error in the approximation.
N	Output. The number of function evaluations actually performed.
ICON ..	Output. Condition code. See Table AQEI-1.

Table AQEI-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The desired accuracy is not attained due to rounding-off errors.	The approximation obtained so far is put out in S. The accuracy has reached the attainable limit.
11000 12000 13000	1,2,3 at the place of 1000 means that the function value converges too late to 0 when $x^{\to -\infty}$, $x^{\to \infty}$, $x^{\to \pm\infty}$,respectively.	Processing continues with a relaxed tolerance.
20000	The desired accuracy is not attained though the number of integrand evaluations has reached the upper limit.	Processing stops. S is an approximation obtained so far, but is not accurate.
21000 22000 23000	After the occurrence of any event of codes 11000 - 13000, the number of integrand evaluations reached the upper limit.	
25000	The table for abscissas(i.e. work area) has been exhausted.	Processing stops. S is an approximation by the smallest step-size allowed in this subroutine.
30000	One of the followings occurred. 1 EPSA<0.0 2 EPSR<0.0 3 NMIN<0 4 NMAX<NMIN	Processing stops.

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH
  FORTRAN basic functions ... AMIN1, ABS, AMAX1, FLOAT, SINH, COSH, EXP

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only an integration variable. Its function name must be declared as EXTERNAL in a calling program. If this function includes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with a main program.
  When this subroutine is called many times, 645 constants (tables of abscissas and weights for the integration formula) are determined only on the first call. This computation is bypassed on subsequent calls.
  This subroutine works successfully even for integrand $f(x)$ which converges relatively slowly to 0 when $x \to \infty$, or $f(x)$ to which Gauss-Hermite's rule cannot be applied. The accuracy is sometimes reduced when $|x|$ has a high peak near the origin or is oscillatory.
  Since function values at large values of $f(x)$ are required, the function subprogram FUN needs to have a defense against overflows and underflows if the desired accuracy is high.

Parameters NMIN and NMAX
This subroutine limits the number of evaluations of the integrand $f(x)$ as follows:

$$\text{NMIN} \leq \text{Number of evaluations} \leq \text{NMAX}$$

This means that $f(x)$ is evaluated at least NMIN times but not more than NMAX times regardless of the result of the convergence test. When a value of S that satisfies (1.1) is not obtained within NMAX evaluations, processing stops with ICON code 20000 - 23000. Or when extremely small NMAX is given, for example NMAX=2, NMAX is increased automatically to a value which is determined by the behavior of $f(x)$.

Accuracy of the integral approximation S
This subroutine obtains a value of S that satisfies the expression (1.1) when constants ε_a and ε_r are given. ε_r=0 means to obtain the approximation with its absolute error within ε_a. Similarly, ε_r=0 means to obtain it with its relative error within ε_r. This purpose is sometimes obstructed by unexpected characteristics of the function, or unexpected value of ε_r. For example, when ε_a or ε_r is extremely small in comparison with arithmetic precision in function evaluation, the effect of rounding-off errors becomes greater, so it is no use to continue the computation even though the number of integrand evaluations has not reached the upper limit. In this case, processing stops with ICON code 10000. At this time, the accuracy of S has reached the attainable limit for the computer used. The approximation sometimes does not converge within NMAX evaluations. In this case, *S* is the approximation obtained so far, and is not accurate. This is indicated by ICON within the code range 20000 - 23000. In addition, ICON is set to 25000 when the approximation does not converge though the smallest step-size defined in this subroutine is used. To determine the accuracy of integration, this subroutine always puts out an estimate of its absolute error in parameter ERR, as well as the integral approximation S.

- Example
  The integral

$$\int_{-\infty}^{\infty} \frac{1}{10^{-2}+x^2} dx$$

  is obtained in the program below.

```
C      **EXAMPLE**
       EXTERNAL FUN
       EPSA=1.0E-5
       EPSR=0.0
       NMIN=20
       NMAX=645
       CALL AQEI(FUN,EPSA,EPSR,NMIN,NMAX,
      *S,ERR,N,ICON)
       WRITE(6,600) ICON,S,ERR,N
       STOP
  600 FORMAT(' ',30X,'ICON=',I5,5X,
      *'S=',E15.7,5X,'ERR=',E15.7,
      *5X,'N=',I5)
       END
       FUNCTION FUN(X)
       IF(ABS(X).GT.1.0E+35) GO TO 10
       IF(ABS(X).LT.1.0E-35) GO TO 20
       FUN=1.0/(1.0E-2+X*X)
       RETURN
   10 FUN=0.0
       RETURN
   20 FUN=100.0
       RETURN
       END
```

Method
This subroutine uses the automatic integration method based on Takahashi-Mori's double exponential formula.
As for the principles and computing procedures of this method, refer to the descriptions under the heading Method of subroutine AQE. To add a description for subroutine AQEI, the variable transformation applied to the integration variable x is

$$x = \phi(t) = \sinh\left(\frac{3}{2}\sinh t\right)$$

thus, to a weight function $\phi'(t)$;

$$\phi'(t) = \frac{3}{2}\cosh t \cdot \cosh\left(\frac{3}{2}\sinh t\right)$$

is used.

G24-13-0101 AQMC8, DAQMC8

Multiple integration of a function by a modified Clenshaw-Curtis rule
CALL AQMC8(M,LSUB,FUN,EPSA,EPSR,NMIN, NMAX,S,ERR,N,ICON)

Function

A multiple integral of dimension $m(1 \le m \le 3)$ is defined here by

$$I = \int_{\varphi_1}^{\psi_1} dx_1 \int_{\varphi_2}^{\psi_2} dx_2 \cdots \int_{\varphi_m}^{\psi_m} dx_m \, f(x_1, x_2, \cdots, x_m) \qquad (1.1)$$

The upper limits and lower limits are given by

$\varphi_1 = a$ (constant) , $\psi_1 = b$ (constant)
$\varphi_2 = \varphi_2(x_1)$, $\psi_2 = \psi_2(x_1)$
: :
$\varphi_m = \varphi_m(x_1, x_2, ..., x_{m-1})$, $\psi_m = \psi_m(x_1, x_2, ..., x_{m-1})$

Then, this subroutine obtains an approximation ***S*** which satisfies

$$|S - I| \le \max(\varepsilon_a, \varepsilon_r |I|) \qquad (1.2)$$

for ε_a, ε_r given by a modified Clenshaw-Curtis rule applied to each dimension.

Parameters

M ..... Input. Dimension m of the integral

LSUB .. Input. The name of the subroutine sub-program which evaluates the lower limit φ_k and upper limit ψ_k. The form of the subroutine is as follows:
SUBROUTINE LSUB(K,X,A,B)
where,
K ... Input. Index k of integration variable. $1 \le k \le m$.
X ... Input. One-dimensional array of size (M-1) which corresponds to X(1)=x_1, X(2)=x_2,...,X(M-1)=x_{m-1}
A ... Output. The value of the lower limit $\varphi_k(x_1, x_2, ..., x_{k-1})$
B ... Output. The value of the upper limit $\psi_k(x_1,x_2,...,x_{k-1})$

FUN ... Input. The name of the function subprogram which evaluates the integrated $f(x_1,x_2,...,x_m)$
The form of the subroutine is as follows:
FUNCTION FUN(X)
Parameter X is a one-dimensional array of size M which corresponds to X(1)=x_1, X(2)=x_2, ..., X(M)=x_m.

EPSA ... Input. The absolute error tolerance ε_a (≥0.0) for the integral.

EPSR ... Input. The relative error tolerance ε_r(≥ 0.0) for the integral.

NMIN ... Input. Lower limit (≥0) on the number of evaluations of the integrand function when integrating in each integral variable. A proper value is 7.

NMAX ... Input. Upper limit (NMAX ≥ NMIN) on the number of evaluations of the integrand function when integrating in each integral variable. A proper value is 511. (When a value larger than 511 is specified the value is assumed to be 511.) (See Note.)

S .... Output. An approximation (See Note.)

ERR ... Output. An estimate of the absolute error in the approximation.

N ... Output. Total number of integrand evaluations actually performed. It must be a 4-byte integer variable

ICON .. Output. Condition code. See Table AQMC8-1.

Table AQMC8-1 Condition code

Code	Meaning	Processing
0	No error	
100 1000 1100 10000 10100 11000 11100	When integrating the function in the direction of a certain coordinate axis, the requested accuracy in that direction could not be obtained because of round-off error. The third place indicates that during integration in the direction of axis x_3, the difficulty occurred for various pairs of (x_1,x_2). The fourth place indicates that during integration in the direction of axis x_2, the difficulty occurred for various values of x_1. The fifth place indicates that the difficulty occurred in the direction of axis x_1.	The obtained approximation is output to S. When the condition code is in the range of 100 through 1100, the accuracy satisfies the request or reaches the limit of arithmetic precition. When the condition code is in the range of 10000 through 11100 the accuracy reaches the limit. In either case, the error is output to ERR.
200 2000 2200 20000 20200 22000 22200	When integrating the function in the direction of a certain coordinate axis, the number of evaluation of the integrand function reached the upper limit, but the requested accuracy in the direction of the axsis could not be obtained. The indication of the places is the same as condition codes 100 through 11100.	The obtained approximation is output to S. When the condition code is in the range of 200 through 2200, the required accuracy may be satisfied or not satisfied. In either case, the error is output to ERR. If a large value of NMAX is given, the precision may be modified (the limit of NMAX is 511).

Table AQMC8-1 - continued

Code	Meaning	Processing
300 | 23300	The events indicated in condition codes 100 through 11100 and those of 200 through 22200 occurred concurrently.	The obtained approximation is output to S. The required accuracy may be satisfied or may not be satisfied. In either case, the error is output to ERR.
30000	One of the following detected. 1 EPSA<0.0 2 EPSR<0.0 3 NMIN<0 4 NMAX<NMIN 5 M ≤ 0, or M ≥ 4	Bypassed

Comments on use

- Subprogram used
  SSL II ... MGSSL, AMACH
  FORTRAN basic functions ... ABS, AMAX1, FLOAT, MAX0, MIN0, SQRT

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable vector X. Its function name must be declared as EXTERNAL in a calling program. If the integrand includes an auxiliary variable, this must be declared in the COMMON statement for the purpose of communicating with the calling program (see the example). Also, the name of the subprogram associated with parameter LSUB must be declared as EXTERNAL in a calling program.
  When this subroutine is called many times, 511 constants (tables of abscissas and weights for the integration formula) are determined only on the first call and this computation is bypassed on subsequent calls, thus the computation time is shortened.
  This subroutine works successfully not only for smooth integrands $f(x_1,x_2,...,x_m)$ but also for oscillatory ones.
  This subroutine limits the number of evaluations n_i where i=1,2,...,m in the direction of each coordinate axis of the integrand function in the range of:
  NMIN≤n_i≤NMAX
  This means that $f(x_1,x_2,...,x_m)$ is evaluated at least NMIN times, but not more than NMAX times, regardless of the result of the convergence test. If the approximation does not converge within NMAX times in the direction of a certain coordinate axis, the processing stops with ICON code 200 – 22200. The position of numeric character 2 indicates the type of axis x_3, x_2 or x_1 corresponding to 100, 1000 or 10000 respectively. When NMAX is specified as less than seven, seven is assumed.

Accuracy of the integral approximation ***S***
This subroutine tries to obtain an approximation ***S*** which satisfies (1.2) when ε_a and ε_r are given. When ε_r=0, an approximation is obtained with an absolute error within ε_a. Similarly, when ε_a=0, an approximation is obtained with a relative error within ε_r. This is sometimes obstructed by an unexpected characteristic of the function or an unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small compared to the arithmetic precision in function evaluations, the effect of the rounding-off errors becomes greater, so there is no use in continuing the computation even though the number of integrand evaluations has not reached the upper limit(NMAX). In this case, ICON is set to 100 - 11100. The position of numeric character 1 indicates the type of axis x_3, x_2 and x_1 corresponding to 100, 1000 or 10000 respectively. Generally speaking, even when ICON is set to 100 - 1100, that is the effect of round-off errors becomes greater on axis x_3 and x_2, this may not effect the total accuracy of integral S. It may satisfy the required accuracy. This should be checked depending on the error estimation ERR.
As mentioned above, an approximation may not converge within NMAX times. In this case, ICON is set to 200 - 22200, the position of the number 2 indicating the type of axis. When ICON is set to 200 - 22200, the obtained integral may satisfy the required accuracy. When the events indicated by condition codes 100 - 11100 and 200 - 22200 occur, ICON is set to 300 - 23300.
To determine the accuracy of the integration, this subroutine always outputs an estimate of its absolute error in parameter ERR, as well as the integral approximation ***S***.

- Example
  Increasing the value of auxiliary variable ρ from 1.0 to 3.0 by 1.0 at a time, this example computes the integral.

$$I=\int_{-1}^{1}dx\int_{-2}^{2}dy\int_{-3}^{3}dz\frac{1}{\cos(px)\cos(py)\cos(pz)+2}$$

```
C       **EXAMPLE**
        INTEGER*4 N
        COMMON P
        EXTERNAL FUN,LSUB
        EPSA=1.0E-5
        EPSR=1.0E-5
        NMIN=7
        NMAX=511
        M=3
        DO 10 I=1,3
        P=FLOAT(I)
```

```
      CALL AQMC8(M,LSUB,FUN,EPSA,EPSR,
     *NMIN,NMAX,S,ERR,N,ICON)
   10 WRITE(6,600) P,ICON,S,ERR,N
      STOP
  600 FORMAT(' ',30X,'P=',F6.1,5X,'ICON=',
     *I5,5X,'S=',E15.7,5X,'ERR=',E15.7,5X,
     *'N=',I5)
      END
      SUBROUTINE LSUB(K,X,A,B)
      DIMENSION X(2)
      GO TO (10,20,30),K
   10 A=-1.0
      B=1.0
      RETURN
   20 A=-2.0
      B=2.0
      RETURN
   30 A=-3.0
      B=3.0
      RETURN
      END

      FUNCTION FUN(X)
      DIMENSION X(3)
      COMMON P
      FUN=1.0/(COS(P*X(1))*COS(P*X(2))
     **COS(P*X(3))+2.0)
      RETURN
      END
```

Method

This subroutine uses a product formulas in which a modified Clenshaw-Curtis rule is applied repeatedly to each of dimension. Since AQC8 works successfully for smooth functions and oscillatory functions, this subroutine has the same characteristic with AQC8.

For detailed information on the Clenshaw-Curtis rule, see AQC8.

- The product formula

  Consider the following triple integral.

$$I = \int_{\varphi_1}^{\psi_1} dx_1 \int_{\varphi_2}^{\psi_2} dx_2 \int_{\varphi_3}^{\psi_3} dx_3\, f(x_1, x_2, x_3) \quad (4.1)$$

The lower limit and upper limit of the integral are given by

$\varphi_1=a$, $\psi_1=b$
$\varphi_2=\varphi_2(x_1)$, $\psi_2=\psi_2(x_1)$
$\varphi_3=\varphi_3(x_1,x_2)$, $\psi_3=\psi_3(x_1,x_2)$

The integral I in (4.1) is obtained by computing step by step as follows:

$$I_2(x_1, x_2) = \int_{\varphi_3}^{\psi_3} f(x_1, x_2, x_3)dx_3 \quad (4.2)$$

$$I_1(x_1) = \int_{\varphi_2}^{\psi_2} I_2(x_1, x_2)dx_2 \quad (4.3)$$

$$I = \int_{\varphi_1}^{\psi_1} I_1(x_1)dx_1 \quad (4.4)$$

Therefore, I is obtained when the function $I_1$is integrated for x_1. In this state the function $I_1(x_1)$ can be obtained by fixing x_1 when the function $I_2(x_1,x_2)$ is integrated fixing x_1 and x_2 on $[\varphi_3(x_1,x_2),\psi_3((x_1,x_2)]$ for x_3 of $f(x_1,x_2,x_3)$. This subroutine uses a modified Clenshaw-Curtis rule to integrate I along coordinate axis x_1, x_2, or x_3.

- Computing procedures

Procedure 1 ... Initialization
Determine the initial value of various variables and constants used for tests.

Procedure 2 ... Determination of abscissas and weights
Obtain the abscissas and weights to be used for the Clenshaw-Curtis rule by a recurrence formula. Then put them into one-dimensional work arrays. Since the upper limit on the number of abscissas is 511(=2^9–1), each size of the one-dimensional array required for the abscissas and weights is 256(=2^8), respectively.

Procedure 2 is performed only on the first call and this procedure is bypassed on subsequent calls.

Triple integration is considered in the following procedure 3, 4, and 5.

Procedure 3 ... Initialization for integration in the direction of axis x_1
Set seven points on the interval $[\varphi_1,\psi_1]$ which are defined as x_1^i (i=1,2,3,...7). These are the first seven abscissas obtained in Procedure 2 and linearly transformed to the interval $[\varphi_1,\psi_1]$.

Procedure 4 ... Initialization for integration in the direction of axis x_2
One of x_1^i(i=1,2,...,7) or one of the eight abscissas added on axis x_1 is assumed to be X_1. Set the seven points on interval $[\varphi_2(X_1),\psi_2(X_1)]$ of variable x_2 for this X_1. They are dencted by x_2^i (i = 1, 2, ..., 7). These are the points as in Procedure 3 which are obtained and linearly transformed to the interval $[\varphi_1,\psi_1]$.

Procedure 5 ... Initialization for integration in the direction of axis x_3
Assume one of x_2^i(i=1,2,...,7) or one of the eight abscissas added on axis x_2 to be X_2. Set seven points on the interval $[\varphi_3(X_1,X_2),\psi_3(X_1,X_2)]$ of variable x_3 for (X_1,X_2). They are denoted by x_3^i(i=1,2,...,7). These are the points as in Procedure 3 which are obtained in Procedure 2 and the first seven of them are linearly transformed.

Procedure 6 ... Integration in the direction of axis x_3
Compute the values of function f at the points (X_1,X_2,x_3^i) (i=1,2,...,7) and obtain an approximation $SI_7(X_1,X_2)$. The obtained approximation is the initial approximation for (4.2) when fixing (x_1,x_2).

Furthermore, define x_3^i where i=8,9,...,15 from the points obtained in Procedure 2. Then compute the values of function f at (X_1,X_2,x_3^i) for i=8,9,...15. The approximation $SI_{15}(X_1,X_2)$ for (4.2) is obtained based on the 15 points. Repeat this computation adding eight points each time until the approximation converges. If the approximation does not converge even with NMAX number of abscissas or the error cannot be improved any more because of round-off error, set the value of ICON depending upon the behavior of approximation.

Procedure 7 ... Integration in the direction of axis x_2
Execute procedures 5 and 6 for all x_2^i (i=1,2,...,7) defined in Procedure 4 and obtain approximation $SI_7(X_1)$ of the integral in the direction of axis x_2. This is an approximation to (4.3) when fixing x_1. Furthermore, define x_2^i (i=8,9,...,15) from the points obtained in Procedure 2 and execute Procedure 5 and 6 to obtain $SI_{15}(X_1)$. Repeat adding eight points each time until the approximation converges. When the approximation does not converge, set the value of ICON.

Procedure 8 ... Integration in the direction of axis x_1
Execute Procedures 4, 5, 6 and 7 for all x_1^i (i=1,2,...,7) defined in Procedure 3. Then obtain the approximation SI_7 of the integral in the direction of axis x_1. This is the initial approximation for the triple integration (4.4). Furthermore, define x_1^i (i=8,9,...,15) from the points obtained in Procedure 2 and execute procedure 4, 5, 6 and 7 to obtain SI_{15}. Repeat adding eight points each time until the approximation converges. When the approximation does not converge, set the value of ICON and terminate the processing.

- Error evaluation
  In the triple integration (4.1), to transfer the integral interval $[\varphi_1,\psi_1]$ to [-1,1]. the following variable transformation is performed:

$$x_1=\frac{\psi_1-\varphi_1}{2}x'_1+\frac{\psi_1+\varphi_1}{2}\equiv\alpha_1x'_1+\beta_1$$

The variable transformation for x_2 and x_3 is performed at one time to transfer the integral interval $[\varphi_2(x_1),\psi_2(x_1)]$, $[\varphi_3(x_1,x_2),\psi_3(x_1,x_2)]$ to [-1,1] as follows:

$$x_2=\frac{\psi_2(x_1)-\varphi_2(x_1)}{2}x'_2+\frac{\psi_2(x_1)+\varphi_2(x_1)}{2}$$
$$\equiv\alpha_2(x'_1)x'_2+\beta_2(x'_1)$$
$$x_3=\frac{\psi_3(x_1,x_2)-\varphi_3(x_1,x_2)}{2}x'_3+\frac{\psi_3(x_1,x_2)+\varphi_3(x_1,x_2)}{2}$$
$$\equiv\alpha_3(x'_1,x'_2)x'_3+\beta_3(x'_1,x'_2)$$

The integration (4.1) is obtained as follows:

$$I=\alpha_1\int_{-1}^{1}dx'_1\,h(x'_1) \tag{4.5}$$

$$h(x'_1)=\alpha_2(x'_1)\int_{-1}^{1}dx'_2\,g(x'_1,x'_2) \tag{4.6}$$

$$g(x'_1,x'_2)=\alpha_3(x'_1,x'_2)\int_{-1}^{1}dx'_3\,f(\alpha_1x'_1+\beta_1,\alpha_2(x'_1)x'_2+\beta_2(x'_1),\alpha_3(x'_1,x'_2)x'_3+\beta_3(x'_1,x'_2)) \tag{4.7}$$

When the Clenshaw-Curtis rule is used for the integration on the righthand side in (4.7), and the discretization error is denoted by $p(x_1',x_2')$, the following holds:

$$g(x'_1,x'_2)=\tilde{g}(x'_1,x'_2)+\alpha_3(x'_1,x'_2)p(x'_1,x'_2) \tag{4.8}$$

, where

$$\tilde{g}(x'_1,x'_2)=\alpha_3(x'_1,x'_2)\left[\sum_{k=0}^{7}A_{-1,k}W_{-1,k}+\sum_{i=0}^{P}\sum_{k=0}^{7}A_{i,k}W_{i,k}\right] \tag{4.9}$$

This is the approximation based upon (8P+7) points. And $A_{i,k}$ is the value depending upon (x'_1,x'_2) and $W_{i,k}$ is a weight. $p(x'_1,x'_2)$ is estimated by

$$|p(x'_1,x'_2)|=(|A_{P,5}|+|A_{P,7}|)\cdot|W_{P+1,1}| \tag{4.10}$$

Then substitute (4.8) in (4.6) and obtain the following:

$$h(x'_1)=\alpha_2(x'_1)\int_{-1}^{1}\tilde{g}(x'_1,x'_2)dx'_2+\alpha_2(x'_1)\int_{-1}^{1}\alpha_3(x'_1,x'_2)p(x'_1,x'_2)dx'_2 \tag{4.11}$$

When the Clenshaw-Curtis rule is used for the integration of the first item on the righthand side, and the discretization error is denoted by $q(x'_1)$, then the first item can be written as follows:

$$\left.\begin{array}{l}\alpha_2(x'_1)\int_{-1}^{1}\tilde{g}(x'_1,x'_2)dx'_2=\tilde{h}(x'_1)+\alpha_2(x'_1)q(x'_1)\\ \text{, where}\\ \tilde{h}(x'_1)=\alpha_2(x'_1)\left[\sum_{k=0}^{7}B_{-1,k}W_{-1,k}+\sum_{i=0}^{Q}\sum_{k=0}^{7}B_{i,k}W_{i,k}\right]\end{array}\right\} \tag{4.12}$$

$q(x'_1)$ is estimated as (4.10) by

$$|q(x'_1)|=(|B_{Q,5}|+|B_{Q,7}|)\cdot|W_{Q+1,1}| \tag{4.13}$$

Substitute the first equation of (4.12) in (4.11) and substitute the obtained equation in (4.5) to obtain the following:

$$I = \alpha_1 \int_{-1}^{1} \tilde{h}(x'_1)dx'_1 + \alpha_1 \int_{-1}^{1} \alpha_2(x'_1)q(x'_1)dx'_1$$
$$+\alpha_1 \int_{-1}^{1} \alpha_2(x'_1) \int_{-1}^{1} \alpha_3(x'_1, x'_2)p(x'_1, x'_2)dx'_2\, dx'_1 \tag{4.14}$$

Also, when the Clenshaw-Curtis rule is used for the integration of the first item on the righthand side of (4.14), and the discretization error is denoted by *r*, the following holds:

$$\left.\begin{aligned} &\alpha_1 \int_{-1}^{1} \tilde{h}(x'_1)dx'_1 = \tilde{I} + \alpha_1 r \\ &\text{, where} \\ &\tilde{I} = \alpha_1 \left[\sum_{k=0}^{7} C_{-1,k}W_{-1,k} + \sum_{i=0}^{R}\sum_{k=0}^{7} C_{i,k}W_{i,k} \right] \end{aligned}\right\} \tag{4.15}$$

r is estimated by

$$|r| = \left(|\mathrm{C}_{R,5}| + |\mathrm{C}_{R,7}|\right)\cdot |\mathrm{W}_{R+1,1}| \tag{4.16}$$

Therefore $\tilde{I}$ in (4.15) is the approximation to the triple integration and the error $|I-\tilde{I}|$ is obtained by substituting the first equation in (4.15) in (4.14) as follows:

$$|I-\tilde{I}| \le |\alpha_1 r| + \left|\alpha_1 \int_{-1}^{1} \alpha_2(x'_1)q(x'_1)dx'_1\right|$$
$$+\left|\alpha_1 \int_{-1}^{1} \alpha_2(x'_1) \int_{-1}^{1} \alpha_3(x'_1, x'_2)p(x'_1, x'_2)dx'_2\, dx'_1\right| \tag{4.17}$$

This indicates how the discretization errors $p(x'_1,x'_2)$, $q(x'_1)$ and *r* effect the entire error. The estimations of $p(x'_1,x'_2)$, $q(x'_1)$, and *r* can be obtained by (4.10), (4.13) and (4.16) respectively. These equations hold when the approximation converges rapidly. When the approximation does not converge rapidly, another estimation is considered. Define the approximation to the integral when using the Clenshaw-Curtis rule in which $2^n - 1$ points are used, as $I_{2^n} - 1$. The error for the approximation to the integral using the Clenshaw-Curtis rule in which $2^{n-1} - 1$ points are used can be evaluated by $|\mathrm{I}_{2^n-1} - \mathrm{I}_{2^{n-1}-1}|$. When the error evaluation of the Clenshaw-Curtis rule is assumed to be $\mathrm{E}(2^{n-1} - 1)$, it will be an underestimation if the approximation does not converge rapidly. Here the Clenshaw-Curtis rule contains $2^{n-1} - 1$ number of branch points using expansion coefficients and weight coefficients as in (4.10), (4.13) and (4.16).

Therefore the following equation holds:

$$\left|\mathrm{I}_{2^n-1} - \mathrm{I}_{2^{n-1}-1}\right| > E\left(2^{n-1} - 1\right) \tag{4.18}$$

When (4.18) is met, the following is used as the error evaluation for P in $2^n - 1 \le 8P+7 < 2^{n+1} - 1$ instead of E(8P+7).

$$\mathrm{E}(8\mathrm{P}+7)\cdot\left|\mathrm{I}_{2^n-1} - \mathrm{I}_{2^{n-1}-1}\right| \big/ \mathrm{E}\left(2^{n-1}-1\right) \tag{4.19}$$

I_{2^n-1}, $\mathrm{I}_{2^{n-1}-1}$ corresponds to $\tilde{g}(x'_1, x'_2), \tilde{h}(x'_1)$ and *I* when using 2^{n-1} or $2^{n-1} - 1$ number of branch points respectively.

- Convergence criterion
  In order $|I-\tilde{I}| \le \varepsilon_a$ (an absolute error tolerance), it is sufficient that each term of the righthand side in (4.17) can be bounded as:

$$|\alpha_1 r| \le \varepsilon_a/3$$
$$\left|\alpha_1 \int_{-1}^{1} \alpha_2(x'_1)q(x'_1)dx'_1\right| \le \varepsilon_a/3$$
$$\left|\alpha_1 \int_{-1}^{1} \alpha_2(x'_1) \int_{-1}^{1} \alpha_3(x'_1, x'_2)p(x'_1, x'_2)dx'_2\, dx'_1\right| \le \varepsilon_a/3$$

The following are sufficient for the above.

$$\left.\begin{aligned} &|r| \le \frac{\varepsilon_a}{3\alpha_1} \equiv \varepsilon_a^{(1)} \\ &|q(x'_1)| \le \frac{\varepsilon_a}{3\times 2\alpha_1\alpha_2(x'_1)} \equiv \varepsilon_a^{(2)} \\ &|p(x'_1, x'_2)| \le \frac{\varepsilon_a}{3\times 2\times 2\alpha_1\alpha_2(x'_1)\alpha_3(x'_1, x'_2)} \equiv \varepsilon_a^{(3)} \end{aligned}\right\} \tag{4.20}$$

Here the following are assumed:

$\alpha_1>0$, $\alpha_2(x'_1)>0$, $\alpha_3(x'_1,x'_2)>0$

On the other hand, computational error bounds for $\tilde{g}(x'_1, x'_2)/\alpha_3(x'_1, x'_2)$, $\tilde{h}(x'_1)/\alpha_2(x'_1)$ and $\tilde{I}/\alpha_1$ denoted by $\rho^{(3)}(x'_1,x'_2), \rho^{(2)}(x_1), \rho^{(1)}$, respectively, are estimated as follows using the round off unit.

$$\left.\begin{aligned} &\rho^{(3)}(x'_1, x'_2) = 8u(P+1)\|f\|_\infty \\ &\rho^{(2)}(x'_1) \quad = 16u(Q+1)\|\tilde{g}\|_\infty \\ &\rho^{(1)} \qquad = 32u(R+1)\|\tilde{h}\|_\infty \end{aligned}\right\} \tag{4.21}$$

where,

$$\|f\|_\infty = \max_{x'_3} \left| f(\alpha_1 x'_1 + \beta_1, \alpha_2 x'_2 + \beta_2, \alpha_3 x'_3 + \beta_3) \right|$$
$$\|\tilde{g}\|_\infty = \max_{x'_2} \left|\tilde{g}(x'_1, x'_2)\right|$$
$$\|\tilde{h}\|_\infty = \max_{x'_1} \left|\tilde{h}(x'_1)\right|$$

Summarizing the above discussions, the convergence criterion constants for each step are defined as follows:

$$\tau^{(3)}(x'_1, x'_2) = \max\left(\varepsilon_a^{(3)}, \varepsilon_r \left|\tilde{g}(x'_1, x'_2)/\alpha_3(x'_1, x'_2)\right|, \rho^{(3)}(x'_1, x'_2)\right)$$
$$\tau^{(2)}(x'_1) = \max\left(\varepsilon_a^{(2)}, \varepsilon_r \left|\tilde{h}(x'_1)/\alpha_2(x'_1)\right|, \rho^{(2)}(x'_1)\right)$$
$$\tau^{(1)} = \max\left(\varepsilon_a^{(1)}, \varepsilon_r \left|\tilde{I}/\alpha_1\right|, \rho^{(1)}\right)$$

where, $\tilde{g}$, and $\tilde{h}$, and $\tilde{I}$ of the righthand sides are approximations which have been obtained so far. The convergence criterion is performed as follows:
When the following equation is satisfied, $g(x'_1,x'_2)$ is assumed to have converged.

$$\left|p\left(x'_1, x'_2\right)\right| \le \tau^{(3)}\left(x'_1, x'_2\right) \qquad (4.22)$$

When the following is satisfied, $\tilde{h}(x'_1)$ is assumed to have converged.

$$\left|q\left(x'_1\right)\right| \le \tau^{(2)}\left(x'_1\right) \qquad (4.23)$$

Then the following is satisfied, the obtained $\tilde{I}$ is output to parameter S as an approximation to the multiple integral.

$$|r| \le \tau^{(1)} \qquad (4.24)$$

G24-13-0201 AQME, DAQME

Multiple integration of a function by double exponential formula
CALL AQME(M,INT,LSUB,FUN,EPSA,EPSR,NMIN, NMAX,S,ERR,N,ISF,ICON)

Function

A multiple integral of dimension m$(1 \le m \le 3)$ is defined here by

$$I = \int_{\varphi_1}^{\psi_1} dx_1 \int_{\varphi_2}^{\psi_2} dx_2 \cdots \int_{\varphi_m}^{\psi_m} dx_m \, f(x_1, x_2, \cdots, x_m) \qquad (1.1)$$

Generally the lower limits and upper limits are given as follows:

$$\begin{aligned} \varphi_1 &= a(\text{constant}) &, \ \psi_1 &= b(\text{constant}) \\ \varphi_2 &= \varphi_2(x_1) &, \ \psi_2 &= \psi_2(x_1) \\ &: & &: \\ \varphi_m &= \varphi_m(x_1, x_2, ..., x_{m-1}) &, \ \psi_m &= \psi_m(x_1, x_2, ..., x_{m-1}) \end{aligned}$$

This subroutine handles semi-infinite or infinite regions along with finite regions. In other words, the integral interval $[\varphi_k, \psi_k]$ for x_k can independently be $[0,\infty)$ or $(-\infty,\infty)$. Under these conditions, this subroutine obtains an approximation S that satisfies, for ε_a, ε_r given,

$$|S - I| \le \max(\varepsilon_a, \varepsilon_r |I|) \qquad (1.2)$$

by using Takahashi-Mori double exponential formula repeatedly.

Parameters

M ..... Input. Dimension m of the integral

INT .... Input. Information indicating the type of intervals for each integration variable. One-dimensional array of size m. The k-th element INT(K) indicates the type of the integration interval for the k-th variable x_k, and should be specified either of 1, 2, or 3 according to the following rule:
1 ... Finite interval
2 ... Semi-infinite interval
3 ... Infinite interval
For example, for the triple integration

$$I = \int_0^\infty dx_1 \int_0^\pi dx_2 \int_0^{2\pi} dx_3 \, f(x_1, x_2, x_3)$$

INT(1)=2, INT(2)=1 and INT(3)=1.

LSUB ... Input. The name of the subroutine Subprogram which evaluates the lower limit φ_k and upper limit ψ_k.
The form of the subroutine is as follows:
SUBROUTINE LSUB(K,X,A,B)
where,

K ... Input. Index k of integration variable. $1 \le k \le m$.

X ... Input. One-dimensional array of size (M−1) which corresponds to X(1)=x_1, X(2)=x_2,.....,X(M−1)=x_{m-1}−1.

A ... Output. The value of the lower limit $\varphi_k(x_1,x_2,...,x_{k-1}$−1)

B ... Output. The value of the upper limit $\psi_k(x_1,x_2,...,x_{k-1})$

However if the interval $[\varphi_k, \psi_k]$ is either $[0,\infty)$ or $(-\infty,\infty)$, it is not necessary to define values of A and B for corresponding k.

FUN ... Input. The name of the function subprogram which evaluates the integrand $f(x_1,x_2,...,x_m)$
The form of subroutine is as follows:
FUNCTION FUN(X)
where, parameter X is a one-dimensional array of size M with the correspondence X(1)=x_1,X(2)=x_2,...,X(M)=x_m.

EPSA ... Input. The absolute error tolerance ε_a (≥0.0) for the integral.

ERSR ... Input. The relative error tolerance ε_r (≥0.0) for the integral.

NMIN ... Input. Lower limit (≥0) on the number of evaluations of the integrand function when integrating in each integration variable. A proper value is 20.

NMAX ... Input. Upper limit (NMAX≥NMIN) on the number of the evaluation of the integrand function when integrating in each integration variable. A proper value is 705 (if the value exceeding 705 is specified, 705 is assumed). (See Notes)

S ..... Output. An approximation (See Notes)

ERR ... Output. An estimate of the absolute error in the approximation S.

N ..... Output. Total number of integrand evaluations actually performed. It must be a 4-byte integer variable.

ISF ... Output. Information on the behavior of the integrand when the value of ICON is in 25000's. ISF is a 3-digit positive integer in decimal. Representing ISF by

ISF=100j_1+10j_2+j_3

j_3, j_2 or j_1 indicates the behavior of the integrand function in the direction of axis x_3, x_2 or x_1 respectively. Each j_1 assumes 1, 2, 3 or 0 which is explained as follows:

1 ... The function value increases rapidly near the lower limit of integration or if the interval is infinite, the function values tend to zero very slowly as $x \to -\infty$.

2 ... The function value increases rapidly near the upper limit of integration or if the interval is semi-infinite or infinite, the function values tend to zero very slowly as $x \to \infty$.

3 ... The events indicated in the above 2 and 3 occur concurrently.

0 ... The above mentioned event does not occur.

ICON ... Output. Condition code (See Table AQME-1).

Comments on use

- Subprograms used
  SSL II ... MGSSL,AMACH,AFMAX,AFMIN, UAQE1,UAQE2,UAQE3,UFN10,UFN20,UFN30, UFN40
  FORTRAN basic function ... ABS,ALOG,SQRT, EXP,FLOAT,COSH,SINH,MAX0,AMAX1,AMIN1, MOD

Table AQME-1 Condition code

Code	Meaning	Processing
0	No error	
10001 │ 10077	When integrating in the direction of axis x_3 or x_2, the following events occur. The first place indicates the direction of axis x_3 and the second place indicates the direction of axis x_2. Each assumes the value of 0 through 7 (there is no case when both of them are zero). 1. The required accuracy in the direction of the axis cannot be obtained due to the round-off error. 2. The required accuracy cannot be obtained even if the number of the evaluations of the integrand function in the direction of the axis reaches the upper limit(NMAX). 4. The required accuracy in the direction of the axis cannot be obtained even if integrating by the minimum step size defined in the subroutine. 3. The events indicated above in 1 and 2 occur concurrently. 5. The events indicated above in 1 and 4 occur concurrently. 6. The events indicated above in 2 and 4 occur concurrently. 0. No events indicated above occur.	The obtained approximation is output to S. The required accuracy may be satisfied.
10100 │ 10177	When integrating in the direction of axis x_1, the required accuracy cannot be obtained due to the round-off error. The lower two digits mean the same as those in codes 10001 - 10077.	The obtained approximation is output to S. The accuracy almost reaches a limit attainable.
20200 │ 20277	When integrating in the direction of axis x_1, even if the number of evaluations of the integrand function reaches the upper limit(NMAX), the required accuracy cannot be obtained. The lower two digits mean the same as those in codes 10001 - 10077.	The obtained approximation is output to S. The required accuracy is not always guaranteed. If the value of NMAX is taken as a larger one, the accuracy may be improved (up to NMAX=705).
20400 │ 20477	When integrating in the direction of axis x_1 even by the minimum step size defined in the subroutine, the required accuracy cannot be obtained. The lower two digits mean the same as those in codes 10001 - 10077.	The obtained approximation is output to S.
25000 │ 25477	When integrating in the direction of a certain axis, the value of the function rapidly increases near the lower limit or upper limit of the integration interval, or when the integration interval is semi-infinite or infinite, the integrand function slowly converges to 0 as the integration variable tends to infinite. The lower three digits mean the same as those in codes 10001 - 10077.	Processing is continued after relaxing the required accuracy. The obtained approximation is output to S. Even when the integral does not exist theoretically, this range of code may be returned. For detailed information on the behavior of the integrand, refer to parameter ISF.
30000	One of the following occurred: 1 EPSA<0.0 2 EPSR<0.0 3 NMIN<0 4 NMAX<NMIN 5 M≤0 or M≥4 6 Some value other than 1, 2, or 3 is input for the element containing INT.	Processing terminates.

- Notes

  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable vector X. Its function name must be declared as EXTERNAL in a calling program. If the integrand includes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with the calling program.

  When this subroutine is called many times, constants (table of abscissas and weights for the integration formula) are determined only on the first call, and this computation is bypassed on subsequent calls, thus the computation time is shortened.

  This subroutine works successfully even when the integrand function changes rapidly in the neighborhood of the boundary of integration region. This subroutine is most recommended when algebraic or logorithmic singularities are located on the boundary. If the integral domain is finite and the integrand is smooth or oscillatory, you should use subroutine AQMC8.

  This subroutine works successfully for the integrand $f(x_1,x_2,...,x_m)$ which converges to zero rather slowly when $x\rightarrow\pm\infty$. However, if the function is extremely oscillatory in the region, high accuracy may not be attained.

  This subroutine does not evaluate the function on the boundary, therefore it is possible for the function value to be infinity. However, points which tend to infinity must not be contained in the region.

  When the integration interval in the direction of a certain axis(say, *i*-th axis) is infinite, the function values for large $|x_i|$ is required, therefore if the desired accuracy is high, the function subprogram FUN needs to have a defence against overflows and underflows.

Parameters NMIN and NMAX

Parameters NMIN and NMAX must be specified considering that this subroutine limits the number of evaluations of integrand in the direction of each coordinate axis as

NMIN $\le n_i \le$ NMAX

This means that $f(x_1,x_2,...,x_m)$ is evaluated at least NMIN times in the direction of each coordinate axis, but not more than NMAX times, regardless of the result of the convergence test. When the integral does not converge within NMAX evaluations, this information is output to the first, second, or third digit of ICON corresponding to the axis x_3, x_2 or x_1 respectively.

When extremely small NMAX is given, for example NMAX=2, NMAX is increased automatically to a value which is determined by the behavior of $f(x_1,x_2,...,x_m)$.

Accuracy of the integral approximation S

This subroutine tries to obtain an approximation S which satisfies (1.2) when ε_a and ε_r are given. When ε_r=0, the approximation is obtained with its absolute error within ε_a. Similarly, when ε_a=0, the approximation is obtained with its relative error within ε_r. This is sometimes obstructed by unexpected characteristics of the function or an unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small compared to the arithmetic precision in function evaluation, the effect of rounding-off errors becomes greater, so there is no use in continuing the computation even though the number of integrand evaluations has not reached the upper limit (NMAX). Depending upon the axis, this information is output to the third, second or first place of ICON. Generally speaking, even when the effect of the rounding-off error on axis x_3 or x_2 is large, this may not effect the total accuracy of integral approximation S. It may satisfy the required accuracy. This must be checked depending on the estimate ERR.

As mentioned in "Parameters NMIN and NMAX", the approximation sometimes does not converge within NMAX evaluations. In this case, this information is output to ICON. Therefore, even if this event occurs on axis x_3 or x_2, the obtained integral approximation sometimes will satisfy the required accuracy.

In addition, the approximation may not converge though the smallest step-size defined in this subroutine is used. Although this information is output to ICON, even if this event occurs on axis x_3 or x_2, the required accuracy may be satisfied.

To determine the accuracy of the integration, this subroutine always outputs an estimate of its absolute error in parameter ERR, as well as the integral approximation S.

- Example

  The integral

$$I=\int_0^\infty dx_1\int_0^{x_1}dx_2\int_0^{1-x_2}\frac{e^{-x_1}}{x_1\sqrt{x_2+x_3}}dx_3$$

  is computed in the following program.

```
C      **EXAMPLE**
       INTEGER*4 N
       EXTERNAL FUN,LSUB
       DIMENSION INT(3)
       INT(1)=2
       INT(2)=1
       INT(3)=1
       EPSA=1.0E-3
       EPSR=1.0E-3
```

```
      NMIN=20
      NMAX=705
      M=3
      CALL AQME(M,INT,LSUB,FUN,EPSA,EPSR,
     *      NMIN,NMAX,S,ERR,N,ISF,ICON)
      WRITE(6,600) ICON,S,ERR,N
      IF(ICON.GE.25000) WRITE(6,610) ISF
      STOP
  600 FORMAT(' ',10X,'ICON=',I6,5X,'S=',
     *E15.7,5X,'ERR=',E15.7,5X,'N=',I6)
  610 FORMAT(' ',20X,'ISF=',I3)
      END
      SUBROUTINE LSUB(K,X,A,B)
      DIMENSION X(2)
      A=0.0
      GO TO (10,20,30),K
   10 RETURN
   20 B=X(1)
      RETURN
   30 B=1.0-X(2)
      RETURN
      END
      FUNCTION FUN(X)
      DIMENSION X(3)
      Y=X(2)+X(3)
      IF(Y.LT.1.0E-30) GO TO 20
      IF(X(1).GT.80.0) GO TO 20
      Y=X(1)*SQRT(Y)
      IF(Y.LT.1.0E-30) GO TO 20
      FUN=EXP(-X(1))/Y
      RETURN
   20 FUN=0.0
      RETURN
      END
```

Method

This subroutine uses the direct product multiple integration method in which automatic integration method based upon Takahashi-Mori's double exponential formula is repeated in the direction of each axis. Since the methods used in the following subroutines are used for this, you should refer to each subroutine for details. The way to apply the double exponential formula to multiple integration and the way to compute it are described below.

- Multiple integration using the double exponential formula

  In m-multiple integration defined in (1.1), when m=3, the variable transformation

$$I=\int_{\varphi 1}^{\psi 1}dx_1\int_{\varphi 2}^{\psi 2}dx_2\int_{\varphi 3}^{\psi 3}dx_3 f\left(x_1,x_2,x_3\right) \tag{4.1}$$

  is performed for each integral variable of the integration

$$x_k=\phi_k(t_k)\ ,\ \ k=1,2,3 \tag{4.2}$$

  When the integration (4.1) is transformed and the interval $[\varphi_k,\psi_k]$ is transformed to $(-\infty,\infty)$, the following integration will be obtained.

$$I=\int_{-\infty}^{\infty}dt_1\int_{-\infty}^{\infty}dt_2\int_{-\infty}^{\infty}dt_3 f\left(\phi_1(t_1),\phi_2(t_2),\phi_3(t_3)\right) \times\phi_1'(t_1)\phi_2'(t_2)\phi_3'(t_3) \tag{4.3}$$

(4.3) can be expressed step by step as follows:

$$I_2\left(\phi_1(t_1),\phi_2(t_2)\right)=\int_{-\infty}^{\infty}f\left(\phi_1(t_1),\phi_2(t_2),\phi_3(t_3)\right)\phi_3'(t_3)dt \tag{4.4}$$

$$I_1\left(\phi_1(t_1)\right)=\int_{-\infty}^{\infty}I_2\left(\phi_1(t_1),\phi_2(t_2)\right)\phi_2'(t_2)dt_2 \tag{4.5}$$

$$I_0=\int_{-\infty}^{\infty}I_1\left(\phi_1(t_1)\right)\phi_1'(t_1)dt_1\equiv I \tag{4.6}$$

Using the trapezoidal rule to obtain infinite integration I_{k-1}(k=1,2,3) in (4.4), (4.5) and (4.6), the following integration formula can be obtained.

$$I_{k-1}=h_k\sum_{n_k=-\infty}^{\infty}I_k\left(\phi_1(n_1h_1),...,\phi_k(n_kh_k)\right)\phi_k'(n_kh_k) \quad ,\quad k=1,2,3 \tag{4.7}$$

where, h_k is the step size for t_k.

This subroutine uses the following as variable transformation of (4.2):

1) When φ_k=A, ψ_k=B for finite A and B

$$\phi_k(t_k)=\frac{\mathrm{B}-\mathrm{A}}{2}\tanh\left(\frac{3}{2}\sinh t_k\right)+\frac{\mathrm{A}+\mathrm{B}}{2} \tag{4.8}$$

2) When φ_k=0, $\psi_k\to\infty$

$$\phi_k(t_k)=\exp\left(\frac{3}{2}\sinh t_k\right) \tag{4.9}$$

3) when $\varphi_k\to-\infty$, $\psi_k\to\infty$

$$\phi_k(t_k)=\sinh\left(\frac{3}{2}\sinh t_k\right) \tag{4.10}$$

- Multiple integration convergence criterion

  For each I_k in (4.4), (4.5) and (4.6), the following inequalities

$$\left|S_2-I_2\right|\le\max\left(\varepsilon_a,\varepsilon_r\left|I_2\right|\right) \tag{4.11}$$

$$\left|S_1-I_1\right|\le\max\left(\varepsilon_a,\varepsilon_r\left|I_1\right|\right) \tag{4.12}$$

$$\left|S_0-I_0\right|\le\max\left(\varepsilon_a,\varepsilon_r\left|I_0\right|\right) \tag{4.13}$$

are solved for each S$_k$, and the last S$_0$ is used as S in (1.2).

For detailed information on how to set the step size in the equally spaced step-size trapezoidal rule, information on convergence criterion, information on how to get the threshold to approximate the finite sum (4.7) by using infinite sum, or information on how to detect the influence of the rounding-off error, refer to Method of subroutine AQE.

- Computing procedure
  Procedure 1 ... Initialization
  Define the initial value of several variables and constants for criterions.

  Procedure 2 ... Computation of abscissas and weight coefficients
  Transforms the integral variables and compute abscissas ($\phi_k(t_k)$) and values of weight ($\phi'_k(t_k)$) according to INT value. The number of abscissas and weights are calculated as $2\times352\times3$. This computation is performed only when this subroutine is called for the first time. It is bypassed on subsequent calls.

  Procedure 3 ... Integration in the direction axis x_1-Computing I_0
  Approximation S_0 for I_0 in (4.6) is computed. Convergence is tested in (4.13). $I_1(\phi_1(t_1))$ value required for computing I_0 is computed in procedure 4. The obtained S_0 and the error estimate are output to parameter S and ERR.

  Procedure 4 ... Integration in the direction of axis x_2-Computing $I_1(\phi_1(t_1))$.
  Approximation S_1 for $I_1(\phi_1(t_1))$ in (4.5) is computed. The convergence is tested in (4.12). The value of $I_1(\phi_1(t_1))$ is computed in Procedure 5.

  Procedure 5 ... Integration in the direction of axis x_3-
  Approximation S_2 for $I_2(\phi_1(t_1),\phi_2(t_2))$ in (4.4) which is used to compute $I_2(\phi_1(t_1), \phi_2(t_2))$ is computed. The convergence is tested in (4.11).

G23-11-0201 AQN9, DAQN9

Integration of a function by adaptive Newton-Cotes 9 point rule
CALL AQN9 (A,B,FUN,EPSA,EPSR,NMIN,NMAX,S,ERR,N, ICON)

Function

Given a function $f(x)$ and constant a, b, ε_a, ε_r, this subroutine obtains an approximation S that satisfies

$$\left|S-\int_a^b f(x)dx\right| \le \max\left(\varepsilon_a, \varepsilon_r \cdot \left|\int_a^b f(x)dx\right|\right) \qquad (1.1)$$

by adaptive Newton-Cotes 9 point rule.

Parameters

A	Input. Lower limit a of the interval.
B	Input. Upper limit b of the interval.
FUN ...	Input. The name of the function subprogram which evaluates the integrand $f(x)$. (See the example.)
EPSA ..	Input. The absolute error tolerance $\varepsilon_a(\ge 0.0)$ for the integral.
EPSR ..	Input. The relative error tolerance $\varepsilon_r(\ge 0.0)$ for the integral.
NMIN ..	Input. Lower limit on the number of function evaluations. A proper value is 21. $0\le$NMIN<150.
NMAX ..	Input. Upper limit on the number of function evaluations. A proper value is 2000. (See Notes.)
S	Output. An approximation to the integral (see "Notes".)
ERR ...	Output. An estimate of the absolute error in the approximation.
N	Output. The number of function evaluations actually performed.
ICON ..	Output. Condition code. See Table AQN9-1.

Table AQN9-1 Condition codes

Code	Meaning	Processing
0	No error	
10000 | 13111	Irregular points such as singular points are found. 1000 means algebraic singularities 2000, Cauchy's singularities, and 3000, both of the two; 100 means logarithmic singularities; 10 means discontinuity points; 1 means other irregular points	Processing continues. For algebraic singularity, or discontinuity points only, S will usually satisfy the desired accuracy.
20000 | 23111	The desired accuracy is not attained though the number of integrand evaluations has reached the upper limit. Meanings of last four digits are the same as those described above.	Processing stops. S is an approximation obtained so far, but not accurate.
30000	One of the followings detected. 1 EPSA<0.0; 2 EPSR<0.0; 3 NMIN<0, NMIN≥150; 4 NMAX≤NMIN	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL,AMACH
  FORTRAN basic functions ... ABS, MAX0, AMAX1, AMIN1, ALOG, DLOG, MOD, FLOAT

- Notes
  The function subprogram associated with parameter FUN must be defined as a subprogram whose argument is only the integration variable. Its function name must be declared as EXTERNAL in its calling program. If the integrand includes auxiliary variables, they must be declared in the COMMON statement for the purpose of communicating with the calling program (see the example).

This subroutine may be used for a broad class of function: it can handle successfully even those integrands which have peaks, or irregular points such as algebraic singularities, logarithmic singularities, or discontinuities at places which can be accessed in the manner of bisection, such as endpoints, the midpoint or quartered points. Consequently this subroutine should be used with first priority for such class of integrands as well as ones whose properties are not clearly understood. However, in order to improve the accuracy of the solution, it is desirable to change the integration variable if necessary so that difficult points may be located only at endpoints.

It should also be noted, however, that subroutine AQC8 will best handle oscillatory functions as well as smooth ones in the sense of efficiency, while AQE does for functions having singularities only at endpoints.

When the value of the integrand $f(x)$ goes to infinity ($f(x)\rightarrow\pm\infty$) at a certain point within the interval $[a,b]$, it is necessary to replace the value of $f(x)$ at that point with some finite value (such as 0), as shown in the example below.

Parameters NMIN and NMAX

This subroutine limits the number of evaluation of integrand $f(x)$ as follows:

NMIN≤Number of evaluations≤NMAX

This means that $f(x)$ is evaluated at lease NMIN times and not more than NMAX times, regardless of the result of the convergence test. When a value of S that satisfies expression (1.1) is not obtained within NMAX evaluations, processing stops with ICON value 20000 - 21111. In addition, if the value of NMAX is less than 21, a default value of 21 is used.

Accuracy of the approximation *S*
This subroutine obtains a value of *S* satisfying (1.1) when constants ε_a, ε_r are given. Consequently, $\varepsilon_r=0$ means to obtain the approximation with its absolute error within ε_a, similarly, $\varepsilon_a = 0$ means to obtain it with its relative error within ε_r. This purpose is sometimes obstructed by unexpected characteristics of the function or unexpected values of ε_a or ε_r. For example, when ε_a or ε_r is extremely small in comparison with arithmetic precision in function evaluations, the number of function evaluations is increased and sometimes the approximation does not converge within NMAX times. In this case, S is only an interim approximation and not accurate. This is indicated by ICON within the code range from 20000 to 21111. As well as the integral approximation *S*, this subroutine puts out an estimate of the absolute error in parameter ERR for checking the actual accuracy of approximation.

- Example
  Increasing the value of the auxiliary variable p from 0.1 to 0.9 by 0.1 at a time, this example computes the integral

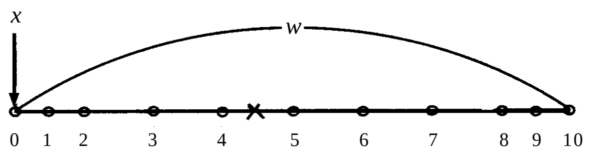
```
C     **EXAMPLE**
      COMMON P
      EXTERNAL FUN
      A=0.0
      B=1.0
      EPSA=1.0E-4
      EPSR=1.0E-4
      NMIN=21
      NMAX=2000
      DO 10 I=1,9
      P=FLOAT(I)/10.0
      CALL AQN9(A,B,FUN,EPSA,EPSR,NMIN,
     *     NMAX,S,ERR,N,ICON)
   10 WRITE(6,600) P,ICON,S,ERR,N
      STOP
  600 FORMAT(' ',30X,'P=',F6.3,5X,
     *'ICON=',I5,5X,'S=',E15.7,5X,
     *'ERR=',E15.7,5X,'N=',I5)
      END
      FUNCTION FUN(X)
      COMMON P
      FUN=0.0
      IF(X.GT.0.0) FUN=X**(-P)+SIN(P*X)
      RETURN
      END
```

Method
This subroutine uses an adaptive automatic integration based on the Newton-Cotes 9 point rule. Adaptive automatic integration is a method which automatically places abscissas densely where the integrand $f(x)$ changes rapidly, or sparsely where it changes gradually. This is currently the best method for automatic integration in terms of reliability and economy.

- Computing procedures

Procedure 1 ... Initialization
Initializes the left end point x, width w, and approximation S to the integral as $x=a$, $w=w_0=(b-a)$, and S=0; and initializes various variables and constants to be used for various judgment. Calculates function values to f_0, f_2, f_3, f_4, f_5, f_6, f_7, f_8, and f_{10}, at octasectional points of the integration interval including both end points, and f_1, and f_9 at the hexadecasectional end points. (See Figure AQN9-1.) The average value $\overline{f}$ of $f(x)$ in the integration interval is calculated with these 11 function values by means of the modified Newton-Cotes 9-point rule (expression 4.5).

Procedure 2 ... Bisection and information string
Bisects the current subinterval (left end point x, width w). Stores the function values f_6, f_7, f_8, f_9 and f_{10} relevant to the right-half portion, value of width w, and estimate e of the truncation error in the stack. Calculates the function value at the second right-most octasectional point (marked with x in Figure AQN9-1) in the left portion and let it be denoted by f_8. Proceeds to procedure 3 to manipulate function values in the left portion.



Points 0, 2 ~ 8, 10 are octasectional points
Points 1 and 9 are hexadecasectional points

Fig. AQN9-1 Point locations

Procedure 3 ... Convergence criterion
Calculates those function values at octasectional points and at hexadecasectional end points which are not available yet, namely, four values f_1, f_4, f_6, and f_9. Then estimates the truncation error e by (4.4) of the approximate integral over the current subinterval, and makes convergence test to e. If the test is satisfactory proceeds to procedure 7, otherwise to procedure 4.

Procedure 4 ... Detection of irregular points
Checks whether there is an irregularity at the end points of the current subinterval. If there is an irregularity, control is passed to procedure 6, and when width *w* becomes less than the tolerable minimum value determined by input variables such as *a*, *b*, ε_a, and ε_r, control is passed to procedure 5. In other cases, control is passed back to procedure 2.

Procedure 5 ... Confirmation of the existance of irregularity
Checks irregularity using the relaxed criterion. If the test is satisfactory proceeds to procedure 6; otherwise to procedure 7.

Procedure 6 ... Handling of irregular points
Calculates the approximate integral over the current subinterval by an analytical formula depending on the type of detected irregularity.

Procedure 7 ... Accumulation of the approximation retrieval
Normally, the approximate integral over the current subinterval is calculated by means of the modified 9-point rule(4.5) and accumulated to *S*. In the case of irregularity, the value obtained in procedure 6 is accumulated to *S*. The information stored in the stack is retrieved and the function value f_2 in a new subinterval is calculated, then control is passed back to procedure 3. If the stack becomes empty, processing is stopped and the value of *S* is assumed to be the integral.

- Newton-Cotes 9-point rule and error estimation
  Expressions (4.1) and (4.2) give the approximation *S*(*x*,*w*) and the truncation error *e* when the Newton-Cotes 9-point rule is applied to the interval with width *w* (Figure AQN9-1), where *I* is the true value of the integral and f_i is the function value at point *i*.

$$\begin{aligned} S(x,w) &= \frac{w}{28350}\{989(f_0+f_{10})+5888(f_2+f_8) \\ &\quad -928(f_3+f_7)+10496(f_4+f_6)-4540f_5\} \quad (4.1)\\ &= I+e \end{aligned}$$

$$e=\frac{37w^{11}f^{(10)}}{62783697715200} \qquad (4.2)$$

where $f^{(10)}$ denotes the 10th derivative of *f*(*x*) at some point in the interval. Conventionally, value *e* has been estimated as follows:
The Newton-Cotes 9-point rule is applied to both the bisected intervals to obtain *S*(*x*,*w*/2) and *S*(*x*+*w*/2,*w*/2) then *e* is estimated as

$$e \approx \frac{1024}{1023}\{S(x,w)-(S(x,w/2)+S(x+w/2,w/2))\} \qquad (4.3)$$

In this case, however, eight function values must be additionally calculated. As is clear from expression (4.2), it is only necessary to estimate $f^{(10)}$, so only 11 function values are sufficient to do this; therefore, only two additional function values need be calculated. This subroutine estimates value *e* from the 10th order differentiation formula (4.4) based on the octasectional points and two hexadecasectional points 1 and 9 of the interval (Figure AQN9-1) as follows:

$$\begin{aligned} e \approx \frac{2383w}{468242775}&\{3003(f_0+f_{10})-16384(f_1+f_9) \\ &+27720(f_2+f_8)-38220(f_3+f_7) \\ &+56056(f_4+f_6)-64350f_5\} \end{aligned} \qquad (4.4)$$

When the value *e* satisfies the convergence criterion (expression (4.7)), *e* is subtracted from *S*(*x*,*w*) to obtain a more accurate approximation. In this subroutine, however this is done directly by means of the modified Newton-Cotes formula shown below which is derived from the difference between the righthand sides of (4.1) and (4.4).

$$\begin{aligned} \frac{w}{936485550}&\{18447429(f_0+f_{10})+77594624(f_1+f_9) \\ &+63216384(f_2+f_8)+150355296(f_3+f_7) \\ &+81233152(f_4+f_6)+154791780f_5\} \end{aligned} \qquad (4.5)$$

In (4.5) the signs of weights to be multiplied by the function values are not constant, whereas the signs are constant in (4.5); thus, (4.5) is an good integration formula with high stability in numerical sense.

- Convergence criterion
  Unlike a global automatic integration method, an adaptive automatic integration requires convergence judgment for each subinterval. The following criterion has been used to make the resultant approximation *S* satisfy the requirement (1.1):

$$|e| \le \max(\varepsilon_a, \varepsilon_r|S'|)w/w_0 \qquad (4.6)$$

  where *S'* is an approximation to $\int_a^b f(x)dx$ and w_0 is the width (*b* – *a*) of the whole integration interval. This formula is based on a natural idea that the tolerable errors relevant to subintervals in proportion to each width. However, it is generally true that an approximation obtained by eliminating the dominant term in such a way done in the subroutine would be much more accurate than required. In this case, it seems to be too conservative to use (4.6) as it is. Instead it is advantageous to relax the criterion for short intervals having little effect on the total accuracy, to save the number of function evaluations. This subroutine employs (4.7) for this purpose:

$$|e| \le \max\left(\varepsilon_a, \varepsilon_r |S'|\right) w/w_0 \cdot \log_2\left(w_0/w\right) \qquad (4.7)$$

Although (4.7) is not based on a theoretical background, it is effective in practical use. The value S' in (4.7) is calculated using the average $\overline{f}$ of $f(x)$ obtained in procedure 1 as follows:

$$S' = S + \overline{f}\left(b - x\right) \qquad (4.8)$$

- Detection and handling of irregular points
  Consider that the integration interval includes irregularities such as discontinuities, logarithmic singularities, and algebraic singularities. The Cauchy's singularities for which the integral diverges in the normal sense but converges as the Cauchy's principal integration is also considered. If the function value is infinite at these irregularities, the value shall be replaced with an appropriate finite value, for example, zero.
  Since the value e in (4.4) does not satisfy the convergence criterion when the subinterval includes a irregularity, subdivision will be repeated to produce a sequence of intervals each including the irregularity. Assume that the irregularity is at one of 2^m -sectional point in the whole integration interval, where m is a positive integer. In this case, from some stage of repeated subdivision a sequence of subintervals $\{I_i, i=1,2,...\}$ are generated which share the irregularity at a fixed end point, with the length of each subinterval being halved. For simplicity, assume the irregularity is at the origin (x=0) and at the left end point of the interval. (See Figure AQN9-2)

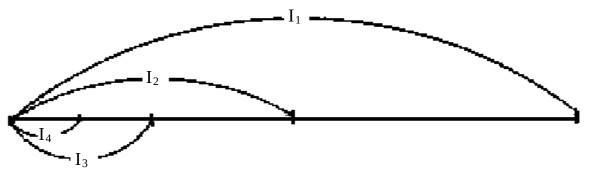


Fig. AQN9-2 Detection of irregularity

Consider a normalized error $\tilde{e}=e/w$ derived from normalizing the truncation error e by w. Let $\tilde{e}_i$ denote the $\tilde{e}$ relevant to the interval I_i. Since $\tilde{e}_i$ is a homogeneous linear combination of function values at similar points with respect to the irregular point and has coefficients independent of interval width, moreover the sum of the coefficients is zero, the following can be seen:

Discontinuity point	$\longleftrightarrow$	Value $\tilde{e}_i$ aproaches a constant value $\tilde{e}$.
Logarithmic singularity	$\longleftrightarrow$	Value $\Delta\tilde{e}_i = \tilde{e}_{i+1} - \tilde{e}_i$ approaches a constant value d.
Algebraic singularity	$\longleftrightarrow$	Value $\Delta\tilde{e}_i/\Delta\tilde{e}_{i+1}$ approachesa constant value r.

When $\tilde{e}_i$ exhibits one of the above properties over the subinterval is calculated as follows:

1) Discontinuity point ... Let the jump at the irregular point be δ, then it is obtained from (4.4) as follows:

$$\delta = \frac{468242775}{2383 \times 3003}\tilde{e} \qquad (4.8)$$

Since a discrepancy equal to δ has been added to the value of f_0, it is only necessary to substitue $f_0 - \delta$ for f_0 and perform calculation of (4.5).

2) Logarithmic singularity ... Assume the function in the vicinity of the irregular point to be:

$$f(x)=\alpha\log x+\beta \qquad (4.9)$$

Then, the value of α is obtained from (4.4) as follows:

$$\alpha = \frac{468242775}{2383 \times 3003 \times \log 2} d \qquad (4.10)$$

Since

$$I = \int_0^w f(x)dx = w\left(\alpha\left(\log w - 1\right) + \beta\right),$$
$$f_5 = f\left(w/2\right) = \log\left(w/2\right) + \beta$$

the value of I is calculated as:

$$I=w(f_5+\alpha(\log 2 - 1)) \qquad (4.11)$$

3) Algebraic singularity ... Assume the function in the vicinity of the irregular point to be:

$$f\left(x\right) = \alpha x^p + \beta x^{p+1} + \gamma \qquad (4.12)$$

Order p of the irregular point is obtained as:

$$p=\log_2 r \tag{4.13}$$

Calculate α, β and γ using f_0, $f_5=f(w/2)$, $f_{10}=f(w)$, γ, $\tilde{e}$ (last $\tilde{e}_i$), and $\tilde{e}'$ ($\tilde{e}$ previous to $\tilde{e}_i$); then, calculate the following analytically with these values:

$$I=\int_0^w f(x)dx=w\left(\frac{\alpha w^p}{p+1}+\frac{\beta w^{p+1}}{p+2}+\gamma\right) \tag{4.14}$$

4) Cauchy's singularity ... Assume an algebraic singularity to be at the right end of an interval and order p of the singularity is equal to or less than -1. If the singularity is also an algebraic singularity at the left end of the interval on the right of the interval and its order is p, and the combination of the primary portion there of and that of the left interval results in zero, this point is assumed to be a Cauchy's singularity. The integral over this interval is calculated as the integral of the integrand minus the primary portion.

- Information storing
  In procedure 2, the following 10 items are stored in the stack: five function values f_6, f_7, f_8, f_9 and f_{10} relevant to the right-half portion f_3, f_5, f_7, f_8 of the interval (to be used as and f_{10} later; width w and the value of $\log_2(w_0/w)$; and three values of $\tilde{e}_i$, $\Delta\tilde{e}_i$, and $\Delta\tilde{e}_i/\Delta\tilde{e}_{i+1}$, necessary to detect irregularities. In procedure 7, these items are retrieved in reverse order and processed again. The stack depth (the number of stack fields) is 60 in any case, that is, the required storage capacity is 600 elements.
  (See also references [96] and [97] for details.)

A21-12-0101 ASSM, DASSM

Addition of two matrices (real symmetric + real symmetric)
CALL ASSM(A,B,C,N,ICON)

Function

These subroutines perform addition of $n \times n$ real symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

$$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}$$

where $\boldsymbol{C}$ is an $n \times n$ real symmetric matrix. $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
B	Input. Matrix $\boldsymbol{B}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
C	Output. Matrix $\boldsymbol{C}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$. (Refer to "Comment on use".)
N	Input. The order n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.
ICON ...	Output. Condition codes. Refer to Table ASSM-1.

Table ASSM-1 Condition code

Code	Meaning	Processing
0	No error	
30000	n<1	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... none

- Notes
  Saving the storage area:
  When the contents of array A or B are not required save the area as follows;
  – When the contents of array A is not needed.

  CALL ASSM(A,B,A,N,ICON)

  – When the contents of array B is not needed.

  CALL ASSM(A,B,B,N,ICON)

  In these cases, matrix $\boldsymbol{C}$ is stored in array A or B.

- Example
  The following shows an example of obtaining the addition of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Here, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),C(5050)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
   10 READ(5,100) N
      IF(N.EQ.0) STOP
      WRITE(6,150)
      NT=N*(N+1)/2
      READ(5,200) (A(I),I=1,NT)
      READ(5,200) (B(I),I=1,NT)
      CALL ASSM(A,B,C,N,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PSM(IA,1,A,N)
      CALL PSM(IB,1,B,N)
      CALL PSM(IC,1,C,N)
      GOTO 10
  100 FORMAT(I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX ADDITION **')
      END
```

Subroutine PSM in the example is for printing the real symmetric matrix. This program is shown in the example for subroutine MGSM.

A25-31-0201 ASVD1, DASVD1

Singular value decomposition of a real matrix (Householder method, QR method)
CALL ASVD1(A,KA,M,N,ISW,SIG,U,KU,V,KV,VW, ICON)

Function

Singular value decomposition is performed for $m \times n$ real matrix $\boldsymbol{A}$ using the Householder method and the QR method.

$$\boldsymbol{A}=\boldsymbol{U\Sigma V}^{\mathrm{T}} \tag{1.1}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are matrices of $m \times l$ and $n \times l$ respectively, l=min(m,n).

When $l=n(m\geq n)$, $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{VV}^{\mathrm{T}}=\boldsymbol{I}_n$

When $l=m(m<n)$, $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{UU}^{\mathrm{T}}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{I}_m$

$\boldsymbol{\Sigma}$ is an $l \times l$ diagonal matrix expressed by $\boldsymbol{\Sigma}$=diag(σ_i),$\sigma_i\geq0$ and σ_i is a singular value of A. Singular values σ_i are the positive square root of the eigenvalues of matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and the i-th row of $\boldsymbol{V}$ is the eigenvector corresponding to the eigenvalue σ_i.

$m\leq1$, $n\leq1$.

For dimensions of matrices $\boldsymbol{A}$, $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$ see Fig. ASVD1-1.

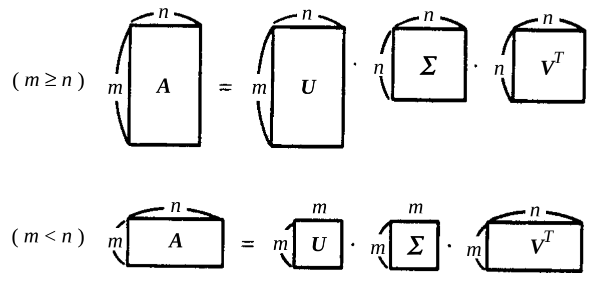


Fig. ASVD1-1 Relationship between dimension of the matrices in singular value decomposition

Parameters

A ..... Input. Matrix $\boldsymbol{A}$. Two-dimensional array A(KA,N). (See "Comments on use".)

KA ..... Input. Adjustable dimension (≥M) of array A.

M ..... Input. Number of rows in matrices $\boldsymbol{A}$ and $\boldsymbol{U}$.

N ..... Input. Number of columns in matrices $\boldsymbol{A}$ and $\boldsymbol{U}$, and number of rows of $\boldsymbol{V}$.

ISW ... Input. Control information.
ISW=$10d_1 + d_0$ with $0 \leq d_0$, $d_1 \leq 1$, specified as follows:
d_1=0 ... Matrix $\boldsymbol{U}$ is not obtained.
d_1=1 ... Matrix $\boldsymbol{U}$ is obtained.
d_0=0 ... Matrix $\boldsymbol{V}$ is not obtained.
d_0=1 ... Matrix $\boldsymbol{V}$ is obtained.

SIG .... Output. Singular values of matrix $\boldsymbol{A}$. One-dimensional array of size l+1. (See "Comments on use".)

U ..... Output. Matrix $\boldsymbol{U}$. Two-dimensional array U(KU,N). (See "Comments on use".)

KU ..... Input. Adjustable dimension (≥M) of array U.

V ..... Output. Matrix $\boldsymbol{V}$. Two-dimensional array V(KV,K), where K= min(M+1,N). (See "Comments on use".)

KV .... Input. Adjustable dimension (≥N) of array V.

VW ..... Work area. One-dimensional array of size n+1.

ICON ... Output. Condition code. See Table ASVD1-1.

Table ASVD1-1 Condition code

Code	Meaning	Processing
0	No error	
15000	Some singular values cannot be obtained.	Discontinued
30000	M<1,N<1,KA<M,KU<M,KV< N or ISW≠0,1,10,11.	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL
  FORTRAN basic functions ... MIN0, MOD, SIGN, SQRT, AMAX1, ABS

- Notes
  If you use decomposition factors $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$, from singular value decomposition, generalized inverse $\boldsymbol{A}^{+}$ of the original matrix $\boldsymbol{A}$ or least squares minimal norm solution of linear equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be obtained. (For details, see Section 3.5.)

  In this case, it is effective to use subroutine GINV or LAXLM.

  Although the singular value decomposition can be widely utilized (see Section 3.5), it requires a great amount of computation, this is a weak point.

  Therefore, $\boldsymbol{U}$ and $\boldsymbol{V}$ should be computed when they are required. When there is no need to compute them, since $\boldsymbol{U}$ and $\boldsymbol{V}$ are not referenced, the corresponding real arguments need not be two-dimensional array. ISW can control such requests.

  This subroutine allows rewriting of either $\boldsymbol{U}$ or $\boldsymbol{V}$ on A to reduce storage space. When $\boldsymbol{A}$ does not have to be saved, a real argument corresponding to $\boldsymbol{U}$ or $\boldsymbol{V}$ is written as a real argument and to reduce storage space.

  All singular values are non-negative and are stored in descending order. When ICON is set to 15000, only the non-negative values are singular values, and the rest are −1and are arranged randomly.

  The relationship of the number of columns m and the number of rows n of matrix $\boldsymbol{A}$ is not constrained in this subroutine. This subroutine can perform singular value decomposition for any types of $m \times n$ matrices $\boldsymbol{A}$'s under any of the following conditions:

- $m > n$
- $m = n$
- $m < n$

- Example
  Singular value decomposition is performed for $m \times n$ matrix $\boldsymbol{A}$. All singular values and the corresponding columns of $\boldsymbol{V}$ are output. However, $\boldsymbol{U}$ is not computed and $\boldsymbol{V}$ is computed on $\boldsymbol{A}$, where $1 \le n \le 100$ and $1 \le m \le 100$.
  Subroutine SVPRT in this example is used to print singular values and eigenvectors.

```
C     **EXAMPLE**
      DIMENSION A(100,100),SIG(100),VW(100)
   10 READ(5,500) M,N
      IF(M.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,M),J=1,N)
      WRITE(6,600) M,N
      DO 20 I=1,M
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL ASVD1(A,100,M,N,1,SIG,A,100,
     *           A,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.15000) GO TO 10
      CALL SVPRT(SIG,A,100,N,N)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',20X,'ORIGINAL MATRIX',
     *5X,'M=',I3,5X,'N=',I3/)
  610 FORMAT('0',4(5X,'A(',I3,',',I3,')=',
     *E14.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END

      SUBROUTINE SVPRT(SIG,V,K,N,M)
      DIMENSION SIG(N),V(K,M)
      WRITE(6,600)
      DO 20 INT=1,M,5
      LST=MIN0(INT+4,M)
      WRITE(6,610) (J,J=INT,LST)
      WRITE(6,620) (SIG(J),J=INT,LST)
      DO 10 I=1,N
   10 WRITE(6,630) I,(V(I,J),J=INT,LST)
   20 CONTINUE
      RETURN
  600 FORMAT('1',20X,
     *'SINGULAR VALUE AND EIGENVECTOR')
  610 FORMAT('0',10X,5I20)
  620 FORMAT('0',5X,'SV',3X,5E20.7/)
  630 FORMAT(5X,I3,3X,5E20.7)
      END
```

Method

The following singular value decomposition is performed for $m \times n$ matrix $\boldsymbol{A}$ by using the Householder and QR methods.

$$\boldsymbol{A}=\boldsymbol{U\Sigma V}^{\mathrm{T}} \tag{4.1}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $m \times l$ and $n \times l$ matrices respectively, l=min(m,n) and $\boldsymbol{\Sigma}$ is an $l \times l$ diagonal matrix expressed by $\boldsymbol{\Sigma}$=diag(σ_i),$\sigma_i \ge 0$, and

- when $l=n(m \ge n)$,$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{VV}^{\mathrm{T}}=\boldsymbol{I}_n$
- when $l=m(m<n)$,$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{UU}^{\mathrm{T}}=\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=I_m$

The value of σ_i is called the singular value of $\boldsymbol{A}$.
This subroutine does not constrain the size of m or n. It can decompose any type of matrices. Since $m \times n$ matrix $\boldsymbol{A}$, where $m \ge n$, is generally used, it is shown below.

- Computation procedures
  This subroutine performs singular value decomposition of (4.1) by the following two stages.
  (a) Reduction to the upper bidiagonal matrix (the Householder method)
  Operating two finite sequences $\boldsymbol{P}_1,...,\boldsymbol{P}_n$ and $\boldsymbol{Q}_1,...,\boldsymbol{Q}_{n-2}$ of Householder transformations from the right and left sides of matrix $\boldsymbol{A}$ alternatively

$$\boldsymbol{J}_0=\boldsymbol{P}_n \cdots \boldsymbol{P}_1\boldsymbol{AQ}_1 \cdots \boldsymbol{Q}_{n-2} \tag{4.2}$$

The matrix $\boldsymbol{A}$ is reduced to an upper bidiagonal matrix shown in Fig. ASVD1-2.

$$\boldsymbol{J}_0 \equiv \left[\begin{array}{ccccc} q_1 & l_2 & & & \\ & q_2 & l_3 & & 0 \\ & & \ddots & \ddots & \\ & 0 & & \ddots & l_n \\ & & & & q_n \\ \hdashline & & 0 & & \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \\ \}(m-n) \times n \end{array}$$

Fig. ASVD1-2 Structure of an upper bidiagonal matrix

Starting with $\boldsymbol{A}_1$=$\boldsymbol{A}$, the following are defined:

$$\boldsymbol{A}_{k+1/2} = \boldsymbol{P}_k\boldsymbol{A}_k,\ k = 1,...,n \tag{4.3}$$

$$\boldsymbol{A}_{k+1} = \boldsymbol{A}_{k+1/2}\boldsymbol{Q}_k, \quad k = 1,...,n-2 \tag{4.4}$$

Let $\boldsymbol{A}_k=(a_{ij}^{(k)})$, $\boldsymbol{P}_k$ should be such that

$$a_{ik}^{(k+1/2)} = 0 \quad , j = k+1,...,m$$

and $\boldsymbol{Q}_k$ should be such that

$$a_{kj}^{(k+1)} = 0 \quad , j = k+2,...,n$$

Therefore, $\boldsymbol{P}_k$ and $\boldsymbol{Q}_k$ can be chosen as follows:

$$\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{x}_k \boldsymbol{x}_k^{\mathrm{T}} / b_k , k = 1,...,n \tag{4.5}$$

$$\boldsymbol{Q}_k = \boldsymbol{I} - \boldsymbol{y}_k \boldsymbol{y}_k^{\mathrm{T}} / c_k , k = 1,...,n-2 \tag{4.6}$$

where,

$$\left.\begin{aligned}
\boldsymbol{x}_k &= \left(0,...,0, a_{k,k}^{(k)} + d_k , a_{k+1,k}^{(k)} ,..., a_{n,k}^{(k)}\right)^{\mathrm{T}} \\
d_k &= \left(\left(a_{k,k}^{(k)}\right)^2 + ... + \left(a_{n,k}^{(k)}\right)^2 \right)^{1/2} \mathrm{sign}\left(a_{k,k}^{(k)}\right) \\
b_k &= d_k^2 + d_k a_{k,k}^{(k)} \\
\boldsymbol{y}_k &= \left(0,...,0, a_{k,k+1}^{(k+1/2)} + e_k , a_{k,k+2}^{(k+1/2)} ,..., a_{k,n}^{(k+1/2)}\right)^{\mathrm{T}} \\
e_k &= \left(\left(a_{k,k+1}^{(k+1/2)}\right)^2 + ... + \left(a_{k,n}^{(k+1/2)}\right)^2 \right)^{1/2} \mathrm{sign}\left(a_{k,k+1}^{(k+1/2)}\right) \\
c_k &= e_k^2 + e_k a_{k,k+1}^{(k+1/2)}
\end{aligned}\right\} \tag{4.7}$$

(b) Reduction to the diagonal matrix (the QR method)
Choosing two orthogonal transformation sequences $\boldsymbol{S}_i$ and $\boldsymbol{T}_i$, for i=0,1,..., the matrix $\boldsymbol{J}_{i+1}$

$$\boldsymbol{J}_{i+1} = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{J}_i \boldsymbol{T}_i \tag{4.8}$$

can converge to the diagonal matrix $\boldsymbol{\Sigma}$.
Namely, matrix $\boldsymbol{\Sigma}$ can be repressed by using (4.9) for a certain integer q.

$$\Sigma = \boldsymbol{S}_q^{\mathrm{T}} \cdots \boldsymbol{S}_0^{\mathrm{T}} \boldsymbol{J}_0 \boldsymbol{T}_0 \cdots \boldsymbol{T}_q \tag{4.9}$$

(a) With these operations, matrix $\boldsymbol{A}$ will be transformed to a diagonal matrix and transformation matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are expressed as follows by using (4.1), (4.2) and (4.9)

$$\boldsymbol{U} = \boldsymbol{P}_1 \cdots \boldsymbol{P}_n \boldsymbol{S}_0 \cdots \boldsymbol{S}_q \tag{4.10}$$

$$\boldsymbol{V} = \boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2} \boldsymbol{T}_0 \cdots \boldsymbol{T}_q \tag{4.11}$$

These can be generated by multiplying the transformation matrices from the right and left sides alternatively.
In this case, matrix $\boldsymbol{T}_i$ in (4.8) must be defined such that symmetric tridiagonal matrix $\boldsymbol{M}_i = \boldsymbol{J}_i^{T} \boldsymbol{J}_i$ converges to a diagonal matrix while matrix $\boldsymbol{S}_i$ must be defined such that all $\boldsymbol{J}_i$ converges to bidiagonal matrices.

- How to select transformation matrices in QR method
This section describes how to select transformation matrices $\boldsymbol{T}_i$ and $\boldsymbol{S}_i$ in (4.8). For notation, use the following notation:

$$\boldsymbol{J} \equiv \boldsymbol{J}_i , \overline{\boldsymbol{J}} \equiv \boldsymbol{J}_{i+1} , \boldsymbol{S} \equiv \boldsymbol{S}_i , \boldsymbol{T} \equiv \boldsymbol{T}_i , \boldsymbol{M} \equiv \boldsymbol{J}^{\mathrm{T}} \boldsymbol{J} , \overline{\boldsymbol{M}} \equiv \overline{\boldsymbol{J}}^{\mathrm{T}} \boldsymbol{J}$$

$\boldsymbol{J}$ is transformed to $\overline{\boldsymbol{J}}$ by alternatively operating the two-dimensional Givens rotation from left and right sides. Thus,

$$\overline{\boldsymbol{J}} = \boldsymbol{L}_n^{\mathrm{T}} \boldsymbol{L}_{n-1}^{\mathrm{T}} \cdots \boldsymbol{L}_2^{\mathrm{T}} \boldsymbol{J} \boldsymbol{R}_2 \boldsymbol{R}_3 \cdots \boldsymbol{R}_n \tag{4.12}$$

where,

$$\boldsymbol{L}_{\mathrm{k}} = \begin{array}{c} \begin{array}{ccccccc} & & (k-1) & (k) & & & \end{array} \\ \left[\begin{array}{ccc|c|ccc} 1 & & 0 & & 0 & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ 0 & & & & & & \\ & & \cos\theta_{\mathrm{k}} & -\sin\theta_{\mathrm{k}} & \text{——} & & \\ & & \sin\theta_{\mathrm{k}} & \cos\theta_{\mathrm{k}} & \text{——} & & \\ & & & & 1 & 0 & \\ & & & & & \ddots & \\ & & 0 & & 0 & & 1 \end{array}\right] \begin{array}{c} \\ \\ \\ \\ (k-1) \\ (k) \\ \\ \\ \\ \end{array} \end{array}$$

and $\boldsymbol{R}_k$ is defined in the same way using ϕ_k instead of θ_k. Angle θ_k, k=2,...,n and angle ϕ_k, k=3,...,n can be defined so that $\overline{\boldsymbol{J}}$ can be an upper bidiagonal matrix for arbitrary ϕ_2.
Thus, the following steps should be taken with $\boldsymbol{J}$=(j_{ij}).
– $\boldsymbol{R}_2$ generates non-zero element j_{21}.
– $\boldsymbol{L}_2^{\mathrm{T}}$ generates non-zero element j_{13} by eliminating j_{21}.
– $\boldsymbol{R}_3$ generates non-zero element j_{32} by eliminating j_{13}.
– $\boldsymbol{R}_n$ generates non-zero element $j_{nn\text{-}1}$ by eliminating $j_{n\text{-}2n}$.
– $\boldsymbol{L}_n^{T}$ eliminates $j_{nn\text{-}1}$.
Figure ASVD1-3 shows these steps with n=5.

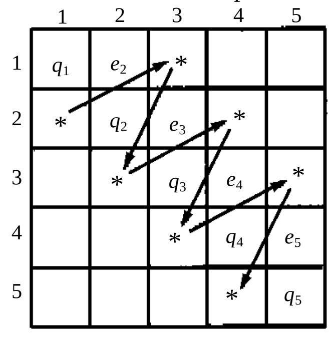


Fig. ASVD1-3 Behavior of non-zero elements(n=5)

Let

$$\boldsymbol{S} = \boldsymbol{L}_2 \boldsymbol{L}_3 \cdots \boldsymbol{L}_n \tag{4.13}$$

$$\boldsymbol{T} = \boldsymbol{R}_2 \boldsymbol{R}_3 \cdots \boldsymbol{R}_n \tag{4.14}$$

then from (4.12), the following equation holds.

$$\boldsymbol{J} = \boldsymbol{S}^{\mathrm{T}} \boldsymbol{J} \boldsymbol{T} \tag{4.15}$$

since $\boldsymbol{J}$ is an upper bidiagonal matrix, the $\overline{\boldsymbol{M}}$

$$\overline{\boldsymbol{M}} = \boldsymbol{J}^{\mathrm{T}} \overline{\boldsymbol{J}} = \boldsymbol{T}^{\mathrm{T}} \boldsymbol{M} \boldsymbol{T} \tag{4.16}$$

will be symmetric tridiagonal matrix such as $\boldsymbol{M}$.

The first rotating angle ϕ_2, that is, the first transformation $\boldsymbol{R}_2$ was not defined. ϕ_2 can be defined such that the transformation in (4.16) is a QR transformation, with an origin shift s.

Therefore, the following QR transformation can hold

$$\overline{\boldsymbol{M}}_s = \boldsymbol{T}_s^{\mathrm{T}} \boldsymbol{M} \boldsymbol{T}_s \tag{4.17}$$

where

$$\boldsymbol{M} - s\boldsymbol{I} = \boldsymbol{T}_s \boldsymbol{R}_s$$
$$\boldsymbol{R}_s \boldsymbol{T}_s + s\boldsymbol{I} = \overline{\boldsymbol{M}}_s$$
$$\boldsymbol{T}_s^{\mathrm{T}} \boldsymbol{T}_s = \boldsymbol{I}$$
$$\boldsymbol{R}_s : \text{upper triangular matrix}$$

To hold QR transformation, the first column of $\boldsymbol{R}_2$ should be proportioned to that of $\boldsymbol{M} - s\boldsymbol{I}$. As the usual QR method, the origin shift s may be defined as the eigenvalue at the smallest absolute value, in the two-dimensional submatrix in lower right of $\boldsymbol{M}$. Actually $\boldsymbol{T}_s$ and $\boldsymbol{T}$ are equal.

- Convergence criterion and direct sum in QR method
  Convergence is tested as follows:
  When

$$|e_n| \le \varepsilon_1 \tag{4.18}$$

is satisfied, decrease the order of matrices by 1 after adopting as a singular value $|q_n|$.
where,

$$\varepsilon_1 = \|\boldsymbol{J}_0\|_\infty u \tag{4.19}$$

and u is a unit round-off.
If the following holds for some k ($k \ne n$),

$$|e_k| \le \varepsilon_1 \tag{4.20}$$

the matrix is split into direct sum of two submatrices, which will be processed independently.

If the following holds for k ($k \ne 1$).

$$|q_k| \le \varepsilon_1 \tag{4.21}$$

operates the two-dimensional Givens rotation associated with the k-th column form the right side of $\boldsymbol{J}$ as follows:
$j_{k-2,k}$ and $j_{k,k-1}$ are generated when $e_k = j_{k-1,k}$ is eliminated.
$j_{k-3,k}$ and $j_{k,k-2}$ are generated when $j_{k-2,k}$ is eliminated.
....

Thus

$$\overline{\boldsymbol{J}} = \begin{array}{cc} \left[\begin{array}{cccccccc} \bar{q}_1 & \bar{e}_2 & & & & & & \\ & \ddots & \ddots & & & 0 & & \\ 0 & & & \bar{e}_{k-1} & & & & \\ & & & \bar{q}_{k-1} & 0 & & & \\ \delta_1 & \delta_2 & \cdots & \delta_{k-1} & \bar{q}_k & e_{k+1} & & \\ & & & & & \ddots & \ddots & \\ & & 0 & & & & & e_n \\ & & & & & & & q_n \end{array}\right] & \begin{array}{c} \\ \\ \\ \\ (k) \\ \\ \\ \\ \end{array} \\ \begin{array}{cccccccc} & & & (k-1) & (k) & & & \end{array} & \end{array}$$

However, the following will be held by orthogonality of transformation matrix.

$$\delta_1^2 + \delta_2^2 + \cdots + \delta_{k-1}^2 + \bar{q}_k^2 = q_k^2 \le \varepsilon_1^2 \tag{4.22}$$

All absolute values of $\delta_1, \delta_2, \ldots, \delta_{k-1}$ are less than ε_1 then matrix $\overline{\boldsymbol{J}}$ can be split into two submatrices.
Every singular value is obtained within 30 times iteration of QR method. Otherwise, the processing is terminated and unobtained singular values are defined -1.

For details, see Reference [11].

A52-31-0302 BDLX, DBDLX

A system of linear equations with a positive-definite symmetric band matrix decomposed into the factors L, D and L^T
CALL BDLX(B,FA,N,NH,ICON)

Function

This subroutine solves a system of linear equations.

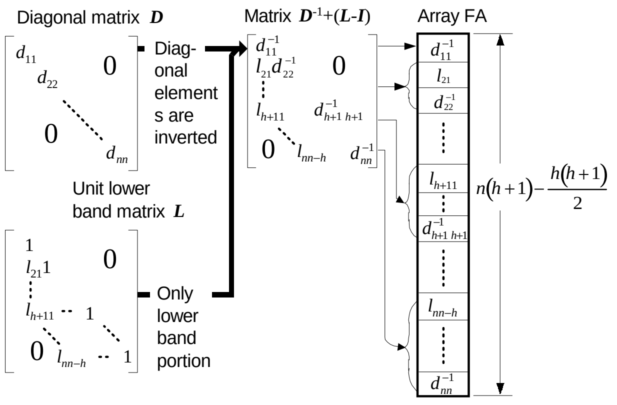
$$\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{b} \tag{1.1}$$

Where $\boldsymbol{L}$ is an $n \times n$ unit lower band matrix with band width h, $\boldsymbol{D}$ is an $n \times n$ diagonal matrix, $\boldsymbol{b}$ is an n-dimensional real constant vector, $\boldsymbol{x}$ is an n-dimensional solution vector, and $n>h\geq0$.

Parameters

B ..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One dimensional array with size n.

FA .... Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$. See Fig. BDLX-1.
FA is a one-dimensional array of size $n(h+1)-h(h+1)/2$ to contain $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$ in the compressed mode for symmetric band matrices.



Note:
The diagonal and lower band portions of the matrix D^{-1}+(L-I) are contained in one-dimensional array FA in the compressed mode for a symmetric band matrix.

Fig. BDLX-1 How to contain matrices L and D^{-1}

N .... Input. Order n of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$.

NH ...... Input. Lower band width h.

ICON .. Output. Condition code. Refer to Table BDLX-1

Table BDLX-1 Condition code

Code	Meaning	Processing
0	No error	
10000	The coefficient matrix was not positive-definite.	Continued
30000	NH<0 or NH≥N	By passed

Comments on use

- Subprograms used
  SSL II .... MGSSL
  FORTRAN basic functions ... None

- Notes
  This subroutine omits the operations concerning the elements out of the band so that the processing speed is faster than subroutine LDLX provided for a positive-definite symmetric matrix.
  Note that in this subroutine the decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$ contained in the compressed mode for symmetric band matrix are required. A system of linear equations can be solved by calling this subroutine following the subroutine SBDL. However, subroutine LSBX can be usually called to solved such equations in one step.

- Example
  A system of linear equations is solved after first LDL$^{\mathrm{T}}$ decomposition of $n \times n$ coefficient matrix with band width h using subroutine SBDL. $n\leq100$ and $h\leq50$.

```
C     **EXAMPLE**
      DIMENSION A(3825),B(100)
   10 READ(5,500)N,NH
      IF(N.EQ.0)STOP
      NH1=NH+1
      NT=N*NH1-NH*NH1/2
      READ(5,510)(A(I),I=1,NT)
      WRITE(6,640)
      L=0
      LS=1
      DO 20 I=1,N
      L=L+MIN0(I,NH1)
      JS=MAX0(1,I-NH)
      WRITE(6,600)I,JS,(A(J),J=LS,L)
   20 LS=L+1
      CALL SBDL(A,N,NH,1.0E-6,ICON)
      WRITE(6,610)ICON
      IF(ICON.GE.20000)STOP
      READ(5,510)(B(I),I=1,N)
      CALL BDLX(B,A,N,NH,ICON)
      WRITE(6,610)ICON
      DET=A(1)
      L=1
      DO 30 I=2,N
      L=L+MIN0(I,NH1)
   30 DET=DET*A(L)
      DET=1.0/DET
      WRITE(6,620)(B(I),I=1,N)
      WRITE(6,630)DET
      GO TO 10
```

```
500 FORMAT(2I5)
510 FORMAT(4E15.7)
600 FORMAT(' ','(',I3,',',I3,')'
   */(10X,5E17.8))
610 FORMAT(/10X,'ICON=',I5)
620 FORMAT(/10X,'SOLUTION VECTOR'
   *//(10X,5E17.8))
630 FORMAT(/10X,
   *'DETERMINANT OF COEFFICIENT MATRIX=',
   *E17.8)
640 FORMAT(/10X,'INPUT MATRIX')
    END
```

Method

Solving a system of linear equations (4.1)

$$\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{b} \tag{4.1}$$

is equivalent to solving the following equations (4.2) and (4.3).

$$\boldsymbol{Ly}=\boldsymbol{b} \tag{4.2}$$

$$\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{D}^{-1}\boldsymbol{y} \tag{4.3}$$

- Solving $\boldsymbol{Ly}=\boldsymbol{b}$ (Forward substitution)
  $\boldsymbol{Ly}=\boldsymbol{b}$ can be serially solved using equations (4.4).

$$y_i=b_i-\sum_{k=\max(1,i-h)}^{i-1} l_{ik}y_k \quad ,i=1,...,n \tag{4.4}$$

Where $\boldsymbol{L}=(l_{ij})$, $\boldsymbol{y}^{\mathrm{T}}=(y_1,...,y_n)$, $\boldsymbol{b}^{\mathrm{T}}=(b_1,...,b_n)$

- Solving $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{D}^{-1}\boldsymbol{y}$ (Backward substitution)
  $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{D}^{-1}\boldsymbol{y}$ can be serially solved using equations (4.5).

$$x_i=y_id_i^{-1}\sum_{k=i+1}^{\min(i+h,n)} l_{ki}x_k, \quad i=n,...,1 \tag{4.5}$$

Where $\boldsymbol{D}^{-1}=\mathrm{diag}(d_i^{-1})$, $\boldsymbol{x}^{\mathrm{T}}=(x_1, ..., x_n)$.
For further details, see Reference [7].

E12-32-1102 BICD1, DBICD1

B-spline two-dimensional interpolation coefficient calculation (I-I).
CALL BICD1(X,NX,Y,NY,FXY,K,M,C,VW,ICON)

Function

Given function values $f_{ij}=f(x_i,y_j)$ at point (x_i,y_j), $(x_1<x_2<...<x_{nx}, y_1<y_2<...<y_{ny})$, on the xy plane, as well as partial derivatives $f_{i,j}^{(\lambda,\mu)}$, i=1,n_x, j=1, n_y, λ=1, 2, ..., $(m-1)/2$, μ=1,2, ..., $(m-1)/2$ at the boundary points, the interpolation coefficients $c_{\alpha,\beta}$'s in the dual m-th (m:odd integer) degree B-spline two-dimensional interpolating function

$$s(x,y)=\sum_{\beta=-m+1}^{n_y-1}\sum_{\alpha=-m+1}^{n_x-1}c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (1.1)$$

are obtained with restriction $m\geq3$, $n_x\geq2$ and $n_y\geq2$.

For later use, we introduce below a notation $\hat{f}_{i,j}$ with $l=(m-1)/2$ for convenience. And a matrix consisting of $\hat{f}_{i,j}$ as elements is shown in Fig. BICD1-1.

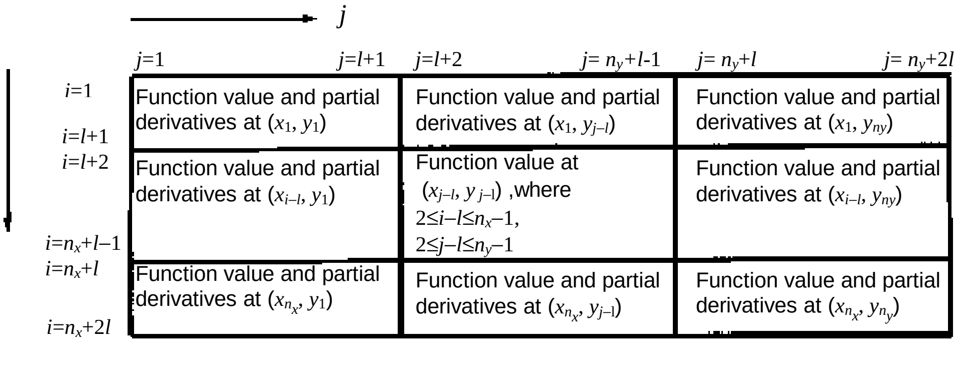
$$\begin{aligned}
\hat{f}_{i,j} &= f_{11}^{(l-i+1,l-j+1)} && : i = 1,2,...,l+1\\
& && : j = 1,2,...,l+1\\
\hat{f}_{i,j} &= f_{i-l,1}^{(0,l-j+1)} && : i = l+2,l+3,...,n_x+l-1\\
& && : j = 1,2,...,l+1\\
\hat{f}_{i,j} &= f_{n_x,1}^{(i-n_x-l,l-j+1)} && : i = n_x+l,n_x+l+1,...,n_x+2l\\
& && : j = 1,2,...,l+1\\
\hat{f}_{i,j} &= f_{1,j-l}^{(l-i+1,0)} && : i = 1,2,...,l+1\\
& && : j = l+2,l+3,...,n_y+l-1\\
\hat{f}_{i,j} &= f_{i-l,j-l} && : i = l+2,l+3,...,n_x+l-1\\
& && : j = l+2,l+3,...,n_y+l-1\\
\hat{f}_{i,j} &= f_{n_x,j-l}^{(i-n_x-l,0)} && : i = n_x+l,n_x+l+1,...,n_x+2l\\
& && : j = l+2,l+3,...,n_y+l-1\\
\hat{f}_{i,j} &= f_{1,n_y}^{(l-i+1,j-n_y-l)} && : i = 1,2,...,l+1\\
& && : j = n_y+l,n_y+l+1,...,n_y+2l\\
\hat{f}_{i,j} &= f_{i-l,n_y}^{(0,j-n_y-l)} && : i = l+2,l+3,...,n_x+l-1\\
& && : j = n_y+l,n_y+l+1,...,n_y+2l\\
\hat{f}_{i,j} &= f_{n_x,n_y}^{(i-n_x-l,j-n_y-l)} && : i = n_x+l,n_x+l+1,...,n_x+2l\\
& && : j = n_y+l,n_y+l+1,...,n_y+2l
\end{aligned}$$

Parameters

X	Input Discrete points x_j's in the x-direction.
NX	Input. Number of the x_i's.
Y	Input. Discrete points y_j's in the y-direction. One-dimensional array of size n_y.
NY	Input. Number of the y_i's.
FXY ...	Input. Function values and partial derivatives, $\hat{f}_{i,j}$'s. Two-dimensional array as FXY (K,NY+M−1). FXY (I,J) is assigned $\hat{f}_{i,j}$. See Fig. BICD1−1.
K	Input. Adjustable dimension for arrays FXY and C(K≥NX+M−1).
M	Input. Degree of the B-spline. See Note.
C	Output. Interpolation coefficients, $c_{\alpha\beta}$'s. Two-dimensional array as C(K,NY+M−1) $C_{-m+i,-m+j}$ is put out in C(I,J).
VW	Work area. One-dimensional array of size. $\{\max(n_x,n_y)+1\}(m+2)-3+(m+1)^2/2$.
ICON ..	Output. Condition code. See Table BICD1-1.

Comments on use

- Subprograms used
  SSL II ... MGSSL, UMIO1, UCIO1, UBAS1 and ULUI1
  FORTRAN basic functions ... MOD and FLOAT



j
i
j=1
j=l+1
j=l+2
j= n_y+l-1
j= n_y+l
j= n_y+2l
i=1
i=l+1
i=l+2
i=n_x+l−1
i=n_x+l
i=n_x+2l
Function value and partial derivatives at (x_1, y_1)
Function value and partial derivatives at (x_1, y_{j-l})
Function value and partial derivatives at (x_1, y_{ny})
Function value and partial derivatives at (x_{i-l}, y_1)
Function value at (x_{j-l}, y_{j-l}) ,where $2\leq i-l\leq n_x-1$, $2\leq j-l\leq n_y-1$
Function value and partial derivatives at (x_{i-l}, y_{ny})
Function value and partial derivatives at (x_{n_x}, y_1)
Function value and partial derivatives at (x_{n_x}, y_{j-l})
Function value and partial derivatives at (x_{n_x}, y_{n_y})

Fig. BICD1-1 Function value and derivatives $\hat{f}_{i,j}$'s ($l = (m-1)/2$)

Table BICD1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the followings occurred. (a) M is not an odd number. (b) x_i which satisfies $x_i \geq x_{i+1}$ exists. (c) y_j which satisfies $y_j \geq y_{j+1}$ exists. (d) M<3 (e) NX<2 or NY<2	Bypassed

- Notes

  By calling the subroutine BIFD1 after subroutine BICD1, the interpolated values based on the B-spline interpolating function (1.1), as well as derivatives and/or integrals can be obtained. The parameter values of X, NY, Y, NY, K, M and C are passed from BICD1 to input to BIFD1.

  The degree *m* is preferably 3 or 5. In double precision, if the original function is smooth and f_{ij}'s are given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example

  See the example given for subroutine BIFD1.

Method

The B-spline two-dimensional interpolating function S(*x*,*y*) to be obtained here is a direct extension of the B-spline interpolating function (I) obtained by subroutine BIC1. In other words, S(*x*,*y*) is defined in the region $R = \left\{(x,y) \middle| x_1 \leq x < x_{nx}, y_x \leq y \leq y_{ny}\right\}$ and satisfies the following conditions:

(a) The S(*x*,*y*) is a dual *m*-th degree, at most, polynomial in each of the partial region

$$R_{i,j} = \left\{(x,y) \middle| x_i \leq x < x_{i+1}, y_j \leq y < y_{j+1}\right\}$$

The dual *m*-th degree means that the function is of degree *m* with respect to both *x* and *y*.

(b) $S(x,y) \in C^{m-1,m-1}[\boldsymbol{R}]$ i.e., for λ=0, 1, ..., $C^{m\text{-}1,\, m\text{-}1}$, and μ = 0, 1, ... , *m*–1,

$$\frac{\partial^{\lambda+\mu}}{\partial x^{\lambda} \partial y^{\mu}} S(x,y)$$

exists and also is continuous.

(c) $S(x_i, y_i) = f_{i,j}, i = 1,2,...,n_x, j = 1,2,...,n_y$

$$S^{(\lambda,\mu)}(x_i, y_j) = f_{i,j}^{(\lambda,\mu)}, i = 1, n_x, j = 1, n_y$$
$$\lambda = 1,2,...,(m-1)/2, \mu = 1,2,...,(m-1)/2$$

The *S*(*x*,*y*) which satisfies (a) and (b) above can be represented as

$$S(x,y) = \sum_{\beta=-m+1}^{n_y-1} \sum_{\alpha=-m+1}^{n_x-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \qquad (4.1)$$

with $c_{\alpha\beta}$'s being arbitrary constants. The $N_{\alpha m+1}(x)$, $N_{\beta m+1}(y)$ are both *m*-th degree B-splines and represented respectively by

$$N_{\alpha,m+1}(x) = (s_{\alpha+m+1} - s_{\alpha}) g_{m+1}[s_{\alpha}, s_{\alpha+1}, ..., s_{\alpha+m+1}; x] \qquad (4.2)$$

$$N_{\beta,m+1}(y) = (t_{\beta+m+1} - t_{\beta}) g_{m+1}[t_{\beta}, t_{\beta+1}, ..., t_{\beta+m+1}; y] \qquad (4.3)$$

where sequences of $\{s_i\}$ and $\{t_j\}$ are the same as for the one-dimensional B-spline interpolating function (I).

Let's define $\hat{N}_{\alpha,i}$ and $\hat{N}_{\beta,i}$ as follows:

$$\hat{N}_{\alpha,i} = \begin{cases} N_{\alpha,m+1}^{(l-i+1)}(x_1) & : i = 1,2,...,l+1 \\ N_{\alpha,m+1}(x_{i-l}) & : i = l+2, l+3, ..., n_x + l - 1 \\ N_{\alpha,m+1}^{(i-n_x-l)}(x_{n_x}) & : i = n_x + l, n_x + l + 1, ..., n_x + 2l \end{cases}$$

$$\hat{N}_{\beta,j} = \begin{cases} N_{\beta,m+1}^{(l-j+1)}(y_1) & : j = 1,2,...,l+1 \\ N_{\beta,m+1}(y_{j-l}) & : j = l+2, l+3, ..., n_y + l - 1 \\ N_{\beta,m+1}^{(j-n_y-l)}(y_{n_y}) & : j = n_y + l, n_y + l + 1, ..., n_y + 2l \end{cases}$$

Then the coefficients in (4.1) can be uniquely determined by the interpolation condition (c). By using (4.1), the condition (c) can be stated as

$$\sum_{\beta=-m+1}^{n_y-1} \left\{ \sum_{\alpha=-m+1}^{n_x-1} c_{\alpha,\beta} \hat{N}_{\alpha,i} \right\} \hat{N}_{\beta,j} = \hat{f}_{i,j} \qquad (4.4)$$

and this can be further rewritten to a simpler form by defining several matrices as follows:

- $\boldsymbol{F}$ is an (n_x+*m* – 1) by (n_y+*m* – 1) matrix with the elements $\hat{f}_{i,j}$'s
- $\boldsymbol{C}$ is an (n_x+*m* – 1) by (n_y+*m* – 1) matrix with the elements $c_{\alpha,\beta}$'s
- $\boldsymbol{\Phi}$ is an (n_x+*m* – 1) by (n_x+*m* – 1) matrix whose *i*-th row consists of $\hat{N}_{\alpha,i}$'s, where $\alpha = -m+1, -m+2, ...,$ $n_x - 1$
- $\boldsymbol{\Psi}$ is an (n_y+*m*–1) by (n_y+*m* – 1) matrix whose *j*-th row consists of $\hat{N}_{\beta,j}$'s, where $\beta = -m+1, -m+2, ..., n_y-1$

By using these matrices, Eq.(4.4) can be rewritten to

$$\boldsymbol{\Phi C \Psi}^{\mathrm{T}} = \boldsymbol{F} \qquad (4.5)$$

Objective matrix $\boldsymbol{C}$ can be solved as follows. First, consider

$$\boldsymbol{\Psi X} = \boldsymbol{F}^{\mathrm{T}} \qquad (4.6)$$

as (n_x+m-1) systems of linear equations of order (n_y+m-1) and then we can solve them for matrix $\boldsymbol{X}$. Next, considering

$$\boldsymbol{\Phi C} = \boldsymbol{X}^{\mathrm{T}} \tag{4.7}$$

as (n_y+m-1) systems of linear equations of order (n_x+m-1) then they can be solved for matrix $\boldsymbol{C}$. Matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are of exactly the same form as the coefficient matrices in the linear equations when obtaining the one-dimensional B-spline interpolating function (I). See the explanation for subroutine BIC1.

This subroutine solves the linear equations, (4.6) and (4.7), by using Crout method (LU decomposition method) in the slave subroutines UMIO1, UCIO1 and ULUI1.

E12-32-3302 BICD3, DBICD3

B-spline two-dimensional interpolation coefficient calculation (III-III)
CALL BICD3(X,NX,Y,NY,FXY,K,M,C,XT,VW,ICON)

Function

Given function values $f_{ij}=f(x_i,y_j)$ at points (x_i,y_j) $(x_1<x_2<...<x_{nx}, y_1<y_2<...<y_{ny})$, on the xy-plain, the interpolation coefficients $c_{\alpha,\beta}$'s of dual m-th degree (m odd integer) B-spline two-dimensional interpolationg function.

$$S(x,y)=\sum_{\beta=-m+1}^{n_y-m}\sum_{\alpha=-m+1}^{n_x-m}c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (1.1)$$

are obtained. The knots of $S(x,y)$ are as shown in (1,2) as for x-direction, and as shown in (1.3) as for y-direction. (See Fig. BICD3-1.)

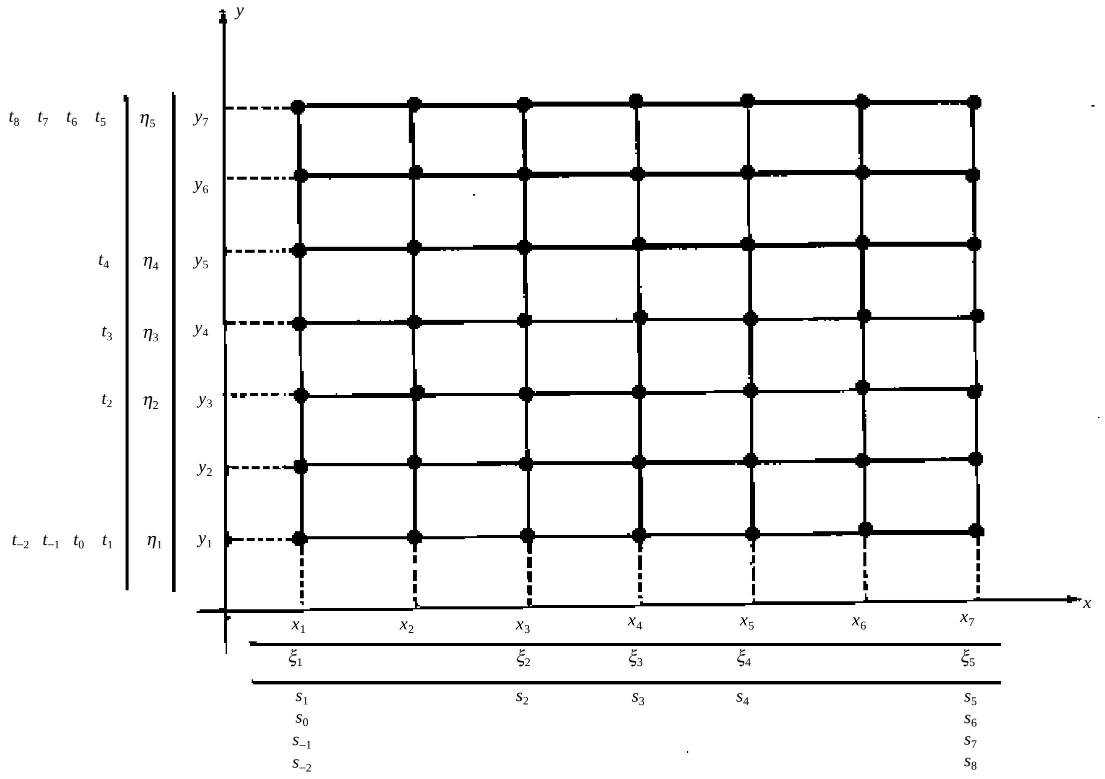
$$\xi_i=\begin{cases}x_1 & ,\ i=1\\ x_{i+(m-1)/2} & ,\ i=2,3,...,n_x-m\\ x_{n_x} & ,\ i=n_x-m+1\end{cases} \qquad (1.2)$$

$$\eta_j=\begin{cases}y_1 & ,\ j=1\\ y_{j+(m-1)/2} & ,\ j=2,3,...,n_y-m\\ y_n & ,\ j=n_y-m+1\end{cases} \qquad (1.3)$$

Here $m\geq3$, $n_x\geq m+2$ and $n_y\geq m+2$

Parameters

X	Input. Discrete points x_i's in the x-direction. One-dimensional array of size n_x.
NX	Input. Number of the x_i's, n_x.
Y	Input. Discrete points y_j's, in the y-direction. One-dimensional array of size n_y
NY	Input. Number of the y_j's, n_y
FXY ...	Input. Function values f_{ij}. Two-dimensional array as FXY(K,NY). FXY(I,J) is to be assigned f_{ij}
K	Input. Adjustable dimension for arrays FXY and C(K≥NX).
M	Input. Degree of the B-spline, m. See Note.
C	Output. Interpolation coefficients $c_{\alpha,\beta}$. Two-dimensional array as C(K,NY). $c_{-m+i,-m+i}$ is put out to C(I,J).
XT ...	Output. Knots $\{\xi_i\}$ and $\{\eta_j\}$. One dimensional array of size $(n_x-m+1)+(n_y-m+1)$. $\{\xi_i\}$ is put out first followed by $\{\eta_j\}$.
VW	Work area. One dimensional array of the following size: $\{\max(n_x,n_y)-2\}m+2(m+1)+2\max(n_x,n_y)$.
ICON ..	Output. Condition code. See Table BICD3-1.



y
t_8 t_7 t_6 t_5 η_5 y_7
y_6
t_4 η_4 y_5
t_3 η_3 y_4
t_2 η_2 y_3
y_2
t_{-2} t_{-1} t_0 t_1 η_1 y_1
x
x_1 x_2 x_3 x_4 x_5 x_6 x_7
ξ_1 ξ_2 ξ_3 ξ_4 ξ_5
s_1 s_2 s_3 s_4 s_5
s_0 s_6
s_{-1} s_7
s_{-2} s_8

Fig. BICD3-1 Knots $\{\xi_i\}$ and $\{\eta_j\}$ (for $n_x=n_y=7$ and $m=3$)

Table BICD3-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	One of the followings occurred: (a) M is not an odd integer. (b) NX<M+2 or NY<M+2 (c) x_i which satisfies $x_i \ge x_{i+1}$ exists. (d) y_j which satisfies $y_j \ge y_{j+1}$ exists. (e) M<3	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UMIO3, UCIO3, UBAS1 and ULUI3
  FORTRAN basic functions ... MOD and FLOAT

- Notes
  By calling the subroutine BIFD3 after subroutine BICD3, the interpolated values based on the B-spline interpolating function (1.1), as well as derivatives and/or integrals can be obtained. The parameter values of X, NX, Y, NY, K, M, C and XT are passed from BICD3 to input to BIFD3.
  The degree m is preferably 3 or 5. In double precision, if the original function is smooth and f_{ij}'s are given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example
  See the example given for subroutine BIFD3.

Method

The B-spline two-dimensional interpolating function $S(x,y)$ to be obtained here is a direct extension of the B-spline interpolating function (III) obtained by subroutine BIC3. In other words, taking knots $\{\xi_i\}$ and $\{\eta_j\}$ in the x- and y-directions as shown in (1.2) and (1.3), $S(x,y)$ is defined in the region $R=\{(x,y) \mid x_1 \le x \le x_{nx},\ y_1 \le y \le y_{ny}\}$and satisfies the following conditions:

(a) The $S(x,y)$ is a dual m-th degree, at most, polynomial in each of the partial region $R_{i,j}$= $\{(x,y) | \xi_i \le x \le \xi_{i+1},$ $\eta_j \le y \le \eta_{j+1}\}$. The dual m-th degree means that the function is of degree m with respect to both x and y.

(b) $S(x,y) \in C^{m-1,m-1}$ [R] i.e., for λ = 0, 1, ..., $m-1$, and $\mu = 0,1,...,m-1, \dfrac{\partial^{\lambda+\mu}}{\partial x^{\lambda} \partial y^{\mu}} S(x,y)$ exists and also is continuous.

(c) $S(x_i,y_j)=f_{ij}$, $i=1,2,...,n_x$, $j=1,2,...,n_y$.

The $S(x,y)$ which satisfies (a) and (b) above, can be represented as

$$S(x,y)=\sum_{\beta=-m+1}^{n_y-m}\sum_{\alpha=-m+1}^{n_x-m} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \qquad (4.1)$$

with the $c_{\alpha,\beta}$'s being arbitrary constants. The $N_{\alpha,m+1}(x)$ and $N_{\beta,m+1}(y)$ are both the m-th degree B-splines and represented respectively by

$$N_{\alpha,m+1}(x) = (s_{\alpha+m+1} - s_{\alpha}) g_{m+1}[s_\alpha, s_{\alpha+1},...,s_{\alpha+m+1}; x] \qquad (4.2)$$

$$N_{\beta,m+1}(y) = (t_{\beta+m+1} - t_{\beta}) g_{m+1}[t_\beta, t_{\beta+1},...,t_{\beta+m+1}; y] \qquad (4.3)$$

where knots $\{s_i\}$ and $\{t_j\}$ are the same as for the one-dimensional B-spline interpolating function (III). An example is shown in Fig. BICD3-1.

The coefficients in (4.1) can be uniquely determined by the interpolation condition (c). By using (4.1), condition (c) canbe stated as

$$\sum_{\beta=-m+1}^{n_y-m}\left\{\sum_{\alpha=-m+1}^{n_x-m} c_{\alpha,\beta} N_{\alpha,m+1}(x_i)\right\} N_{\beta,m+1}(y_j) = f_{ij} \qquad (4.4)$$
$$, i=1,2,...,n_x,\ j=1,2,...,n_y$$

and can be further rewritten to a simpler form by defining several matrices as follows:

- $\boldsymbol{F}$ is an $n_x \times n_y$ matrix with the elements f_{ij}
- $\boldsymbol{C}$ is an $n_x \times n_y$ matrix with the elements $c_{\alpha,\beta}$
- $\boldsymbol{\Phi}$ is an $n_x \times n_x$ matrix whose i-th row consists of $N_{\alpha,m+1}(x_i)$, where $\alpha = -m+1,..., n_x - m$
- $\boldsymbol{\Psi}$ is an $n_y \times n_y$ matrix whose j-th row consists of $N_{\beta,m+1}(y_j)$, where $\beta = -m+1,..., n_y - m$

By using these matrices above, Eq.(4.4) can be rewritten as

$$\boldsymbol{\Phi C \Psi}^{\mathrm{T}} = \boldsymbol{F} \qquad (4.5)$$

Objective matrix $\boldsymbol{C}$ can be solved as follows. First, consider

$$\boldsymbol{\Psi X} = \boldsymbol{F}^{\mathrm{T}} \qquad (4.6)$$

as n_x systems of linear equations of order n_y and then we can solve them for the matrix $\boldsymbol{X}$. Next, considering

$$\boldsymbol{\Phi C} = \boldsymbol{X}^{\mathrm{T}} \qquad (4.7)$$

as n_y systems of linear equations of order n_x then they can be solved for matrix $\boldsymbol{C}$. The matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are exactly of the same form as the coefficient matrix in the linear equations when obtaining the one-dimensional B-spline interpolating function (III). (See the explanation for subroutine BIC3.)
This subroutine solves the linear equations, (4.6) and (4.7), by using Crout method (LU decomposition method) in the slave subroutines UMIO3, UCIO3 and ULUI3.

E12-31-0102 BIC1, DBIC1

B-spline interpolation coefficient calculation (I)
CALL BIC1(X,Y,DY,N,M,C,VW,ICON)

Function

Given function values $y_i = f(x_i)$, i=1, 2, ..., n, at the discrete points $x_1, x_2, ..., x_n (x_1 < x_2 < ... < x_n)$, as well as derivatives $y_l^{(1)} = f(x_1)$ and $y_n^{(1)} = f(x_n)$, l=1, 2, ..., $(m-1)/2$, at both end points x_1 and x_n, the interpolation coefficients, c_j's, $j= -m+1, -m+2, ..., n-1$, in the interpolating function represented as a linear combination of B-splines of degree m(odd integer),

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{1.1}$$

are obtained.
The obtained $S(x)$ satisfies

$$\begin{cases} S^{(l)}(x_1)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}^{(l)}(x_1)=y_1^{(l)} \\ l=0,1,...,(m-1)/2 \\ S(x_i)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x_i)=y_i \\ i=2,3,...,n-1 \\ S^{(l)}(x_n)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}^{(l)}(x_n)=y_n^{(l)} \end{cases} \tag{1.2}$$
$$l=(m-1)/2,(m-1)/2-1,...,0$$

Here m≥3 and n≥2.

Parameters

X	Input. Discreate points, x_i's. One-dimensional array of size n.
Y	Input. Function values, y_i's. One-dimensional array of size n.
DY	Input. Derivatives at end points x_1 and x_n. Two-dimensional array of DY(2,$(m-1)/2$). DY(1,l) and DY(2, l) are assigned $y_1^{(l)}$ and $y_n^{(l)}$, respectively for l=1,2,...,$(m-1)/2$.
N	Input. Number of the discrete points, n.
M	Input. Degree of the B-spline, m See Note.
C	Output. Interpolating coefficients c_j's. One-dimensional array of size $n+m-1$.
VW	Work area. One-dimensional array of size $(n-2)m+(m+1)^2/2+(m+1)$.
ICON ..	Output. Condition code. See Table BIC1-1.

Table BIC1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the followings occurred: 1 M is not an odd integer. 2 x_i which satisfies $x_i \geq x_{i+1}$ exists. 3 M<3. 4 N<2.	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UMIO1, UCIO1, UBAS1 and ULUI1
  FORTRAN basic functions ... MOD and FLOAT

- Notes
  By calling the subroutine BIF1 after subroutine BIC1, the interpolated values based on the B-spline interpolating function (1.1), as well as derivatives and/or integrals can be obtained. The parameter values of X, N, M and C are passed from BIC1 to input to BIF1.
  The degree m is preferably 3 or 5. In double precision, if the original function is smooth and y_i's are given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example
  See the example given for subroutine BIF1.

Method

The m-th degree B-spline interpolating function $S(x)$ to be obtained here is a function defined in the interval $[x_1,x_n]$ and satisfying the following conditions:

(a) $S(x)$ is polynomial at most of degree m in the subinterval $[x_i, x_{i+1})$, i=1, 2, ..., $n-1$.
(b) $S(x) \in C^{m-1}[x_1,x_n]$,i.e., $S(x)$ and its derivatives of up to order $(m-1)$ are conditnous in the interval $[x_1,x_n]$.
(c) $S(x_i)=y_n$, i=2,3,...,$n-1$
(d) $S^{(l)}(x_1) = y_1^{(l)}$,
$S^{(l)}(x_n) = y_n^{(l)}$, l=0, 1, ..., $(m-1)/2$

A m-th degree spline function defined with knots $\{t_j\}$, $j = -m+1,-m+2,...,n+m$, taken as

$$t_j=\begin{cases} x_1 : j=-m+1,-m+2,...,1 \\ x_j : j=2,3,...,n-1 \\ x_n : j=n,n+1,...,n+m \end{cases} \tag{4.1}$$

can be represented as

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{4.2}$$

where, $N_{j,m+1}(x)$ is the m-th degree B-spline and given by

$$N_{j,m+1}(x) = (t_{j+m+1} - t_j)g_{m+1}[t_j, t_{j+1}, ..., t_{j+m+1}; x]$$

(For details, see Chapter 7, 7.1)

Considering locality of $N_{j,m+1}(x)$,

$$N_{j,m+1}(x_i) = \begin{cases} > 0: \; j = i-m, i-m+1, ..., i-1 \\ = 0: \begin{cases} j = -m+1, -m+2, ..., i-m-1 \\ j = i, i+1, ..., n-1 \end{cases} \end{cases}$$

$$N^{(l)}_{j,m+1}(x_1) = \begin{cases} \neq 0: \; j = -m+1, -m+2, ..., -m+1+l \\ = 0: \; j = -m+2+l, -m+3+l, ..., n-1 \end{cases}$$

$$N^{(l)}_{j,m+1}(x_n) = \begin{cases} = 0: \; j = -m+1, -m+2, ..., n-2-l \\ \neq 0: \; j = n-1-l, n-l, ..., n-1 \end{cases} \tag{4.3}$$

and applying the interpolation conditions (c) and (d) described above to Eq. (4.2), a system of $(n+m-1)$ linear equations

$$\begin{cases} \displaystyle\sum_{j=-m+1}^{-m+1+l} c_j N^{(l)}_{j,m+1}(x_1) = y^{(l)}_1 \; : \; l = 0, 1, ..., (m-1)/2 \\ \displaystyle\sum_{j=i-m}^{i-1} c_j N_{j,m+1}(x_i) = y_i \; : \; i = 2, 3, ..., n-1 \\ \displaystyle\sum_{j=n-1-l}^{n-1} c_j N^{(l)}_{j,m+1}(x_n) = y^{(l)}_n \; : \; l = (m-1)/2, (m-1) \\ \qquad\qquad\qquad\qquad /2-1, ..., 0 \end{cases} \tag{4.4}$$

are given with c_j's; $j = -m+1, -m+2, ..., n-1$, unknown.

By solving these equations all of the interpolation coefficients c_j's can be obtained.

The form of the coefficient matrix in the linear equations (4.4) is shown in Fig. BIC1-1 as an example for m=5 and n=8.

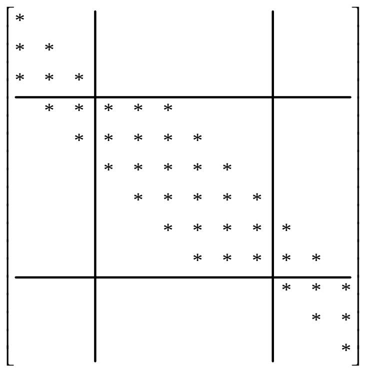


Fig. BIC1-1 Coefficient matrix (in the case of m=5 and n=8)

The subroutine solves the linear equations by using the Crout method (LU decomposition).
Subroutines UMIO1, ULUI1 and UCIO1 are called.

E12-31-0202 BIC2, DBIC2

B-spline interpolation coefficient calculation II
CALL BIC2(X,Y,DY,N,M,C,VW,ICON)

Function

Given function value $y_i=f(x_i)$, i=1,2,...,n at the discrete points x_1, x_1, ..., $x_n(x_1<x_2<...<x_n)$, as well as derivatives $y_1^{(l)}=f(x_1)$ and $y_n^{(l)}=f(x_n)$, l=(m+1)/2, (m+1)/2+1, ..., m – 1 at both end points x_1 and x_n, the interpolating coefficients c_j's, j=– m+1, – m+2, ..., n – 1 in the interpolating function represented as a linear combination of B-splines of degree m (odd integer),

$$S(x)=\sum_{j=-m+1}^{n-1}c_jN_{j,m+1}(x) \qquad (1.1)$$

are obtained.

The $S(x)$ to be obtained satisfies

$$\begin{cases} S^{(l)}(x_1)=\sum_{j=-m+1}^{n-1}c_jN_{j,m+1}^{(l)}(x_1)=y_1^{(l)} \\ \qquad l=(m+1)/2,(m+1)/2+1,...,m-1 \\ S(x_i)=\sum_{j=-m+1}^{n-1}c_jN_{j,m+1}(x_i)=y_i \\ \qquad i=1,2,...,n \\ S^{(l)}(x_n)=\sum_{j=-m+1}^{n-1}c_jN_{j,m+1}^{(l)}(x_n)=y_n^{(l)} \\ \qquad l=m-1,m-2,...,(m+1)/2 \end{cases} \qquad (1.2)$$

Here, m≥3, and n≥(m+1)/2.

Parameters

X	Input. Desecrate Points, x_i's. One-dimensional array of size n.
Y	Input. Function values y_i's. One dimensional array of size n.
DY	Input. Derivatives at end points x_1 and x_n. Two-dimensional array of DY(2,(m – 1)/2). DY(1,l-(m – 1)/2) and DY(2,l(m – 1)/2) are assigned $y_1^{(l)}$ and $y_n^{(l)}$, respectively, for l=(m+1)/2,(m+1)/2+1,...,m – 1
N	Input. Number of the discrete points, n.
M	Input. Degree of the B-spline, m. See Note.
C	Input. Interpolation coefficients c_j's. One-dimensional array of size n+m – 1.
VW	Work area. One-dimensional array of size m(n+m – 3)+2(m+1).
ICON ..	Output. Condition code. See Table BIC2-1.

Table BIC2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the followings happened. 1 M is not an odd integer. 2 x_i which satisfies $x_i \le x_{i+1}$ exists. 3 M<3 4 N<(M+1)/2	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UMIO2, UCIO2, UBAS1 and ULUI1
  FORTRAN basic functions ... MOD and FLOAT

- Notes
  By calling the subroutine BIF2 after subroutine BIF2, the interpolated values based on the B-spline interpolating function (1.1), as well as derivatives and/or integrals can be obtained. The parameter values of X, N, M and C are passed from BIC2 to input to BIF2.
  The degree m is preferably 3 or 5. In double precision, if the original function is smooth and y_i's are given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example
  See the example given for subroutine BIF2.

Method

The m-th degree B-spline interpolating function $S(x)$ to be obtained here is a function defined in the interval $[x_1,x_n]$ and satisfying the following conditions:

(a) $S(x)$ is a polynomial at most of degree m in the subinterval $[x_i,x_{i+1}]$, i=1, 2, ..., n – 1.
(b) $S(x)\in C^{m-1}$ $[x_1,x_n]$ i.e., $S(x)$ and its derivatives of up to order (m – 1) are continuos in the interval $[x_1,x_n]$.
(c) $S(x_i)=y_i$, i=1,2,..,n
(d) $S^{(l)}(x_1)=y_1^{(l)}$,
$S^{(l)}(x_n)=y_n^{(l)}$, l = (m+1)/2, (m+1)/2+1, ..., m – 1

A m-th degree spline function defined with knots $\{t_j\}$, j = –m+1, –m+2, ..., n+m, taken as

$$t_j=\begin{cases} x_1 & :-m+1,-m+2,...,1 \\ x_j & :2,3,...,n-1 \\ x_n & :n,n+1,...,n+m \end{cases} \qquad (4.1)$$

can be represented as

$$S(x)=\sum_{j=-m+1}^{n-1}c_jN_{j,m+1}(x) \qquad (4.2)$$

where, $N_{j,m+1}(x)$ is the m-th degree B-spline and given by

$$N_{j,m+1}(x) = \left(t_{j+m+1} - t_j\right) g_{m+1}[t_j, t_{j+1}, \ldots, t_{j+m+1}; x] \quad (4.2)$$

(For details, see Section 7.1)
Considering locality of $N_{j,m+1}(x)$,

$$N_{j,m+1}(x_i)\begin{cases} > 0: \ j = i-m, i-m+1, \ldots i-1 \\ = 0: \begin{cases} j = -m+1, -m+2, \ldots, i-m-1 \\ j = i, i+1, \ldots, n-1 \end{cases} \end{cases}$$

$$N^{(l)}_{j,m+1}(x_1)\begin{cases} \neq 0: \ j = -m+1, -m+2, \ldots, -m+1+l \\ = 0: \ j = -m+2+l, -m+3+l, \ldots, n-1 \end{cases} \quad (4.3)$$

$$N^{(l)}_{j,m+1}(x_n)\begin{cases} = 0: \ j = -m+1, -m+2, \ldots, n-2-l \\ \neq 0: \ j = n-1-l, n-l, \ldots, n-1 \end{cases}$$

and applying the interpolation conditions (c) and (d) described above to Eq. (4.2), a system of $(n+m-1)$ linear equations

$$\begin{cases} \displaystyle\sum_{j=-m+1}^{-m+1+l} c_j N^{(l)}_{j,m+1}(x_1) = y_1^{(l)} \ : \ l = (m+1)/2, (m+1)/2+1, \\ \qquad\qquad\qquad\qquad\qquad\qquad \ldots, m-1 \\ \displaystyle\sum_{j=i-m}^{i-1} c_j N_{j,m+1}(x_i) = y_i \quad : \ i = 1, 2, \ldots, n \\ \displaystyle\sum_{j=n-1-l}^{n-1} c_j N^{(l)}_{j,m+1}(x_n) = y_n^{(l)} \ : \ l = m-1, m-2, \ldots, (m+1)/2 \end{cases}$$

(4.4)

are given with c_j's; $j= -m+1, -m+2, \ldots, n-1$, unknown.
By solving these equations all of the interpolation coefficients c_j's can be obtained.
The form of the coefficient matrix in the linear equations (4.4) is shown in Fig. BIC2-1 as an example for m=5 and n=8.

$$\left[\begin{array}{c|cccccccccc|c}
* & & & & & & & & & & & \\
\hline
* & * & * & * & & & & & & & & \\
* & * & * & * & * & & & & & & & \\
 & * & * & * & * & * & & & & & & \\
 & & * & * & * & * & * & & & & & \\
 & & & * & * & * & * & * & & & & \\
 & & & & * & * & * & * & * & & & \\
 & & & & & * & * & * & * & * & & \\
 & & & & & & * & * & * & * & * & \\
 & & & & & & & * & * & * & * & * \\
 & & & & & & & & * & * & * & * \\
\hline
 & & & & & & & & & & & *
\end{array}\right]$$

Fig. BIC2-1 Coefficient matrix (n=7 and m=3)

The subroutine solves the linear equations by using the Crout method (LU decomposition).
Subroutines UMIO2, ULUI1 and UCIO2 are called.

E12-31-0302 BIC3, DBIC3

B-spline interpolation coefficient calculation (III)
CALL BIC3(X,Y,N,M,C,XT,VW,ICON)

Function

Given function values $y_i=f(x_i)$, i=1,2,...,n for discrete points x_1, x_2, ..., $x_n(x_1<x_2<...<x_n)$ this subroutine obtains the interpolating spline $S(x)$of degree m represented as a linear combination of B-splines:

$$S(x)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x) \tag{1.1}$$

The knots of the spline are taken as

$$\begin{aligned} &\xi_1 = x_1 \\ &\xi_i = x_{i+(m-1)/2} \quad ,i = 2,3,...,n-m \\ &\xi_{n-m+1} = x_n \end{aligned}$$

where m is odd integer greater than 2 and $n \geq m+2$ must be satisfied.

Parameters

X	Input. Desecrate Points, x_i. One-dimensional array of size n.
Y	Input. Function values, y_i. One-dimensional array of size n.
N	Input. Number n of the discrete points.
M	Input. Degree m of the B-spline. (See comment)
C	Output. Interpolation coefficients c_j. One-dimensional array of size n.
XT	Output. The knots ξ_i. One-dimensional array of size $n-m+1$.
VW	Work area. One-dimensional array of size $mn+2$.
ICON ..	Output. Condition code. See Table BIC3-1.

Table BIC3-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	(1) M is not an odd number, or (2) N<M+2, or (3) $x_i \leq x_{i+1}$, or (4) M<3.	Bypassed

Comments on use

- Subprograms used
  SSL II .... MGSSL, UMIO3, UCIO3, UBAS1 and ULUI3
  FORTRAN basic function .... MOD

- Notes
  The interpolated values or derivative or integrals based on the interpolating spline (1.1) may be determined by the subroutine BIF3 following this subroutine. In that case the values of parameters X, N, M, C, and XT are input to the subroutine BIF3.
  The preferred degree m is 3 or 5.
  In double precision, however, if the original function does not change obruptly and y_i is given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example
  See the example for the subroutine BIF3.

Method

Given function values $y_i=f(x_i)$. i=1,2,...,n for discrete points $x_1,.x_2,...,x_n(x_1<x_2<...<x_n)$ the interpolating spline of degree m to be obtained here is a function which is defined on the interval $[x_1,x_n]$ and satisfies the following requirements.

(a) $S(x)$ is polynomial of degree m at most on each subinterval $[\xi_i, \xi_{i+1}]$, i=1, 2, ..., $n-m$, $\xi_1=x_1$, $\xi_i=x_{i+(m-1)/2}$, i=2, 3, ..., $n-m$, $\xi_{n-m+1}=x_n$
(b) $S(x)\in C^{m-1}[x_1,x_n]$. That is, $S(x)$ and its derivatives of up to order m-1 are continuous on the interval $[x_1,x_n]$.
(c) $S(x_i)=y_i$, i=1,2,..,n

The $S(x)$ satisfying (a), (b) and (c) can be given by

$$S(x)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x) \tag{4.1}$$

where c_j's, $j=-m+1, -m+2,...,n-m$, are constants, $N_{j,m+1}(x)$ is a normalized m-th degree B-spline and defined by

$$N_{j,m+1}(x)=(t_{j+m+1}-t_j)g_{m+1}[t_j,t_{j+1},...,t_{j+m+1};x] \tag{4.2}$$

(For details, see Section 7.1)
In this subroutine, the knots $\{t_r\}$ of the B-spline is given by

$$t_r=\begin{cases}\xi_1 & , r=-m+1,-m+2,...,1 \\ \xi_r & , r=2,3,...,n-m \\ \xi_{n-m+1} & , r=n-m+1,n-m+2,...,n+1\end{cases} \tag{4.3}$$

A typical behavior of $N_{j,m+1}(x)$ with the knots $\{t_r\}$ is shown in Fig. BIC3-1.
The interpolation coefficients c_j's can be determined by solving the linear equations:

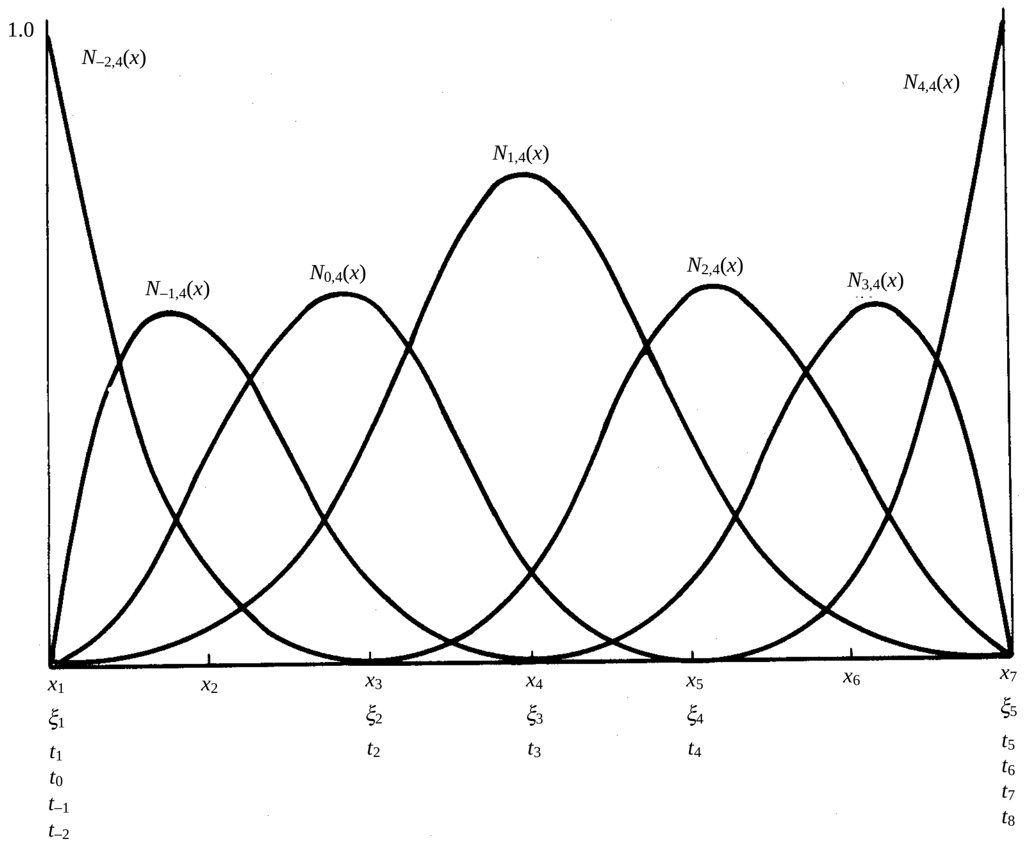


Fig.BIC3-1 Example of B-spline $N_{j,m+1}(x)$: μ = 3, with seven discrete points at equal intervals

$$S(x_i)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x_i)=y_i \quad ,i=1,2,...,n \qquad (4.4)$$

which satisfies the interpolation conditions.

The coefficient matrix of the equations have many null elements because of locality of the B-spline and therefore, it has a similar form to a banded matrix. An example of the coefficient matrix is given in Fig. BIC3-2 for the case, n=7 and m=3.

$$\begin{bmatrix} * & & & & & & \\ * & * & * & * & & 0 & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & * & * & * & \\ & 0 & & * & * & * & * \\ & & & & & & * \end{bmatrix}$$

Fig. BIC3-2 Coefficient matrix (n =7 and m =3)

The subroutine solves these linear equations by using Crout method (LU decomposition method). Subroutine ULUI3 is called.

E12-31-0402 BIC4, DBIC4

B-spline interpolation coefficient calculation (IV)
CALL BIC4(X,Y,N,M,C,VW,ICON)

Function

Given periodic function values $y_i=f(x_i)$, $i=1, 2, ..., n$, (where $y_1=y_n$) at the discrete points $x_1, x_2, ..., x_n$ ($x_1<x_2<...<x_n$) with the period (x_n-x_1), the interpolation coefficients c_j's, $j=-m+1, -m+2, ..., n-1$ in the interpolating function represented as a linear combination of B-splines of degree m (odd integer),

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \qquad (1.1)$$

are obtained.

The obtained $S(x)$ is a periodic function with the period (x_n-x_1) similarly to $f(x)$ and satisfies the boundary conditions:

$$S^{(l)}(x_1)=S^{(l)}(x_n) \quad ,l=0,1,...,m-1 \qquad (1.2)$$

Here $m \ge 3$ and $n \ge m+2$.

Parameters

X	Input. Discrete points, x_i's. One-dimensional array of size n.
Y	Input. Function values y_i's. One dimensional array of size n. Must be $y_1=y_n$. If $y_1 \ne y_n$, y_n is taken.
N	Input. Number of the discrete points, n.
M	Input. Degree of the B-spline, m. See Notes.
C	Input. Interpolation coefficients c_j. One-dimensional array of size $n+m-1$.
VW	Work area. One-dimensional array of size $(n-1)(2m-1)+m+1$
ICON ..	Output. Condition code. See Table BIC4-1.

Table BIC4-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the followings occurred. (a) M is not an odd integer. (b) N<M+2 (c) x_i satisfying $x_i \le x_{i+1}$ exists. (d) M<3	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL, UMIO4, UCIO4, UBAS4, ULUI4 and UPEP4
  FORTRAN basic functions ... MOD and FLOAT

- Notes
  By calling the subroutine BIF4 after subroutine BIC4, the interpolated values based on the B-spline interpolating function (1.1), as well as derivatives and/or integrals can be obtained. The parameter values of X, N, M and C are passed from BIC4 to input to BIF4.
  The degree m is preferably 3 or 5. In double precision, if the original function is smooth and y_i's are given with high accuracy, the degree may be increased above 3 or 5, but not beyond 15.

- Example
  See the example given for subroutine BIF4.

Method

The m-th degree B-spline interpolating function $S(x)$ to be obtained here is a function defined in the interval $[x_1,x_n]$ and satisfying the following conditions:

Condition (c) is necessary for $S(x)$ to be periodic and is peculiar to this subroutine.

(a) $S(x)$ is polynomial at most of order m in the interval $[x_i,x_{i+1})$, $i=1, 2, ... n-1$.
(b) $S(x)\in C^{m-1}[x_1,x_n]$, i.e., the $S(x)$ and its derivatives of up to degree $m-1$ are continuous in the interval $[x_1,x_n]$.
(c) $S^{(l)}(x_1) = S^{(l)}(x_n)$, $l=0,1,...,m-1$
(d) $S(x_i)=y_i$, $i=2,3,..,n$ ($y_1=y_n$ is assumed)

First, the spline function satisfying condition (c) will be explained. We take the knots $\{t_j\}$ of the B-spline as follows (see Fig. BIC4-1):

$$t_j=\begin{cases} x_{n-1-j}-(x_n-x_1) & ,-m+1\le j\le 0 \\ x_j & ,1\le j\le n \\ x_{-n+1+j}+(x_n-x_1) & ,n+1\le j\le n+m \end{cases} \qquad (4.1)$$

The m-th degree spline function based on this knots can be expressed with c_j's being arbitrary constants:

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \qquad (4.2)$$

where $N_{j,m+1}(x)$ is the m-th degree B-spline and given by

$$N_{j,m+1}(x)=(t_{j+m+1}-t_j)g_{m+1}[t_j,t_{j+1},...,t_{j+m+1};x]$$

(For further details, see Section 7.1)
Adding the condition:

$$c_j = c_{j+n-1} \quad , j = -m+1,...,0 \qquad (4.3)$$

to Eq. (4.2), the periodic condition (c) described above is satisfied.
Next, the interpolation condition (d) is written as

$$\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x_i) = y_i \quad , i = 2,...,n \qquad (4.4)$$

with m=2l – 1($l \geq 2$), Eq. (4.3) is rewritten to

$$\begin{aligned} c_j &= c_{j+n-1} \quad , j = -2l+2,...,-l+1 \\ c_j &= c_{j-n+1} \quad , j = n-l+1,...,n-1 \end{aligned} \qquad (4.5)$$

Rewriting Eq. (4.4) by using the above relationships,

$$\begin{aligned} &\sum_{j=-l+2}^{0} c_j \left\{ N_{j,2l}(x_i) + N_{j+n-1,2l}(x_i) \right\} + \sum_{j=1}^{n-2l} c_j N_{j,2l}(x_i) \\ &+ \sum_{j=n-2l+1}^{n-l} c_j \left\{ N_{j-n+1,2l}(x_i) + N_{j,2l}(x_i) \right\} = y_i \\ &\quad i = 2,3,...,n \end{aligned} \qquad (4.6)$$

can be obtained.
Eq. (4.6) is a system of linear equations with (n – 1) unknowns c_j's (j=–l+2,–l+3,...,n – l).
Solving the equations and using the relationships (4.5), all of the interpolation coefficients c_j's of the spline function (4.2) can be obtained. The coefficient matrix of equations (4.6) has a form similar to a banded matrix. An example is given in Fig. BIC4-2 for the case of m=5(l=3) and n=9.

$$\left[\begin{array}{cccccccc} * & * & * & & & & * & * \\ * & * & * & * & & & & * \\ * & * & * & * & * & & 0 & \\ & * & * & * & * & * & & \\ & & * & * & * & * & * & \\ & 0 & & * & * & * & * & * \\ * & & & & * & * & * & * \\ * & * & & & & * & * & * \end{array}\right]$$

Fig. BIC4-2 Coefficient matrix (m=5 and n=9)

The subroutine solves the linear equations given above by using Crout method (LU decomposition method). Subroutines UMIO4, ULUI4 and UCIO4 are called.

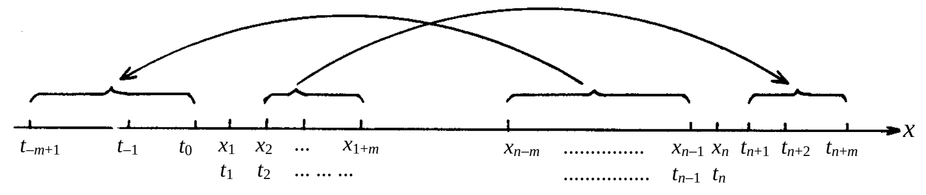


Fig. BIC4-1 Knots {t_j}

E11-32-1101 BIFD1, DBIFD1

B-spline two-dimensional interpolation, differentiation and integration (I-I)
CALL BIFD1(X,NX,Y,NY,M,C,K,ISWX,VX,IX,ISWY, VY,IY,F,VW,ICON)

Function

When, on the xy-plane, the function values $f_{ij}= f(x_i, y_j)$ are given at the points (x_i, y_i), $(x_1 < x_2 < \cdots < x_{nx}, y_1 < y_2 < \cdots < y_{ny})$, and also the following partial derivatives are given at the boundary points, an interpolated value or a partial derivative at point P(v_x, v_y) or a double integral on the area $\{(x,y) \mid x_1 \le x \le v_x , y_1 \le y \le v_y\}$are obtained. (See Fig. BIFD1-1)

$$f_{1,j}^{(\lambda,0)} = f^{(\lambda,0)}(x_1,y_j) \ , \ f_{n_x,j}^{(\lambda,0)} = f^{(\lambda,0)}(x_{n_x},y_j)$$
$$f_{i,1}^{(0,\mu)} = f^{(0,\mu)}(x_i,y_1) \ , \ f_{i,n_y}^{(0,\mu)} = f^{(0,\mu)}(x_i,y_{n_y})$$
$$f_{11}^{(\lambda,\mu)} = f^{(\lambda,\mu)}(x_1,y_1) \ , \ f_{n_x,1}^{(\lambda,\mu)} = f^{(\lambda,\mu)}(x_{n_x},y_1)$$
$$f_{1,n_y}^{(\lambda,\mu)} = f^{(\lambda,\mu)}(x_1,y_{n_y}), \ f_{n_x,n_y}^{(\lambda,\mu)} = f^{(\lambda,\mu)}(x_{n_x},y_{n_y})$$
$$i = 1,2,...,n_x , j = 1,2,...,n_y$$
$$\lambda = 1,2,...,(m-1)/2, \mu = 1,2,...,(m-1)/2$$

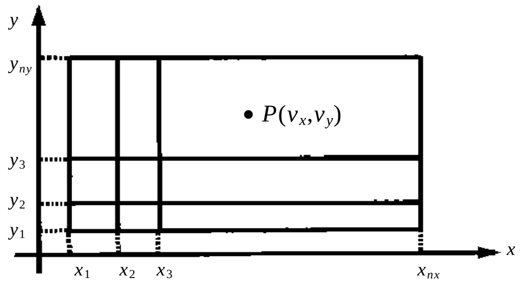


Fig. BIFD1-1 Point p in the area $R=\{(x,y) \mid x_1 \le x \le x_{nx} , y_1\le y\le y_{ny}\}$

However, subroutine BICD1 must be called before using subroutine BIFD1 to calculate the interpolating coefficients $C_{\alpha,\beta}$ in the B-spline two-dimensional interpolating function,

$$S(x,y) = \sum_{\beta=-m+1}^{n_y-1} \sum_{\alpha=-m+1}^{n_x-1} c_{\alpha,\beta} N_{\alpha,m+1}(x) N_{\beta,m+1}(y) \qquad (1.1)$$

where m is an odd integer and denotes the degree of the B-splines, $N_{\alpha,m+1}(x)$ and $N_{\beta,m+1}(y)$. Here $x_1\le v_x\le x_{nx}$, $y_1\le v_y\le y_{ny}$, $m\ge3$, $n_x\ge2$ and $n_y\ge2$.

Parameters

X	Input. Discrete points, x_i's in the x direction. One-dimensional array of size n_x.
NX	Input. Number of x_i's, n_x.
Y	Input. Discrete points y_i's in the y direction. One-dimensional array of size n_y.
NY	Input. Number of y_i's, n_y.
M	Input. Degree of the B-spline. See Notes.
C	Input. Interpolating coefficients $C_{\alpha,\beta}$ (output from BICD1). Two-dimensional array as C(K,NY+M−1).
K	Input. Adjustable dimension for array C. (K≥NX+M−1).
ISWX ..	Input. An integer which specifies the type of calculation in the direction of x. $-1\le$ISWX$\le m$ See the parameter F.
VX	Input. x-coordinate at point $P(v_x,v_y)$.
IX	Input. Value i which satisfies $x_i\le v_x<x_{i+1}$. If $v_x=x_{nx}$ then IX=n_x-1 Output. Value i which satisfies $x_i\le v_x<x_{i+1}$.
ISWY ..	Input. An integer which specifies the type of calculation in the direction of y. $-1\le$ISWY$\le m$(See the parameter F)
VY	Input. y-coordinate at point $P(v_x,v_y)$.
IY	Input. Value j which satisfies $y_j\le v_y<y_{j+1}$. If $v_y=y_{ny}$, then IY=n_y-1 Output. Value j which satisfies $y_j\le v_y<y_{j+1}$. See Notes.
F	Output. Interpolated value, partial derivative or integral obtained. Setting ISWX=λ and ISWY=μ, one of the following values is put out depending on combination of λ and μ. When $0\le\lambda,\mu$

$$\text{F} = \frac{\partial^{\lambda+\mu}}{\partial x^{\lambda} \partial y^{\mu}} S(v_x,v_y)$$

The interpolated value can be obtained by setting $\lambda=\mu=0$.

When $\lambda= -1$, $0\le\mu$

$$\text{F} = \int_{x_1}^{v_x} \frac{\partial^{\mu}}{\partial y^{\mu}} S(x,v_y)\, dx$$

When $\lambda \le 0$, $\mu \le -1$

$$\text{F} = \int_{y_1}^{v_y} \frac{\partial^{\lambda}}{\partial x^{\lambda}} S(v_x,y)\, dy$$

When $\lambda = \mu = -1$

$$\text{F} = \int_{y_1}^{v_y} dy \int_{x_1}^{v_x} S(x,y)\, dx$$

VW	Work area. One-dimensional array of size $4(m+1) + \max(n_x,n_y)+m-1$
ICON ..	Output. Condition code. See Table BIFD1-1.

Table BIFD1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Either X(IX)≤VX<X(IX+1) or Y(IY)≤VY<Y(IY+1) is not satisfied.	IX or IY shown on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: 1 VX<X(1) or VX>X(NX) 2 VY<Y(1) or VY>Y(NY) 3 ISWX<-1 or ISWX>M 4 ISWY<-1 or ISWY>M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAD1 and UBAS1
  FORTRAN basic function ... FLOAT

- Notes
  The subroutine, based on the B-spline two-dimensional interpolating function (1.1) given by using subroutine BICD1, obtains an interpolated value, a partial derivative or a double integral.
  Therefore subroutine BICD1 must be called to obtain the interpolating function (1.1) before calling this subroutine to obtain an interpolated value, etc. Also parameters X, NX, Y, NY, K, M and C must be directly passed from subroutine BICD1.
  Parameters IX and IY should satisfy the relationships X(IX)≤VX<X(IX+1) and Y(IY)≤VY<Y(IY+1), respectively. If not, IX and IY that satisfy those relationships are searched for to continue the processing.

- Example
  By inputting points (x_i,y_i), function values f_{ij}, i=1,2,...,n_x, j=1, 2, ..., n_y, partial derivatives $f_{i,j}^{(\lambda,\mu)}$,i=1,n_x, j=1,n_y,λ=1, 2, ..., $(m-1)/2$, μ=1, 2, ..., $(m-1)/2$, and degree m, interpolated values or partial derivatives at points (v_{ir},u_{js}); $v_{ir}=x_i+(x_{i+1}-x_i)\cdot(r/4)$, $u_{js}=y_j+(y_{j+1}-y_j)\cdot(s/4)$, i=1, 2, ..., n_x-1, j=1, 2, ..., n_y-1 , r=0, 1, ..., 4, s=0,1,...,4, or integrals over the area $\{(x,y) \mid x_1 \le x \le v_{ir} , y_1 \le y \le u_{js}\}$ are obtained.
  Here n_x≤30, n_y≤30 and m≤5. Further, the data input to FXY(I,J) must be given as follows, when m=5, for example

$$\begin{bmatrix}
f_{11}^{(2,2)} & f_{11}^{(1,2)} & f_{11}^{(0,2)} & f_{21}^{(0,2)} & \cdots & f_{n_x,1}^{(0,2)} & f_{n_x,1}^{(1,2)} & f_{n_x,1}^{(2,2)} \\
f_{11}^{(2,1)} & f_{11}^{(1,1)} & f_{11}^{(0,1)} & f_{21}^{(0,1)} & \cdots & f_{n_x,1}^{(0,1)} & f_{n_x,1}^{(1,1)} & f_{n_x,1}^{(2,1)} \\
f_{11}^{(2,0)} & f_{11}^{(1,0)} & f_{11} & f_{21}, f_{31} & \cdots & f_{n_x,1} & f_{n_x,1}^{(1,0)} & f_{n_x,1}^{(2,0)} \\
. & . & . & . & . & . & . & . \\
f_{1,n_y}^{(2,0)} & f_{1,n_y}^{(1,0)} & f_{1,n_y} & f_{2,n_y} & \cdots & f_{n_x,n_y} & f_{n_x,n_y}^{(1,0)} & f_{n_x,n_y}^{(2,0)} \\
f_{1,n_y}^{(2,1)} & f_{1,n_y}^{(1,1)} & f_{1,n_y}^{(0,1)} & f_{2,n_y}^{(0,1)} & \cdots & f_{n_x,n_y}^{(0,1)} & f_{n_x,n_y}^{(1,1)} & f_{n_x,n_y}^{(2,1)} \\
f_{1,n_y}^{(2,2)} & f_{1,n_y}^{(1,2)} & f_{1,n_y}^{(0,2)} & f_{2,n_y}^{(0,2)} & \cdots & f_{n_x,n_y}^{(0,2)} & f_{n_x,n_y}^{(1,2)} & f_{n_x,n_y}^{(2,2)}
\end{bmatrix}$$

```
C     **EXAMPLE**
      DIMENSION X(30),Y(30),FXY(32,32),
     *C(32,32),VW(232),R(5,5)
      READ(5,500) NX,NY,M
      READ(5,510) (X(I),I=1,NX),
     *            (Y(J),J=1,NY)
      NXM=NX+M-1
      NYM=NY+M-1
      READ(5,510) ((FXY(I,J),I=1,NXM),
     *J=1,NYM)
      WRITE(6,600) M,(I,X(I),I=1,NX)
      WRITE(6,610) (J,Y(J),J=1,NY)
      WRITE(6,620) ((I,J,FXY(I,J),I=1,NXM),
     *J=1,NYM)
      K=32
      CALL BICD1(X,NX,Y,NY,FXY,K,M,C,
     *VW,ICON)
      IF(ICON.EQ.0) GO TO 20
      WRITE (6,630)
      STOP
   20 NX1=NX-1
      NY1=NY-1
      M2=M+1
      DO 80 LY2=1,M2
      ISWY=LY2-2
      DO 70 LX2=1,M2
      ISWX=LX2-2
      WRITE(6,640) ISWX,ISWY
      DO 60 IY=1,NY1
      HY=(Y(IY+1)-Y(IY))*0.25
      YJ=Y(IY)
      DO 50 IX=1,NX1
      HX=(X(IX+1)-X(IX))*0.25
      XI=X(IX)
      DO 40 J=1,5
      VY=YJ+HY*FLOAT(J-1)
      DO 30 I=1,5
      VX=XI+HX*FLOAT(I-1)
      IXX=IX
      IYY=IY
      CALL BIFD1(X,NX,Y,NY,M,C,K,ISWX,
     *VX,IXX,ISWY,VY,IYY,F,VW,ICON)
      R(I,J)=F
   30 CONTINUE
   40 CONTINUE
```

```
    WRITE(6,650) IXX,IYY,((R(I,J),I=1,5),
   *J=1,5)
 50 CONTINUE
 60 CONTINUE
 70 CONTINUE
 80 CONTINUE
    STOP
500 FORMAT(3I6)
510 FORMAT(2F12.0)
600 FORMAT('1'//10X, 'INPUT DATA',3X,'M=',
   *I2//20X,'NO.',10X,'X'/(20X,I3,E18.7))
610 FORMAT(//20X,'NO.',10X,'Y'/(20X,I3,
   *E18.7))
620 FORMAT(3(10X,'FXY(',I2,',',I2,')=',
   *E15.7))
630 FORMAT('0',10X,'ERROR')
640 FORMAT(//5X,'ISWX=',I2,3X,'ISWY=',
   *I2//)
650 FORMAT(10X,'IX=',I3,2X,'IY=',I3/
   *(15X,5(5X,E15.7)))
    END
```

Method

Suppose that the interpolating coefficient $c_{\alpha,\beta}$ in the dual m-th degree B-spline two dimensional interpolating function.

$$S(x,y)=\sum_{\beta=-m+1}^{n_y-1}\sum_{\alpha=-m+1}^{n_x-1}c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (4.1)$$

is already obtained by subroutine BICD1.

Subroutine BIFD1 calculates an interpolated value, partial derivative and/or an integral based on the interpolating function (4.1). The method is given in Section 7.1 "Definition, representation and calculation method of bivariate spline function".

E11-32-3301 BIFD3, DBIFD3

B-spline two-dimensional interpolation, differentiation and integration (III-III)
CALL BIFD3(X,NX,Y,NY,M,C,K,XT,ISWX,VX,IX, ISWY,VY,IY,F,VW,ICON)

Function

Given the function values $f_{ij}=f(x_i,y_j)$ at points (x_i, y_j), $(x_1 < x_2 < ... < x_{nx}, y_1 < y_2 < ... < y_{ny})$ on the xy -plane, an interpolated value or a partial derivativee at the point $P(v_x,v_y)$ and/or a double integral over the area $[x_1 \le x \le v_x\ ,\ y \le y \le v_y]$, is obtained. (See Fig. BIFD3-1)

Before using subroutine BIFD3, the knots $\{\xi_j\}$ in the x-direction and the knots $\{\eta_j\}$ in the y-direction, and also the interpolating coefficients $c_{\alpha,\beta}$ in the B-spline two-dimensional interpolating function

$$S(x,y)=\sum_{\beta=-m+1}^{n_y-m}\sum_{\alpha=-m+1}^{n_x-m}c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (1.1)$$

must be calculated by subroutine BICD3, where m is an odd integer and denotes the degree of the B-splines $N_{\alpha,m+1}(x)$ and $N_{\beta,m+1}(y)$. Here $x_1 \le v_x \le x_{nx}$

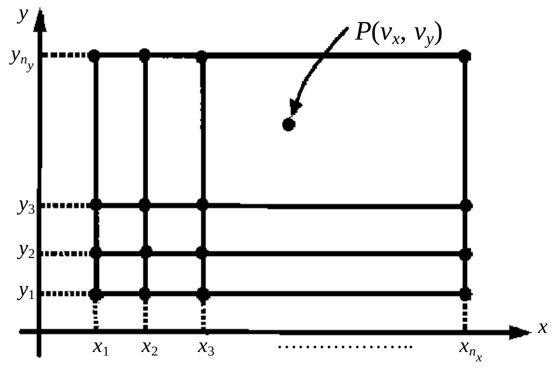


Fig. BIFD3-1 Point p in the area R={(x,y) | $x_1 \le x \le x_{nx}$, $y_1 \le y \le y_{ny}$}

Parameters

X	Input. Discrete points, x_i's in the x-direction. One-dimensional array of size n_x.
NX	Output. Number of x_i's, n_x.
Y	Input. Discrete points y_j's in the y-direction. One dimensional array of size n_y.
NY	Input. Number of y_j's, n_y.
M	Input. The degree of the B-spline, m. See Note.
C	Input. Interpolation coefficients $c_{\alpha,\beta}$ (output from BICD3). Two-dimensional array as C(K,NY).
K	Input. Adjustable dimension for array C.
XT	Input. Knots in the x- and y-direction (output from BICD3). One-dimensional array of size$(n_x-m+1)+(n_y-m+1)$.
ISWX ..	Input. An integer which specifies type of calculation associated with x-direction. $-1 \le$ISWX$\le m$. See parameter F.
VX	Input. x-coordinate of the point $P(v_x,v_y)$.
IX	Input. The i which satisfies $x_i \le v_x < x_{i+1}$. If $v_x=x_{nx}$, then IX=n_x-1. Output. The i which satisfies $x_i \le v_x < x_{i+1}$. See Note.
ISWY ..	Input. An integer which specifies type of calcuration associated with y-direction. $-1 \le$ISWY$\le m$. See the parameter F.
VY	Input. y-coordinate of the point $P(v_x,v_y)$.
IY	Input. The j value which satisfies $y_j \le v_y < y_{j+1}$. If $v_y=y_{ny}$ then IY=n_y-1 Output. The j which satisfies $y_j \le v_y < y_{j+1}$. See Note.
F	Output. Interpolated value, partial derivative or integral value. Setting ISWX=λ and ISWY=μ, one of the following value is put out depending on combination of λ and μ:

- When $0 \le \lambda,\mu$

$$F=\frac{\partial^{\lambda+\mu}}{\partial x^{\lambda}\partial y^{\mu}}S(v_x,v_y)$$

The interpolated value can be obtained by setting $\lambda = \mu = 0$.

- When $\lambda=-1$, $0 \le \mu$

$$F=\int_{x_1}^{v_x}\frac{\partial^{\mu}}{\partial y^{\mu}}S(x,v_y)dx$$

- When $\lambda \ge 0$, $\mu=-1$

$$F=\int_{y_1}^{v_y}\frac{\partial^{\lambda}}{\partial x^{\lambda}}S(v_x,y)dy$$

- When $\lambda = \mu = -1$

$$F=\int_{y_1}^{v_y}dy\int_{x_1}^{v_x}S(x,y)\,dx$$

VW	Work area. One-dimensional array of size $4(m+1)$ + max(n_x,n_y)
ICON ..	Output. Condition code. See Table BIFD3-1.

- Subprograms used
  SSL II ... MGSSL, UCAD1 and UBAS1
  FORTRAN basic function ... FLOAT

Table BIFD3-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	X(IX)≤VX<X(IX+1) or Y(IY)≤VY<Y(IY+1) is not satisfied.	IX or IY shown on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: 1 VX<X(1) or VX>X(NX) 2 VY<Y(1) or VY>Y(NY) 3 ISWX<-1 or ISWX>M 4 ISWY<-1 or ISWY>M	Bypassed

- Notes

  The subroutine, based on the B-spline two-dimensional interpolating function (1.1) given by using subroutine BICD3, obtains an interpolated value, a partial derivative or a double integral. Therefore subroutine BICD3 must be called to obtain the interpolating function (1.1) before calling this subroutine to obtain an interpolated value, etc. Also parameters X, NX, Y, NY, K, M, C and XT must be directly passed from subroutine BICD3.

  Parameters IX and IY should satisfy the relationships X(IX)≤VX<X(IX+1) and Y(IY)≤VY<Y(IY+1). If not, IX and IY that satisfy those relationships are searched for to continue the processing.

- Example

  By inputting points (x_i,y_j), function values f_{ij}, $i=1,2,...,n_x$, $j=1,2,...,n_y$, and the degree m, interpolated values or partial derivatives at the point (v_{ir},u_{js}) ; $v_{ir}=x_i+(x_{i+1}-x_i)(r/4)$, $u_{js}=y_j+(y_{j+1}-y_j)(s/4)$, $i=1,2,...,n_x-1$, $j=1,2,...,n_y-1$, $r=0,1,...,4$ and $s=0,1,...,4$, and/or integrals over the area$[x_1 \le x \le v_{ir}, y_1 \le y \le u_{js}]$ are obtained. Here $n_x \le 30$, $n_y \le 30$ and $m \le 5$.

```
C     **EXAMPLE**
      DIMENSION X(30),Y(30),FXY(30,30),
     *C(30,30),XT(52),VW(182),R(5,5)
      READ(5,500) NX,NY,M
      READ(5,510) (X(I),I=1,NX),
     *            (Y(J),J=1,NY)
      READ(5,510) ((FXY(I,J),J=1,NY),I=1,NX)
      WRITE(6,600) M,(I,X(I),I=1,NX)
      WRITE(6,610) (J,Y(J),J=1,NY)
      WRITE(6,620)
      DO 10 I=1,NX
   10 WRITE(6,630) (I,J,FXY(I,J),J=1,NY)
      K=30
      CALL BICD3(X,NX,Y,NY,FXY,K,M,C,
     *XT,VW,ICON)
      IF(ICON.EQ.0) GO TO 20
      WRITE(6,640)
      STOP
   20 NX1=NX-1
      NY1=NY-1
      M2=M+2
      DO 80 LX2=1,M2
      ISWX=LX2-2
      DO 70 LY2=1,M2
      ISWY=LY2-2
      WRITE(6,650) ISWX,ISWY
      DO 60 IX=1,NX1
      HX=(X(IX+1)-X(IX))*0.25
      XI=X(IX)
      DO 50 IY=1,NY1
      HY=(Y(IY+1)-Y(IY))*0.25
      YJ=Y(IY)
      DO 40 I=1,5
      VX=XI+HX*FLOAT(I-1)
      DO 30 J=1,5
      VY=YJ+HY*FLOAT(J-1)
      IXX=IX
      IYY=IY
      CALL BIFD3(X,NX,Y,NY,M,C,K,XT,
     *ISWX,VX,IXX,ISWY,VY,IYY,F,VW,ICON)
      R(I,J)=F
   30 CONTINUE
   40 CONTINUE
      WRITE(6,660)IXX,IYY,((R(I,J),J=1,5),
     *I=1,5)
   50 CONTINUE
   60 CONTINUE
   70 CONTINUE
   80 CONTINUE
      STOP
  500 FORMAT(3I6)
  510 FORMAT(2F12.0)
  600 FORMAT('1'//10X,'INPUT DATA',3X,
     *'M=',I2//20X,'NO.',10X,'X'/
     *(20X,I3,E18.7))
  610 FORMAT(//20X,'NO.',10X,'Y'/
     *(20X,I3,E18.7))
  620 FORMAT(//20X,'FXY'/)
  630 FORMAT(3(10X,'FXY(',I2,',',I2,')=',
     *E15.7))
  640 FORMAT ('0',10X,'ERROR')
  650 FORMAT(//5X,'ISWX=',I2,3X,'ISWY=',
     *I2//)
  660 FORMAT(10X,'IX=',I3,2X,'IY=',I3/
     *(15X,5(5X,E15.7)))
      END
```

Method

Suppose that the dual m-th degree B-spline two-dimensional interpolating function.

$$S(x,y)=\sum_{\beta=-m+1}^{n_y-m}\sum_{\alpha=-m+1}^{n_x-m} c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (4.1)$$

is already obtained by subroutine BICD3. Subroutine BIFD3 calculates interpolated values, partial derivative and/or an integrals based on the interpolating function (4.1). The method is described in Section 7.1 "Definition, representation and calculation method of spline function".

E11-31-0101 BIF1, DBIF1

B-spline interpolation, differentiation and integration (I)
CALL BIF1(X,N,M,C,ISW,V,I,F,VW,ICON)

Function

Given function values $y_i=f(x_i)$, $i=1,2,...,n$ at discrete points $x_1,x_2...,x_n(x_1<x_2<...<x_n)$ and derivatives $y_1^{(l)} = f^{(l)}(x_1)$ and $y_n^{(l)} = f^{(l)}(x_n)$, $l=1,2,...,(m-1)/2$ at each end point x_1 and x_n, then an interpolated value, a derivative at the point $x=v\in[x_1, x_n]$, or an integral from x_1 to v are obtained. However, subroutine BIC1 must be called before using subroutine BIF1 to calculate the interpolating coefficients c_j, $j=-m+1, -m+2, ..., n-1$ in the B-spline interpolating function,

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{1.1}$$

, where m is an odd integer and is the degree of the B-spline $N_{j,m+1}(x)$ and $x_1\le v\le x_n$, $m\le 3$ and $n\le 2$.

Parameters

X ..... Input. Discrete points x_i.
One-dimensional array of size n.

N ..... Input. Number of the discrete points, n.

M ..... Input. Degree of the B-spline, m.
See Note.

C ..... Input. Interpolating coefficients c_j(output from BIC1) One-dimensional array of size $n+m-1$.

ISW ... Input. An integer which specifies the type of calculation.
If ISW=0, interpolated value F=$S(v)$.
If ISW=l $(1 \le l \le m)$, l-th order derivative F=$S^{(l)}(v)$.
If ISW=–1, integral $F=\int_{x_1}^{v} S(x)dx$ are calculated, respectively.

V ..... Input. The point v at which the interpolated value etc. is to be obtained.

I ..... Input. Value of i which satisfies $x_i\le v<x_{i+1}$.
If $v=x_n$ the parameter should be given $n-1$.
Output. Value of i which satisfies $x_i\le v<x_{i+1}$.
See Note.

F ..... Output. Interpolated value, l-th order derivative or integral. See parameter ISW.

VW .... Work area. One-dimensional array of size $m+1$.

ICON .. Output. Condition code.
See Table BIF1-1.

Table BIF1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	X(I)≤V<X(I+1) is not satisfied.	The I given on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: (a) V<X(1) or V>X(N) (b) ISW<-1 or ISW>M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAR1 and UBAS1
  FORTRAN basic function ... FLOAT

- Notes
  The subroutine obtains an interpolated value, derivative or integral based on the B-spline interpolating function (1.1) which is obtained by using subroutine BIC1. Therefore, BIC1 must be called before calling this subroutine. Parameters, X, N, M and C must be directly passed from BIC1 to subroutine BIF1.
  Parameter I should satisfy the relationships X(I)≤V<X(I+1). If not, the value of I which satisfies the relationship is searched for to continue the processing.

- Example
  By inputting discrete points x_i, function values y_i, $i=1,2,...$; n derivatives $y_1^{(l)}$, $l=1,2,...(m-1)/2$, and $y_n^{(l)}$, $l=1,2,...,(m-1)/2$ at each end point, and the degree m, the following values are obtained; an integral value from x_1 to $v_{ij}=x_i+(x_{i+1}-x_i)(j/5)$, $i=1,2,...,n-1$, $j=0,1,...,5$, an interpolated value or derivatives of order one through m. Here $n\le 101$, and $m\le 5$.

```
C     **EXAMPLE**
      DIMENSION X(101),Y(101),C(105),
     *DY(2,2),VW(519),R(6)
      READ(5,500) N,M
      LM1=(M-1)/2
      READ(5,510) (X(I),Y(I),I=1,N)
     *,((DY(I,L),I=1,2),L=1,LM1)
      WRITE(6,600) N,M,(I,X(I),Y(I),I=1,N)
      CALL BIC1(X,Y,DY,N,M,C,VW,ICON)
      IF(ICON.EQ.0) GO TO 10
      WRITE(6,610)
      STOP
   10 N1=N-1
      M2=M+2
      DO 40 L2=1,M2
      ISW=L2-2
      WRITE(6,620) ISW
      DO 30 I=1,N1
      H=(X(I+1)-X(I))/5.0
      XI=X(I)
      DO 20 J=1,6
      V=XI+H*FLOAT(J-I)
      II=I
      CALL BIF1(X,N,M,C,ISW,V,II,F,VW,ICON)
      R(J)=F
   20 CONTINUE
      WRITE(6,630) II,(R(J),J=1,6)
   30 CONTINUE
   40 CONTINUE
      STOP
  500 FORMAT(2I6)
  510 FORMAT(2F12.0)
  600 FORMAT('1'//10X,'INPUT DATA',3X,
     *'N=',I3,3X,'N=',I2//20X,'NO.',10X,
     *'X',17X,'Y'//(20X,I3,2E18.7))
  610 FORMAT('0',10X,'ERROR')
  620 FORMAT('1'//10X,'L=',I2/)
  630 FORMAT(6X,I3,6E18.7)
      END
```

Method

Suppose that the m-th degree B-spline interpolating function is already obtained by the subroutine BIC1 as follows:

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{4.1}$$

The subroutine, based on Eq. (4.1), obtains an interpolated value, l-th order derivative and/or integral from Eqs. (4.2), (4.3) and (4.4), respectively

$$S(v)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(v) \tag{4.2}$$

$$S^{(l)}(v)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}^{(l)}(v) \tag{4.3}$$

$$I=\int_{x_1}^{v} S(x)dx \tag{4.4}$$

The method of calculating these three values is explained in section 7.1 "Calculating spline function". The subroutine described here performs calculation of $N_{j,m+1}(x)$ and its derivative and integral by using the subroutine UBAS1.

E11-31-0201 BIF2, DBIF2

B-spline interpolation, differentiation and integration (II)
CALL BIF2(X,N,M,C,ISW,V,I,F,VW,ICON)

Function

Given function values $y_i=f(x_i)$, i=1,2,...,n at discrete points $x_1,x_2...,x_n(x_1<x_2<...<x_n)$, and derivatives $y_1^{(l)}=f^{(l)}(x_1)$ and $y_n^{(l)}=f^{(l)}(x_n)$, $l=(m+1)/2,(m+1)/2+1,...,m-1$ at each end point x_1 and x_n then an interpolated value, a derivative at the point $x=v\in[x_1, x_n]$, or an integral from x_1 to v are obtained. However, subroutine BIC2 should be called before using subroutine BIF2 to calculate the interpolating coefficients c_j, $j=-m+1, -m+2, ..., n-1$ in the B-spline interpolating function,

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \qquad (1.1)$$

where m is an odd integer and is the degree of the B-spline $N_{j+1}(x)$ and $x_1\le v\le x_n$, $m\ge3$ and $n\ge(m+1)/2$.

Parameters

X	Input. Discrete points x_i. One-dimensional array of size n.
N	Input. Number of the discrete points, n.
M	Input. Degree of the B-spline, m. See Note.
C	Input. Interpolating coefficients c_j(output from BIC2) One-dimensional array of size $n+m-1$.
ISW ...	Input. An integer which specifies the type of calculation. If ISW=0, interpolated value F=S(v). If ISW=l ($1 \le l \le m$), l-th order derivative F=$S^{(l)}(v)$. If ISW= −1, integral $F=\int_{x_1}^{v} S(x)dx$ are calculated, respectively.
V	Input. The point v at which the interpolated value etc. is to be obtained.
I	Input. Value of i which satisfies $x_i\le v<x_{i+1}$. If $v=x_n$ the parameter should be given n–1. Output. Value of i which satisfies $x_i\le v<x_{i+1}$. See Note.
F	Output. Interpolated value, l-th order derivative or integral. See parameter ISW.
VW	Work area. One-dimensional array of size m+1.
ICON ..	Output. Condition code. See Table BIF2-1.

Table BIF2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	X(I)≤V<X(I+1) is not satisfied.	The I given on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: (a) V<X(1) or V>X(N) (b) ISW<-1 or ISW>M	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAR1 and UBAS1
  FORTRAN basic function ... FLOAT

- Notes
  The subroutine obtains an interpolated value, derivative or integral based on the B-spline interpolating function (1.1) which is obtained by using subroutine BIC2. Therefore, BIC2 must be called before calling this subroutine. Parameters, X, N, M and C must be directly passed from BIC2 to subroutine BIF2.
  Parameters I should satisfy the relationships X(I)≤V<X(I +1). If not, the value I which satisfies the relationship is searched for to continue the processing.

- Example
  By inputting discrete points x_i, function values y_i, i=1,2,...,n, derivatives $y_1^{(l)}$ and $y_n(l)$, $l=(m+1)/2,(m+1)/2+1,...,m-1$, at each end point, and the degree m, the following values are obtained; an integral value from x_1 to $v_{ij}=x_i+(x_{i+1}-x_i)(j/5)$, i=1,2,...,$n-1$, j=0,1,...,5, an interpolated value or derivatives of order one through m. Here $n\le101$, and $m\le5$.

```
C     **EXAMPLE**
      DIMENSION X(101),Y(101),C(105),
     *DY(2,2),VW(527),R(6)
      READ(5,500) N,M
      LM1=(M-1)/2
      READ(5,510) (X(I),Y(I),I=1,N)
     * ,((DY(I,L),I=1,2),L=1,LM1)
      WRITE(6,600) N,M,(I,X(I),Y(I),I=1,N)
      CALL BIC2(X,Y,DY,N,M,C,VW,ICON)
      IF(ICON.EQ.0) GO TO 10
      WRITE(6,610)
      STOP
   10 N1=N-1
      M2=M+2
      DO 40 L2=1,M2
      ISW=L2-2
      WRITE(6,620) ISW
```

```
    DO 30 I=1,N1
    H=(X(I+1)-X(I))/5.0
    XI=X(I)
    DO 20 J=1,6
    V=XI+H*FLOAT(J-I)
    II=I
    CALL BIF2(X,N,M,C,ISW,V,II,F,VW,ICON)
    R(J)=F
 20 CONTINUE
    WRITE(6,630) II,(R(J),J=1,6)
 30 CONTINUE
 40 CONTINUE
    STOP
500 FORMAT(2I6)
510 FORMAT(2F12.0)
600 FORMAT('1'//10X,'INPUT DATA',3X,
   *'N=',I3,3X,'N=',I2//20X,'NO.',10X,
   *'X',17X,'Y'//(20X,I3,2E18.7))
610 FORMAT('0',10X,'ERROR')
620 FORMAT('1'//10X,'L=',I2/)
630 FORMAT(6X,I3,6E18.7)
    END
```

Method

Suppose that the *m*-th degree B-spline interpolating function is already obtained by the subroutine BIC2 as follows:

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{4.1}$$

The subroutine, based on Eq. (4.1), obtains an interpolated value, *l*-th order derivative and/or integral from Eqs. (4.2), (4.3) and (4.4), respectively

$$S(v)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(v) \tag{4.2}$$

$$S^{(l)}(v)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}^{(l)}(v) \tag{4.3}$$

$$I=\int_{x_1}^{v} S(x)dx \tag{4.4}$$

The method of calculating these three values is explained in section 7.1 "Calculating spline function". The subroutine described here performs calculation of $N_{j,m+1}(x)$ and its derivative and integral by using the subroutine UBAS1.

E11-31-0301 BIF3, DBIF3

B-spline interpolation, differentiation and integration (III)
CALL BIF3(X,N,M,C,XT,ISW,V,I,F,VW,ICON)

Function

Given function values $y_i=f(x_i)$, i=1,2,...,n at discrete points $x_1,x_2,...,x_n(x_1<x_2<...<x_n)$, this subroutine obtains interpolated value or derivative at $x=v$ or integral over the interval $[x_1, v]$.
Before using this subroutines, it is necessary a sequence of knots ξ_i, i=1,2,...,n–m+1, and interpolating coefficients c_j, j=–m+1,–m+2,...,n – m of the B-spline interpolation:

$$S(x)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x)$$

have been given by the subroutine BIC3. Where m is an odd number which denotes the degree of B-spline $N_{j,m+1}(x)$.

$x_1 \le v \le x_n$, $m \ge 3$ and $n \ge m+2$

Parameters

X	Input. Discrete points x_i. X is a one-dimensional array of size n.
N	Input. Number of the discrete points n.
M	Input. The degree of the B-spline: m. (See Note)
C	Input. Interpolating coefficients c_j (output from BIC3) C is a one-dimensional array of size n.
XT	Input. The knots ξ_i, (output from DIC3). XT is a one-dimensional array of size n–m+1.
ISW ...	Input. An integer which specifies the type of calculation. When ISW=0, interpolated value $F=S(v)$. When ISW=l (l=1,2,...,m), the derivative of order l: $F=S^{(l)}(v)$, and When ISW=–1, integral $F=\int_{x_1}^{v} S(x)dx$ are calculated, respectively.
V	Input. The points v at which the interpolated value etc. is to be obtained.
I	Input. An integer i which satisfies $x_i \le v < x_{i+1}$. When $v=x_n$ the parameter should be given n–1. Output. An interger i which satisfies $x_i \le v < x_{i+1}$. (See Note.)
F	Output. Interpolated value or derivative of order l or integral. (See ISW)
VW	Work area. VW is a one-dimensional array of size 2m+2.
ICON ..	Output. Condition code. Refer to Table BIF3-1.

Table BIF3-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	X(I)≤V<X(I+1) is not satisfied.	I is searched for in the subroutine and the processing is continued.
30000	1 V<X(1) or V>X(N), or 2 ISW<-1 or ISW>M	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAR1 and UBAS1
  FORTRAN basic function ... FLOAT

- Notes
  The subroutine determines interpolated values or derivative or integral based on the B-spline interpolating functions determined by the subroutine BIC3.
  Therefore, the subroutine BIC3 must be called to determine the interpolating function (1.1) before calling this subroutine to determine interpolated values, etc. Parameters, X, N, Y, C and XT must be identical with those of the BIC3.
  Parameters I should preferably satisfy X(I)≤V<(I+1). If the parameter does not satisfy the condition, I that satisfies X(I)≤V<X(I+1) is searched for to continue processing.

- Example
  Discrete point x_i, function value y_i, i=1,2,...,n and degree m are input, and integrals from x_1, interpolated values and differentials of the first through the m-th degree in $v_{ij}=x_i+(x_{i+1}-x_i)\times(j/5)$, i=1,2,...,n – 1, j=0,1,...,5 are determined. n≤101, n≤5.

```
C      **EXAMPLE**
       DIMENSION X(101),Y(101),C(101),XT(99),
      *VW(507),R(6)
       READ(5,500) N,M
       READ(5,510) (X(I),Y(I),I=1,N)
       WRITE(6,600) N,M,(I,X(I),Y(I),I=1,N)
       CALL BIC3(X,Y,N,M,C,XT,VW,ICON)
       IF(ICON.EQ.0) GO TO 10
       WRITE(6,610)
       STOP
    10 N1=N-1
       M2=M+2
       DO 40 L2=1,M2
       ISW=L2-2
       WRITE(6,620) ISW
       DO 30 I=1,N1
       H=(X(I+1)-X(I))/5.0
       XI=X(I)
    DO 20 J=1,6
    V=XI+H*FLOAT(J-1)
    II=I
    CALL BIF3(X,N,M,C,XT,ISW,V,II,F,
   *          VW,ICON)
    R(J)=F
 20 CONTINUE
    WRITE(6,630) II,(R(J),J=1,6)
 30 CONTINUE
 40 CONTINUE
    STOP
500 FORMAT(2I6)
510 FORMAT(2F12.0)
600 FORMAT('1'//10X,'INPUT DATA',3X,
   *'N=',I3,3X,'N=', I2//20X, 'NO.',10X,
   *'X',17X,'Y'//(20X,I3,2E18.7))
610 FORMAT('0',10X,'ERROR')
620 FORMAT('1'//10X,'L=',I2/)
630 FORMAT(6X,I3,6E18.7)
    END
```

Method

Suppose that the *m*-th degree B-spline interpolating function is already obtained by the subroutine BIC3 as follows:

$$S(x)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x) \tag{4.1}$$

The subroutine, based on Eq. (4.1), obtains an interpolated value, *l*-th order derivative and/or integral by Eqs. (4.2), (4.3) and (4.4), respectively.

$$S(v)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(v) \tag{4.2}$$

$$S^{(l)}(v)=\sum_{j=-m+1}^{n-m} c_j N^{(l)}_{j,m+1}(v) \tag{4.3}$$

$$I=\int_{x_1}^{v} S(x)dx \tag{4.4}$$

The method of calculating these three values is explained in section 7.1 "Calculating spline function".

The subroutine described here performs calculation of $N_{j,m+1}(x)$ and its derivative and integral by using the subroutine UBAS1.

E11-31-0401 BIF4, DBIF4

B-spline interpolation, differentiation and integration (IV)
CALL BIF4(X,N,M,C,ISW,V,I,F,VW,ICON)

Function

Given a periodic function values $y_i=f(x_i)$, $i=1,2,...,n$ (where $y_1=y_n$), of period (x_n-x_1) at the discrete points $x_1,x_2...,x_n (x_1<x_2<...<x_n)$, then an interpolated value, a derivative or an integral from x_1 to v are obtained.

However, subroutine BIC4 must be called before using this subroutine BIF4 to calculate the interpolating coefficients c_j, $j=-m+1,-m+2,...,n-1$ in the B-spline interpolation function,

$$S(x)=\sum_{j=-m+1}^{n-m} c_j N_{j,m+1}(x) \tag{1.1}$$

which satisfies the periodic condition, where m is an odd integer and denotes the degree of the B-spline $N_{j,m+1}(x)$. Here $x_1 \le v \le x_n$, $m \ge 3$ and $n \ge m+2$

Parameters

X	Input. Discrete points x_i. One-dimensional array of size n.
N	Input. Number of the discrete points n.
M	Input. Degree of the B-spline, m. (See Note.)
C	Input. Interpolating coefficients c_j (output from BIC4) One-dimensional array of size $n+m-1$.
ISW ...	Input. An integer which specifies the type of calculation. If ISW=0, the interpolated value $F=S(v)$. If ISW=l $(1 \le l \le m)$, l-th order derivative $F=S^{(l)}(v)$. If ISW=–1, integral $F=\int_{x_1}^{v} S(x)dx$ are calculated, respectively.
V	Input. The point v at which the interpolated value etc. is to be obtained.
I	Input. Value of i which satisfies $x_i \le v < x_{i+1}$. If $v=x_n$ the parameter should be given $n-1$. Output. Value of i which satisfies $x_i \le v \le x_{i+1}$. See Note.
F	Output. Interpolated value, l-th order derivative or integral. See parameter ISW.
VW	Work area. One-dimensional array of size $m+1$.
ICON ..	Output. Condition code. See Table BIF4-1.

Table BIF4-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	X(I)≤V<X(I+1) is not satisfied.	The I given on the left is searched for in the subroutine and the processing is continued.
30000	Either of the followings occurred: (a) V<X(1) or V>X(N) (b) ISW<-1 or ISW>M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAR4, UBAS4 and UPEP4
  FORTRAN basic function ... FLOAT

- Notes
  The subroutine obtains an interpolated values, derivatives or integral based on the B-spline interpolating function (1.1) which is obtained by using subroutine BIC4. Therefore, BIC4 must be called before calling this subroutine. Parameters X, N, M and C must be directly passed from BIC4 to subroutine BIF4.
  Parameters I should satisfy the relationship X(I)≤V<X(I+1). If not, the value of I which satisfies the relationship is searched for to continue the processing.

- Example
  By inputting discrete points x_i, function values y_i, $i=1,2,...,n$ (with period (x_n-x_1)), and degree m, the following values are obtained; integrals form x_1 to $v_{ij}=x_i+(x_{i+1}-x_i)\cdot(j/5)$, $i=1,2,...,n-1$, and j=0,1,2,...,5 interpolated values, and derivatives of order one through m.
  Here $n \le 101$, and $m \le 5$.

```
C     **EXAMPLE**
      DIMENSION X(101),Y(101),C(105),
     *VW(906),R(6)
      READ(5,500) N,M
      READ(5,510) (X(I),Y(I),I=1,N)
      WRITE(6,600) N,M,(I,X(I),Y(I),I=1,N)
      CALL BIC4(X,Y,N,M,C,VW,ICON)
      IF(ICON.EQ.0) GO TO 10
      WRITE(6,610)
      STOP
   10 N1=N-1
      M2=M+2
      DO 40 L2=1,M2
      ISW=L2-2
      WRITE(6,620) ISW
      DO 30 I=1,N1
      H=(X(I+1)-X(I))/5.0
      XI=X(I)
      DO 20 J=1,6
      V=XI+H*FLOAT(J-1)
      II=I
      CALL BIF4(X,N,M,C,ISW,V,II,F,
     *VW,ICON)
      R(J)=F
   20 CONTINUE
      WRITE(6,630) II,(R(J),J=1,6)
   30 CONTINUE
   40 CONTINUE
      STOP
  500 FORMAT(2I6)
  510 FORMAT(2F12.0)
  600 FORMAT('1'//10X,'INPUT DATA',3X,
     *'N=',I3,3X,'N=',I2//20X,'NO.',10X,
     *'X',17X,'Y'//(20X,I3,2E18.7))
  610 FORMAT('0',10X,'ERROR')
  620 FORMAT('1'//10X,'L=',I2/)
  630 FORMAT(6X,I3,6E18.7)
      END
```

Method
Suppose that the *m*-th degree B-spline interpolating function has been already obtained by subroutine BIC4 as follows:

$$S(x)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(x) \tag{4.1}$$

where the $S(x)$ is a periodic function with period (x_n-x_1) which satisfies

$$S^{(l)}(x_1)=S^{(l)}(x_n),\ l=0,1,\ldots,m-1 \tag{4.2}$$

The subroutine, based on Eq. (4.1), obtaines an interpolated value, *l*-th order derivative or integral by using Eqs. (4.3), (4.4) and (4.5), respectively

$$S(v)=\sum_{j=-m+1}^{n-1} c_j N_{j,m+1}(v) \tag{4.3}$$

$$S^{(l)}(v)=\sum_{j=-m+1}^{n-1} c_j N^{(l)}_{j,m+1}(v) \tag{4.4}$$

$$\mathrm{I}=\int_{x_1}^{v} S(x)dx \tag{4.5}$$

The method of calculating these three values is explained in Section 7.1 "Calculating spline function". Subroutine BIF4 perfoms calculation of $N_{j,m+1}(x)$ and its derivatives and integral by using the subroutine UBAS4.

I11-81-1201 BIN, DBIN

Integer order modified Bessel function of the first kind $I_n(x)$
CALL BIN(X,N,BI,ICON)

Function

This subroutine computes integer order modified Bessel function of the first kind

$$I_n(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{2k+n}}{k!(n+k)!}$$

by the Taylor expansion and recurrence formula.

Parameters

X	Input. Independent variable x.
N	Input. Order n of $I_n(x)$.
BI	Output. Function value $I_n(x)$.
ICON ..	Output. Condition code. See Table BIN−1.

When N=0 or N=1, ICON is handled the same as in ICON of BI0 andBI1.

Table BIN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	One of the following was true with respect to the values of X and N: · |X| > 100 · 1/8 ≤ |X| < 1 and |N| ≥ 19 |X|+29 · 1 ≤ |X| < 10 and |N| ≥ 4.7 |X| + 43 · 10 ≤ |X| ≤ 100 and |X| ≤ 1.83 |X| + 71	BI is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... AFMAX, AFMIN, AMACH, BI0, BI1, MGSSL, ULMAX
  FORTRAN basic functions ... ABS, IABS, FLOAT, EXP, MAX0 and SQRT.

- Notes
  The ranges of |X| and |N| are indicated by the null area in Fig. BIN-1. These limits are provided to avoid overflow and underflow in the calculations. (See "Method".)

When calculating $I_0(x)$ and $I_1(x)$, subroutines BI0 and BI1 should be used instead.

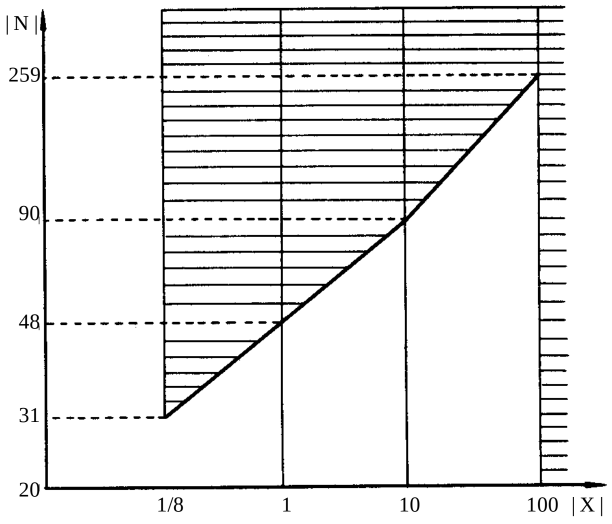


Fig. BIN-1 Calculation range of the arguments

- Example
  The following example generates a table of $I_n(x)$ for range of from 0 to 10 with increment 1and for the range of N from 20 to 30 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 20 N=20,30
       DO 10 K=1,11
       X=K-1
       CALL BIN(X,N,BI,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,N,BI
       IF(ICON.NE.0) WRITE(6,620) X,N,BI,ICON
    10 CONTINUE
    20 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',5X,'N',8X,
      *'IN(X)'/)
   610 FORMAT(' ',F8.2,I5,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'N=',I5,5X,'BI=',E17.7,5X,
      *'CONDITION=',I10)
       END
```

Method

With $|x|$=1/8 as the boundary, the formula used to calculate Bessel function $I_n(x)$ changes.
Since $I_{-n}(x)=I_n(x)$, $I_n(-x)=(-1)^n I_n(x)$, n and x will be used instead of $|n|$ and $|x|$ in the following.

- For $0 \le x < 1/8$
  The computation is based on the Taylor expansion:

$$I_n(x) = \frac{x_n}{2^n n!}\left\{1 + \frac{x^2}{2(2n+2)} + \frac{x^4}{2.4(2n+2)(2n+4)} + \cdots\right\} \tag{4.1}$$

The value of $I_n(x)$ is computed as a partial sum of the first N terms with N being taken large enough so that the last term no longer affects the value of the partial sum significantly.

- For $1/8 \le x \le 100$
  Letting $\boldsymbol{M}$ be a certain integer sufficiently greater than n and x the recurrence formula:

$$F_{k-1}(x) = \frac{2k}{x} F_k(x) + F_{k+1}(x) \tag{4.2}$$

  is evaluated for $k=\boldsymbol{M}, \boldsymbol{M}-1,...,1$ where $F_{M+1}(x)=0$, $F_M(x)=fl_{min}$

Then, $I_n(x)$ is computed as

$$I_n(x) = e^x F_n(x) / \left\{ F_0(x) + 2\sum_{i=1}^{M} F_i(x) \right\} \tag{4.3}$$

[Determination of $\boldsymbol{M}$]

$$M = 2\left[\left\{m_0 + 1 + \max(n - n_0, 0)\right\} / 2\right] \tag{4.4}$$

where [ ] denotes the Gaussian notation, and m_0 is taken as follows:

(a) For $1/8 \le x < 1$
  Single precision: $m_0=4.8x+6.1$, $n_0=3.6x+2.1$ (4.5)
  Double precision: $m_0=9.4x+9.5$
  $n_0=5.0x+4.5$ (4.6)

(b) For $1 \le x < 10$
  Single precision: $m_0=1.4x+9.6$, $n_0=0.9x+5.1$ (4.7)
  Double precision: $m_0=2.5x+15.5$
  $n_0=1.4x+8.6$ (4.8)

(c) For $10 \le x \le 100$
  Single precision: $m_0=0.75x+17.5$
  $n_0=0.28x+11.2$ (4.9)
  Double precision: $m_0=0.77x+32.3$
  $n_0=0.45x+17.5$ (4.10)

For more information, see References [81] and [82].

I11-83-0301 BIR, DBIR

Real order modified Bessel function of the first kind $I_v(x)$
CALL BIR(X,V,BI,ICON)

Function

This subroutine computes the value of real order modified Bessel function of the first kind

$$I_v(x)=\left(\frac{1}{2}x\right)^v\sum_{k=0}^{\infty}\frac{\left(\frac{1}{4}x^2\right)^k}{k!\Gamma(v+k+1)}$$

by using the power series expansion (above expression) and the recurrence formula.

Parameters

X	Input. Independent variable $x(x \geq 0)$.
V	Input. Order v of $I_v(x)$ $(v \geq 0)$.
BI	Output. Value of function $I_v(x)$.
ICON ..	Output. Condition code. See Table BIR-1.

Table BIR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X>log(fl_{max})	BI is set to 0.0.
30000	X<0 or V<0.	BI is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... AFMAX, AMACH, AFMIN, MGSSL, ULMAX
  FORTRAN basic function ... FLOAT, ABS, GAMMA, AMAX1, EXP

- Notes
  $0 \leq X \leq \log(fl_{max})$ and $V \geq 0$.
  When computing $I_0(x)$ and $I_1(x)$, subroutines BI0 and BI1 are used instead.
  When a set of function values $I_v(x)$, $I_{v+1}(x)$, $I_{v+2}(x)$, ..., $I_{v+M}(x)$ is needed at the same time, $I_{v+M}(x)$ and $I_{v+M-1}(x)$ are computed with this subroutine, and next, $I_{v+M-2}(x)$, $I_{v+M-3}(x)$, ..., $I_v(x)$ should be computed in sequence from high order to low order, by using the recurrence formula continuously. Conversely, it should be avoided in computing $I_{v+2}(x)$ and $I_{v+3}(x)$, ..., $I_{v+M}(x)$ by the recurrence formula, after computing $I_v(x)$ and $I_{v+1}(x)$ with this subroutine, in sequence from low order to high order.

- Example
  The following example generates a table of $I_v(x)$ for the range of x from 0 to 10 with increment 1 and for the range of v from 0.4 to 0.6 with increment 0.01.

```
C      **EXAMPLE**
       DO 20 K=1,11
       X=K-1
       DO 10 NV=40,60
       V=FLOAT(NV)/100.0
       CALL BIR(X,V,BI,ICON)
       IF(ICON.EQ.0) WRITE(6,600) X,V,BI
    10 CONTINUE
    20 CONTINUE
       STOP
   600 FORMAT(' ',F8.2,F8.3,E17.7)
       END
```

Method

When it is known that the value of $I_v(x)$ will be the underflowed value, the following computations are skipped and the result 0.0 is output.
The computation of $I_v(x)$ depends on the range of x.

- $0 \leq x \leq 1$
  With the power series expansion

$$I_v(x)=\left(\frac{1}{2}x\right)^v\sum_{k=0}^{\infty}\frac{\left(\frac{1}{4}x^2\right)^k}{k!\Gamma(v+k+1)} \tag{4.1}$$

, it is computed until the k-th term is less than the unit round-off in relative to the first term.

- $1<x\leq\log(fl_{max})$
  Recurrence formula is used.
  Let's suppose that m is a sufficiently large integer (determined by x, v, and the desired precision), and that δ is a sufficiently small constant (positive smallest number allowed for the computer used) and moreover that n and α are determined by

  $v=n+\alpha(n;\text{integer},0 \leq \alpha < 1)$

  Initial values

  $G_{\alpha+m+1}(x)=0,\ G_{\alpha+m}(x)=\delta$

  are set, and recurrence formula

$$G_{\alpha+k-1}(x)=\frac{2(\alpha+k)}{x}G_{\alpha+k}(x)+G_{\alpha+k+1}(x) \tag{4.2}$$

is repeatedly applied to $k=m,m-1,...,1$. Then the value of function $I_v(x)$ is obtained as

$$I_v(x)\approx\frac{1}{2}\left(\frac{x}{2}\right)^{\alpha}\frac{\Gamma(2\alpha+1)}{\Gamma(\alpha+1)}e^xG_{\alpha+n}(x)\Bigg/\left(\sum_{k=0}^{m}\frac{(\alpha+k)\Gamma(2\alpha+k)}{k\,!}G_{\alpha+k}(x)\right) \tag{4.3}$$

For the method of determining of m and other details, see Reference [81].

I11-81-0601 BI0,DBI0

Zero order modified Bessel function of the first kind $\boldsymbol{I}_0(x)$
CALL BI0(X,BI,ICON)

Function

This subroutine computes the zero order modified Bessel function of the first kind $I_0(x)$

$$I_0(x)=\sum_{k=0}^{\infty}\frac{(x/2)^{2k}}{(k!)^2}$$

by polynomial approximations and the asymptotic expansion.

Parameters

X	Input. Independent variable x.
BI	Output. Function value $I_0(x)$.
ICON ..	Output. Condition code. See Table BI0-1.

Table BI0-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert X\rvert > \log(fl_{max})$	BI is set to fl_{max}.

Comments on use

- Subprograms used
  SSL II ... AFMAX, MGSSL, ULMAX
  FORTRAN basic function ... ABS, EXP, and SQRT

- Notes
  [The range of argument X]
  $\lvert X\rvert \le \log(fl_{max})$
  If |X| exceeds the limits, an overflow will occur during the calculation of e^x. This limit is provided for that reason. (See "Method".)

- Example
  The following example generates a table of $I_0(x)$ from 0 to 100 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       X=K-1
       CALL BI0(X,BI,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BI
       IF(ICON.NE.0) WRITE(6,620) X,BI,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',9X,'I0(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BI=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With $\lvert \boldsymbol{X}\rvert = 8$ as the boundary, the approximation formula used to calculate modified Bessel function $I_0(x)$ changes. Since $I_0(-x)=I_0(x)$, x is used instead of $\lvert x\rvert$ in the following.

- For $0\le x<8$
  The power series expansion of $I_0(x)$

$$I_0(x)=\sum_{k=0}^{\infty}\frac{(x/2)^{2k}}{(k!)^2} \tag{4.1}$$

is calculated using the approximation formulas in (4.2) and (4.3).
Single precision:

$$I_0(x)=\sum_{k=0}^{11}a_k x^{2k} \tag{4.2}$$

Double precision:

$$I_0(x)=\sum_{k=0}^{16}a_k x^{2k} \tag{4.3}$$

- For $8 \le x \le \log(fl_{max})$
  The asymptotic expansion of $I_0(x)$

$$I_0(x)=\frac{e^x}{\sqrt{2\pi x}}\left\{1+\frac{1}{8x}+\frac{1^2\cdot 3^2}{2!}\cdot\frac{1}{(8x)^2}+\frac{1^2\cdot 3^2\cdot 5^2}{3!}\cdot\frac{1}{(8x)^3}+\cdots\right\} \tag{4.4}$$

is calculated using the approximation formulas in (4.5) and (4.6)
Single precision:

$$I_0(x)=\frac{e^x}{\sqrt{x}}\left(\sum_{k=0}^{5}a_k z^k\right), \quad z=8/x \tag{4.5}$$

Double precision:

$$I_0(x)=\frac{e^x}{\sqrt{x}}\left(\sum_{k=0}^{11}a_k z^k \Big/ \sum_{k=0}^{11}b_k z^k\right), \quad z=(x-8)/x \tag{4.6}$$

I11-81-0701 BI1,DBI1

First order modified Bessel function of the first kind $I_1(x)$
CALL BI1(X,BI,ICON)

Function

This subroutine computes the first order modified Bessel function of the first kind

$$I_1(x)=\sum_{k=0}^{\infty}\frac{(x/2)^{2k+1}}{(k!)(k+1)!}$$

by polynomial approximations and the asymptotic expansion.

Parameters

X	Input. Independent variable x.
BI	Output. Function value $I_1(x)$.
ICON ..	Output. Condition code. See Table BI1-1.

Table BI1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X>log(fl_{max}) or X<log(fl_{max})	BI is set to fl_{max} or BI is set to $-fl_{max}$.

Comments on use

- Subprograms used
  SSL II ... AFMAX, MGSSL, ULMAX
  FORTRAN basic function ... ABS, EXP, and SQRT

- Notes
  [Range of argument X]
  |X| ≤ log(fl_{max})
  If |X| exceeds the above limits, an overflow will occur in the calculation of x. This limit is provided for that reason. (See "Method".)

- Example
  The following example generates a table of $I_1(x)$ from 0 to 100 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       X=K-1
       CALL BI1(X,BI,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BI
       IF(ICON.NE.0) WRITE(6,620) X,BI,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',9X,'I1(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BI=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With $|x|$ as the boundary, the approximation formula used to calculate the modified Bessel function $I_1(x)$ changes. Since $I_1(-x)= -I_1(x)$, x is used instead of $|x|$ in the following.

- For $0\le x<8$
  The power series expansion of $I_1(x)$

$$I_1(x)=\sum_{k=0}^{\infty}\frac{(x/2)^{2k+1}}{(k!)(k+1)!} \tag{4.1}$$

is calculated using the approximation formulas in (4.2) and (4.3).
Single precision:

$$I_1(x)=\sum_{k=0}^{10}a_k x^{2k+1} \tag{4.2}$$

Double precision:

$$I_1(x)=\sum_{k=0}^{16}a_k x^{2k+1} \tag{4.3}$$

- For $8 \le x \le \log(fl_{max})$
  The asymptotic expansion of $I_1(x)$

$$I_1(x)=\frac{e^x}{\sqrt{2\pi x}}\left\{1-\frac{3}{8x}+\frac{3\cdot(-5)}{2!}\cdot\frac{1}{(8x)^2}\right.\\ \left.-\frac{3\cdot(-5)\cdot(-21)}{3!}\cdot\frac{1}{(8x)^3}+\cdots\right\} \tag{4.4}$$

is calculated using the approximation formulas in (4.5) and (4.6)
Single precision:

$$I_1(x)=\frac{e^x}{\sqrt{x}}\left(\sum_{k=0}^{5}a_k z^k\right), z=8/x \tag{4.5}$$

Double precision:

$$I_1(x)=\frac{e^x}{\sqrt{x}}\left(\sum_{k=0}^{11}a_k z^k \Big/ \sum_{k=0}^{11}b_k z^k\right), z=(x-8)/x \tag{4.6}$$

I11-81-1001 BJN, DBJN

Integer order Bessel function of the first kind $J_n(x)$
CALL BJN(X,N,BJ,ICON)

Function

This subroutine computes integer order Bessel function of the first kind

$$J_n(x)=\sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{2k+n}}{(k!)(n+k)!}$$

by Taylor expansion and the asymptotic expansion.

Parameters

X ..... Input. Independent variable x.
N ..... Input. Order n of $J_n(x)$.
BJ .... Output. Function value $J_n(x)$.
ICON .. Output. Condition code. See Table BJN-1. When N=0, or N=1, the same handling as for the ICON of BJ0 and BJ1 applies.

Table BJN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	One of the following conditions was true with respect to the value of X and N. · |X| > 100 · 1/8 ≤ |X| < 1 and |N| ≥ 19 |X| + 29 · 1 ≤ |X| < 10 and |N| ≥ 4.7 |X| + 43 · 10 ≤ |X| ≤ 100 and |N| ≥ 1.83 |X| + 71	BJ is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... AFMIN, AMACH, BJ0, BJ1, MGSSL and UTLIM
  FORTRAN basic function ... ABS, IABS, FLOAT, MAX0, DSIN, DCOS, and DSQRT

- Notes
  The range of |X| and |N| are indicated by the null area in Fig. BJN-1. These limits are provided to avoid overflow and underflow during calculations (refer to "Method").

When calculating $J_0(x)$ and $J_1(x)$, subroutines BJ0 and BJ1 should be used instead.

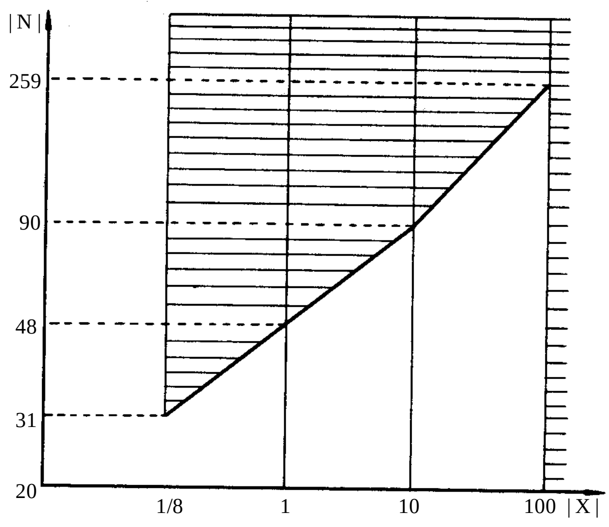


Fig. BJN-1 Calculation range of the arguments

- Example
  The following example generates a table of $J_n(x)$ for the range of x from 0.0 to 10.0 with increment 1.0 and for the range of N from 20 to 30 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 20 N=20,30
       DO 10 K=1,11
       X=K-1
       CALL BJN(X,N,BJ,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,N,BJ
       IF(ICON.NE.0) WRITE(6,620) X,N,BJ,ICON
    10 CONTINUE
    20 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',5X,'N',8X,
      *'JN(X)'/)
   610 FORMAT(' ',F8.2,I5,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'N=',I5,5X,'BJ=',E17.7,5X,
      *'CONDITION=',I10)
       END
```

Method

With $|x|$ = 1/8 as the boundary, the calculation formula of Bessel function $J_n(x)$ changes.
Since $J_{-n}(x)=J_n(-x)=(-1)^nJ_n(x)$, n and x will be used instead of $|n|$ and $|x|$ in the following.

- For $0\le x<1/8$
  The computation is based on the Taylor expansion:

$$J_n(x)=\frac{x^n}{2^n n!}\left\{1-\frac{x^2}{2(2n+2)}+\frac{x^4}{2\cdot 4(2n+2)(2n+4)}-\cdots\right\} \quad (4.1)$$

The value of $J_n(x)$ is computed as a partial sum of the first N terms with N being taken large enough so that the last term no longer affects the value of the partial sum significantly.

- For $1/8 \le x \le 100$
  Letting $\boldsymbol{M}$ be a certain integer sufficiently greater than n and x the recurrence formula:

$$F_{k-1}(x) = \frac{2k}{x} F_k(x) - F_{k+1}(x) \qquad (4.2)$$

is evaluated for $k=\boldsymbol{M},\boldsymbol{M}-1,...,1$, where $F_{M+1}(x)=0$, $F_M(x)=fl_{min}$.
Then, $J_n(x)$ is computed as

$$J_n(x) = F_n(x) \Big/ \left\{ F_0(x) + 2\sum_{i=1}^{[M/2]} F_{2i}(x) \right\} \qquad (4.3)$$

[Determination of $\boldsymbol{M}$]

$$M = 2\left[\left\{ m_0 + 1 + \max\left(n - \frac{m_0 + x}{2}, 0 \right) \right\} / 2\right] \qquad (4.4)$$

where [ ] denotes the Gaussian notation, and m_0 is taken as follows:

(a) For $1/8 \le x < 1$
  Single precision: $m_0=5.5x+5$ (4.5)
  Double precision: $m_0=8x+10$ (4.6)

(b) For $1 \le x < 10$
  Single precision: $m_0=1.8x+9$ (4.7)
  Double precision: $m_0=2x+19$ (4.8)

(c) For $10 \le x \le 100$
  Single precision: $m_0=1.25x+18$ (4.9)
  Double precision: $m_0=1.3x+34$ (4.10)

For more information, see References [81] and [82].

I11-83-0101 BJR, DBJR

Real order Bessel function of the first kind $J_v(x)$
CALL BJR(X,V,BJ,ICON)

Function

This subroutine evaluates real order Bessel function of the first kind

$$J_v(x) = \left(\frac{1}{2}x\right)^v \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{4}x^2\right)^k}{k!\Gamma(v+k+1)}$$

by using power series expansion (above expression), recurrence formula, and asymptotic expansion.

Parameters

X	Input. Independent variable x ($x \geq 0$).
N	Input. Order v of $J_v(x)$ ($v \geq 0$).
BJ	Output. Value of function $J_v(x)$.
ICON ..	Output. Condition code. See Table BJR-1.

Table BJR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Any of these errors. · X>100 and V>15 · X≥t_{max}	BJ is set to 0.0.
30000	X<0 or V<0	BJ is set to 0.0.

Comments on use

- Subprograms used

  SSL II ... AMACH, AFMIN, MGSSL, UTLIM

  FORTRAN basic function ... FLOAT, ABS, GAMMA, AMAX1, MOD, SQRT, COS, SIN

- Notes

  X≥0, V≥0.

  X, V, must be within in the range shown as the white part in Fig. BJR-1. And, X<t_{max} are required since values of sin(X–(1/2·V+1/4)π) and cos(X–(1/2·V+1/4)π) in the asymptotic expansion are not computed accurately when X is great. See Method (4.4) expression.

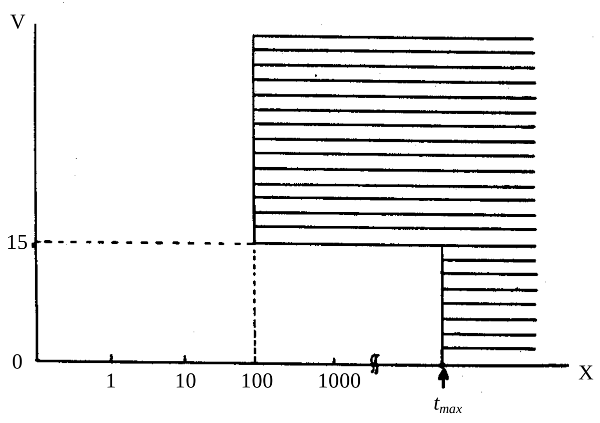


Fig. BJR-1 Argument range

To evaluate $J_0(x)$ or $J_1(x)$, it is better to use BJ0 or BJ1 respectively rather than this subroutine.

When a set of function values $J_v(x)$, $J_{v+1}(x)$, $J_{v+2}(x)$, ..., $J_{v+M}(x)$ is needed at the same time, $J_{v+M}(x)$ and $J_{v+M-1}(n)$ are computed with this subroutine first, and next, $J_{v+M-2}(x)$, $J_{v+M-3}(x)$, ..., $J_v(x)$ should be computed in sequence from high order to low order, by using the recurrence formula continuously. Conversely, it should be avoided in computing $J_{v+2}(x)$, $J_{v+3}(x)$, ..., $J_{v+M}(x)$ by recurrence formula after computing $J_v(x)$ and $J_{v+1}(x)$ with this subroutine, in sequence from low order to high order.

- Example

  The following example generates a table of $J_v(x)$ for the range of x from 0 to 10 with increment 1 and for the range of v from 0.4 to 0.6 with increment 0.01.

```
C      **EXAMPLE**
       DO 20 K=1,11
       X=K-1
       DO 10 NV=40,60
       V=FLOAT(NV)/100.0
       CALL BJR(X,V,BJ,ICON)
       IF(ICON.EQ.0) WRITE(6,600) X,V,BJ
    10 CONTINUE
    20 CONTINUE
       STOP
   600 FORMAT(' ',F8.2,F8.3,E17.7)
       END
```

Method

When it is known that the value of $J_v(x)$ will underflow (has a value less than about 10^{-75}), the following computations are skipped and the result 0.0 is output.

Different computations of $J_v(x)$ are used corresponding to ranges of x and v.

- $0 \le x < 1$
  With the power series expansion

$$J_v(x) = \left(\frac{1}{2}x\right)^v \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{4}x^2\right)^k}{k!\,\Gamma(v+k+1)} \qquad (4.1)$$

, it is computed until the k-th term becomes less than unit round-off, in relative to the first term.

- Single precision: $1 \le x \le 16$, or $16 < x \le 100$ and $v > 0.115x+4$
  Double precision: $1 \le x < 30$, or $30 \le x \le 100$ and $v > 0.115x+4$
  The recurrence formula is used for the computation.
  Let's suppose that m is a sufficiently large integer (determined by x, v, and the desired precision), and that δ is a sufficiently small constant (positive smallest number allowed for the computer used), and moreover that n and α are determined by

$v = n + \alpha$ (n: integer, $0 \le \alpha < 1$)

Initial values

$$F_{\alpha+m+1}(x) = 0, F_{\alpha+m}(x) = \delta$$

are set, and recurrence formula

$$F_{\alpha+k-1}(x) = \frac{2(\alpha+k)}{x} F_{\alpha+k}(x) - F_{\alpha+k+1}(x) \qquad (4.2)$$

is repeatedly applied to $k = m, m-1, \ldots, 1$. Then the value of function $\boldsymbol{J}_v(x)$ is obtained as

$$J_v(x) \approx \left(\frac{x}{2}\right)^\alpha F_{\alpha+n}(x) \Big/ \left(\sum_{k=0}^{[m/2]} \frac{(\alpha+2k)\Gamma(\alpha+k)}{k!} \times F_{\alpha+2k}(x) \right) \qquad (4.3)$$

For the method of determining of m and other details, see Reference [81].

- Single precision: $100 < x < t_{max}$ and $v \le 15$, or $16 < x \le 100$ and $v \le 0.115x+4$
  Double precision: $100 < x < t_{max}$ and $v \le 15$, or $30 \le x \le 100$ and $v \le 0.115x+4$
  The asymptotic expansion

$$J_v(x) = \sqrt{\frac{2}{\pi x}} \left\{ P(x,v)\cos\left(x - \left(\frac{1}{2}v + \frac{1}{4}\right)\pi\right) - Q(x,v)\sin\left(x - \left(\frac{1}{2}v + \frac{1}{4}\right)\pi\right) \right\} \qquad (4.4)$$

is used for the computation. Where

$$P(x,v) = \sum_{k=0}^{\infty} (-1)^k \frac{(v,2k)}{(2x)^{2k}} \qquad (4.5)$$

$$Q(x,v) = \sum_{k=0}^{\infty} (-1)^k \frac{(v,2k+1)}{(2x)^{2k}} \qquad (4.6)$$

$$(v,k) = \frac{\left(v^2 - \left(\frac{1}{2}\right)^2\right)\left(v^2 - \left(\frac{3}{2}\right)^2\right) \cdots\cdots \left(v^2 - \left(\frac{2k-1}{2}\right)^2\right)}{k!} \qquad (k \ne 0)$$

$$(v,0) = 1$$

P(x,v) and Q(x,v) is computed until each k-th term relative to the first term is less than Single precision: max (unit round-off, 10^{-10}) Double precision: max (unit round-off, 10^{-20})

I11-81-0201 BJ0,DBJ0

Zero order Bessel function of the first kind $\boldsymbol{J}_0(x)$
CALL BJ0(X,BJ,ICON)

Function

This subroutine computes zero order Bessel function of the first kind

$$J_0(x)=\sum_{k=0}^{\infty}\frac{(-1)^k (x/2)^{2k}}{(k!)^2}$$

by rational approximations and the asymptotic expansion.

Parameters

X	Input. Independent variable x.
BJ	Output. Function value $\boldsymbol{J}_0(x)$.
ICON ..	Output. Condition code. See Table BJ0-1

Table BJ0-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert X\rvert \geq t_{max}$	BJ is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... ABS, DSIN, DCOS, and DSQRT

- Notes
  [Range of argument]
  $\lvert X\rvert \leq t_{max}$
  The limits are set since $\sin(x-\pi/4)$ and $\cos(x-\pi/4)$ lose accuracy if |X| becomes large.
  (See (4.4) in the Method section.)

- Example
  The following example generates a table of $\boldsymbol{J}_0(x)$ from 0.0 to 100.0 with increment 1.0.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       X=K-1
       CALL BJ0(X,BJ,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BJ
       IF(ICON.NE.0) WRITE(6,620) X,BJ,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',9X,'J0(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BJ=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With |x|=8 as the boundary, the form of the approximation formula used for calculating Bessel function $\boldsymbol{J}_0(x)$ changes. Since $\boldsymbol{J}_0(-x)=\boldsymbol{J}_0(x)$, x is used instead of |x| in the following.

- For $0 \leq x \leq 8$
  The expansion of $\boldsymbol{J}_0(x)$ into power series

$$J_0(x)=\sum_{k=0}^{\infty}\frac{(-1)^k (x/2)^{2k}}{(k!)^2} \tag{4.1}$$

is calculated using the following rational approximations.
Single precision:

$$J_0(x)=\sum_{k=0}^{5}a_k x^{2k} \Big/ \sum_{k=0}^{5}b_k x^{2k} \tag{4.2}$$

Theoretical precision = 8.54 digits

Double precision:

$$J_0(x)=\sum_{k=0}^{8}a_k x^{2k} \Big/ \sum_{k=0}^{8}b_k x^{2k} \tag{4.3}$$

Theoretical precision = 19.22 digits

- For x>8
  The asymptotic expansion of $\boldsymbol{J}_0(x)$

$$\begin{aligned} J_0(x) = \sqrt{\frac{2}{\pi x}}\{P_0(x)\cos(x-\pi/4) \\ - Q_0(x)\sin(x-\pi/4)\} \end{aligned} \tag{4.4}$$

is evaluated through use of the following approximate expressions of $P_0(x)$ and $Q_0(x)$:
Single precision:

$$P_0(x)=\sum_{k=0}^{2}a_k z^{2k} \Big/ \sum_{k=0}^{2}b_k z^{2k}, z=8/x \tag{4.5}$$

Theoretical precision = 10.66 digits

$$Q_0(x)=\sum_{k=0}^{1}c_k z^{2k+1} \Big/ \sum_{k=0}^{2}d_k z^{2k}, z=8/x \tag{4.6}$$

Theoretical precision = 9.58 digits

BJ0

Double precision:

$$P_0(x) = \sum_{k=0}^{5} a_k z^{2k} \Big/ \sum_{k=0}^{5} b_k z^{2k}, z = 8/x \qquad (4.7)$$

Theoretical precision = 18.16 digits

$$Q_0(x) = \sum_{k=0}^{5} c_k z^{2k+1} \Big/ \sum_{k=0}^{5} d_k z^{2k}, z = 8/x \qquad (4.8)$$

Theoretical precision = 18.33 digits

For more information, see Reference [78]pp.141~149.

I11-81-0301 BJ1, DBJ1

First order Bessel function of the first kind $\boldsymbol{J}_1(x)$
CALL BJ1(X,BJ,ICON)

Function

This subroutine computes first order Bessel function of the first kind

$$J_1(x)=\sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{2k+1}}{k!(k+1)!}$$

by rational approximations and the asymptotic expansion.

Parameters

X	Input. Independent variable x.
BJ	Output. Function value $\boldsymbol{J}_1(x)$.
ICON ..	Output. Condition code. See Table BJ1-1

Table BJ1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert X\rvert \geq t_{max}$	BJ is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL, UTLIM
  FORTRAN basic function ... ABS, DSIN, DCOS, and DSQRT

- Notes
  [Range of argument]
  $|X| \leq t_{max}$
  The range limits are set since $\sin(x-3\pi/4)$ and $\cos(x-3\pi/4)$ lose accuracy if |X| becomes too large. (See "Method".)

- Example
  The following example generates a table of $\boldsymbol{J}_1(x)$ from 0.0 to 100.0 with increment 1.0.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,101
      X=K-1
      CALL BJ1(X,BJ,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,BJ
      IF(ICON.NE.0) WRITE(6,620) X,BJ,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF BESSEL ',
     *'FUNCTION'///6X,'X',9X,'J1(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     *E17.7,5X,'BJ=',E17.7,5X,'CONDITION=',
     *I10)
      END
```

Method

With $|x|=8$ as the boundary, the form of the approximation used to calculate Bessel function $\boldsymbol{J}_1(x)$ changes. Since $\boldsymbol{J}_1(-x)=-\boldsymbol{J}_1(x)$, x is used instead of $|x|$ in the following.

- For $0 \leq x \leq 8$
  The expansion of $\boldsymbol{J}_1(x)$ into power series

$$J_1(x)=\sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{2k+1}}{k!(k+1)!} \quad (4.1)$$

is calculated using the following rational approximations.
Single precision:

$$J_1(x)=\sum_{k=0}^{4}a_kx^{2k+1}\Big/\sum_{k=0}^{5}b_kx^{2k} \quad (4.2)$$

Theoretical precision = 8.19 digits
Double precision:

$$J_1(x)=\sum_{k=0}^{7}a_kx^{2k+1}\Big/\sum_{k=0}^{8}b_kx^{2k} \quad (4.3)$$

Theoretical precision = 18.68 digits

- For $x>8$
  The asymptotic expansion of $\boldsymbol{J}_1(x)$

$$J_1(x)=\sqrt{\frac{2}{\pi x}}\{P_1(x)\cos(x-3\pi/4) - Q_1(x)\sin(x-3\pi/4)\} \quad (4.4)$$

is evaluated through use of the following approximate expressions of $P_0(x)$ and $Q_0(x)$:
Single precision:

$$P_1(x)=\sum_{k=0}^{2}a_kz^{2k}\Big/\sum_{k=0}^{2}b_kz^{2k}, z=8/x \quad (4.5)$$

Theoretical precision = 10.58 digits

$$Q_1(x)=\sum_{k=0}^{1}c_kz^{2k+1}\Big/\sum_{k=0}^{2}d_kz^{2k}, z=8/x \quad (4.6)$$

Theoretical precision = 9.48 digits

Double precision:

$$P_1(x) = \sum_{k=0}^{5} a_k z^{2k} \Bigg/ \sum_{k=0}^{5} b_k z^{2k}, z = 8/x \tag{4.7}$$

Theoretical precision = 18.11 digits

$$Q_1(x) = \sum_{k=0}^{5} c_k z^{2k+1} \Bigg/ \sum_{k=0}^{5} d_k z^{2k}, z = 8/x \tag{4.8}$$

Theoretical precision = 18.28 digits
For more information, see Reference [78]pp.141~149.

I11-81-1301 BKN, DBKN

Integer order modified Bessel function of the second kind $K_n(x)$
CALL BKN(X,N,BK,ICON)

Function

This subroutine computes integer order modified Bessel function of the second kind

$$K_n(x) = (-1)^{n+1} I_n(x)\{\gamma + \log(x/2)\}$$
$$+\frac{1}{2}\sum_{k=0}^{n-1}\frac{(-1)^k(n-k-1)!}{k!}(x/2)^{2k-n}$$
$$+\frac{(-1)^n}{2}\sum_{k=0}^{\infty}\frac{(x/2)^{n+2k}}{k!(n+k)!}\left(\sum_{m=1}^{k}1/m+\sum_{m=1}^{k+n}1/m\right)$$

for x>0 by the recurrence formula, where, $\boldsymbol{I}_n(x)$ is integer order modified Bessel function of the first kind, and γ denotes the Euler's constant, and also the assumption $\sum_{m=1}^{0}1/m = 0$ is made.

Parameters

X	Input. Independent variable x.
N	Input. Order n of $\boldsymbol{K}_n(x)$.
BK	Output. Function value $\boldsymbol{K}_n(x)$.
ICON ..	Output. Condition code. See Table BKN-1.

When N=0 or N=1, ICON is handled the same as the ICON of BK0 and BK1.

Table BKN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	N>log(fl_{max})	BK is set to 0.0.
30000	X≤0	BK is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... BK0, BK1, MGSSL, and ULMAX
  FORTRAN basic function ... IABS, FLOAT, ALOG, EXP, and SQRT
- Notes
  [Range of the argument X]
  0<X≤log(fl_{max})
  If X is outside of the above range, overflow and underflow will occur in the calculation of e^{-x}. The limit is provided for that reason. (See (4.4) and (4.5) in the Method sections of BK0 and BK1.)
  When calculating $\boldsymbol{K}_0(x)$ and $\boldsymbol{K}_1(x)$, BK0 and BK1 should be used.

- Example
  The following example generates a table of $\boldsymbol{K}_n(x)$ for the range of x from 1 to 10 with increment 1 and for the range of N from 20 to 30 with increment 1.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 20 N=20,30
      DO 10 K=1,10
      X=K
      CALL BKN(X,N,BK,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,N,BK
      IF(ICON.NE.0) WRITE(6,620) X,N,BK,ICON
   10 CONTINUE
   20 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF BESSEL ',
     *'FUNCTION'//6X,'X',5X,'N',8X,
     *'KN(X)'/)
  610 FORMAT(' ',F8.2,I5,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     *E17.7,5X,'N=',I5,5X,'BK=',E17.7,5X,
     *'CONDITION=',I10)
      END
```

Method

Bessel function $K_n(x)$ is calculated using the following recurrence formula

$$K_{k+1}(x) = \frac{2k}{x}K_k(x) + K_{k-1}(x), k = 1,2,...,n-1 \qquad (4.1)$$

where, both $\boldsymbol{K}_0(x)$ and $\boldsymbol{K}_1(x)$ are calculated by using BK0 and BK1.

I11-83-0401 BKR, DBKR

Real order modified Bessel function of the second kind $\boldsymbol{K}_v(x)$
CALL BKR(X,V,BK,ICON)

Function

This subroutine computes real order modified Bessel function of the second kind:

$$K_v(x)=\frac{\pi}{2}\frac{I_{-v}(x)-I_v(x)}{\sin(v\pi)}$$

by the method by Yoshida and Ninomiya.
$\boldsymbol{I}_v(x)$ is modified Bessel function of the first kind.
Where $x>0$.

Parameters

X	Input. Independent variable x.
V	Input. Order v of $\boldsymbol{K}_v(x)$.
BK	Output. Value of function $\boldsymbol{K}_v(x)$.
ICON ..	Output. Condition code.

See Table BKR-1.

Table BKR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X=0.0. Or BK was large enough to overflow.	The maximum value of the floating point is output to BK.
30000	X<0.0	BK=0.0.

Comments on use

- Subprograms used
  SSL II ... AMACH, AFMAX, MGSSL, ULMAX
  FORTRAN basic functions ... FLOAT, ALOG, AMAX1, ALGAMA, GAMMA, ABS, SQRT, EXP

- Notes
  X>0.0 must be satisfied.
  When computing $\boldsymbol{K}_0(x)$ or $\boldsymbol{K}_1(x)$, BK0 or BK1 should be used for efficiency instead of this subroutine.
  When the values of $\boldsymbol{K}_v(x)$, $\boldsymbol{K}_{v+1}(x)$, $\boldsymbol{K}_{v+2}(x)$,..., $\boldsymbol{K}_{v+M}(x)$ are required at one time, first obtain $\boldsymbol{K}_v(x)$ and $\boldsymbol{K}_{v+1}(x)$ by this subroutine and obtain others in sequence from low order to high order as $\boldsymbol{K}_{v+2}(x)$, $\boldsymbol{K}_{v+3}(x)$, ..., $\boldsymbol{K}_{v+M}(x)$.
  When the subroutine is called repeatedly with a fixed value of v but with various, large values of x in magnitude, the subroutine computes $\boldsymbol{K}_v(x)$ efficiently by bypassing a common part of computation.

- Example
  The following example generates a tale of $\boldsymbol{K}_v(x)$ for the range of x from 1 to 10 with increment 1 and for the range of v from 0.4 to 0.6 with increment 0.01.

```
C       **EXAMPLE**
        DO 20 NV=40,60
        V=FLOAT(NV)/100.0
        DO 10 K=1,10
        X=FLOAT(K)
        CALL BKR(X,V,BK,ICON)
        WRITE(6,600) V,X,BK,ICON
   10   CONTINUE
   20   CONTINUE
        STOP
  600   FORMAT(' ',F8.3,F8.2,E17.7,I7)
        END
```

Method

The vth-order modified Bessel function of the second kind $\boldsymbol{K}_v(x)$ is defined as

$$K_v(x)=\frac{\pi}{2}\frac{I_{-v}(x)-I_v(x)}{\sin(v\pi)} \tag{4.1}$$

using modified Bessel function of the first kind $\boldsymbol{I}_v(x)$ and $\boldsymbol{I}_{-v}(x)$. When the order v is an integer n, the function is defined as the limit as $v\rightarrow n$.
Since

$$\boldsymbol{K}_{-v}(x)=\boldsymbol{K}_v(x) \tag{4.2}$$

holds, computation is required only for $v\geq0$.
This subroutine directly computes the value of $\boldsymbol{K}_v(x)$ when $0\leq\mu\leq2.5$.
When $v\geq2.5$, let μ denote the fractional part of v.
When $\mu\geq0.5$, this subroutine directly obtains $\boldsymbol{K}_{\mu+1}(x)$ and $\boldsymbol{K}_{\mu+2}(x)$ and when $\mu>0.5$, it directly obtains $\boldsymbol{K}_{\mu}(x)$ and $\boldsymbol{K}_{\mu+1}(x)$. And then it computes the value of $\boldsymbol{K}_v(x)$ by

$$K_{v+1}(x)=\frac{2v}{x}K_v(x)+K_{v-1}(x) \tag{4.3}$$

The Yoshida-Ninomiya method for the computation of $\boldsymbol{K}_v(x)$ when $0\leq v\leq2.5$ is explained below.
The method for computing $\boldsymbol{K}_v(x)$ depends on x and v. In Fig. BKR-1, one method is used in the domain A and another in the domain B.

- Method in the domain A
  The method here uses the series expansions of $\boldsymbol{I}_{-v}(x)$ and $\boldsymbol{I}_v(x)$.

$$I_{-v}(x)=\left(\frac{x}{2}\right)^{-v}\left\{\frac{1}{\Gamma(1-v)}+\frac{\left(\frac{x}{2}\right)^2}{1!\Gamma(2-v)}+\frac{\left(\frac{x}{2}\right)^4}{2!\Gamma(3-v)}+\cdots\right\} \tag{4.4}$$

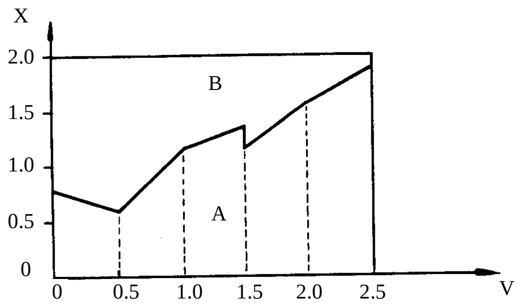


Fig. BKR-1 Domain A and B

$$I_v(x)=\left(\frac{x}{2}\right)^v\left\{\frac{1}{\Gamma(1+v)}+\frac{\left(\frac{x}{2}\right)^2}{1!\Gamma(2+v)}+\frac{\left(\frac{x}{2}\right)^4}{2!\Gamma(3+v)}+\cdots\right\} \quad (4.5)$$

$\phi_1(v,x)$ and $\phi_2(v,x)$ are defined, for later use, as

$$\left.\begin{aligned}\phi_1(v,x)&=\frac{\left(\frac{x}{2}\right)^{-v}-1}{v}\\ \phi_2(v,x)&=\frac{1-\left(\frac{x}{2}\right)^{v}}{v}\end{aligned}\right\} \quad (4.6)$$

The approximation formula depends on whether $0\le v\le 0.5$, $0.5<v\le 1.5$ or $1.5<v\le 2.5$.

Here, an example with $0\le v\le 0.5$ is explained.

From (4.4), (4.5) and (4.6), the following equation can be obtained:

$$I_{-v}(x)-I_v(x)=v\sum_{k=0}^{\infty}\{A_k(v,x)+B_k(v,x)\} \quad (4.7)$$

where,

$$\left.\begin{aligned}A_k(v,x)&=\left(\frac{x}{2}\right)^{2k}\left\{\frac{1}{k!v}\left(\frac{1}{\Gamma(k+1-v)}-\frac{1}{\Gamma(k+1+v)}\right)\right\}\\ B_k(v,x)&=\left(\frac{x}{2}\right)^{2k}\left\{\frac{1}{k!}\left(\frac{\phi_1(v,x)}{\Gamma(k+1-v)}+\frac{\phi_2(v,x)}{\Gamma(k+1+v)}\right)\right\}\end{aligned}\right\} \quad (4.8)$$

When these values are computed, cancellation of digits may occur when $v\approx 0$. To avoid this, take the following procedures.

First, in the computation of $B_k(v,x)$, since $\phi_1(v,x)$ and $\phi_2(v,x)$ are of the same sign, no cancellation occurs in the addition.

However, $\phi_1(v,x)$ and $\phi_2(v,x)$ can be computed according to their definitions only when $(x/2)^v<1/2$ or when $(x/2)^v>2$.

When $1/2\le(x/2)^v\le 2$, that is $-\log 2\le v\log(x/2)\le\log 2$, cancellation may occurs (where, in the addition of $a+b$, when max $(|a|, |b|)/|a+b|\ge 2$, a round-off error occurs. This is equivalent to having a binary round-off error of one digit or more).

To compute $\phi_1(v,x)$ and $\phi_2(v,x)$ without any cancellation, it is sufficient to find the best approximation in relative sense to the function in the range of $-\log 2\le t\le\log 2$.

$$f(t)=\frac{e^t-1}{t}=\frac{1}{1!}+\frac{t}{2!}+\frac{t^2}{3!}+\cdots \quad (4.9)$$

By the approximation, we can compute

$$\left.\begin{aligned}\phi_1(v,x)&=-f\left(-v\log\left(\frac{x}{2}\right)\right)\log\left(\frac{x}{2}\right)\\ \phi_2(v,x)&=-f\left(v\log\left(\frac{x}{2}\right)\right)\log\left(\frac{x}{2}\right)\end{aligned}\right\} \quad (4.10)$$

This subroutine uses the following approximation to $f(t)$:

$$f(t)\approx\frac{2\sum_{k=0}^{M}p_kt^{2k}}{\sum_{k=0}^{N}q_kt^{2k}-t\sum_{k=0}^{M}p_kt^{2k}} \quad (4.11)$$

(For detailed information on concrete values of M, N, p_k and q_k, refer to [88])

Also, as for $A_k(v,x)$ in (4.8), to avoid cancellations in computing the value of $(1/\Gamma(k+1-v)-1/\Gamma(k+1+v))/(k!v)$, the best approximation in relative sense should be provided. If we denote the expression in { } by $\tilde{A}_k(v)$ we can find,

$$\tilde{A}_k(v)=\frac{1}{(k+v)(k-v)}\left[\frac{1}{k!}\left\{\frac{1}{\Gamma(k-v)}+\frac{1}{\Gamma(k+v)}\right\}+\tilde{A}_{k-1}(v)\right] \qquad (k\ge 1) \quad (4.12)$$

This means that only an approximation to $\tilde{A}_0(v)$ should be provided. However, for k=1, (4.12) causes cancellation. Therefore it can be used for $k\ge 2$. As a result, the best approximation to $\tilde{A}_0(v)$ and $\tilde{A}_1(v)$ in the range of $0\le v\le 0.5$ is required.

This subroutine uses the best approximation polynomials of the form

$$\left.\begin{aligned}\tilde{A}_0(v)&\approx\sum_{k=0}^{M}p_kv^{2k}\\ \tilde{A}_1(v)&\approx\sum_{k=0}^{N}q_kv^{2k}\end{aligned}\right\} \quad (4.13)$$

(For detailed information on M, p_k, N, q_k, refer to [88].)

Thus $A_k(v,x)$ and $B_k(v,x)$ can be obtained without cancellations. If $0\le v\le 0.5$, then $A_0(v,x)\le 0$ and $A_k(v,x)\ge 0$ (k=1,2,...) if $x\le 2$, then $B_k(v,x)\ge 0$ and if $x>2$, then $B_k(v,x)<0$ (k=0,1,2,...). Therefore, in the addition in (4.7), cancellations may occur. Testing results imply that no cancellation occurs in the domain A when $0\le v\le 0.5$ in Fig.BKR-1. The items used in the sum in (4.7) becomes small enough as k becomes large. Therefore, the items used for convergence to the required accuracy is less.

Consequently, if we use the best approximation (for detailed information on M and p_k, refer to [88]),

$$g(v)=\frac{2}{\pi v}\sin(v\pi) \quad (4.15)$$

to

$$g(v)\approx\sum_{k=0}^{M}p_kv^{2k} \quad (4.14)$$

the value of $\boldsymbol{K}_v(x)$ can be computed as

$$K_v(x) = \frac{I_{-v}(x) - I_v(x)}{\frac{2}{\pi}\sin(v\pi)} \approx \frac{\sum_{k=0}^{\infty}\{A_k(v,x) + B_k(v,x)\}}{g(v)} \tag{4.16}$$

When v=0, $\boldsymbol{K}_0(x)$ can be computed more efficiently from the expression,

$$K_0(x) = \sum_{k=0}^{\infty} \frac{\left(\frac{x}{2}\right)^{2k}}{(k!)^2}\left\{-\left(\gamma + \log\left(\frac{x}{2}\right)\right) + \Phi_k\right\} \tag{4.17}$$

which is the limit of (4.1) as $v \to 0$,. instead of using (4.16). In (4.17), γ denotes the Euler's constant,

$$\Phi_0 = 0, \quad \Phi_k = \sum_{m=0}^{k} \frac{1}{m} (k \ge 1)$$

If the required relative accuracy is assumed to be ε, when

$$v < 1.723 x \varepsilon^{\frac{1}{2}} \tag{4.18}$$

$\boldsymbol{K}_v(x)$ and $\boldsymbol{K}_0(x)$ are the same in the sense of relative accuracy. Then $\boldsymbol{K}_v(x)$ is computed by using (4.17).

For the calculation when $0.5<v\le1.5$ and $1.5<v\le2.5$, the methods are almost the same as that above.

- Method in the domain B

  The method below is a generalization of the τ-method for computing $\boldsymbol{K}_n(x)$ of integer order.

  The method is based on the expression for $\boldsymbol{K}_v(x)$ of the form

$$K_v(x) = \sqrt{\frac{\pi}{2x}} e^{-x} f_v\left(\frac{1}{x}\right) \tag{4.19}$$

and uses the approximation to $f_v(1/x)$. Letting t=1/x, and $f_v(x)$ satisfies

$$t^2 f_v''(t) + 2(t+1) f_v'(t) - \left(v^2 - \frac{1}{4}\right) f_v(t) = 0 \tag{4.20}$$

When adding the shifted Ultraspherical polynomial orthogonal on the interval $[0,\eta]$ multiplied by τ on the right of (4.20),

$$t^2 f_v''(t) + 2(t+1) f_v'(t) - \left(v^2 - \frac{1}{4}\right) f_v(t) = \tau\, C_m^{*(\alpha)}\left(\frac{t}{\eta}\right) \tag{4.21}$$

where,

$$C_m^{*(\alpha)}(t) = \sum_{i=0}^{m} C_{mi}^{*(\alpha)} t^i \tag{4.22}$$

is the shifted Ultraspherical polynomial. Equation (4.21) has the following mth degree polynomial solution.

$$f_{vm}(t) = \tau \sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)} \sum_{i=0}^{k} a_i t^i}{(k+1) a_{k+1} \eta^k} \tag{4.23}$$

where,

$$\left.\begin{aligned} a_0 &= 1 \\ a_k &= \frac{(4v^2 - 1^2)(4v^2 - 3^2)\cdots(4v^2 - (2k-1)^2)}{k!8^k} \\ &\quad (k \ge 1) \end{aligned}\right\} \tag{4.24}$$

Here $f_{vm}(t)$ is considered as the approximation polynomial to $f(t)$. From the initial condition $f_{vm}(0)$=1 (as $t \to 0$, $f_v(t) \to 1$) τ can be determined. Then

$$f_{vm}(t) = \frac{\sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)} \sum_{i=0}^{k} a_i t^i}{(k+1) a_{k+1} \eta^k}}{\sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)}}{(k+1) a_{k+1} \eta^k}} \tag{4.25}$$

can be obtained. Although (4.25) contains α and η as unknown, when α=0.5 ($C_m^{*(\alpha)}(t)$ is a shifted Legendre polynomial) and η=t, the accuracy reaches the best.
In this case,

$$f_{vm}(t) = \frac{\sum_{k=0}^{m} \frac{P_{mk}^{*} \sum_{i=0}^{k} a_i t^i}{(k+1) a_{k+1} t^k}}{\sum_{k=0}^{m} \frac{P_{mk}^{*}}{(k+1) a_{k+1} t^k}} \tag{4.26}$$

holds. Where, $\boldsymbol{P}_{mk}^{*}$ is a coefficient of a shifted Legendre polynomial,

$$P_m^{*}(t) = \sum_{i=0}^{m} P_{mi}^{*} t^i \tag{4.27}$$

By multiplying both the numerator and the denominator by t^m, we obtain

$$f_{vm}(t) = \frac{\sum_{i=0}^{m} G_i(m,v) t^i}{\sum_{i=0}^{m} H_i(m,v) t^i} \tag{4.28}$$

where,

$$G_i(m,v) = \sum_{k=0}^{i} \frac{P_{m,m-i+k}^{*} a_k}{(m+1-i+k) a_{m+1-i+k}} \tag{4.29}$$

$$H_i(m,v) = \frac{P_{m,m-i}^{*}}{(m-i+1) a_{m-i+1}} \tag{4.30}$$

Equation (4.29) can be expressed using the power of v^2 as follows:

$$G_i(m,v)=\frac{1}{a_{m+1}}\sum_{j=0}^{i} b_{ij}\left(v^2\right)^j \tag{4.31}$$

where,

$$b_{ij}=2^{2j-3i}\sum_{l=0}^{j}\sum_{k=j-l}^{i-l}\frac{P^*_{m,m-k}\,p_{i-k,l}\,q_{k,j-l}}{(i-k)!\prod_{n=0}^{k}(m-n+1)} \tag{4.32}$$

With initial values

$$p_{0,0}=1,\,p_{1,0}=-1,\,p_{1,1}=1 \tag{4.33}$$

$$q_{0,0}=1,\,q_{1,0}=-(2m+1)^2,\,q_{1,1}=1 \tag{4.34}$$

$p_{k,l}$ and $q_{k,l}$ can be computed using the following recurrence formulas.

$$\left.\begin{aligned}p_{k,0}&=-(2k-1)^2\,p_{k-1,0}\\ p_{k,l}&=p_{k-1,l-1}-(2k-1)^2\,p_{k-1,l}\ (1\le l\le k-1)\\ p_{k,k}&=p_{k-1,k-1}\end{aligned}\right\} \tag{4.35}$$

$$\left.\begin{aligned}q_{k,0}&=-(2m-2k+3)^2\,q_{k-1,0}\\ q_{k,l}&=q_{k-1,l-1}-(2m-2k+3)^2\,q_{k-1,l}\ (1\le l\le k-1)\\ q_{k,k}&=q_{k-1,k-1}\end{aligned}\right\} \tag{4.36}$$

Equation (4.30) may be expressed as

$$H_i(m,v)=\frac{1}{a_{m+1}}c_i\psi_i \tag{4.37}$$

where,

$$c_i=\frac{(m-i)!\,P^*_{m,m-i}}{(m+1)!(-2)^i} \tag{4.38}$$

$$\left.\begin{aligned}\psi_0&=1\\ \psi_i&=\prod_{l=0}^{i-1}\left\{\left(m-l+\frac{1}{2}\right)^2-v^2\right\}(i\ge 1)\end{aligned}\right\} \tag{4.39}$$

Using (4.19), (4.28), (4.31) and (4.37), we obtain as approximation to $\boldsymbol{K}_v(x)$

$$\mathrm{K}_v(x)\approx\sqrt{\frac{1}{x}}e^{-x}\frac{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i\sum_{j=0}^{i}d_{ij}\left(v^2\right)^j}{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i e_i\psi_i} \tag{4.40}$$

where,

$$\left.\begin{aligned}d_{ij}&=b_{ij}/b_{1,1}\\ e_i&=\sqrt{\frac{2}{\pi}}\,c_i/b_{1,1}\end{aligned}\right\} \tag{4.41}$$

In (4.40), when m is fixed, the relationship is such that the larger the x, the higher the accuracy. Also when x is fixed, the relationship is such that the larger the m, the higher the accuracy. Equation (4.40) is used for the domain B in Fig. BKR-1. This subroutine sets m as follows in consideration of efficiency.

Single precision: when $x<2$, $m=9$; when $2\le x<10$, $m=6$; when $10\le x$, $m=4$ (4.42)

Double precision: when $x<2$, $m=28$; when $2\le x<10$, $m=16$; when $10\le x$, $m=11$ (4.43)

This subroutine contains a table in which constants d_{ij} and e_i are stored in the data statement.

I11-81-0801 BK0, DBK0

Zero order modified Bessel function of the second kind $\boldsymbol{K}_0(x)$
CALL BK0(X,BK,ICON)

Function

This subroutine computes the values of zero order modified Bessel function of the second kind

$$K_0(x)=\sum_{k=1}^{\infty}\frac{(x/2)^{2k}}{(k\,!)^2}\left(\sum_{m=1}^{k}1/m\right)-I_0(x)\{\gamma+\log(x/2)\}$$

(where $\boldsymbol{I}_0(x)$: zero order modified Bessel function of the first kind and γ: Euler's constant) by polynomial approximations and the asymptotic expansion.
Where, x>0.

Parameters

X	Input. Independent variable x.
BK	Output. Function value $\boldsymbol{K}_0(x)$.
ICON ..	Output. Condition code. See Table BK0-1.

Table BK0-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X>log(fl_{max})	BK is set to 0.0.
30000	X≤0	BJ is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL, ULMAX
  FORTRAN basic function ... ALOG, EXP, and SQRT

- Notes
  [Range of argument X].
  0<X≤log(fl_{max})
  If X exceeds the limits, and underflow will occur during the calculation of e^{-x}. This limit is provided for that reason. (See (4.5) and (4.6) in "Method".)

- Example
  The following example generates a table of $\boldsymbol{K}(x)$ from 1 to 100 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,100
       X=K
       CALL BK0(X,BK,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BK
       IF(ICON.NE.0) WRITE(6,620) X,BK,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',9X,'K0(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BK=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With x=2 as the boundary, the approximation formula used to calculate modified Bessel function $\boldsymbol{K}_0(x)$ changes.

- For 0<x<2
  The power series expansion of $\boldsymbol{K}_0(x)$

$$K_0(x)=\sum_{k=1}^{\infty}\frac{(x/2)^{2k}}{(k!)^2}\left(\sum_{m=1}^{k}1/m\right)-I_0(x)\{\gamma+\log(x/2)\} \qquad (4.1)$$

(where $\boldsymbol{I}_0(x)$: zero order modified Bessel function of the first kind, γ: Euler's constant) is calculated using the approximation formulas (4.2) and (4.3).
Single precision:

$$K_0(x)=\log\left(\frac{x}{2}\right)\sum_{k=0}^{5}a_k x^{2k}+\sum_{k=0}^{5}b_k x^{2k} \qquad (4.2)$$

Double precision:

$$K_0(x)=\log\left(\frac{x}{2}\right)\sum_{k=0}^{9}a_k x^{2k}+\sum_{k=0}^{9}b_k x^{2k} \qquad (4.3)$$

- For 2 ≤ x ≤ log(fl_{max})
  The asymptotic expansion of $\boldsymbol{K}_0(x)$

$$K_0(x)=\sqrt{\frac{\pi}{2x}}e^{-x}\left\{1+\frac{(-1)}{8x}+\frac{(-1)(-9)}{2!}\right.\left.\frac{1}{(8x)^2}+\frac{(-1)(-9)(-25)}{3!}\cdot\frac{1}{(8x)^3}+\cdots\right\} \qquad (4.4)$$

is calculated using the approximation formulas (4.5) and (4.6)
Single precision:

$$K_0(x)=\frac{e^{-x}}{\sqrt{x}}\left(\sum_{k=0}^{8}a_k z^k\right), \quad z=2/x \qquad (4.5)$$

Double precision:

$$K_0(x)=\frac{e^{-x}}{\sqrt{x}}\left(\sum_{k=0}^{8}a_k x^k\Big/\sum_{k=0}^{8}b_k x^k\right) \qquad (4.6)$$

I11-81-0901 BK1, DBK1

First order modified Bessel function of the second kind $\boldsymbol{K}_1(x)$
CALL BK1(X,BK,ICON)

Function

This subroutine computes first order modified Bessel function of the second kind $\boldsymbol{K}_1(x)$

$$K_1(x) = I_1(x)\{\gamma + \log(x/2)\} + \frac{1}{x}$$
$$-1/2 \cdot \sum_{k=0}^{\infty} \frac{(x/2)^{2k+1}}{k!(k+1)!} \left(\sum_{m=1}^{k} 1/m + \sum_{m=1}^{k+1} 1/m \right)$$

(where $\boldsymbol{I}_1(x)$: first order modified Bessel function of the first kind, γ: Euler's constant) by polynomial approximations and the asymptotic expansion.
Where, $x>0$.

Parameters

X	Input. Independent variable x.
BK	Output. Function value $\boldsymbol{K}_1(x)$.
ICON ..	Output. Condition code. See Table BK1-1.

Table BK1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X>log(fl_{max})	BK is set to 0.0.
30000	X≤0	BK is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL, ULMAX
  FORTRAN basic function ... ALOG, EXP, and SQRT

- Notes
  [Range of argument X].
  0< X≤log(fl_{max})
  If X exceeds the limits, and underflow will occur in the calculation of e^{-x}. This limit is provided for that reason. (See (4.5) and (4.6) in "Methods".)

- Example
  The following example generates a table of $\boldsymbol{K}_1(x)$ from 1 to 100 with increment 1.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,100
      X=K
      CALL BK1(X,BK,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,BK
      IF(ICON.NE.0) WRITE(6,620) X,BK,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF BESSEL ',
     *'FUNCTION'///6X,'X',9X,'K1(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     *E17.7,5X,'BK=',E17.7,5X,'CONDITION=',
     *I10)
      END
```

Method

With x=2 as the boundary, the approximation formula used to calculate modified Bessel function $\boldsymbol{K}_1(x)$ changes.

- For $0<x<2$

$$K_1(x) = I_1(x)\{\gamma + \log(x/2)\} + 1/x$$
$$-1/2 \cdot \sum_{k=0}^{\infty} \frac{(x/2)^{2k+1}}{k!(k+1)!} \left(\sum_{m=1}^{k} 1/m + \sum_{m=1}^{k+1} 1/m \right) \quad (4.1)$$

(where $\boldsymbol{I}_1(x)$: first order modified Bessel function of the first kind, γ: Euler's constant) is calculated using the approximation formulas (4.2) and (4.3).
Single precision:

$$K_1(x) = \left\{ \log\left(\frac{x}{2}\right) \sum_{k=0}^{5} a_k x^{2k} + \sum_{k=0}^{5} b_k x^{2k} \right\} x + \frac{1}{x} \quad (4.2)$$

Double precision:

$$K_1(x) = \left\{ \log\left(\frac{x}{2}\right) \sum_{k=0}^{9} a_k x^{2k} + \sum_{k=0}^{9} b_k x^{2k} \right\} x + \frac{1}{x} \quad (4.3)$$

- For $2 \le x \le \log(fl_{max})$
  The asymptotic expansion of $\boldsymbol{K}_1(x)$

$$K_1(x) = \sqrt{\frac{\pi}{2x}} e^{-x} \left\{ 1 + \frac{3}{8x} + \frac{3 \cdot (-5)}{2!} \cdot \frac{1}{(8x)^2} + \frac{3 \cdot (-5) \cdot (-21)}{3!} \cdot \frac{1}{(8x)^3} + \cdots \right\} \quad (4.4)$$

is calculated using the approximation formulas (4.5) and (4.6)
Single precision:

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \left(\sum_{k=0}^{8} a_k z^k \right), \quad z = 2/x \quad (4.5)$$

Double precision:

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \left(\sum_{k=0}^{8} a_k x^k \Big/ \sum_{k=0}^{8} b_k x^k \right) \quad (4.6)$$

B21-11-0202 BLNC, DBLNC

Balancing of a real matrix
CALL BLNC(A,K,N,DV,ICON)

Function

The diagonal similarity transformation shown in (1.1) is applied to an n-order matrix $\boldsymbol{A}$. By this transformation, sums of the norms of elements in the corresponding i-th row and i-th column (i=1,...,n) are almost equalized for the transformed real matrix $\tilde{\boldsymbol{A}}$.

$$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D} \qquad (1.1)$$

$\boldsymbol{D}$ is a diagonal matrix. $n \geq 1$.

Parameters

A	Input. Real matrix $\boldsymbol{A}$. Output. Balanced real matrix $\tilde{\boldsymbol{A}}$. A(K,N) is a two-dimensional array.
K	Input. Adjustable dimension of array A.
N	Input. Order n on $\boldsymbol{A}$ and $\tilde{\boldsymbol{A}}$.
DV	Output. Scaling factor (diagonal elements of $\boldsymbol{D}$). DV is a one-dimensional array of size n.
ICON ..	Output. Condition code. See Table BLNC-1.

Table BLNC-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	Balancing was not performed.
30000	N<1 or K<N	Bypassed

Comments on use

- Subprograms used
  SSL II ... IRADIX and MGSSL
  FORTRAN basic function ... ABS

- Notes
  If there are large differences in magnitude of elements in a matrix, the precision of the computed eigenvalues and eigenvectors on that matrix can be adversely affected. This routine is used to avoid the adverse effects.
  If each element of a matrix is nearly the same magnitude, this routine should be omitted.
  If all elements of a row or column (except the diagonal element) are zero, balancing of the row and corresponding column is bypassed.
  In order to obtain the eigenvectors $\boldsymbol{x}$ of real matrix $\boldsymbol{A}$, the back transformation of (3.1) must be applied to eigenvectors $\tilde{\boldsymbol{x}}$ of matrix $\tilde{\boldsymbol{A}}$ which has been balanced by this routine.

$$\boldsymbol{x} = \boldsymbol{D}\tilde{\boldsymbol{x}} \qquad (3.1)$$

The back transformation of (3.1) can be performed using subroutine HBK1. (See the section on HBK1.)

- Example
  After balancing an n-order real matrix $\boldsymbol{A}$, it is reduced to a real Hessenburge matrix $\boldsymbol{H}$ using subroutine HES1, then the eigenvalues are determined using subroutine HSQR. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),DV(100),PV(100)
     *,ER(100),EI(100)
   10 CONTINUE
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL BLNC(A,100,N,DV,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL HES1(A,100,N,PV,ICON)
      CALL HSQR(A,100,N,ER,EI,L,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,630) (ER(I),EI(I),I=1,L)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',5X,
     *'N=',I3/)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     *E14.7))
  620 FORMAT('0',20X,'ICON=',I5)
  630 FORMAT('0',5X,'EIGENVALUE'/'0',20X,
     *'REAL',16X,'IMAG'/('0',15X,2(E15.7,
     *5X)))
      END
```

Method

An n-order real matrix $\boldsymbol{A}$ is balanced through iterations of the diagonal similarity transformation.

$$\boldsymbol{A}_s = \boldsymbol{D}_s^{-1}\boldsymbol{A}_{s-1}\boldsymbol{D}_s \quad s = 1,2,... \qquad (4.1)$$

Where $\boldsymbol{A}_0 = \boldsymbol{A}$, $\boldsymbol{D}_s$ is the diagonal matrix shown in (4.2), and s is the number of iterations.

$$\boldsymbol{D}_s = \begin{bmatrix} d_1^{(s)} & & & & \\ & d_2^{(s)} & & 0 & \\ & & \cdot & & \\ & 0 & & \cdot & \\ & & & & \cdot \\ & & & & & d_n^{(s)} \end{bmatrix} \qquad (4.2)$$

In the balancing of (4.1), excluding the diagonal elements, the sum of the magnitude of elements in the i-th row of $\boldsymbol{A}_s$ is made almost equal to that of the magnitude of elements in the i-th column. Letting $\boldsymbol{A}_s$

$=(a_{ij}^{(s)})$, balancing is performed such that

$$\sum_{\substack{k=1\\k\neq i}}^{n} \left|a_{ik}^{(s)}\right| \approx \sum_{\substack{k=1\\k\neq i}}^{n} \left|a_{ki}^{(s)}\right| \tag{4.3}$$

is saturated. If $\boldsymbol{D}_{si}$ is defined as shown in (4.4), $\boldsymbol{D}_s$ of (4.2) can be expressed as (4.5).

$$\boldsymbol{D}_{si} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{l} \begin{array}{ccccccc} & & & i & & & \end{array} \\ \left[\begin{array}{ccccccc} 1 & & & & & & \\ & \ddots & & & 0 & & \\ & & 1 & & & & \\ & & & d_i^{(s)} & & & \\ & & & & 1 & & \\ & 0 & & & & \ddots & \\ & & & & & & 1 \end{array}\right] \end{array} \begin{array}{c} \\ \\ \\ \\ i \\ \\ \\ \end{array} \tag{4.4}$$

$$\boldsymbol{D}_s = \boldsymbol{D}_{s1}\boldsymbol{D}_{s2}\cdots\boldsymbol{D}_{sn} \tag{4.5}$$

The equation (4.5) shows that the transformation of (4.1) can be performed by executing the transformation of (4.6) for i=1,2,...,n.

$$\boldsymbol{A}_{si} = \boldsymbol{D}_{si}^{-1}\boldsymbol{A}_{si-1}\boldsymbol{D}_{si} \tag{4.6}$$

where $\boldsymbol{A}_{s0} = \boldsymbol{A}_{s-1}$ and $\boldsymbol{A}_s = \boldsymbol{A}_{sn}$

$\boldsymbol{D}_{si}$ that is $\boldsymbol{d}_i^{(s)}$ is defined such that the transformed ***i***-th row and corresponding column satisfy the equation (4.3). If they already satisfy the equation (4.3) at the time of transformation, $\boldsymbol{d}_i^{(s)}=1$. Iterations of (4.1) continue until (4.3) is satisfied for all rows and corresponding columns. A brief description of this procedure follows:

- Excluding the diagonal element, the sums of magnitudes of elements in ***i***-th row and corresponding column are computed.

$$C = \sum_{\substack{k=1\\k\neq i}}^{n} \left|a_{ki}\right| \tag{4.7}$$

and

$$R = \sum_{\substack{k=1\\k\neq i}}^{n} \left|a_{ik}\right| \tag{4.8}$$

- $\boldsymbol{d}_i^{(s)}$ is defined

$$\boldsymbol{d}_i^{(s)} = \rho^{k} \tag{4.9}$$

where $\rho = \begin{cases} 2 & \text{for binary digits} \\ 16 & \text{for hexadecimal digits} \end{cases}$

k is defined to satisfy the condition

$$R\cdot\rho > \mathrm{C}\cdot\rho^{2k} \geq R/\rho \tag{4.10}$$

From (4.10), when $\boldsymbol{C} < \boldsymbol{R}/\rho$, $k > 0$, and when $\boldsymbol{C} \geq \boldsymbol{R}/\rho$, $k \leq 0$.

- Whether or not transformation is necessary is determined by

$$\left(C\cdot\rho^{2k} + R\right)/\rho < 0.95\left(C + R\right) \tag{4.11}$$

If (4.11) is satisfied, transformation is performed where $d_i^{(s)} = \rho^k$. If (4.11) is not satisfied, transformation is bypassed.

- Balancing ends when transformation can no longer be performed on any row or column. Then, the diagonal elements of $\boldsymbol{D}$ shown in (4.12) are stored as the scaling factor in the array DV.

$$\boldsymbol{D} = \boldsymbol{D}_1\boldsymbol{D}_2\cdots\boldsymbol{D}_s \tag{4.12}$$

For further information see Reference [13] pp.315-326.

A52-11-0302 BLUX1, DBLUX1

A system of linear equations with a real general band matrix decomposed into the factors *L* and *U*
CALL BLUX1(B,FA,N,NH1,NH2,FL,IP,ICON)

Function

This subroutine solves a system of linear equations

$$\boldsymbol{LUx}=\boldsymbol{b} \qquad (1.1)$$

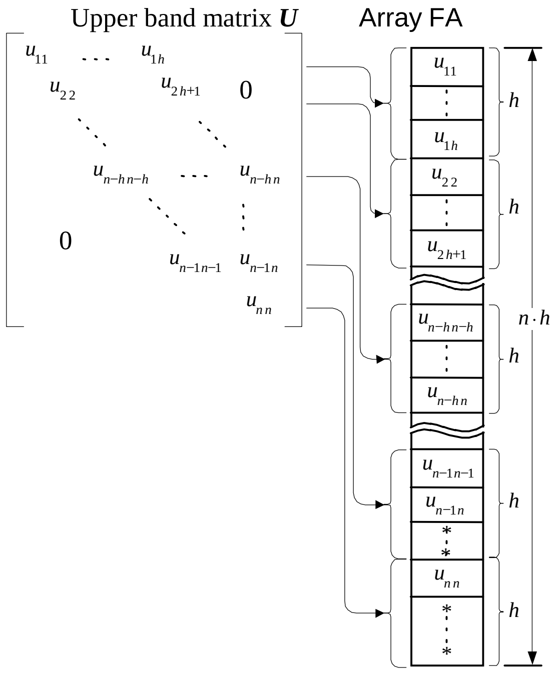
where $\boldsymbol{L}$ is a unit lower band matrix with band width h_1, $\boldsymbol{U}$ is an upper band matrix with band width $h(=\min(h_1+h_2+1,n)$ and b is an n-dimensional real constant vector. Further, the order, the lower and upper band widths of the original matrix which has been LU-decomposed are n, h_1 and h_2 respectively, where $n>h_1\geq0$ and $n>h_2\geq0$.

Parameters

B	Input. Constant vector b. Output. Solution vector x. One-dimensional array of size n.
FA	Input. Matrix $\boldsymbol{U}$. **See Figure BLUX1-1.** One-dimensional array of size $n\cdot h$.
N	Input. Order n of matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.
NH1 ...	Input. Lower band width h_1 of the LU-decomposed original matrix.
NH2 ...	Input. Upper band width h_2 of the LU-decomposed original matrix.
FL	Input. Matrix $\boldsymbol{L}$. One-dimensional array of size $(n-1)\cdot h_1$. **See Fig. BLUX1-2.**
IP	Input. Transposition vector which indicates the history of the row exchange in partial pivoting. One-dimensional array of size n.
ICON ..	Output. Condition code. **See Table BLUX1-1.**

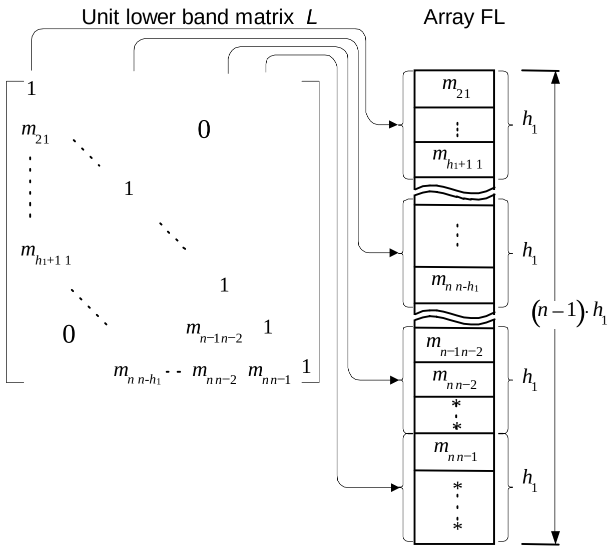
Table BLUX1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The coefficient matrix was singular.	Discontinued
30000	N≤NH1, N≤NH2, NH1<0, NH2<0 or there was error in IP.	Bypassed



Note: The elements marked by *⋯* are arbitrary values.

Fig. BLUX1-1 Storage of elements of $\boldsymbol{U}$ in array FA



Note: The diagonal elements are not stored.
The elements marked by *⋯* are arbitrary values.

Fig. BLUX1-2 Storage of elements of $\boldsymbol{L}$ in array FL

Comments on use

- Subprograms used
  SSL II .... MGSSL
  FORTRAN basic function ... MIN0

- Notes
  A system of linear equations can be solved by calling this subroutine following subroutine BLU1. At this case, this subroutine requires the output parameters from subroutine BLU1 as the input parameters (except

for the constant vector). However, such equations can be solved by calling subroutine LBX1 in one step.

This subroutine, by making use of band matrix characteristics, saves a data storage area. In some cases, however, depending on the size of the band width, a larger data storage area may be required than used by subroutine LUX provided for real general matrices.

If that is the case, subroutine LUX may be used to more save data storage area.

This subroutine is especially useful for the case where the upper and lower band widths of the coefficient matrix of order n are approximately less than $n/3$, provided both the band widths are equal.

- Example

This example shows that the subroutine BLU1 is once called to decompose the $n \times n$ matrix with lower band width h_1 and upper band width h_2 and then l systems of linear equations with the decomposed coefficient matrix are solved. $n \le 100$, $h_1 \le 20$ and $h_2 \le 20$.

```
C     **EXAMPLE**
      DIMENSION A(4100),B(100),FL(1980),
     *IP(100),VW(100)
      CHARACTER*4 NT1(6),NT2(6),NT3(6)
      DATA NT1/'CO  ','EF  ','FI  ','CI  ',
     *'EN  ','T   '/,
     *NT2/'CO  ','NS  ','TA  ','NT  ',
     *2*'    '/,
     *NT3/'SO  ','LU  ','TI  ','ON  ',
     *2*'    '/
      READ(5,500) N,NH1,NH2,L
      NT=N*MIN0(NH1+NH2+1,N)
      READ(5,510) (A(I),I=1,NT)
      M=1
      WRITE(6,600) N,NH1,NH2
      CALL PBM(NT1,6,A,N,NH1,NH2)
      CALL BLU1(A,N,NH1,NH2,1.0E-6,
     *IS,FL,IP,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.EQ.0) GO TO 10
      WRITE(6,620)
      STOP
   10 READ(5,510) (B(I),I=1,N)
      CALL PGM(NT2,6,B,N,N,1)
      CALL BLUX1(B,A,N,NH1,NH2,
     *FL,IP,ICON)
      CALL PGM(NT3,6,B,N,N,1)
      M=M+1
      IF(L.GT.M) GO TO 10
      WRITE(6,630)
      STOP
  500 FORMAT(4I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'///5X,
     *'LINEAR EQUATIONS  AX=B'
     */5X,'ORDER=',I4
     */5X,'SUB-DIAGONAL LINES=',I4
     */5X,'SUPER-DIAGONAL LINES=',I4)
  610 FORMAT(' ',4X,'ICON=',I5)
  620 FORMAT(' '/5X,'** ABNORMAL END **')
  630 FORMAT(' '/5X,'** NORMAL END **')
      END
```

Subroutines PBM and PGM in these example are used to print out a band matrix and a real matrix, respectively. Those programs are described in the examples for subroutines LBX1 and MGSM.

Method

A system of linear equations

$$\boldsymbol{LUx}=\boldsymbol{b} \tag{4.1}$$

can be solved by solving two following equations

$$\boldsymbol{Ly}=\boldsymbol{b} \tag{4.2}$$
$$\boldsymbol{Ux}=\boldsymbol{y} \tag{4.3}$$

where $\boldsymbol{L}$ is an $n \times n$ unit lower band matrix with band width h_1 and $\boldsymbol{U}$ is an $n \times n$ upper band matrix with band width h (=min(h_1+h_2+1,n)).

This subroutine assumes that the triangular matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ have been formed using the Gaussian elimination method.

Particularly, $\boldsymbol{L}$ is represented by

$$\boldsymbol{L}=(\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}...\boldsymbol{M}_1\boldsymbol{P}_1)^{-1} \tag{4.4}$$

where $\boldsymbol{M}_k$ is the matrix to eliminate the elements below the diagonal element in the k-th column at the k-th step of the Gaussian elimination method, and $\boldsymbol{P}_{\mathrm{k}}$ is the permutation matrix which performs the row exchanging required in partial pivoting. (For further details, refer to method for subroutine BLU1).

- Solving $\boldsymbol{Ly}=\boldsymbol{b}$ (Forward substitution)

  The equation (4.5) can be derived from the equation (4.4) by the equation (4.2).

$$\boldsymbol{y}=\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}...\boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{b} \tag{4.5}$$

The equation (4.5) is successively computed using the following equations.

$$\begin{aligned}
\boldsymbol{b}^{(1)} &= \boldsymbol{b} \\
\boldsymbol{b}^{(2)} &= \boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{b}^{(1)} \\
\boldsymbol{b}^{(3)} &= \boldsymbol{M}_2\boldsymbol{P}_2\boldsymbol{b}^{(2)} \\
&\vdots \\
\boldsymbol{b}^{(n)} &= \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\boldsymbol{b}^{(n-1)} \\
\boldsymbol{y} &= \boldsymbol{b}^{(n)}
\end{aligned}$$

where $\boldsymbol{M}_k$ is the following matrix.

$$M_k = \begin{bmatrix} 1 & & & & & \\ & 1 & & & 0 & \\ & & \ddots & & & \\ & & & 1 & & \\ & & & -m_{k+1k} & & \\ & 0 & & -m_{k+2k} & 1 & \\ & & & \vdots & 0 & \ddots \\ & & & -m_{nk} & & & 1 \end{bmatrix}$$

In this subroutine, $\boldsymbol{b}^{(k+1)}$ (k=0,1,...,n – 1) are successively obtained by the following sequence:

Corresponding to the row exchange at the k-th step of the Gaussian elimination process, the elements of constant vector $\boldsymbol{b}^{(k)}$ are exchanged such that

$$\tilde{\boldsymbol{b}}^{(k)} = \boldsymbol{P}_k \boldsymbol{b}^{(k)}$$

Then

$$b_1^{(k+1)} = \tilde{b}_i^{(k)} - m_{ik}\tilde{b}_k^{(k)}, i = k+1,...,\min(k+h_1, n)$$
$$\tilde{b}^{(k)} = \left(\tilde{b}_1^{(k)}, \tilde{b}_2^{(k)}, ..., \tilde{b}_n^{(k)}\right)^{\mathrm{T}}, L = (m_{ij}),$$
$$y = (y_1, y_2, ..., y_n)^{\mathrm{T}}$$

- Solving $\boldsymbol{Ux}$=$\boldsymbol{y}$ (Backward substitution)
  $\boldsymbol{Ux}$=$\boldsymbol{y}$ can be serially solved using equation (4.6).

$$x_k = y_k - \sum_{i=k+1}^{\min(n,k+h)} u_{ki}x_i \quad , k = n, n-1,...,1$$

By increasing the precision of the inner products in the equation (4.6), the effects of rounding errors is minimized.

A52-11-0202 BLU1, DBLU1

LU-decomposition of a real general band matrix (Gaussian elimination method)
CALL BLU1(A,N,NH1,NH2,EPSZ,IS,FL,IP,VW,ICON)

Function

An $n \times n$ real general band matrix with lower band width h_1 and upper band width h_2 is LU decomposed using the Gaussian elimination method.

$$\boldsymbol{A}=\boldsymbol{LU} \tag{1.1}$$

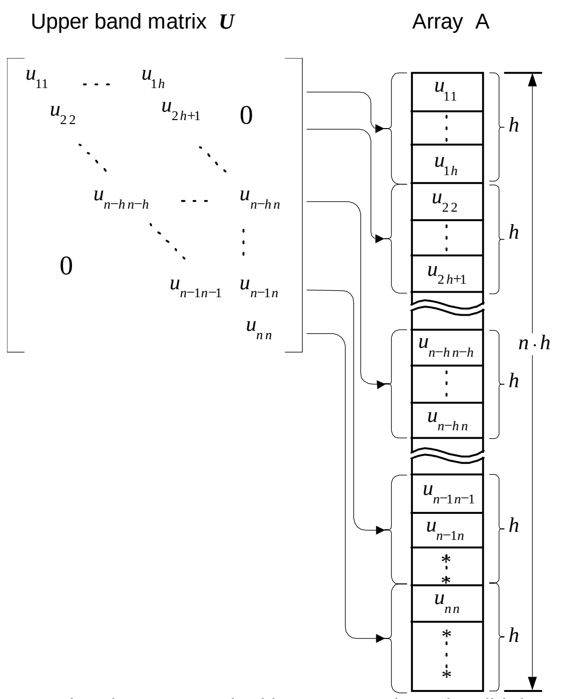
Where $\boldsymbol{L}$ is an unit lower band matrix and $\boldsymbol{U}$ is an upper band matrix. $n>h_1\geq0$ and $n>h_2\geq0$.

Parameters

A	Input. Matrix $\boldsymbol{A}$. Output. Matrix $\boldsymbol{U}$. See Fig. BLU1-1. Matrix $\boldsymbol{A}$ is stored in a one-dimensional array of size $n\cdot\min(h_1+h_2+1,n)$ in the compressed mode for band matrices.
N	Input. Order n of matrix $\boldsymbol{A}$.
NH1	Input. Lower band width h_1 of matrix $\boldsymbol{A}$.
NH2	Input. Upper band width h_2 of matrix $\boldsymbol{A}$.
EPSZ ..	Input. Tolerance for relative zero test of pivots in decomposition process of matrix $\boldsymbol{A}$ (≥0.0). When EPSZ is 0.0, the standard value is used. (See Notes.)
IS	Output. Information for obtaining the determinant of matrix $\boldsymbol{A}$. (See Notes.)
FL	Output. The matrix $\boldsymbol{L}$. See Fig. BLU1-2. One-dimensional array of size $(n-1)h_1$.
IP	Output. The transposition vector which indicates the history of row exchanging that occurred in partial pivoting. One dimensional array of size n. (See Notes.)
VW	Work area. One-dimensional array of size n.
ICON ..	Output. Condition code. See Table BLU1-1.

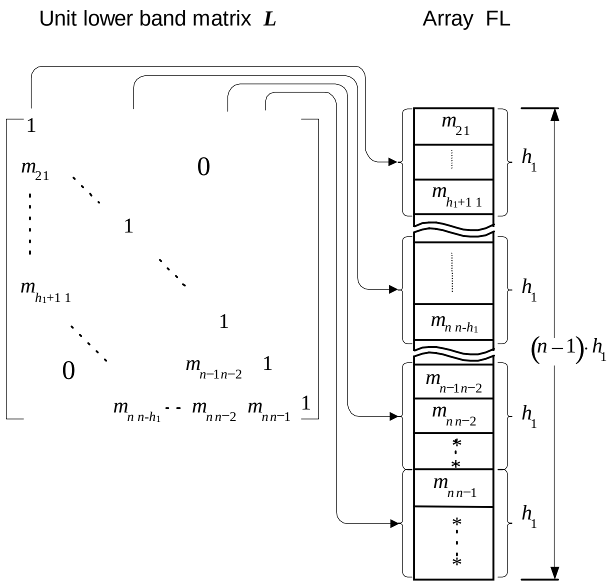
Table BLU1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The relatively zero pivot occurred. It is highly probable that the matrix is singular.	Discontinued
30000	N≤NH1, N≤NH2, NH1<0, NH2<0 or EPSZ<0.0	Bypassed



Note: The elements marked by *..* may have invalid data. h=min(h_1+h_2+1,n)

Fig. BLU1-1 Storage of the elements of $\boldsymbol{U}$ in array A.



Note: The diagonal elements are not stored. The elements marked by *..* may have invalid data.

Fig. BLU1-2 Storage of the elements of $\boldsymbol{L}$ in array FL

Comments on use

- Subprograms used
  SSL II .... AMACH, MGSSL
  FORTRAN basic functions .... ABS, MIN0

- Notes
  This subroutine assumes that the relatively zero pivot occurs when the absolute value of the pivot is smaller than the largest absolute value of the elements, in the

coefficient matrix, multiplied by EPSZ in the LU-decomposition using the Gaussian elimination method. In such a case, the processing is discontinued with ICON=20000. The standard value of EPSZ is 16 *u*, *u* being the unit round off. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value, but the result is not always be guaranteed.

The elements of matrix ***U*** are stored in array A as shown in Fig. BLU1-1. Therefore, the determinant of matrix ***A*** can be obtained by multiplying the product of the *n* diagonal elements, i.e., *n* array elements A($i\times h+1$), $i=0,1,...,n-1$, by the IS value, where $h=\min(h_1+h_2+1,n)$.

In this subroutine, the elements of array A are actually exchanged in partial pivoting. That is, if the I-th row (I ≥ J) has been selected as the pivotal row in the J-th stage (J=1,2,...,$n-1$) of decomposition, the elements of the I-th and J-th rows of matrix ***A*** are exchanged. Then, I is stored into IP(J) in order to record the history of this exchange.

It is possible to solve a system of linear equations by calling subroutine BLUX1 following this subroutine. However, instead of these subroutines, subroutine LBX1 can be called to solve such an equations in one step.

This subroutine, by making use of band matrix characteristics, saves data storage area. In some cases, however, depending on the size of the band width, a larger data storage area may be required (including work area) than used by subroutine ALU provided for real general matrices. If that is the case, subroutine ALU may be used to save more data storage area.

This subroutine is especially useful for the case where the upper and lower band widths of the matrix of order *n* are approximately less than *n*/3, provided both the band widths are equal.

- Example

  For an $n\times n$ matrix with lower band width h_1 and upper band width h_2, the LU-decomposition is computed. $n\le100$, $h_1\le20$ and $h_2\le20$.

```
C      **EXAMPLE**
       DIMENSION A(4100),FL(1980),IP(100),
      * VW(100)
       CHARACTER*4 NT1(6),NT2(15),NT3(10)
       DATA NT1/'MA  ','TR  ','IX  ',
      *         3*'    '/,
      *NT2/'LU  ','-D  ','EC  ','OM  ',
      *    'PO  ','SE  ','D   ','MA  ',
      *    'TR  ','IX  ',5*'    '/,
      *NT3/'TR  ','AN  ','SP  ','OS  ',
      *    'IT  ','IO  ','N   ','VE  ',
      *    'CT  ','OR  '/
       READ(5,500) N,NH1,NH2
       NT0=MIN0(NH1+NH2+1,N)
       NT=N*NT0
       READ(5,510) (A(I),I=1,NT)
       WRITE(6,600) N,NH1,NH2
       CALL PBM(NT1,6,A,N,NH1,NH2)
       CALL BLU1(A,N,NH1,NH2,1.0E-6,
      *          IS,FL,IP,VW,ICON)
       WRITE(6,610) ICON
       IF(ICON.EQ.0) GO TO 10
       WRITE(6,620)
       STOP
   10  CALL PBM(NT2,15,A,N,NH1,NH2)
       CALL PGM(NT3,10,IP,N,N,1)
       DET=IS
       DO 20 I=1,NT,NT0
       DET=DET*A(I)
   20  CONTINUE
       WRITE(6,630) DET
       WRITE(6,640)
       STOP
  500  FORMAT(3I5)
  510  FORMAT(4E15.7)
  600  FORMAT('1'
      *///5X,'DETERMINANT COMPUTATION'
      */5X,'ORDER=',I4
      */5X,'SUB-DIAGONAL LINES=',I4
      */5X,'SUPER-DIAGONAL LINES=',I4)
  610  FORMAT(' ',4X,'ICON=',I5)
  620  FORMAT(' '/5X,'** ABNORMAL END **')
  630  FORMAT(' ',4X,'DET=',E16.8)
  640  FORMAT(' '/5X,'** NORMAL END **')
       END
```

Subroutines PBM and PGM are used to print out a general band matrix and a real general matrix, respectively. Those programs are described in the examples for subroutines LBX1 and MGSM, respectively.

Method

- Gaussian elimination method

  Generally, $n\times n$ non-singular matrix ***A*** can be decomposed, with partial pivoting, into a unit lower triangular matrix ***L*** and an upper triangular matrix ***U*** as equation (4.1).

$$\boldsymbol{A}=\boldsymbol{LU} \tag{4.1}$$

Now, let $\boldsymbol{A}^{(k)}$ represent the matrix at the *k*-th step of Gaussian elimination method, where $\boldsymbol{A}^{(1)}=\boldsymbol{A}$.

The first step of the elimination is to obtain $\boldsymbol{A}^{(2)}$ by eliminating the elements below the diagonal element in the first column. This process can be described as the form (4.2) in a matrix expression,

$$\boldsymbol{A}^{(2)}=\boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{A}^{(1)} \tag{4.2}$$

where $\boldsymbol{P}_1$ is a permutation matrix which performs the row exchanging required for partial pivoting and $\boldsymbol{M}_1$ is a matrix to eliminate the elements below the diagonal element in the first column of the permuted matrix. Similarly, the *k*-th step of the elimination process can be described as the form (4.3).

$$\boldsymbol{A}^{(k+1)}=\boldsymbol{M}_k\boldsymbol{P}_k\boldsymbol{A}^{(k)} \tag{4.3}$$

where

$$M_k = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & 0 & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & -m_{k+1k} & 1 & & \\ & 0 & & -m_{k+2k} & & \ddots & \\ & & & \vdots & & & \\ & & & -m_{nk} & 0 & & 1 \end{bmatrix} \tag{4.4}$$

$m_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}, \quad \boldsymbol{A}^{(k)} = \left(a_{ij}^{(k)}\right)$

At the final n-th step, it can be described as follows:

$$\begin{aligned} \boldsymbol{A}^{(n)} &= \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\boldsymbol{A}^{(n-1)} \\ &= \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\ldots\boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{A}^{(1)} \end{aligned} \tag{4.5}$$

and then matrix $\boldsymbol{A}$ ($=\boldsymbol{A}^{(1)}$) has been transformed into the upper triangular matrix $\boldsymbol{A}^{(n)}$.
Let matrices $\boldsymbol{U}$ and $\boldsymbol{L}$ represent the forms (4.6) and (4.7).

$$\boldsymbol{U} = \boldsymbol{A}^{(n)} \tag{4.6}$$
$$\boldsymbol{L} = (\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\ldots\boldsymbol{M}_1\boldsymbol{P}_1)^{-1} \tag{4.7}$$

Then, the equation (4.5) can be represented by the equation (4.8).

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U} \tag{4.8}$$

Since

$$M_k^{-1} = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & 0 & & \\ & & \ddots & & & & \\ & & & 1 & & & \\ & & & m_{k+1k} & 1 & & \\ & 0 & & m_{k+2k} & & \ddots & \\ & & & \vdots & 0 & & \\ & & & m_{nk} & & & 1 \end{bmatrix} \tag{4.9}$$

matrix $\boldsymbol{L}$ shown in the equation (4.7) is a lower triangular matrix.
Thus matrix $\boldsymbol{A}$ is decomposed into unit lower triangular matrix $\boldsymbol{L}$ and upper triangular matrix $\boldsymbol{U}$.

- Procedure performed in the subroutine
  In the k-th step (k=1,...,n − 1) of the elimination process, this subroutine obtains each element of the k-th column of matrix $\boldsymbol{L}$ and the k-th row of matrix $\boldsymbol{U}$ using equations (4.10) and (4.11).

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad ,i = k+1, k+2,.., \min(k+h_1, n) \tag{4.10}$$

$$u_{kj} = a_{kj}^{(k)} \quad ,i = k, k+1,.., \min(k+h_1+h_2, n) \tag{4.11}$$

The principal minor $\boldsymbol{A}^{(k+1)}$ which is eliminated in the (k+1)-th step is obtained using equation (4.12),

$$\begin{aligned} &a_{kl}^{(k+1)} = a_{kl}^{(k)} - m_{kj}u_{jl} \\ &\quad k = j+1, j+2,..., \min(j+h_1, n) \\ &\quad l = j+1, j+2,..., \min(j+h_1+h_2, n) \end{aligned} \tag{4.12}$$

where $\boldsymbol{A}^{(k)}=(a_{ij}^{(k)})$, $\boldsymbol{L}=(m_{ij})$, $\boldsymbol{U}=(u_{ij})$
Prior to each step of the elimination process, the pivotal element $a_{kk}^{(k)}$ in equation (4.10) is selected to minimize the effect of rounding errors as follows:
That is, the first step is to select the largest value in the equation (4.13),

$$V_l \left| a_{lk}^{(k)} \right|, \quad l = k, k+1,..., \min(k+h_1, n) \tag{4.13}$$

and the next step is to regard the element $a_{lk}^{(k)}$ as the pivotal element, when $\boldsymbol{V}_l$ is the inverse of the largest absolute element in the l-th row of the matrix $\boldsymbol{A}$, and then the elements of the l-th and k-th rows are exchanged. If the selected pivotal element $a_{kk}^{(k)}$ satisfies

$$\left| a_{kk}^{(k)} \right| \le \max \left| a_{ij} \right| \cdot 16u$$

where $\boldsymbol{A}=(a_{ij})$ and u is a unit round off, matrix $\boldsymbol{A}^{(1)}$ is regarded as numerically singular and the processing is discontinued with ICON=20000. For more information, see References [1], [3], [4] and [8].

A52-21-0302 BMDMX, DBMDMX

A system of linear equations with a real indefinite symmetric band matrix decomposed into factors M, D, and M^T.
CALL BMDMX(B,FA,N,NH,MH,IP,IVW,ICON)

Function

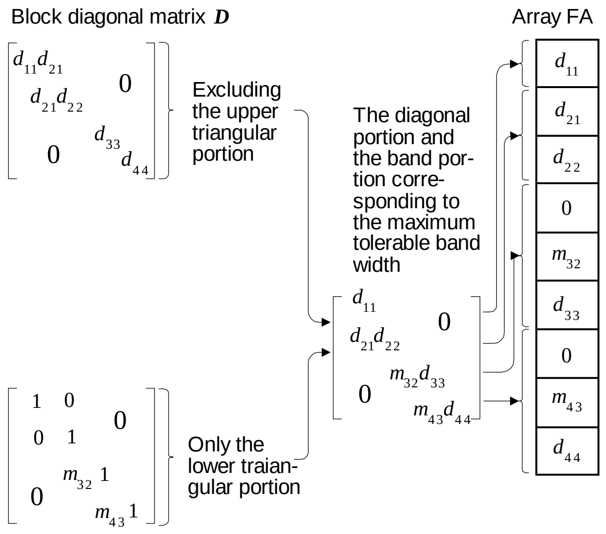
This subroutine solves a system of linear equations with a MDM^T-decomposed real symmetric band matrix.

$$\boldsymbol{P}^{-1}\boldsymbol{M}\boldsymbol{D}\boldsymbol{M}^{\mathrm{T}}\boldsymbol{P}^{-\mathrm{T}}\boldsymbol{x}=\boldsymbol{b} \qquad (1.1)$$

where $\boldsymbol{M}$ is an $n \times n$, unit lower band matrix having band width $\tilde{h}$, $\boldsymbol{D}$ is a symmetric block diagonal matrix comprising symmetric blocks each at most of order 2, $\boldsymbol{P}$ is a permutation matrix to be used to interchange rows in the pivoting procedure when the coefficient matrix is MDM^T-decomposed, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. Further, $m_{k+1,k}=0$ if $d_{k+1,k}\neq 0$, where $\boldsymbol{M}=(m_{ij})$ and $\boldsymbol{D}=(d_{ij})$ for $n>\tilde{h}\geq 0$.

Parameters

B	Input. Constant Vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. One-dimensional array of size n.
FA	Input. Matrices $\boldsymbol{M}$ and $\boldsymbol{D}$ given in the compressed mode for symmetric band matrix assuming $\boldsymbol{M}$ to have band width h_m. (See Figure BMDMX-1) One-dimensional array of size $n(h_m+1)-h_m(h_m+1)/2$.
N	Input. Order n of the matrices $\boldsymbol{M}$ and $\boldsymbol{D}$, constant vector $\boldsymbol{b}$, and solution vector $\boldsymbol{x}$.
NH	Input. Band width $\tilde{h}$ of matrix $\boldsymbol{M}$. (See "Comments on Use".)
MH	Input. Maximum tolerable band width h_m (N>MH≥NH) of matrix $\boldsymbol{M}$. (See "Comments on Use".)
IP	Input. Transposition vector indicating the history of row interchange in the pivoting procedure. One dimensional array of size n. (See "Comments on Use".)
IVW ...	Work area. One-dimensional array of size n.
ICON ..	Output. Condition code. (See Table BMDMX-1.)


Block diagonal matrix D
Array FA
Excluding the upper triangular portion
The diagonal portion and the band portion corresponding to the maximum tolerable band width
Only the lower traiangular portion

Note: In this example, orders of blocks in $\boldsymbol{D}$ are 2, 1, and 1; band width of $\boldsymbol{M}$ is 1, and the maximum tolerable band width is 2.

Fig. BMDMX-1 Storing method of matrices $\boldsymbol{M}$ and $\boldsymbol{D}$

Table BMDMX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The coefficient matrix is singular.	Bypassed.
30000	NH<0, NH>MH, N≥MH, or IP error.	Bypassed.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... MIN0

- Notes
  Simultaneous linear equations can be solved by calling this subroutine after decomposing the coefficient matrix into factors M, D, and M^T using subroutine SBMDM; however, the solution is obtained by calling subroutine LSBIX in an ordinary case.
  Input parameters FA, NH, IP, and MH of this subroutine are the same as output parameters A, NH, IP, and input parameter MH of subroutine SBMDM.

- Example
  Simultaneous linear equations are solved after decomposing an $n \times n$ real symmetric band matrix having band width h with subroutine SBMDM. Where $n \geq 100$ and $\tilde{h} \leq h_m \leq 50$.

```
C      **EXAMPLE**
       DIMENSION A(3825),B(100),IP(100),
      *          IVW(100)
       READ(5,500) N,NH,MH
       WRITE(6,600) N,NH,MH
       NT=(N+N-MH)*(MH+1)/2
       READ(5,510) (A(J),J=1,NT)
       EPSZ=0.0
       CALL SBMDM(A,N,NH,MH,EPSZ,IP,IVW,ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) STOP
       READ(5,510) (B(I),I=1,N)
       CALL BMDMX(B,A,N,NH,MH,IP,IVW,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) STOP
       WRITE(6,630) (B(I),I=1,N)
       STOP
   500 FORMAT(3I4)
   510 FORMAT(4E15.7)
   600 FORMAT('1'/10X,'N=',I3,5X,'NH=',I3,5X,
      *'MH=',I3)
   610 FORMAT(' ',5X,'ICON OF SBMDM=',I6)
   620 FORMAT(' ',5X,'ICON OF BMDMX=',I6)
   630 FORMAT(11X,'SOLUTION VECTOR'
      */(15X,5E15.6))
       END
```

Method

To solve simultaneous linear equations

$$\boldsymbol{P}^{-1}\boldsymbol{MDM}^{\mathrm{T}}(\boldsymbol{P}^{\mathrm{T}})^{-1}\boldsymbol{x}=\boldsymbol{b} \tag{4.1}$$

having coefficient expressed in MDM$^{\mathrm{T}}$-decomposed real symmetric band matrix is reduced to solve

$$\boldsymbol{Mx}^{(1)}=\boldsymbol{Pb} \tag{4.2}$$
$$\boldsymbol{Dx}^{(2)}=\boldsymbol{x}^{(1)} \tag{4.3}$$
$$\boldsymbol{M}^{\mathrm{T}}\boldsymbol{x}^{(3)}=\boldsymbol{x}^{(2)} \tag{4.4}$$
$$(\boldsymbol{P}^{\mathrm{T}})^{-1}\boldsymbol{x}=\boldsymbol{x}^{(3)} \tag{4.5}$$

where $\boldsymbol{M}$ is an $n \times n$ unit lower band matrix, $\boldsymbol{D}$ is a symmetric block diagonal matrix comprising symmetric blocks each at most of order 2, $\boldsymbol{b}$ is a constant vector, and $\boldsymbol{x}$ is a solution vector. In this subroutine, it is assumed that $\boldsymbol{M}$ and $\boldsymbol{D}$ are matrices decomposed by the block diagonal pivoting method, where $\boldsymbol{P}$ is a permutation matrix. (See "Method" for subroutine SBMDM.)

Solving $\boldsymbol{Mx}^{(1)}=\boldsymbol{Pb}$ by backward substitution.
For the 1×1 pivot (the order of blocks in matrix $\boldsymbol{D}$ is 1), the solution is obtained serially from (4.6).

$$x_i^{(1)} = b_i' - \sum_{k=1}^{i-1} m_{ik} x_k^{(1)} \quad ,i = 1,...,n \tag{4.6}$$

If 2×2 pivot (the order of blocks in matrix $\boldsymbol{D}$ is 2) is used in iteration i, $x_{i+1}^{(1)}$ is obtained after $x_i^{(1)}$ from (4.7), then iteration i+2 is processed.

$$x_{i+1}^{(1)} = b_{i+1}' - \sum_{k=1}^{i-1} m_{ik} x_k^{(1)}$$
$$M = (m_{ij}), x^{(1)^{\mathrm{T}}} = (x_1^{(1)},...,x_n^{(1)}), \tag{4.7}$$

$(\boldsymbol{Pb})^{\mathrm{T}}=(\boldsymbol{b}_1',...,\boldsymbol{b}_n')$

Solving $\boldsymbol{Dx}^{(2)}=\boldsymbol{x}^{(1)}$
For 1×1 pivot, the solution is obtained serially from

$$x_i^{(2)} = x_i^{(1)} / d_{ii}, i = 1,...,n \tag{4.8}$$

If 2×2 pivot is used in itenation i, $x_{i+1}^{(2)}$ is obtained after $x_i^{(2)}$ from (4.9), then iteration i+2 is processed.

$$\begin{aligned} x_i^{(2)} &= \left(x_i^{(1)} \cdot d_{i+1,i+1} / d_{i+1,i} - x_{i+1}^{(1)}\right) / d \\ x_{i+1}^{(2)} &= \left(x_{i+1}^{(1)} \cdot d_{ii} / d_{i+1,i} - x_i^{(1)}\right) / d \\ d &= d_{i+1,i+1} \cdot \left(d_{ii} / d_{i+1,i}\right) - d_{i+1,i} \end{aligned} \tag{4.9}$$

where $\boldsymbol{D} = (d_{ij}), \boldsymbol{x}^{(2)^{\mathrm{T}}} = (\boldsymbol{x}_1^{(2)},...,\boldsymbol{x}_n^{(2)})$

Solving $\boldsymbol{M}^{\mathrm{T}}\boldsymbol{x}^{(3)}=\boldsymbol{x}^{(2)}$ by forward substitution
For 1×1 pivot, the solution is obtained serially from

$$x_i^{(3)} = x_i^{(2)} - \sum_{k=i+1}^{n} m_{ki} x_k^{(3)} \quad , i = n,...,1 \tag{4.10}$$

If 2×2 pivot is used in iteration i, $x_{i-1}^{(3)}$ is obtained after $x_i^{(3)}$ from (4.11), then iteration i+2 is processed.

$$x_{i-1}^{(3)} = x_{i-1}^{(2)} - \sum_{k=i+1}^{n} m_{k,i-1} x_k^{(3)} \tag{4.11}$$

where. $x^{(3)^T} = (x_1^{(3)},...,x_n^{(3)})$

Solving $(\boldsymbol{P}^{\mathrm{T}})^{-1}\boldsymbol{x}=\boldsymbol{x}^{(3)}$
Elements $\boldsymbol{x}_i$ of solution vector $\boldsymbol{x}$ are obtained by multiplying the permutation matrix by vector $\boldsymbol{x}^{(3)}$. Actually, however, these elements are obtained by exchanging elements of vector $\boldsymbol{x}^{(3)}$ using values in transposition vector IP. The band structure of the coefficient matrix has been ignored for simplifying explanations above, however, the band structure is effectively used to efficiently process calculations in the actual program.

E32-32-0202 BSCD2, DBSCD2

B-spline two-dimensional smoothing coefficient calculation (variable knots)
CALL BSCD2(X,NX,Y,NY,FXY,KF,SX,SY,M,XT, NXT,YT,NYT,NXL,NYL,RNOT, C, KC,RNOR,VW, IVW,ICON)

Function

Given observed value $f_{ij}=f(x_i,y_j)$, observation error $\sigma_{i,j}=\sigma_{xi}\cdot\sigma_{yj}$, at lattice points (x_i,y_j); $i=1,2,...,n_x$, $j=1,2,...,n_y$, a tolerance for the square sum of residuals δ_t^2 and initial sequence of knots $\xi_1,\xi_2,...,\xi_{ns}$; $\eta_1,\eta_2,...,\eta_{ls}$ in the x- and y-directions, this subroutine obtains a bivariate B-spline smoothing function of degree m to the data in the sense of least squares in which the square sum of residuals is within the tolerance, by adding knots appropriately on the x- and y-axis.

Namely, letting the numbers of knots in the x- and y-directions be n_t and l_t, the subroutine obtains the coefficients $C_{\alpha,\beta}$ in the B-spline smoothing function (1.2), subject to (1.1).

$$\delta^2_{n_t+l_t}=\sum_{j=1}^{n_y}\sum_{i=1}^{n_x}\frac{1}{\left(\sigma x_i\cdot\sigma y_j\right)^2}\left\{f_{i,j}-\bar{S}\left(x_i,y_j\right)\right\}^2\leq\delta_t^2 \qquad (1.1)$$

$$\bar{S}(x,y)=\sum_{\beta=-m+1}^{l_t-1}\sum_{\alpha=-m+1}^{n_t-1}C_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (1.2)$$

This subroutine outputs knots $\xi_1,\xi_2,...,\xi_{nt}$, in the x-direction, $\eta_1,\eta_2,...,\eta_{lt}$, in the y-direction, square sum of residuals (1.3) at each step of adding knots and statistics (1.4) and (1.5), along with coefficient $C_{\alpha,\beta}$.

$$\delta^2_{n_r+l_r}=\sum_{j=1}^{n_y}\sum_{i=1}^{n_x}\frac{1}{\left(\sigma x_i\cdot\sigma y_j\right)^2}\left\{f_{i,j}-\bar{S}\left(x_i,y_j\right)\right\}^2 \qquad (1.3)$$

(where, $\bar{S}(x,y)$ denotes m-th degree B-spline smoothing function with knots $\xi_1,\xi_2,...,\xi_{nr}$ and $\eta_1,\eta_2,...,\eta_{lr}$)

$$\bar{\sigma}^2_{n_r+l_r}=\delta^2_{n_r+l_r}\Big/\left\{n_x\cdot n_y-\left(n_r+m-1\right)\left(l_r+m-1\right)\right\} \qquad (1.4)$$

$$AIC_r=n_x\cdot n_y\log\delta^2_{n_r+l_r}+2\left(n_r+m-1\right)\left(l_r+m-1\right), \qquad (1.5)$$

Here, $\sigma_{xi}>0$, $\sigma_{yj}>0$, $m\geq1$, $n_s\geq2$, $l_s\geq2$ and the initial knots ξ_i, $i=1,2,...,n_s$; η_j, $j=1,2,...,l_s$ should be satisfied.

Parameters

X	Input. Discrete points x_i in the x-direction. One-dimensional array of size n_x.
NX	Input. Number of x_i, namely n_x.
Y	Input. Discrete points y_j in the y-direction.
NY	Input. Number of y_j, namely n_y.
FXY ...	Input. Observed values f_{ij}. Two-dimensional array of FXY(KF,NY).
KF	Input. Adjustable dimension (≥NX) of array FXY.
SX	Input. Observation errors σ_{xi} in the x-direction. One-dimensional array of size n_x.
SY	Input. Observation errors σ_{yj} in y-direction. One-dimensional array of size n_y.
M	Input. Degree m of B-spline (See Notes).
XT	Input. Initial knots ξ_i, $i=1,2,...,n_s$ in the x-direction (See Notes). Output. Final knots ξ_i, $i=1,2,...,n_t$ in the x-direction. The results are output in the order $\xi_1<\xi_2<...<\xi_{nt}$. One dimensional array of size NXL.
NXT ...	Input. The number of initial knots n_s in the x-direction. Output. The number of knots n_t finally used in the x-direction.
YT	Input. Initial knots η_j, $j=1,2,...,l_s$ in the y-direction (See Notes). Output. η_j, $j=1,2,...,l_t$ in the y-direction. The results are output in the order $\eta_1<\eta_2<...<\eta_{lt}$. One-dimensional array of size NYL.
NYT ...	Input. The number of initial knots l_s in the y-direction. Output. The number of knots l_t finally used in the y-direction.
NXL ...	Input. Upper limit ($\geq n_s$) on the number of knots in the x-direction (See Notes).
NYL ...	Input. Upper limit ($\geq l_s$) on the number of knots in the y-direction (See Notes).
RNOT ..	Input. The tolerance δ_t^2 for square sum of residuals. A proper value is $\delta_t^2=n_x\cdot n_y$.
C	Output. Smoothing coefficients $C_{\alpha,\beta}$, $\alpha=-m+1,-m+2,...,l_t-1$; $\beta=-m+1,-m+2,...,n_t-1$. These are stored in C($\alpha+m$, $\beta+m$). Two-dimensional array of C(KC,NYL+M−1).
KC	Input. Ajustable dimension (≥ NXL+M-1) of Array C.
RNOR ..	Output. Values of (1.3), (1.4) and (1.5) at each step of adding knots. Two-dimensional array of RNOR(3,KR), where KR=(NXL−n_s)+(NYL−n_l)+1 Letting $n_r+l_r=n_s+l_s$, n_s+l_s+1, ... n_t+l_t, δ^2_{nr+lr} is stored in RNOR (1,P_r), $\bar{\sigma}^2_{nr+lr}$ is stored in RNOR (2,P_r), and AIC$_r$ is stored in RNOR (3,P_r), where $P_r=(n_r-n_s)+(l_r-l_s)+1$. (See Notes).
VW	Work area. One-dimensional array of size max (s_1, s_2),

where

$$s_1 = (n_x + n_y + 2m)(m+1) + \max\{\max(n_x + m, n_y + m), 2 + \min(n_x + m, n_y + m)(m+1)\}$$

$$s_2 = \{\min(n_x, n_y) + 3\}(m+1) + n_x + n_y + \mathrm{NXL} + \mathrm{NYL} + 2$$

IVW ... Work area.
One-dimensional array of size $n_x + n_y + \max(\mathrm{NXL}, \mathrm{NYL}) \cdot m$

ICON .. Output. Condition code.
See table BSCD2-1.

Table BSCD2-1 Condition code

Code	Meaning	Processing
0	No error	
10000	Although the number of knots in the *x*-direction reached the upper limit, the convergence criterion (1.1) was not satisfied.	Outputs the most recently obtained smoothing function.
11000	Although the number of knots in the *y*-direction reached the upper limit, the convergence criterion (1.1) was not satisfied.	
30000	One of the following occurred: 1 $\sigma x_i \le 0$ 2 $\sigma y_j \le 0$ 3 M<1 4 XT(I)=XT(K), where I≠K or YT(I)=YT(K), where I≠K 5 $n_s<2$ or $l_s<2$ 6 NXL<n_s or NYL<l_s 7 $\min_i(\xi_i) > \min_i(x_i)$ or $\max_i(\xi_i) < \max_i(x_i)$ 8 $\min_j(\eta_j) > \min_j(y_j)$ or $\max_j(\eta_j) < \max_j(y_j)$	Bypassed

Comments on use

- Subprogram used
  SSL II ... MGSSL, UPOB2, UPCA2, UREO1, UBAS0, UCDB2
  FORTRAN basic function ... FLOAT, IFIX, ALOG, ABS

- Notes
  By calling BSFD1 after the subroutine BSCD2, smoothed values, partial derivatives and double integrals can be obtained based on the B-spline smoothing function (1.2).
  At that time, the parameter values of M, XT, NT and C must be the input into BSFD1.

An appropriate value for degree *m* (either even or odd) is 3, but the value should not exceed 5. This is because the normal equation with respect to $C_{\alpha,\beta}$ becomes ill-conditioned as degree *m* is increased.

Initial knots ξ_i, η_j, (i=1,2,...,n_s; j=1,2,...,l_s) can be typically given by

$$n_s = l_s = 2$$
$$\xi_1 = \min_i(x_i), \quad \xi_{n_s} = \max_i(x_i)$$
$$\eta_1 = \min_j(y_j), \quad \eta_{l_s} = \max_j(y_j),$$

The upper limit NXL and NYL on the number of knots in the *x*- and *y*-directions should preferably be given by $n_x/2$ and $n_y/2$ respectively. (If the number of knots increases, the normal equation becomes ill-conditioned.) This subroutine terminates when the number of knots reaches the upper limit regardless of satisfying equation (1.1), setting ICON=10000 (for the *x*-direction) or ICON=11000 (for the *y*-direction).

The information output to RNOR is the history of various criteria in the process of adding knots at each step. The history can be used for checking the obtained smoothing function. These criteria generally decrease according to the addition of knots, the change becoming slow as step goes. Particularly when σ^2_{nr+lr} and AIC*r* are virtually unvarying the smoothing function exhibits good one. The user can check the obtained smoothing functions by printing out the contents of RNOR.

- Example
  The bivariate third degree B-spline smoothing function is obtained by inputting lattice points (x_i,y_j): i=1,2,...,80, j=1,2,...,60, observed values f_{ij}, and observation errors σ_{xi} and σ_{yj} in *x*- and *y*-directions. Initial knots are given as (3.1) and (3.2) in *x*- and *y*-directions respectively. The upper limit on the number of knots are 20 for the *x*-direction and 15 for the *y*-direction.

$$\xi_1 = x_{min} = \min_i(x_i), \xi_2 = x_{max} = \max_i(x_i) \tag{3.1}$$

$$\eta_1 = y_{min} = \min_j(y_j), \eta_2 = y_{max} = \max_j(y_j) \tag{3.2}$$

Then, letting VXr and VYs denote (3.3) and (3.4), subroutine BSFD1 computes the smoothing value at each point and the first-order partial derivatives in the *x*- and *y*-directions.

$$VXr = x_{min} + (x_{max} - x_{min})r/10 \qquad r = 0,1,...,10 \tag{3.3}$$

$$VYs = y_{min} + (y_{max} - y_{min})s/10 \qquad s = 0,1,...,10 \tag{3.4}$$

```
C     **EXAMPLE**
      DIMENSION X(80),Y(60),FXY(80,60),
     *          SX(80),SY(60),XT(20),YT(15),
     *          C(22,17),RNOR(3,32),
     *          VW(670),IVW(200),RR(3)
      NX=80
      NY=60
      READ(5,500) (X(I),SX(I),I=1,NX),
     *(Y(J),SY(J),J=1,NY)
      DO 10 I=1,NX
      DO 10 J=1,NY
      READ(5,510) FXY(I,J)
   10 CONTINUE
      M=3
      NXT=2
      NYT=2
      XMAX=X(1)
      XMIN=X(1)
      YMAX=Y(1)
      YMIN=Y(1)
      DO 20 I=2,NX
      IF(X(I).GT.XMAX) XMAX=X(I)
      IF(X(I).LT.XMIN) XMIN=X(I)
   20 CONTINUE
      DO 30 J=2,NY
      IF(Y(J).GT.YMAX) YMAX=Y(J)
      IF(Y(J).LT.YMIN) YMIN=Y(J)
   30 CONTINUE
      XT(1)=XMIN
      XT(2)=XMAX
      YT(1)=YMIN
      YT(2)=YMAX
      NXL=20
      NYL=15
      RNOT=FLOAT(NX*NY)
C
      CALL BSCD2(X,NX,Y,NY,FXY,80,SX,SY,
     *M,XT,NXT,YT,NYT,NXL,NYL,RNOT,C,22,
     *RNOR,VW,IVW,ICON)
      WRITE(6,600) ICON
      IF(ICON.EQ.30000) STOP
      WRITE(6,610)
      NT=NXT+NYT
      DO 40 I=4,NT
      IADD=I-3
      WRITE(6,620) I,(RNOR(J,IADD),J=1,3)
   40 CONTINUE
C
      HX=(XMAX-XMIN)/10.0
      HY=(YMAX-YMIN)/10.0
      DO 70 I=1,11
      VX=XMIN+HX*FLOAT(I-1)
      WRITE(6,630) VX
      WRITE(6,640)
      DO 60 J=1,11
      VY=YMIN+HY*FLOAT(J-1)
      DO 50 K=1,3
      KM1=K-1
      ISWX=MOD(KM1,2)
      ISWY=KM1/2
      CALL BSFD1(M,XT,NXT,YT,NYT,C,22,
     *ISWX,VX,IX,ISWY,VY,IY,RR(K),VW,ICON)
   50 CONTINUE
      WRITE(6,650) VY,(RR(K),K=1,3)
   60 CONTINUE
   70 CONTINUE
      STOP
  500 FORMAT(2F10.0)
  510 FORMAT(F10.0)
  600 FORMAT(10X,'ICON=',I5//)
  610 FORMAT(8X,'NO. OF KNOTS',9X,
     *'RNOR(1,*)',11X,'RNOR(2,*)',11X,
     *'RNOR(3,*)'/)
  620 FORMAT(10X,I2,8X,3(5X,E15.8))
  630 FORMAT(//5X,'VX=',E15.8)
  640 FORMAT(16X,'VY',13X,'SMOOTHED VALUE',
     *8X,'DS(X,Y)/DX',10X,'DS(X,Y)/DY')
  650 FORMAT(5X,4(5X,E15.8))
      END
```

Method

This subroutine obtains bivariate m-th degree B-spline smoothing function by adding knots so that square sum of residuals may become less than the given tolerance δ_t^2.

Now, let us assume $\xi_1, \xi_2, ..., \xi_{nr}$ and $\eta_1, \eta_2, ..., \eta_{lr}$ are given in the x- and y-direction respectively which satisfy the following:

$\xi_1<\xi_2<.....<\xi_{nr}$

$\eta_1<\eta_2<.....<\eta_{lr}$

At the initial step, $n_r=n_s$ and $l_r=l_s$. The coefficient $C_{\alpha,\beta}$ of the bivariate m-th degree spline function

$$\bar{S}(x,y)=\sum_{\beta=-m+1}^{l_{r-1}}\sum_{\alpha=-m+1}^{n_{r-1}}C_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \quad (4.1)$$

in which the above knots are used, is determined so that the square sum of residuals

$$\sum_{j=1}^{n_y}\sum_{i=1}^{n_x}\frac{1}{\sigma x_i^2\cdot\sigma y_j^2}\left\{f_{i,j}-\bar{S}(x_i,y_j)\right\}^2 \quad (4.2)$$

assumes the minimum. Such $C_{\alpha,\beta}$ can be obtained by solving the system of linear equations (normal equations) resulting from partially differentiating (4.2) with respect to $C_{\alpha,\beta}$ and equating it to zero. The value of (4.2) corresponding to the obtained $C_{\alpha,\beta}$ is assumed to be δ^2_{nr+lr}.

If $\delta^2_{nr+lr}\le\delta^2_t$, (4.1) can be taken as the bivariate m-th degree B-spline smoothing function. If $\delta^2_{nr+lr}>\delta^2_t$, determine the new knots to be added under the following procedures.

Compute residuals $\varepsilon_{ij}=f_{ij}-\bar{S}(x_i,y_j)$, $i=1,2,...n_x$, $j=1,2,...,n_y$ at each point.

Using these values, compute

$$u_i=\sum_{\{r|\xi_i\le x_r<\xi_{i+1}\}}\frac{1}{\sigma x_r^2\cdot n_y}\left\{\sum_{s=1}^{n_y}\frac{\varepsilon_{r,s}}{\sigma y_s}\right\}^2$$

$$v_j=\sum_{\{s|\eta_j\le y_s<\eta_{j+1}\}}\frac{1}{\sigma y_s^2\cdot n_x}\left\{\sum_{r=1}^{n_x}\frac{\varepsilon_{r,s}}{\sigma x_r}\right\}^2 \quad (4.4)$$

The maximum values of $\{u_i|1\le i\le n_r-1\}$, $\{v_j|1\le j\le l_r-1\}$ are assumed to be u_i and v_j. When u_i is greater than v_j, n_r is increased by 1 assuming $\xi=(\xi_i+\xi_{i+1})/2$ to be added. On the other hand, when v_j greater than u_i, l_r is increased by 1 assuming $\eta=(\eta_j+\eta_{j+1})/2$ to be added.

Replacing the updated knots in the *x*- or *y*-direction in the ascending order, repeat the above mentioned procedures.

This is repeated until the square sum of square becomes less than the given value δ_t^2, updating knots and increasing the number of knots subsequently.

B21-21-0502 BSCT1, DBSCT1

Selected eigenvalues of a real symmetric tridiagonal matrix (Bisection method)
CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)

Function

Using the bisection method, the *m* largest or *m* smallest eigenvalues of an *n*-order real symmetric tridiagonal matrix ***T*** are determined. $1 \le m \le n$.

Parameters

D	Input. Diagonal elements of tridiagonal matrix ***T***. D is a one-dimensional array of size *n*.
SD	Input. Subdiagonal elements of tridiagonal matrix ***T***. SD is a one-dimensional array of size *n*. The elements are stored in SD(2) to SD(N).
N	Input. Order *n* of the tridiagonal matrix.
M	Input. M=*m* ... The number of largest eigenvalues desired. M=−*m* ... The number of smallest eigenvalues desired.
EPST ..	Input. The absolute error tolerance used to determine accuracy of eigenvalues (refer to equation (4.9) in the Method section). when a negative value is given, an appropriate default value is used.
E	Output. *m* eigenvalues. Order in descending order if M is positive and in ascending order if M is negative. E is a one-dimensional array of size *m*.
VW	Work area. VW is a one-dimensional array of size *n*+2*m*.
ICON ..	Output. Condition code. Refer to Table BSCT1-1.

Table BSCT1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	E(1)=D(1)
30000	N<|M| or M=0	Bypassed

Comments on use

- Subprogram used
  SSL II ... AMACH and MGSSL
  FORTRAN basic function ... IABS,ABS and AMAX1

- Notes
  Normally, when determining the eigenvalues of a real symmetric matrix, the subroutine TRID1 is used first to reduce that matrix to a tridiagonal matrix, then this routine or TRQL can be used to determine the eigenvalues.
  If *n*/4 or more eigenvalues are being determined, computation time is generally better using TRQL instead of this routine.
  If the possibility of an eigenvalue being zero exists, refer to "Convergence criterion and EPST parameter" in the Method section, then specify EPST accordingly.

- Example
  After an *n*-order real symmetric matrix is first reduced to a triagonal matrix using TRID1, *m* eigenvalues are calculated. *n*≤100.

```
C     **EXAMPLE**
      DIMENSION A(5050),D(100),SD(100),
     *E(100),VW(300)
   10 READ(5,500) N,M,EPST
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N,M
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+1
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      CALL TRID1(A,N,D,SD,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)
      WRITE(6,630)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      MM=IABS(M)
      WRITE(6,640) (I,E(I),I=1,MM)
      GO TO 10
  500 FORMAT(2I5,E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'/
     *11X,'** ORDER =',I5,10X,'** M =',I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT(/11X,'** CONDITION CODE =',I5/)
  630 FORMAT('0'/11X,'** EIGENVALUES')
  640 FORMAT(5X,'E(',I3,')=',E15.7)
      END
```

Method

Using the bisection method, the *m* largest or *m* smallest eigenvalues are determined from *n*-order real symmetric tridiagonal matrix ***T***.

If a subdiagonal element of real symmetric tridiagonal matrix ***T*** shown in Fig. BSCT1-1 is zero, that matrix can be split, into submatrices at that point. If ***T*** can be split the eigenvalues of ***T*** can be obtained by determining the eigenvalues of each submatrix. Since the bisection method is applied to each submatrix, in the explanation below, ***T*** is assumed not to be split, i.e., $b_i \neq 0$.

For the matrix ($\boldsymbol{T}-\lambda\boldsymbol{I}$) shown in Fig. BSCT1-2, the value λ that satisfy

$$\det(\boldsymbol{T}-\lambda\boldsymbol{I}) = 0 \tag{4.1}$$

are the eigenvalues of $\boldsymbol{T}$. Let the leading principle minor of the matrix ($\boldsymbol{T}-\lambda\boldsymbol{I}$) be $\boldsymbol{P}_i(\lambda)$, then the recurrence formula in (4.2) can be developed.

$$\begin{aligned} &P_0(\lambda)=1,\ P_1(\lambda)=c_1-\lambda \\ &P_i(\lambda)=(c_i-\lambda)P_{i-1}(\lambda)-b_i^2P_{i-2}(\lambda), \\ &\qquad\qquad i=2,3,\ldots,n \end{aligned} \tag{4.2}$$

The polynomials $P_0(\lambda)$, $P_1(\lambda)$, ..., $P_n(\lambda)$ of (4.2) is a Sturm sequence. Let $\boldsymbol{L}(\lambda)$ be the number of agreements in sign of consecutive members of this sequence, then $\boldsymbol{L}(\lambda)$ is equal to the number of eigenvalues which are smaller than λ.

However, if $P_i(\lambda)=0$, the sign of $P_{i-1}(\lambda)$ is used. In the actual calculations, (4.2) is replaced by the sequence:

$$q_i(\lambda)=P_i(\lambda)/P_{i-1}(\lambda), i=1,2,\ldots,n \tag{4.3}$$

Using (4.3), $\boldsymbol{L}(\lambda)$ is easily determined. $\boldsymbol{L}(\lambda)$ is the number of cases that the sequence $q_i(\lambda)$ yields a positive or zero value. From (4.2) and (4.3) $q_i(\lambda)$ is

$$\left.\begin{aligned} &q_1(\lambda)=c_1-\lambda \\ &q_i(\lambda)=(c_i-\lambda)-b_i^2/q_{i-1}(\lambda), i=2,3,\ldots,n \end{aligned}\right\} \tag{4.4}$$

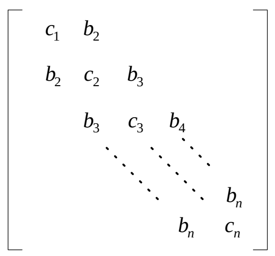


Fig. BSCT1-1 Real symmetric tridiagonal matrix T

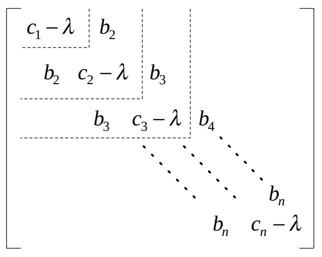


Fig. BSCT1-2 Matrix [T-λI]

If $q_{i-1}(\lambda) = 0$, then

$$q_i(\lambda)=(c_i-\lambda)-|b_i|/u \tag{4.5}$$

where, u is the unit round-off.

Using this method overflow and underflow can be avoided, and $\boldsymbol{L}(\lambda)$ can still be calculated even if $b_i = 0$.

Now consider determining the largest k-th eigenvalue. Suppose the eigenvalues have the relationship

$$\lambda_1\geq\lambda_2\geq.....\geq\lambda_k\geq.....\geq\lambda_n \tag{4.6}$$

1) Using Gershgorin's Method, interval $[l_0,r_0]$ which includes all n eigenvalues is determined.

$$\begin{aligned} r_0 &= \max_{1\leq i\leq n}\{c_i+(|b_i|+|b_{i+1}|)\} \\ l_0 &= \min_{1\leq i\leq n}\{c_i-(|b_i|+|b_{i+1}|)\} \end{aligned} \tag{4.7}$$

where $b_1=0$, $b_{n+1}=0$.

2) Iterations of (4.8) are continued until λ_k is approximately in the midpoint of interval $[l_j,r_j]$

$$\left.\begin{aligned} &j=0,1,2,\cdots\cdots \\ &h_j=(l_j+r_j)/2 \\ \\ &\text{If } L(h_j)<k \text{ then take } l_{j+1}=l_j \text{ and } r_{j+1}=h_j \\ &\text{If } L(h_j)\geq k \text{ then take } l_{j+1}=h_j \text{ and } r_{j+1}=r_j \end{aligned}\right\} \tag{4.8}$$

Interval $[l_j,r_j]$ in which λ_k lies is bisected with each interation of (4.8).

Using 1) and 2), this routine determines the interval in which the m largest eigenvalues lie. Then in that interval, eigenvalues are determined for each submatrix. When m eigenvalues have been obtained, processing is terminated.

Convergence criterion and EPST Parameters

In this routine, convergence is determined by

$$r_j-l_j\leq 2u(|l_j|+|r_j|)+\text{EPST} \tag{4.9}$$

EPST is specified as the allowable absolute error tolerance in determining the eigenvalues. When (4.9) is satisfied, $(l_j+r_j)/2$ is considered an eigenvalue. If EPST=0.0, (4.9) becomes

$$r_j-l_j\leq 2u(|l_j|+|r_j|) \tag{4.10}$$

In this case, bisection of the interval is continued until the least significant digits of l_j and r_j become approximately equal, such that it takes a long time to compute the eigenvalue. EPST is used to indicate the precision at which this computation is terminated. EPST is also a safeguard, since (4.10) would never be satisfied for eigenvalues which are zero.

When a negative EPST is specified, for each submatrix, the following default value is used.

$$\mathrm{EPST} = u \cdot \max(|l_0|, |r_0|) \tag{4.11}$$

Where l_0 and r_0 are the upper and lower limits of the range obtained by the Gerschgorin method which includes eigenvalues of each submatrix. For further information see References [12] pp.299-302, [13] pp.249-256, and [15].

E32-31-0102 BSC1, DBSC1

B-spline smoothing coefficient calculation (with fixed knots)
CALL BSC1(X, Y, W, N, M, XT, NT, C, R, RNOR, VW, IVW, ICON)

Function

Given observed values $y_1, y_2, ..., y_n$ at points $x_1, x_2, ..., x_n$, weighted function values $w_i=w(x_i)$, i=1,2,...,n, and knots of the spline function $\xi_1, \xi_2, ..., \xi_{ni}$, the B-spline smoothing function in the sense of least squares is obtained. In other words, 1et

$$\bar{S}(x)=\sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(x) \qquad (1.1)$$

be the m-th (m: either an odd or even integer) degree B-spline smoothing function to be obtained, and then the smoothing coefficients c_j's which minimize the sum of squares of the weighted residual:

$$\delta_m^2=\sum_{i=1}^{n} w_i\left\{y_i-\bar{S}(x_i)\right\}^2 \qquad (1.2)$$

are obtained.

The interval I_ξ=[min(ξ_j),max(ξ_j)] spanned by the knots {ξ_i} does not always have to contain all of the n discrete points. For example, as shown in Fig. BSC1-1, the I_ξ can be specified as a part of the interval I_x=[min(x_i),max(x_i)] which is spanned by all of the discrete points. If so, $\bar{S}(x)$ given by (1.1) such discrete points, (whose number is, say, n_e) as contained in the interval so, when taking the summation in (1.2), only the discrete points contained in the interval I_ξ have to be taken into consideration.

Here, w_i≥0, m≥1, n_t≥3, $\xi_j \neq \xi_k$ ($j \neq k$) and $n_e \geq n_t+m$-1.

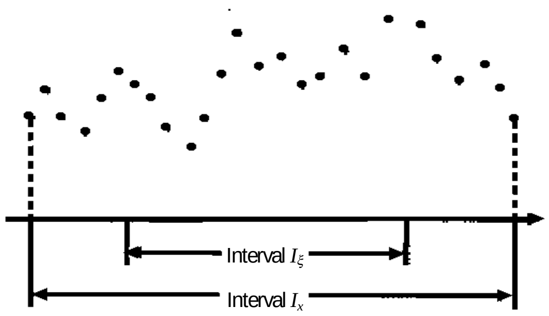


Fig. BSC1-1 Section I_ξ for smoothing function

Parameters

X.......... Input. Discrete points x_i's.
One-dimensional array of size n.

Y.......... Input. Observed values y_i.
One-dimensional array of size n.

W......... Input. Weighted function values.
One-dimensional array of size n.

N.......... Input. Number of the discrete points.

M......... Input. Degree of the B-spline. See Notes.

XT .... Input. Knots ξ_j's. See Notes.
One-dimensional array of size n_t.
Output. If on input
XT(1)<XT(2)<...<XT(n_t)
was not satisfied, XT's are aligned to satisfy it on output.

NT .... Input. Number of the knots, n_t.

C.......... Output. Smoothing coefficients c_j's.
One-dimensional array of size n_t+m-1,

R.......... Output. Residuals $y_i-\bar{S}(x_i)$, i=1,2,...,n. One-dimensional array of size n.

RNOR. Output. Square sum of the weighted residual, δ_m^2.

VW...... Work area. One-dimensional array of size (n_t+m)(m+1).

IVW..... Work area. One-dimensional integer type array of size n.

ICON... Output. Condition code.
See Table BSC1-1.

Table BSC1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the followings occurred: (a) At least one w_i is negative. (b) M<1 (c) XT(I)=XT(K) (I≠K) (d) NT<3 (e) n_e<NT+M-1	Bypassed

Comments on use

- Subprograms used
SSL II ... MGSSL, UREO1, UNCA1, UCDB1, UCAO1 and UBAR1
FORTRAN basic function ... DSQRT

- Notes
By calling subroutine BSF1 after subroutine BSC1, the interpolated values as well as derivatives or integrals can be obtained based on the B-spline smoothing function (1.1). The parameter values of M, XT, NT and C are passed from BSC1 to input to BSF1.
The degree m is preferably 3 but no greater than 5, because the normal equation (see "Method") used when obtaining the smoothing coefficients c_j's become

ill-conditioned as *m* becomes large.

It is important for the knots ξ_i to be located according to the behavior of observed values. In general, a knot should be assigned to the point at which the observed values have a peak or change rapidly. Any knot should not be assigned in an interval where the observed values change slowly. (See Fig. BSC1-2.)

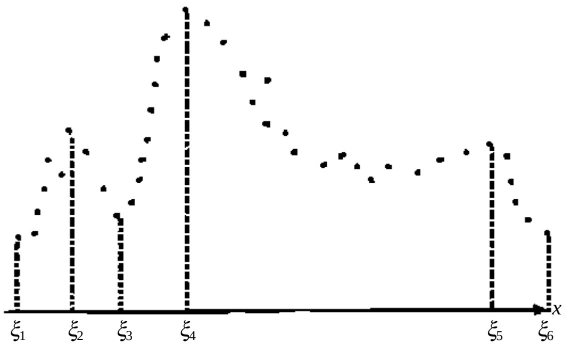


Fig.BSC1-2 Knots ξ_i

- Example
  See the example given for subroutine BSF1.

Method

By setting the *m*-th degree B-spline smoothing function as

$$\bar{S}(x)=\sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(x) \tag{4.1}$$

the subroutine obtains the smoothing coefficients c_j's which minimize the sum of squares of the weighted residuals.

$$\delta_m^2=\sum_{i=1}^{n} w_i \left\{ y_i - \bar{S}(x_i) \right\}^2 \tag{4.2}$$

The interval in which $\bar{S}(x)$ is defined is I_ξ = [min(ξ_j), max(ξ_j)] spanned by the user specified knots {ξ_j}. For simplicity let's assume the knots {ξ_j} satisfy the relationship $\xi_j<\xi_{j+1}$ (j=1,2,...,n_t-1), and all of the *n* discrete points x_i's are contained in the interval [ξ_l,ξ_{nt}].

First, the knots {t_j} of the B-spline $N_{j,m+1}(x)$ are taken by using {ξ_j} as follows (see Fig. BSC1-3):

$$t_j \begin{cases} \xi_1 & ,-m+1 \le j \le 0 \\ \xi_j & ,1 \le j \le n_t \\ \xi_{n_t} & ,n_t+1 \le j \le n_t+m \end{cases} \tag{4.3}$$

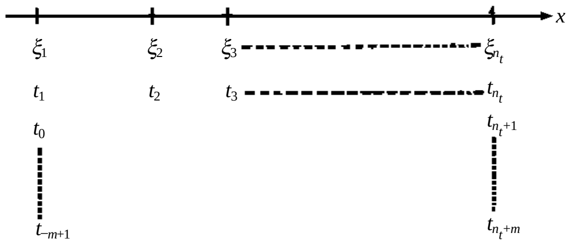


Fig. BSC1-3 Knots {t_j}

Among all the spline functions represented as (4.1), the one which minimizes (4.2) can be obtained by solving so-called a normal equation with respect to c_j's. That is, taking partial derivatives of $\delta_m{}^2$ in (4.2) with respect to c_j's and setting them to zero, the normal equation with respect to c_j's can be obtained as follows.

$$\sum_{j=-m+1}^{n_t-1} \left\{ \sum_{i=1}^{n} w_i N_{k,m+1}(x_i) N_{j,m+1}(x_i) \right\} c_j = \sum_{i=1}^{n} w_i y_i N_{k,m+1}(x_i)$$

$$k=-m+1,\ -m+2,\ .....,\ n_t-1 \tag{4.4}$$

The above is a system of linear equations of order (n_t+*m*-1) and the coefficient matrix is of the form of a symmetric band matrix. Fig. BSC1-4 shows an example of the matrix for *m*=3 an n_t=5.

$$\begin{bmatrix} * & * & * & * & & & 0 \\ * & * & * & * & * & & \\ * & * & * & * & * & * & \\ * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ 0 & & & * & * & * & * \end{bmatrix}$$

Fig. BSC1-4 Coefficient matrix (for *m*=3 and n_t=5)

Therefore, solving Eq. (4.4) gives c_j's. This subroutine solves the linear equations above by using Cholesky method (L^TL decomposition method) with the slave subroutines UNCA1 and UCDB1.

E32-31-0202 BSC2, DBSC2

B-spline smoothing coefficient calculation (variable knots)
CALL BSC2(X,Y,S,N,M,XT,NT,NL,RNOT,C, RNOR,VW,IVW,ICON)

Function

Given observed values $y_1, y_2, ..., y_n$ at discrete points $x_1, x_2, ..., x_n$, observation errors $\sigma_1, \sigma_2, ..., \sigma_n$, a tolerance δ_t^2 for the square sum of residuals, and initial knots $\xi_1, \xi_2, ..., \xi_{n_s}$, then this subroutine obtains a B-spline smoothing function of degree m to the data in the sense of least squares, by adding knots so that the square sum of residuals becomes within the tolerance.

Namely, letting n_t denote the number of knots finally used, and $\delta_{n_t}^2$ the corresponding square sum of residuals, the subroutine obtain the coefficients c_j in the B-spline smoothing function (1.1), subject to (1.2).

$$\delta_{n_t}^2 = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}\left\{y_i - \bar{S}(x_i)\right\}^2 \le \delta_t^2 \qquad (1.1)$$

$$\bar{S}(x) = \sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(x) \qquad (1.2)$$

This subroutine outputs final knots $\xi_1, \xi_2, ..., \xi_{nt}$, square sum of residuals (1.3) at each step in which knots are added and the statistics (1.4) and (1.5).

$$\delta_{n_r}^2 = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}\left\{y_i - \bar{S}(x_i)\right\}^2 \qquad (1.3)$$

(where $\bar{S}(x)$ is an m-th degree B-spline smoothing function in which $\xi_1, \xi_2, ..., \xi_{nr}$ are knots.)

$$\bar{\sigma}_{n_r}^2 = \delta_{n_r}^2 \Big/ \left\{n - (n_r + m - 1)\right\} \qquad (1.4)$$

$$\begin{aligned} &\text{AIC}r = n \log \delta_{n_r}^2 + 2(n_r + m - 1) \\ &n_r = n_s, n_s + 1, ..., n_t \end{aligned} \qquad (1.5)$$

Here, $\sigma_i>0$, $m\ge1$, $n_s\ge2$ and initial knots ξ_j must satisfy
$\min_j(\xi_j) \le \min_i(x_i), \quad \max_j(\xi_j) \ge \max_i(x_i)$

Parameters

X.......... Input. Discrete points x_i.
One-dimensional array of size n.

Y.......... Input. Observed values y_i.
One-dimensional array of size n.

S .... Input. Observation errors σ_i. (See Notes.)
One-dimensional array of size n.

N.......... Input. Number n of the discrete points.

M......... Input. Degree m of the B-spline. (See Notes.)

XT .... Input. Initial knots ξ_j, j=1,2,...,n_s. (See Notes.)
Output. Knots ξ_j, j=1,2,...,n_t finally used. The results are output in the order of $\xi_1<\xi_2<...<\xi_{nt}$.
One-dimensional array of size NL.

NT .... Input. Number n_s of initial knots.
Output. Number n_t of knots finally used.

NL .... Input. The upper limit on the number of knots. (See Notes.)

RNOT.. Input. The tolerance δ_t^2 for the square sum of residuals. A proper value is $\delta_t^2=n$.

C.......... Output. Smoothing coefficients C_j, j=-m+1, -m+2, ..., n_t-1.
C_j is stored in $C(j+m)$.
One-dimensional array of size (NL+M-1)

RNOR.. Output. Values of (1.3), (1.4) and (1.5) at each step in which knots are added.
Two-dimensional array RNOR(3,NL-n_s+1).
Letting n_r=n_s,n_s+1,...,n_t,
$\delta_{n_r}^2$ is stored in RNOR(1,n_r-n_s+1),
$\bar{\sigma}_{n_r}^2$ is stored in RNOR(2,n_r-n_s+1),
AICr is stored in RNOR(3,n_r-n_s+1).
(See Notes.)

VW...... Work area.
One-dimensional array of size (M+1) + (M+2) (NL+M).

IVW..... Work area.
One-dimension/array of size (N+NL+M).

ICON... Output. Condition code.
See Table BSC2-1.

Table BSC2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Although the number of knots reached the upper limit, the convergence criterion (1.1) was not satisfied.	Outputs the most recently obtained smoothing function.
30000	One of the followings occurred: 1 $\sigma_i\le0$ 2 M<1 3 XT(I)=XT(K) where I≠K 4 n_s<2 5 NL<n_s 6 $\min_j(\xi_j) > \min_i(x_i)$ or $\max_j(\xi_j) < \max_i(x_i)$	Bypassed

Comments on use

- Subprograms used
  SSL II ... AFMAX, MGSSL, UPOB1, UPCA1, UREO1, UBAS0, UCDB2
  FORTRAN basic function ... SQRT, IFIX, ABS, FLOAT, ALOG

- Notes

  By calling subroutine BSF1 after this subroutine, smoothed values, derivatives, or integrals can be obtained based on (1.2). At that time, the parameter values of M, XT, NT and C must be the same as those used in BSF1.

  An appropriate value for degree *m* (either even or odd) is 3, but the value should not exceed 5. This is because the normal equation with respect to smoothing coefficient C_j becomes ill-conditioned as degree *m* increases.

  Initial knots ξ_j, j=1,2,...,n_s can be generally given by

$$n_s = 2$$
$$\xi_1 = \min_i(x_i), \quad \xi_{n_s} = \max_i(x_i)$$

  Observation error σ_i is an estimate for the error contained in the observed value y_i. For example, if y_i has d_i significant decimal digits, the value $10^{-d_i}|y_i|$ can be used as σ_i. σ_i is used to indicate how closely $\bar{S}(x)$ should be fit to y_i. The larger σ_i is, the less closely $\bar{S}(x)$ is fit to y_i.

  Upper limit NL on the number of knots should be given a value near *n*/2 (if the number of knots increases, the normal equation becomes ill-conditioned). This subroutine terminates processing by setting ICON=10000 when the number of knots reaches the upper limit regardless of (1.1).

  The information output to RNOR is the history of various criteria in the process of adding knots at each step. The history can be used for checking the obtained smoothing function.

- Example

  By inputting 100 discrete points x_i, observed values y_i and observation errors σ_i, a third degree B-spline smoothing function is obtained.

  Here, the initial knots are taken as

$$\xi_1 = x_{\min} = \min_i(x_i), \quad \xi_2 = x_{\max} = \max_i(x_i)$$

  and the upper limit on the number of knots to be 20.

  Subsequently smoothed values and 1st through 3rd order derivatives are computed at each point of

$$v_j = \xi_1 + (x_{\max} - x_{\min}) \cdot j/50 \qquad j = 0,1,...,50$$

  by using subroutine BSF1.

```
C      **EXAMPLE**
       DIMENSION X(100),Y(100),S(100),XT(20),
      *          C(25),RNOR(3,20),VW(120),
      *          IVW(125),RR(4)
       N=100
       READ(5,500) (X(I),Y(I),S(I),I=1,N)
       M=3
       NT=2
       XMAX=X(1)
       XMIN=X(1)
       DO 10 I=2,N
       IF(X(I).GT.XMAX) XMAX=X(I)
       IF(X(I).LT.XMIN) XMIN=X(I)
    10 CONTINUE
       XT(1)=XMIN
       XT(2)=XMAX
       NL=20
       RNOT=FLOAT(N)
C
       CALL BSC2(X,Y,S,N,M,XT,NT,NL,RNOT,
      *C,RNOR,VW,IVW,ICON)
       WRITE(6,600) ICON
       IF(ICON.EQ.30000) STOP
       WRITE(6,610)
       DO 20 I=2,NT
       IADD=I-1
       WRITE(6,620) I,(RNOR(J,IADD),J=1,3)
    20 CONTINUE
C
       H=(XMAX-XMIN)/50.0
       WRITE(6,630)
       DO 40 J=1,51
       V=XT(1)+H*FLOAT(J-1)
       DO 30 L=1,4
       ISW=L-1
       CALL BSF1(M,XT,NT,C,ISW,V,I,
      *          RR(L),VW,ICON)
    30 CONTINUE
       WRITE(6,640) V,(RR(L),L=1,4)
    40 CONTINUE
       STOP
C
   500 FORMAT(3F10.0)
   600 FORMAT(10X,'ICON=',I5//)
   610 FORMAT(8X,'NO. OF KNOTS',9X,
      *'RNOR(1,*)',11X,'RNOR(2,*)',11X,
      *'RNOR(3,*)'/)
   620 FORMAT(10X,I2,8X,3(5X,E15.8))
   630 FORMAT(//14X,'ARGUMENT',9X,
      *'SMOOTHED VALUE',8X,'1ST DERIV.',
      *10X,'2ND DERIV.',10X,
      *'3RD DERIV.'/)
   640 FORMAT(10X,E15.8,4(5X,E15.8))
       END
```

Method

This subroutine obtain the *m*-th degree B-spline smoothing function such that the square sum of residuals is less than a given tolerance δ_t^2, by adding knots adaptively.

Suppose that knots ξ_1, ξ_2, ..., ξ_{nr} have already been determined and arranged in the order

$$\xi_1<\xi_2<.....<\xi_{nr}$$

where, initially n_r=n_s. The coefficients C_j in *m*-th degree spline with the knots above

$$\bar{S}(x) = \sum_{j=-m+1}^{n_r-1} c_j N_{j,m+1}(x) \tag{4.1}$$

are determined so that the square sum of residuals

$$\sum_{i=1}^{n} \frac{1}{\sigma_i^2}\{y_i - \bar{S}(x_i)\}^2 \tag{4.2}$$

may assume the minimum. (For details, refer toBSC1). Denoting the value of (4.2) with the obtained C_j by δ_{nr}^2 , if $\delta_{nr}^2 \le \delta_t^2$, (4.1) can be taken as the *m*-th degree B-spline smoothing function.

If $\delta_{nr}^2 > \delta_t^2$, the square sum of residuals

$$\delta^2(j, n_r) = \sum_{\xi_j \le x_i < \xi_{j+1}} \frac{1}{\sigma_i^2} \left\{ y_i - \bar{S}(x_i) \right\}^2 \tag{4.3}$$

for each subinterval $[\xi_j, \xi_{j+1}]$ is computed. Then we add, to the current set of knots, the midpoint $\xi = (\xi_i + \xi_{i+1})/2$ of the interval in which $\delta^2(j, n_r)$ takes the maximum.

The updated knots are arranged in ascending order, and letting this be ξ_1, ξ_2, ..., ξ_{nr+1}, then the above process is repeated. This processing is repeated until the square sum of squares becomes less than the tolerance δ_t^2 .

B51-21-0201 BSEG, DBSEG

Eigenvalues and eigenvectors of a real symmetric band matrix (Rutishauser-Schwarz method, bisection method and inverse iteration method)

CALL BSEG (A, N, NH, M, NV, EPST, E, EV, K, VW, ICON)

Function

This subroutine obtains the largest or smallest m eigenvalues of a real symmetric band matrix $\boldsymbol{A}$ of order n and band width h by using the Rutishauser-Schewarz method and the bisection method and also obtains the corresponding n_v eigenvectors by using the inverse iteration method. The eigenvectors are normalized so as to satisfy $\|x\|_2 = 1$. Here, $1 \le m \le n$, $0 \le n_v \le m$ and $0 \le h << n$.

Parameters

A.......... Input. Real symmetric band matrix $\boldsymbol{A}$.
Compressed mode for a symmetric band matrix.
One-dimensional array of size $n(h+1)-h(h+1)/2$
When $n_v \ne 0$, the contents are not destroyed after computation
When $n_v=0$, the contents are altered on output.

N.......... Input. Order n of the matrix $\boldsymbol{A}$.

NH....... Input. Bandwidth h.

M......... Input. The m number of eigenvalues to be obtained.
M=+m ... The largest eigenvalues are obtained.
M=-m ... The smallest eigenvalues are obtained.

NV....... Input. The n_v number of eigenvectors are obtained
When NV=-n_v, replaced as NV=+n_v in the subroutine.
When n_v=0, no eigenvectors are calculated.

EPST ... Input. Upper limit of absolute error used for convergence cirterion to eigenvalues obtained.
See "Method" for the subroutine BSCT1.
If EPST<0, a standard upper limit is set.

E .... Output. Eigenvalues. One-dimensional array of size m.

EV .... Output. Eigenvectors.
The eigenvectors are stored in the column-wise direction.
Two dimensional arrays as represented by EV(K,NV).

K.......... Input. Adjustable dimension of the array EV.
When n_v=0, this is an arbitrary number.

VW...... Work area. One-dimensional array of size max($3n+2m$,$2n(h+1)$).
When n_v=0, the size is as large as $3n+2m$.

ICON ... Output. Condition code.
See Table BSEG-1.

Table BSEG-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	NH=0	Executed normally.
15000	All of the eigenvectors could not be obtained.	Unobtained eigenvectors are set to 0 vectors.
20000	None of the eigenvectors could be obtained.	All of the eigenvectors become 0 vectors.
30000	NH<0, NH≥N, N>K, M=0, $\lvert M\rvert < \lvert NV\rvert$ or $\lvert M\rvert > N$	Bypassed

Comments on use

- Subprograms used
  SSL II ... BTRID, BSCT1, BSVEC, AMACH and MGSSL
  FORTRAN basic function ... IABS

- Notes
  This subroutine is provided for a real symmetric band matrix, and is suitable for obtaining eigenvalues, from either the largest or smallest eigenvalues, to a matrix whose ratio of its bandwidth h to order n, (i.e., h/n), is no larger than 1/6. Although the eigenvectors corresponding to the obtained eigenvalues can be calculated at the same time, since the inverse iteration method used in this subroutine is applied not for a real symmetric tridiagonal matrix, but for directly processing the input band matrix, the method is relatively ineffective. Consequently, unnecessary eigenvectors should not be calculated. If NV=0 is set, no eigenvectors have to be obtained. If a very small number of eigenvalues of a real symmetric band matrix of high order needs to be obtained, from either the largest or smallest eigenvalues in absolute value, and at the same time the corresponding eigenvectors are to be calculated as well, either this method or simultaneous iteration method by Jennings, whichever is better, should be adopted.

- Example
  The largest or smallest m eigenvalues of a real symmetric band matrix
  A of order n and bandwidth h and also the corresponding n_v eigenvectors are obtained in this example below for $n \le 100$, $h \le 10$, and $m \le 10$.

```
C      **EXAMPLE**
       DIMENSION A(1100),E(10),EV(100,10)
      *          ,VW(2100)
   10  READ(5,500) N,NH,M,NV,EPST
       IF(N.EQ.0) STOP
       NN=(NH+1)*(N+N-NH)/2
       READ(5,510) (A(I),I=1,NN)
       WRITE(6,600) N,NH,M,NV
       NE=0
       DO 20 I=1,N
       NI=NE+1
       NE=MIN0(NH+1,I)+NE
   20  WRITE(6,610) I,(A(J),J=NI,NE)
       CALL BSEG(A,N,NH,M,NV,EPST,E,EV,100,
      *          VW,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) GO TO 10
       MM=IABS(M)
       NNV=IABS(NV)
       IF(NNV.EQ.MM) GO TO 40
       NNV1=NNV+1
       DO 30 J=NNV1,MM
       DO 30 I=1,N
   30  EV(I,J)=0.0
   40  CALL SEPRT(E,EV,100,N,MM)
       GO TO 10
  500  FORMAT(4I5,E15.7)
  510  FORMAT(5E15.7)
  600  FORMAT('1',10X,'** ORIGINAL MATRIX'/
      *       11X,'** ORDER =',I5,10X,'NH=',
      *       I3,10X,'M=',I3,10X,'NV=',I3/)
  610  FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620  FORMAT(/11X,'** CONDITION CODE =',I5/)
       END
```

For subroutine SEPRT, see the example of the subroutine SEIG1.

Method

The largest or smallest m eigenvalues of a real symmetric band matrix $\boldsymbol{A}$ of order n and bandwidth h as well as the corresponding n_v eigenvectors are obtained.

First, the matrix $\boldsymbol{A}$ is reduced to a tridiagonal matrix $\boldsymbol{T}$ by the Rutishauser-Schwarz method. Its operation is given by

$$\boldsymbol{T} = \boldsymbol{Q}_S^{\mathrm{T}} \boldsymbol{A} \boldsymbol{Q}_S \tag{4.1}$$

where $\boldsymbol{Q}_s$ is an orthogonal matrix and can be produced as a product of an orthogonal matrix shown in the **Fig. BSEG-1**. Its operation is carried out by using the subroutine BTRID.

Secondly, m egenvalues of $\boldsymbol{T}$ are obtained by using the bisection method in the subroutine BSCT1.

Thirdly, the corresponding eigenvectors $\boldsymbol{x}$ of $\boldsymbol{A}$ are obtained by using the inverse iteration method. The inverse iteration is a method to obtain eigenvectors by iteratively solving

$$(\boldsymbol{A} - \mu \boldsymbol{I})\boldsymbol{x}_r = \boldsymbol{x}_{r-1} \quad ,r = 1,2,... \tag{4.2}$$

when an approximate eigenvalue solution μ is given. where μ is obtained by the bisection method and $\boldsymbol{x}_0$ is an appropriate initial vector. This operation is performed by using the subroutine BSVEC. The eigenvectors are to be normalized so that $\|\boldsymbol{x}\|_2 = 1$.

For further details, see "Method" for the subroutines BTRID, BSCT1 and BSVEC, and References [12] and [13].

$$\begin{array}{c|cccccccc|}
 & & & & i & i+1 & & & \\
 & 1 & & & & & & & \\
 & & \ddots & & & & 0 & & \\
 & & & 1 & & & & & \\
i & & & & \cos\theta & \sin\theta & & & \\
i+1 & & & & -\sin\theta & \cos\theta & & & \\
 & & & & & & 1 & & \\
 & & 0 & & & & & \ddots & \\
 & & & & & & & & 1
\end{array}$$

Fig. BSEG-1 General form of orthogonal similarity transformation matrix used in the Rutishauser-Schwarz method

B51-21-1001 BSEGJ, DBSEGJ

Eigenvalues and eigenvectors of a real symmetric band matrix (Jennings method)
CALL BSEGJ (A, N, NH, M, EPST, LM, E, EV, K, IT, VW, ICON)

Function

The m eigenvalues of a real symmetric band matrix of order n and bandwidth h are obtained starting with the eigenvalue of the largest (or smallest) absolute value first, and also the corresponding eigenvectors are obtained for the given m initial vectors by using the Jennings' simultaneous iteration method with the Jennings' acceleration. When starting with the smallest absolute value, matrix $\boldsymbol{A}$ must be positive definite. The eigenvectors should be normalized such that $\|x\|_2 = 1$.

Here $1 \le m << n$ and $0 \le h << n$.

Parameters

A .... Input. Real symmetric band matrix $\boldsymbol{A}$.
When obtaining the eigenvalues of the smallest absolute value first, the contents are altered on output.
Compressed mode for symmetric band matrix.
One-dimensional array of size $n(h+1)-h(h+1)/2$.

N .... Input. Order n of matrix $\boldsymbol{A}$.

NH .... Input. Bandwidth h of matrix $\boldsymbol{A}$.

M .... Input. The number of eigenvalues and eigenvectors obtained, m.
M=m ... the m largest absolute value of eigenvalues are desired.
M=-m ... the smallest absolute value of eigenvalues are desired.

EPST ... Input. Constant ε used for convergence criterion for the eigenvectors.
If this value is zero or negative, a standard value is set.
See "Comments on use".

LM....... Input. Upper limit for the number of iterations.
If the number of iterations exceeds this number, the processing is terminated. See "Comments on use".

E .... Output. Eigenvalues. Stored in the sequence as specified by parameter M.
One-dimensional array of size m.

EV .... Input. The m initial vectors stored in columnwise direction. See "Comments on use".
Output. Eigenvectors. Two-dimensional array of EV(K,m+2) with the elements stored in column wise direction.

K .... Input. Adjustable dimension of EV.

IT .... Output. The number of iterations performed until eigenvalues and eigenvectors are obtained.

VW ...... Work area. One-dimensional array of a size no less than max(n,2m)+m(3m+1)/2.

ICON ... Output. Condition code. See Table BSEGJ-1.

Table BSEGJ-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The number of iterations exceeded upper limit LM.	Terminated. E and EV contain the approximations of the so far obtained eigenvalues and eigenvectors, respectively.
28000	Orthogonalization of eigenvectors at each iteration cannot be attained.	Discontinued
29000	Matrix A is not positive definite, or possibly singular.	Discontinued
30000	NH<0, NH≥N, N>K, M=0 or |M|>N	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, MSBV, SBDL, BDLX, TRID1, TEIG1, TRBK, UCHLS, USERT, MGSSL
  FORTRAN basic functions ... MAX0, AMAX1, ABS, IABS, FLOAT and SQRT

- Notes
  It is desirable for the initial eigenvectors to be a good approximation to the eigenvectors corresponding to the obtained eigenvalues. If approximate vectors are not available, the standard way to choose initial vectors is to use the first m column vectors of the unit matrix $\boldsymbol{I}$. The number of eigenvalues and eigenvectors, m had better be smaller than n such that $m/n<1/10$. The numbering of the eigenvalues is from the largest (or smallest) absolute value of eigenvalue such as λ_1, λ_2, ..., λ_n. It is desirable, if possible, to choose m in such a way that $|\lambda_{m+1}/\lambda_m|$ <<1 (or λ_{m+1}/λ_m >>1) to achieve convergence faster.
  The parameters EPST is used to examine the convergence of elements of the eigenvector normalized so that $\|x\|_2 = 1$. Whenever an eigenvector converges for the convergence criterion constant ε, the corresponding eigenvalue converges at least with accuracy $\|\boldsymbol{A}\| \cdot \varepsilon$ and in most cases is higher. It is therefore better to choose somewhat a larger EPST value. When defining unit round off as u, the standard value is set ε=16u. When the eigenvalues are very close to each other, however, convergence may not be

attained. If so, it is safe to choose such that $\varepsilon \geq 100u$.

The upper limit LM for the number of iterations is used to forcefully terminate the iteration when convergence is not attained. It should be set taking into consideration the required accuracy and how close the eigenvalues are to each other. The standard value is 500 to 1000.

SSL II is provided with eigenvectors of a real symmetric band matrix by using a direct method. If the same problem is to be solved, a direct method is generally faster except that it needs more storage space in computation of the eigenvectors. Choose an appropriate subroutine after considering the size of the problem, required accuracy, amount of storage and the execution time.

- Example

This example obtains eigenvalues and eigenvectors of a real symmetric band matrix ***A*** of order *n* and bandwidth *h*, when $n \leq 100$, $h \leq 10$ and $m \leq 10$.

```
C      **EXAMPLE**
       DIMENSION A(1100),E(10),EV(100,12)
      *          ,VW(300)
   10 READ(5,500) N,NH,M,EPST
       IF(N.EQ.0) STOP
       MM=IABS(M)
       NN=(NH+1)*(N+N-NH)/2
       READ(5,510) (A(I),I=1,NN),
      *            ((EV(I,J),I=1,N),J=1,MM)
       WRITE(6,600) N,NH,M
       NE=0
       DO 20 I=1,N
       NI=NE+1
       NE=MIN0(NH+1,I)+NE
   20 WRITE(6,610) I,(A(J),J=NI,NE)
       CALL BSEGJ(A,N,NH,M,EPST,500,E,EV,
      *           100,IT,VW,ICON)
       WRITE(6,620) ICON,IT
       IF(ICON.GE.20000) GO TO 10
       CALL SEPRT(E,EV,100,N,MM)
       GO TO 10
  500 FORMAT(3I5,E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1',20X,'ORIGINAL MATRIX',
      *       5X,'N=',I3,5X,'NH=',I3,5X,
      *       'M=',I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',20X,'ICON=',I5,5X,'IT=',I5)
       END
```

The subroutine SEPRT is used in this example to print eigenvalues and eigenvectors of a real symmetric matrix. For details, refer to the example of the subroutine SEIG1.

Method

- Jennings simultaneous iteration method

Letting the eigenvalues of a real symmetric band matrix of order *n*, and of the largest absolute value first be as

$$\lambda_1, \lambda_2, ..., \lambda_n$$

and the corresponding orthogonally normalized eigenvectors be as

$$\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n$$

where diagonal matrix $\boldsymbol{\Lambda}$ and orthogonal matrix ***V*** are defined by

$$\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n)$$
$$\boldsymbol{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n)$$

then

$$\boldsymbol{AV} = \boldsymbol{V\Lambda} \tag{4.1}$$

The simultaneous iteration method is the extension of the power method. Starting from an appropriate $n \times m$ matrix $\boldsymbol{X}_1$

$$\boldsymbol{X}_{i+1} = \boldsymbol{AX}_i \quad i = 1,2,... \tag{4.2}$$

is iterated and $\boldsymbol{X}_i$ is to be converged towards the *m* eigenvectors, that is the first *m* rows of ***V***, corresponding to the largest absolute value first. However, if iteration is performed only for (4.2), each row of $\boldsymbol{X}_i$ converges towards the vector corresponding to the first row $\boldsymbol{v}_1$ of ***V***, thus the purpose is not achieved. In the simultaneous iteration method, iteration matrix $\boldsymbol{X}_i$ is always orthonormalized based upon the properties of the eigenvalues and eigenvectors.

Assume that eigenvectors $\boldsymbol{v}_1,...,\boldsymbol{v}_{k-1}$ corresponding to eigenvalues $\lambda_1,...,\lambda_{k-1}$ have been obtained.

The largest value for $\left| \boldsymbol{v}^{\mathrm{T}}\boldsymbol{Av} \right|$ is obtained when $\boldsymbol{v}=\boldsymbol{v}_k$, and then the following condition is satisfied

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}_i = 0 \quad , i = 1,...,k-1$$

Now

$$\lambda_k = \boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{Av}_k$$

is the eigenvalue containing the *k*-th largest absolute value and $\boldsymbol{v}_k$ is the corresponding eigenvector.

Jennings simultaneous iteration method proceeds as follows:
(indexes used for iteration matrices are omitted)

1) ***Y*** is obtained by multiplying ***A*** by ***X*** from the right.

$$\boldsymbol{Y}=\boldsymbol{AX} \tag{4.3}$$

2) *m*-order symmetric matrix ***B*** is obtained by multiplying $\boldsymbol{X}^{\mathrm{T}}$ by ***Y*** form the right.

$$\boldsymbol{B}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} \tag{4.4}$$

3) The eigenvalues μ_i, i=1,2,...,m of ***B*** and the

corresponding eigenvectors $\boldsymbol{p}_i$, i=1,2,...,m are obtained.

$$\boldsymbol{B}=\boldsymbol{PMP}^{\mathrm{T}} \tag{4.5}$$

where $\boldsymbol{M}$ is a diagonal matrix and $\boldsymbol{P}$ is an orthogonal matrix.

$$\boldsymbol{M}=\mathrm{diag}(\mu_1,\mu_2,\ \dots,\mu_m) \quad ,|\mu_1|\geq\dots\geq|\mu_m|$$
$$\boldsymbol{P}=(\boldsymbol{p}_1,\boldsymbol{p}_2,\dots,\boldsymbol{p}_m)$$

4) $\boldsymbol{Z}$ is obtained by multiplying $\boldsymbol{Y}$ by $\boldsymbol{P}$ from the right.

$$\boldsymbol{Z}=\boldsymbol{YP} \tag{4.6}$$

5) $\boldsymbol{Z}$ is multiplied by $\boldsymbol{Z}^{\mathrm{T}}$ from the left to obtain an m-order symmetric positive-definite matrix and this matrix is $\boldsymbol{LL}^{\mathrm{T}}$ decomposed (this is called LL$^{\mathrm{T}}$ decomposition).

$$\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}=\boldsymbol{LL}^{\mathrm{T}} \tag{4.7}$$

6) $\boldsymbol{X}^*$ is obtained by solving the equation $\boldsymbol{X}^*\boldsymbol{L}^{\mathrm{T}}=\boldsymbol{Z}$

$$\boldsymbol{X}^*=\boldsymbol{ZL}^{-\mathrm{T}} \tag{4.8}$$

$\boldsymbol{X}$* is a normal orthogonal matrix.

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}^*=\boldsymbol{I}_m \tag{4.9}$$

where, $\boldsymbol{I}_m$ is an m-order unit matrix.

7) Convergence is tested for $\boldsymbol{X}^*$. Return to procedure 1) setting $\boldsymbol{X}^*$ as a new $\boldsymbol{X}$, administering to the speed up as required basis.

The coefficient matrix when expanding iteration matrix $\boldsymbol{X}$ by eigenvector $\boldsymbol{V}$ is shown below. When expanding each column of matrix $\boldsymbol{X}$ by eigenvectors, the following is assumed:

$$x_j=\sum_{i=1}^{n}c_{ij}\boldsymbol{v}_i, \quad j=1,\dots,m \tag{4.10}$$

Equation (4.10) can be expressed by coefficient matrix $\boldsymbol{C}$=(c_{ij}) of $n\times m$ matrix as

$$\boldsymbol{X}=\boldsymbol{VC} \tag{4.11}$$

Since $\boldsymbol{X}$ is orthogonal,
$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}=\boldsymbol{I}_m$
and $\boldsymbol{V}$ is orthogonal, $\boldsymbol{C}$ can be orthogonal.

$$\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}=\boldsymbol{I}_m \tag{4.12}$$

$\boldsymbol{C}$ is divided into two parts; the first m rows and the remaining n-m rows,

$$\boldsymbol{C}=\begin{pmatrix}\boldsymbol{C}_1\\ \hdashline \boldsymbol{C}_2\end{pmatrix} \tag{4.13}$$

Corresponding to this, Λ is divided into the following blocks.

$$\Lambda=\left(\begin{array}{c:c}\Lambda_1 & \boldsymbol{O}\\ \hdashline \boldsymbol{O} & \Lambda_2\end{array}\right) \tag{4.14}$$

Based on the preparation of procedures 1) through 7) above, $\boldsymbol{C}$ is tested as follows.

1)' Substituting (4.11) in (4.3), equation (4.1) becomes,

$$\boldsymbol{Y}=\boldsymbol{AX}=\boldsymbol{AVC}=\boldsymbol{V}\Lambda\boldsymbol{C} \tag{4.15}$$

The i-th column of the coefficient matrix is multiplied by λ_i. This means that the elements of the eigenvector corresponding to the eigenvalue in which the absolute value is large in (4.15) are focused and compressed.

2)' Substituting (4.11) and (4.15) in (4.4), considering $\boldsymbol{V}$ to be orthogonal,

$$\boldsymbol{B}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}=\boldsymbol{C}^{\mathrm{T}}\Lambda\boldsymbol{C} \tag{4.16}$$

From (4.13) and (4.14), $\boldsymbol{B}$ may be expressed as,

$$\boldsymbol{B}=\boldsymbol{C}_1^{\mathrm{T}}\Lambda_1\boldsymbol{C}_1+\boldsymbol{C}_1^{\mathrm{T}}\Lambda_2\boldsymbol{C}_2 \tag{4.17}$$

3)' In this procedure, no solution can be obtained if no assumptions are made regarding $\boldsymbol{C}$.

Assume that in (4.13), $\boldsymbol{C}_2=\boldsymbol{0}$ is given. This means that $\boldsymbol{X}$ represents all the first m vectors of $\boldsymbol{V}$. Where (4.17) is expressed as,

$$\boldsymbol{B}=\boldsymbol{C}_1^{\mathrm{T}}\Lambda_1\boldsymbol{C}_1 \tag{4.18}$$

From (4.12) $\boldsymbol{C}$ is an m-order orthogonal matrix. Since (4.5) and (4.18) are performed by orthogonal similar transformation for the same m-order matrix $\boldsymbol{B}$, they are expressed by selecting an appropriate $\boldsymbol{V}$ (for example, by changing the sign of eigenvector $\boldsymbol{v}_i$ or selecting the corresponding eigenvector for multiple eigenvalues) as:

$$\boldsymbol{C}_1\boldsymbol{P}=\boldsymbol{I}_m \tag{4.19}$$
$$\boldsymbol{M}=\Lambda_1 \tag{4.20}$$

4)' From (4.6) and (4.15), $\boldsymbol{Z}$ can be expressed as,

$$\boldsymbol{Z}=\boldsymbol{YP}=\boldsymbol{V}\Lambda\boldsymbol{CP} \tag{4.21}$$

When $\boldsymbol{C}_2=\boldsymbol{0}$,

$$\boldsymbol{Z}=\boldsymbol{V}\begin{pmatrix}\Lambda_1\\ \hdashline \boldsymbol{O}\end{pmatrix} \tag{4.22}$$

5)' From (4.12) and orthogonality of $\boldsymbol{V}$,

$\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}=\boldsymbol{P}^{\mathrm{T}}\boldsymbol{C}^{\mathrm{T}}\boldsymbol{\Lambda}^2\boldsymbol{C}\boldsymbol{P}=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$

When $\boldsymbol{C}_2=\boldsymbol{0}$,

$\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}=\boldsymbol{\Lambda}_1^2=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$

from

$$\boldsymbol{L}=|\boldsymbol{\Lambda}_1|=\mathrm{diag}(|\lambda_1|,\ \ldots\ ,|\lambda_m|) \qquad (4.23)$$

6)' From (4.8) and (4.21),

$$\boldsymbol{X}^*=\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{C}\boldsymbol{P}\boldsymbol{L}^{-\mathrm{T}} \qquad (4.24)$$

When $\boldsymbol{C}_2=\boldsymbol{0}$, then

$$\boldsymbol{X}^*=\boldsymbol{V} \qquad (4.25)$$

Therefore, when $\boldsymbol{C}_2=\boldsymbol{0}$, an eigenvalue and eigenvector can be obtained by one iteration.
When

$$\boldsymbol{C}=\begin{pmatrix}\boldsymbol{I}_m+\boldsymbol{E}_1\\ \cdots\cdots\cdots\\ \boldsymbol{E}_2\end{pmatrix} \qquad (4.26)$$

and the elements of $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ are the first order least value, (4.19) and (4.20) hold if the second-order least value is omitted. And therefore,

$$\boldsymbol{X}^*=\boldsymbol{V}\cdot\begin{pmatrix}\boldsymbol{I}_m\\ \cdots\cdots\cdots\\ \Lambda_2\boldsymbol{C}_2\Lambda_1^{-1}\end{pmatrix} \qquad (4.27)$$

This indicates that the element of eigenvector $\boldsymbol{v}_j$ ($j>m$) contained in $\boldsymbol{x}_i$ ($i\le m$) of $\boldsymbol{X}$ is reduced in each iteration by the ratio
$\boldsymbol{\lambda}_j/\boldsymbol{\lambda}_i$
This means that $\boldsymbol{C}_2$ of $\boldsymbol{C}$ is approaching the least value.

In procedure 1), the elements of an eigenvector corresponding to the eigenvalue of the largest absolute value is compressed. In procedures 2), 3), and 4), the eigenvectors are refined along with obtaining eigenvalues. In procedures 5) and 6), the eigenvectors are made orthonormal.

- Computation procedures
  This subroutine obtains the *m* eigenvalues of the largest (or the smallest) absolute values of a real symmetric band matrix of order *n* and bandwidth *h* and also the corresponding eigenvectors based upon *m* given initial vectors, using the Jennings simultaneous iteration method.

(a) When obtaining the eigenvalues starting with the smallest absolute value and the corresponding eigenvector, the following is used instead of equation (4.3) in procedure 1).

$$\boldsymbol{Y}=\boldsymbol{A}^{-1}\boldsymbol{X} \qquad (4.28)$$

$\boldsymbol{A}$ is decomposed into $\mathrm{LDL}^{\mathrm{T}}$ using subroutine SBDL. If $\boldsymbol{A}$ is not a positive definite matrix or it is singular, the processing is terminated with ICON=29000.

(b) When obtaining $\boldsymbol{B}$ in 2), it is processed along with 1) to reduce storage space as follows, where row vector $\boldsymbol{y}_i$ of $\boldsymbol{Y}$ is computed in (4.29).

$$\boldsymbol{v}=\boldsymbol{x}_i$$

$$\boldsymbol{y}_i=\boldsymbol{A}\boldsymbol{v}\ \text{ or }\ \boldsymbol{y}_i=\boldsymbol{A}^{-1}\boldsymbol{v} \qquad (4.29)$$

$$i=1,\ \ldots,\ m$$

Equation (4.29) is computed by subroutine MSBV or BDLX. Since $\boldsymbol{B}$ is an *m*-order real symmetric matrix, its diagonal and lower triangular elements can be computed as

$$b_{ii}=\boldsymbol{v}^{\mathrm{T}}\boldsymbol{y}_i$$

$$b_{ji}=\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{y}_i \qquad ,\ j=i+1,\ \ldots,\ m$$

Thus, $\boldsymbol{Y}$ is produced directly in the area for $\boldsymbol{X}$ by taking $\boldsymbol{x}_i$ for each column of $\boldsymbol{X}$ and obtaining $\boldsymbol{y}_i$ and b_{ji}.

(c) 3) is performed sequentially by subroutine TRID1, TEIG1 and TRBK. Arrange the eigenvectors corresponding to the eigenvalues in the order of the largest absolute value to the smallest using subroutine UESRT.

(d) 5) is performed by subroutine UCHLS. If LL^{T} decomposition is impossible, processing is terminated with ICON=28000.

(e) In 7), the *m*-th columns $\boldsymbol{x}_m$ and $\boldsymbol{x}_m^*$ of $\boldsymbol{X}$ and $\boldsymbol{X}^*$ are examined to see if

$$d=\left\|\boldsymbol{x}_m^*-\boldsymbol{x}_m\right\|_\infty\le\mathrm{EPST} \qquad (4.30)$$

is satisfied.
If satisfied, the diagonal elements of $\boldsymbol{M}$ obtained in 3) become eigenvalues (when obtaining the smallest absolute values first, the inverse numbers of the diagonal elements become the eigenvalues) and each column of $\boldsymbol{X}^*$ becomes the corresponding eigenvector.

- Jennings' acceleration
  This subroutine uses the original Jennings' method explained above, incorporating the Jennings' acceleration for vector series.

  Principle of acceleration
  $\boldsymbol{A}$ vector series $\boldsymbol{x}^{(k)}$ where k=1,2,... that converges to vector $\boldsymbol{x}$ is expressed as a linear combination of vectors $\boldsymbol{v}_j$ where j=1,2,...,n that are orthogonal to each other and constant $\boldsymbol{\rho}_j$ such as $|\boldsymbol{\rho}_j|<1$ where j=1,2,...,n as follows:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x} + \sum_{j=1}^{n} p_j^k \boldsymbol{v}_j \qquad (4.31)$$

A new vector is generated from three subsequent vectors $\boldsymbol{x}^{(k)}$, $\boldsymbol{x}^{(k+1)}$, and $\boldsymbol{x}^{(k+2)}$ as follows:

$$\overline{\boldsymbol{x}} = \boldsymbol{x}^{(k+2)} + s\left(\boldsymbol{x}^{(k+2)} - \boldsymbol{x}^{(k+1)}\right) \qquad (4.32)$$

where

$$s = \frac{\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k+1)}\right)^T \left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k+2)}\right)}{\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k+1)}\right)^T \left(\boldsymbol{x}^{(k)} - 2\boldsymbol{x}^{(k+1)} + \boldsymbol{x}^{(k+2)}\right)} \qquad (4.33)$$

Substituting (4.31) to (4.33) and using the orthogonal relation of $\boldsymbol{v}_j$, s is expressed as:

$$s = \frac{\sum_{j=1}^{n} p_j z_j}{\sum_{j=1}^{n} (1 - p_j) z_j} \qquad (4.34)$$

where

$$z_j = p_j^{2k} (1 - p_j)^2 \|\boldsymbol{v}_j\|_2^2 \quad , j = 1,2,...,n \qquad (4.35)$$

Substituting (4.31) and (4.34) to (4.32), and the following for s,

$$\overline{p} = \frac{s}{1+s} = \frac{\sum_{j=1}^{n} p_j z_j}{\sum_{j=1}^{n} z_j} \qquad (4.36)$$

then $\overline{\boldsymbol{x}}$ is obtained as:

$$\overline{\boldsymbol{x}} = \boldsymbol{x} + \sum_{j=1}^{n} \frac{p_j - \overline{p}}{1 - \overline{p}} \cdot p_j^{k+1} \boldsymbol{v}_j \qquad (4.37)$$

It is known from (4.36) that $\overline{p}$ is the average of p_j with respect to the positive weight z_j. Assuming $|p_1| > |p_2| > \cdots$, then following is satisfied for a sufficiently large value k:

$$|z_1| >> |z_j| \quad , j > 1$$

Thus, $p \approx p_1$ is obtained, and this proves

$$\overline{\boldsymbol{x}} \approx \boldsymbol{x}$$

This is a brief explanation of the Jennings' acceleration principle.

Assuming, $x_i^{(k)}$ to be vector of column i of the iteration matrix $\boldsymbol{x}_k$ in Jennings' method, the vector is expressed as follows for a sufficiently large value k:

$$\boldsymbol{x}_i^{(k)} = \boldsymbol{v}_i + \sum_{j=m+1}^{n} \left(\lambda_j / \lambda_i\right)^k c_{ji} \boldsymbol{v}_j \qquad (4.38)$$

where $| \lambda_j / \lambda_i | < 1$
This means that Jennings' acceleration is applicable.

Computation procedure

1) Assume the initial value of the constant as the criterion for adopting the acceleration to be:

$$\delta = \frac{1}{2\sqrt{n}} \qquad (4.39)$$

2) Assume another criterion constant for adopting the acceleration to be:

$$\eta = \max\left(\delta^{1.5}, \varepsilon\right) \qquad (4.40)$$

Then, vector number j_a of the vector to be processed is set in m and the following acceleration cycle starts:

3) Stores every other vector obtained from the latest values of $\boldsymbol{x}_{j_a}$ in columns (m+1) and (m+2) of the EV.
4) Performs simultaneous iteration of Jennings' method.
5) Judges convergence of (4.30) only for j_a=m
6) If (4.41) is satisfied, proceeds to step 10) :

$$\left\|\boldsymbol{x}_{j_a}^* - \boldsymbol{x}_{j_a}\right\| \le \eta \qquad (4.41)$$

7) If (4.42) is satisfied, proceeds to step 3) :

$$\left\|\boldsymbol{x}_{j_a}^* - \boldsymbol{x}_{j_a}\right\| \le \delta \qquad (4.42)$$

8) Calculates $\overline{\boldsymbol{x}}_{j_a}$ by performing Jennings' acceleration with every other iterated vector of $\overline{\boldsymbol{x}}_{j_a}$
9) Orthogonalizes the $\overline{\boldsymbol{x}}_{j_a}$ to satisfy the following:

$$\begin{aligned} &\boldsymbol{x}_{j_a}^{\mathrm{T}} \cdot \boldsymbol{x}_i = 0 \quad , j \ne j_a \\ &\left\|\boldsymbol{x}_{j_a}\right\|_2 = 1 \end{aligned} \qquad (4.43)$$

10) Decrements j_a by one, then proceeds to step 3) if $j_a \ge 1$
11) If the convergence condition is satisfied in step 5), iteration is stopped; otherwise, changes δ to:

$$\delta = \max(\delta/1000, \eta, 10\varepsilon) \qquad (4.44)$$

then proceeds to step 2).

• Notes on the algorithm

In Jennings' acceleration, every other vectors are used for the following reasons:

It is clear from (4.37) that Jennings' acceleration is effective when weighted average $\overline{p}$ (the weight is positive) of p_j is close to p_1. If all values of p_j have the same sign, this condition is to be satisfied ordinarily; however, if there is a value having a sign opposite that of p_1 and the absolute value close to that of p_j, considerable effect of Jennings' acceleration cannot be expected. If every other vectors are used in place of successive vectors, that is, p_j is substituted for p_j^2, the difficulty explained above caused by different signs of p_j is eliminated.

Jennings' acceleration is effective for the eigenvalue problem having almost same eigenvalues in magnitude.

(See references [18] and [19] for simultaneous iteration of Jennings' method, and reference [18] for acceleration.)

E-31-32-0101 BSFD1, DBSFD1

B-spline two-dimensional smoothing
CALL BSFD1 (M, XT, NXT, YT, NYT, C, KC, ISWX, VX, IX, ISWY, VY, IY, F, VW, ICON)

Function

Given observed value $f_{i,j}=f(x_i,y_j)$, observation error $\sigma_{i,j}=\sigma x_i\cdot\sigma y_j$ at lattice points (x_i,y_j); i=1,2,...,n_x, j=1,2,...,n_y, this subroutine obtains a smoothed value or a partial derivative at the point P(v_x,v_y), or a double integral over the range [$\xi_1\leq x\leq v_x$, $\eta_1\leq y\leq v_y$] based on the bivariate m-th degree B-spline smoothing function.

However, subroutine BSCD2 must be called before using subroutine BSFD1 to determine the knots ξ_1, ξ_2,..., ξ_{nt} in the x-direction, the knots η_1, η_2,..., η_{lt} in the y-direction, and the smoothing coefficients $C_{\alpha,\beta}$ in the m-th degree B-spline smoothing function.

$$\bar{S}(x,y)=\sum_{\beta=-m+1}^{l_t-1}\sum_{\alpha=-m+1}^{n_t-1}c_{\alpha,\beta}N_{\alpha,m-1}(x)N_{\beta,m+1}(y) \quad (1.1)$$

$m\geq1$, $\xi_1\leq v_x<\xi_{nt}$, $\eta_1\leq v_y\leq\eta_{lt}$

Parameters

M .... Input. Degree of the B-spline.
XT .... Input. Knots ξ_i in the x-direction.
One-dimensional array of size n_t.
NXT........ Input. Number n_t of knots ξ_i in the x-direction.
YT .... Input. Knots η_i in the y-direction.
One-dimensional array of size l_t.
NYT........ Input. Number l_t of knots η_i in the y-direction.
C .... Input. Smoothing coefficient $C_{\alpha,\beta}$.
Two-dimensional array of C(KC,l_t+m-1)
KC .... Input. Adjustable dimension ($\geq n_t+m-1$) for array C.
ISWX...... Input. Type of computations in the x-direction.
-1≤ISWX≤M (See parameter F.)
VX .... Input. Coordinate x at points P(v_x,v_y)
IX............ Input. i satisfying $\xi_i\leq v_x<\xi_{i+1}$.
When $v_x=\xi_{nt}$, IX=n_t-1.
Output. i satisfying $\xi_j\leq v_x<\xi_{i+1}$.
(See Notes.)
ISWY...... Input. Type of computations in the y-direction.
-1≤ISWY≤M (See parameter F.)
VY .... Input. Coordinate y at points P(v_x,v_y).
IY............ Input. j satisfying $\eta_j\leq v_y<\eta_{j+1}$.
When $v_y=\eta_{lt}$, IY=l_t-1.
Output. j satisfying $\eta_j\leq v_y<\eta_{j+1}$
(See Notes.)
F............. Output. Smoothed value, partial devivative or integral. Suppose ISWX=λ and ISWY=μ, one of the following values is output depending upon the combination of λ and μ.

- When $0\leq\lambda,\mu$

$$F=\frac{\partial^{\lambda+\mu}}{\partial x^{\lambda}\partial y^{\mu}}\bar{S}(v_x,v_y)$$

A smoothed value can be obtained by setting $\lambda=\mu=0$.

- When λ=-1, and $0\leq\mu$,

$$F=\int_{\xi_1}^{v_x}\left\{\frac{\partial^{\mu}}{\partial y^{\mu}}\bar{S}(x,v_y)\right\}dx$$

- When $\lambda\geq0$ and μ=-1,

$$F=\int_{\eta_1}^{v_y}\left\{\frac{\partial^{\lambda}}{\partial x^{\lambda}}\bar{S}(v_x,y)\right\}dy$$

- When $\lambda=\mu$=-1,

$$F=\int_{\eta_1}^{v_y}dy\int_{\xi_1}^{v_x}\bar{S}(x,y)dx$$

VW .... Work area.
One-dimensional array of size 5(m+1)+max(n_t,l_t)
ICON ...... Output. Condition code.
See Table BSFD1-1.

Table BSFD1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	XT(IX)≤VX<XT(IX+1) or YT(IY)≤VY<YT(IY+1) is not satisfied.	The IX or IY shown on the left is searched for in the subroutine and processing continues.
30000	One of the following occurred: 1)VX<XT(1) or VX>XT(NXT) 2)VY<YT(1) or VY>YT(NYT) 3)ISWX<-1 or ISWX>M 4)ISWY<-1 or ISWY>M	Bypassed

Comments on use

- Subprogram used
SSL II ... MGSSL, UCAR2, UBAS1
FORTRAN basic function ... FLOAT

- Notes
This subroutine obtains the smoothed value, partial derivative or double integral based upon B-spline two dimensional smoothing function (1.1)

obtained by subroutine BSCD2.

Therefore, subroutine BSCD2 must be called to obtain the smoothing function (1.1) before calling this subroutine. The values of parameters M, XT, NXT, YT, NYT, C, and KC must be directly passed from BSCD2.

Parameters IX and IY should satisfy XT(IX)≤VX<XT(IX+1) and YT(IY)≤VY<YT(IY+1) respectively. If not, IX and IY which satisfy the relationship is found and the processing is continued.

- Example

Refer to Example of subroutine BSCD2.

Method

Suppose that by subroutine BSCD2, the bivariate *m*-th degree B-spline smoothing function

$$\bar{S}(x,y)=\sum_{\beta=-m+1}^{l_t-1}\sum_{\alpha=-m+1}^{n_t-1}c_{\alpha,\beta}N_{\alpha,m+1}(x)N_{\beta,m+1}(y) \qquad (4.1)$$

is obtained.

This subroutine computes smoothed values, partial derivatives or integrals based upon the smoothing function of (4.1).

For detailed information, see Section 7.1 in Part I.

E31-31-0101 BSF1, DBSF1

B-spline smoothing, differentiation and integration (with fixed knots)
CALL BSF1(M,XT,NT,C,ISW,V,I,F,VW,ICON)

Function

Given observed values $y_1, y_2, ..., y_n$ at points $x_1, x_2, ..., x_n$, weighted function values $w_i=w(x_i)$, $i=1,2,...,n$ and knots of the spline function, $\xi_1, \xi_2, ..., \xi_{nt}$ ($\xi_1<\xi_2<...<\xi_{nt}$) then a smoothed value, or derivative at $x=v\in[\xi_1,\xi_{nt}]$ or integral from ξ_1 to v is obtained based on the B-spline smoothing function.

One condition is that the smoothing coefficients c_j's, $j=-m+1, -m+2, ..., n_t-1$ in the B-spline smoothing function

$$\bar{S}(x)=\sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(x) \qquad (1.1)$$

must be calculated in subroutine BSC1 or BSC2 before using subroutine BSF1, where m is the degree of the B-spline $N_{j,m+1}(x)$.

Also $\xi_1 \le v \le \xi_m$, $m\ge1$, $n_t\ge3$, must be satisfied.

Parameters

M	Input. Degree of the B-spline m. See Notes.
XT	Input. Knots ξ_j's. One-dimensional array of size n_t.
NT ...	Input. Number the of knots ξ_i's, n_t.
C	Input. Smoothing coefficient c_j's (output from BSC1 or BSC2) One-dimensional array of size n_t+m-1.
ISW ...	Input. An integer which specifies the type of calculation. If ISW=0, the smoothing value, $F=\bar{S}(v)$ If ISW= $l(1\le l\le m)$,the l-th order derivative, $F=\bar{S}^{(l)}(v)$. If ISW=-1, the integral value, $F=\int_{\xi_1}^{v}\bar{S}(x)dx$ are obtained, respectively.
V	Input. The point v at which the smoothing value, etc.are obtained.
I …..	Input. The i which satisfies $\xi_i\le v<\xi_{i+1}$. If $v=\xi_{nt}$,$I=n_t$-1. Output. The i which satisfies $\xi_i \le v < \xi_{i+1}$. See Notes.
F	Output. Smoothed value, the l-th order derivative or integral value depending on ISW. See parameter ISW.
VW ...	Work area. One-dimensional array of size m+1.
ICON ..	Output. Condition code. See Table BSF1-1.

Table BSF1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	XT(I) ≤V<XT(I+1) is not satisfied.	I satisfying the left relationship is searched for and the processing is continued.
30000	One of the following occurred: (a)V<XT(1) or V>XT(NT) (b)ISW<-1 or ISW>M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, UCAR1and UBAS1
  FORTRAN basic function ... FLOAT

- Notes
  This subroutine obtains a smoothed value, derivative or integral based on the B-spline smoothing function (1.1) obtained by subroutine BSC1 or BSC2.
  Therefore, subroutine BSC1 or BSC2 must be called to obtain the smoothing function (1.1) before calling subroutine BSF1. The parameter values of M, XT, NT, and C must be the same as those used in BSC1 or BSC2.
  Parameter I should satisfy the relationship XT(I)≤V<XT(I+1). If not, the I which satisfies the relationship is searched for to continue the processing.

- Example
  By inputting discrete points x_i's, observed values y_i's, $i=1,2,...,n$, knots ξ_j's $j=1,2,...,n_t$ and the degree m, smoothed values, the 1st to m-th order derivatives at and integrals from ξ_1 to the point.

$$v_{ij}=\xi_i+(\xi_{i+1}-\xi_i)\times(j/5), \quad i=1,2,...,n_t-1,$$
$$j=0,1,...,5$$

are obtained.

Here, the weighted function values w_i, $i=1,2,...,n$ to each of the observed values are all set to 1.0, and also $\min(\xi_j)\le x_i\le\max(\xi_j)$, $i=1,2,...,n$, $n\le101$, $n_t\le10$ and $m\le5$.

```
C     **EXAMPLE**
      DIMENSION X(101),Y(101),W(101),
     *XT(10),C(14),R(101),VW(66),IVW(101),
     *RR(6)
      READ(5,500) N,M
      READ(5,510) (X(I),Y(I),I=1,N)
      READ(5,500) NT
      READ(5,520) (XT(I),I=1,NT)
      WRITE(6,600) N,M,(I,X(I),Y(I),I=1,N)
      WRITE(6,610) NT,(I,XT(I),I=1,NT)
      DO 10 I=1,N
   10 W(I)=1.0
      CALL BSC1 (X,Y,W,N,M,XT,NT,C,R,
     *RNOR,VW,IVW,ICON)
      IF(ICON.EQ.0) GO TO 20
      WRITE(6,620)
      STOP
   20 WRITE(6,630) RNOR
      N1=NT-1
      M2=M+2
      DO 50 L2=1,M2
      ISW=L2-2
      WRITE(6,640) ISW
      DO 40 I=1,N1
      H=(XT(I-1)-XT(I))/5.0
      XI=X(I)
      DO 30 J=1,6
      V=XI+H*FLOAT(J-1)
      II=I
      CALL BSF1(M,XT,NT,C,ISW,V,
     *II,F,VW,ICON)
      RR(J)=F
   30 CONTINUE
      WRITE(6,650) II,(RR(J),J=1,6)
   40 CONTINUE
   50 CONTINUE
      STOP
  500 FORMAT(2I6)
  510 FORMAT(2F12.0)
  520 FORMAT(F12.0)
  600 FORMAT('1'//10X,'INPUT DATA',3X,
     *'N=',I3,3X,'M=',I2//20X,'NO.',10X,
     *'X',17X,'Y'//(20X,I3,2E18.7))
  610 FORMAT(/10X,'INPUT KNOTS',3X,
     *'NT=',I3/20X,'NO.',10X,'XT'//
     *(20X,I3,E18.7))
  620 FORMAT('0',10X,'ERROR')
  630 FORMAT(10X,'SQUARE SUM OF',1X,
     *'RESIDUALS=',E18.7)
  640 FORMAT('1'//10X,'L=',I2/)
  650 FORMAT(6X,I3,6E18.7)
      END
```

Method

Suppose that the m-th degree B-spline smoothing function

$$\bar{S}(x)=\sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(x) \tag{4.1}$$

has been obtained by subroutine BSC1 or BSC2.

Subroutines BSF1, based on the smoothing function (4.1), obtains a smoothed value, l-th order derivative or integral by using the (4.2), (4.3) and (4.4), respectively.

$$\bar{S}(v)=\sum_{j=-m+1}^{n_t-1} c_j N_{j,m+1}(v) \tag{4.2}$$

$$\bar{S}^{(l)}(v)=\sum_{j=-m+1}^{n_t-1} c_j N^{(l)}_{j,m+1}(v) \tag{4.3}$$

$$I=\int_{\xi_1}^{v}\bar{S}(x)dx \tag{4.4}$$

The calculation method for the above is described in Section "Calculating spline function."

This subroutine calculates $N_{j,m+1}(x)$, its derivative and integral by calling subroutine UBAS1.

B51-21-0402 BSVEC, DBSVEC

Eigenvectors of a real symmetric band matrix (Inverse iteration method)
CALL BSVEC(A,N,NH,NV,E,EV,K,VW,ICON)

Function

When n_v number of eigenvalues are given for a real symmetric band matrix $\boldsymbol{A}$ of order n and bandwidth h the corresponding eigenvectors are obtained by using the inverse iteration method.

Parameters

A	Input. Real symmetric band matrix $\boldsymbol{A}$. Compressed mode for symmetric band matrix. One-dimensional array of size $n(h+1)-h(h+1)/2$.
N	Input. Order n of matrix $\boldsymbol{A}$.
NH	Input. Bandwidth h
NV	Input. Number of eigenvectors to be obtained. For NV=$-n_v$ set as NV=$+n_v$.
E	Input. Eigenvalues. One-dimensional array of size at least n_v.
EV	Output. Eigenvectors. The eigenvectors are stored in column wise direction. Two-dimensional array of EV(K,NV)
K	Input. Adjustable dimension of the array EV.
VW	Work area. One-dimensional array of size $2n(h+1)$.
ICON ..	Output. Condition code. See Table BSVEC-1.

Table BSVEC-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	NH=0	Executed normally.
15000	All of the eigenvectors could not be obtained.	The eigenvector is set to zero-vector.
20000	None of the eigenvectors could be obtained.	All of the eigenvectors become zero-vectors.
30000	NH<0, NH≥N, N>K,NV=0 or $\lvert NV \rvert > N$	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL
  FORTRAN basic function ... MAX0, MIN0, ABS, SIGN and SQRT
- Notes
  This subroutine is for a real symmetric band matrix. For obtaining eigenvalues and eigenvectors of a real symmetric matrix, use subroutine SEIG1 or SEIG2. Further, for a real symmetric tridiagonal matrix, use subroutine TEIG1 or TEIG2. If the eigenvalues are close to each other in a small range, the inverse iteration method used to obtain the corresponding eigenvectors may not converge. If this happens, ICON is set to 15000 or 20000 and unobtained eigenvectors become zero-vectors.
- Example
  When a real symmetric band matrix $\boldsymbol{A}$ of order n and bandwidth h and also its n_v number of eigenvalues are given, the corresponding eigenvectors are obtained in this example by using the inverse iteration method for the case n≤100, h≤10 and n_v≤10.

```
C     **EXAMPLE**
      DIMENSION A(1100),E(10),EV(100,10),
     *          VW(2100)
   10 READ(5,500) N,NH,NV
      IF(N.EQ.0) STOP
      NN=(NH+1)*(N+N-NH)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N,NH,NV
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=MIN0(NH+1,I)+NE
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      NNV=IABS(NV)
      READ(5,510) (E(I),I=1,NNV)
      CALL BSVEC(A,N,NH,NV,E,EV,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL SEPRT(E,EV,100,N,NNV)
      GO TO 10
  500 FORMAT(3I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'/
     *       11X,'** ORDER =',I5,'NH=',I3,
     *       10X,'NV=',I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT(/11X,'** CONDITION CODE =',I5/)
      END
```

For subroutine SEPRT, see the example of the subroutine SEIG1.

Method

When a real symmetric band matrix $\boldsymbol{A}$ of order n and bandwidth h and also its n_v number of eigenvalues are given, the corresponding eigenvector are obtained by using the inverse iteration method.
Suppose the true eigenvalues have the relation.

$$\lambda_1>\lambda_2>...>\lambda_n \tag{4.1}$$

Given μ_j as an approximation of λ_j the inverse iteration method iteratively solves

$$(\boldsymbol{A}-\mu_j\boldsymbol{I})\boldsymbol{x}_r=\boldsymbol{x}_{r-1} \quad ,r=1,2,... \tag{4.2}$$

with an appropriate initial vector $\boldsymbol{x}_0$, and chooses $\boldsymbol{x}_r$ as eigenvector when it has satisfied convergence criteria.

First, $\boldsymbol{A}$-$\mu_j\boldsymbol{I}$ is decomposed to a unit lower triangular matrix $\boldsymbol{L}$ and upper triangular matrix $\boldsymbol{U}$

$$\boldsymbol{P}(\boldsymbol{A}-\mu_j\boldsymbol{I})=\boldsymbol{LU} \tag{4.3}$$

and the resulting equation

$$\boldsymbol{LUx}_r=\boldsymbol{Px}_{r-1} \tag{4.4}$$

is solved, where $\boldsymbol{P}$ is a permutation matrix which performs row interchange for a partial pivoting. To solve Eq. (4.4), the following two equations

$$\boldsymbol{Ly}_{r-1}=\boldsymbol{Px}_{r-1} \quad \text{(forward substitution)} \tag{4.5}$$

$$\boldsymbol{Ux}_r=\boldsymbol{y}_{r-1} \quad \text{(backward substitution)} \tag{4.6}$$

are to be solved.

The method described above is the general inverse iteration, but if it is applied to a real symmetric band matrix, the handwidth becomes large because of the row interchange in the Eq. (4.3), and the storage space for an LU-decomposed matrix increases substantially compared to the space for the original matrix $\boldsymbol{A}$. This subroutine, in order to minimize this increase, discards the $\boldsymbol{L}$ component leaving only the $\boldsymbol{U}$ component.

$$\boldsymbol{Ux}_r=\boldsymbol{x}_{r-1} \tag{4.7}$$

Since an appropriate initial eigenvector $\boldsymbol{x}_0$ can be represented as

$$\boldsymbol{x}_0=\sum_{i=1}^{n}\alpha_i^{(0)}\boldsymbol{u}_i \tag{4.8}$$

by the true eigenvectors $\boldsymbol{u}_1,\boldsymbol{u}_2,...,\boldsymbol{u}_n$ which correspond to the true eigenvalues, from Eq. (4.7), let

$$\boldsymbol{x}_1=\boldsymbol{U}^{-1}\boldsymbol{x}_0 \tag{4.9}$$

and substituting Eqs. (4.3) and (4.8) into (4.9), we obtain

$$\boldsymbol{x}_1=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-1}\boldsymbol{P}^T\boldsymbol{L}\sum_{i=1}^{n}\alpha_i^{(0)}\boldsymbol{u}_i$$

$$=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-1}\sum_{i=1}^{n}\alpha_i^{(0)}\boldsymbol{P}^T\boldsymbol{Lu}_i$$

If $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{Lu}_i=\sum_{k=1}^{n}\beta_{ik}\boldsymbol{u}_k$, then

$$\boldsymbol{x}_1=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-1}\sum_{i=1}^{n}\alpha_i^{(0)}\left(\sum_{k=1}^{n}\beta_{ik}\boldsymbol{u}_k\right)$$

$$=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-1}\sum_{k=1}^{n}\left(\sum_{i=1}^{n}\alpha_i^{(0)}\beta_{ik}\right)\boldsymbol{u}_k$$

Rewriting $\sum_{i=1}^{n}\alpha_i^{(0)}\beta_{ik}$ to $\alpha_k^{(1)}$,

$$\boldsymbol{x}_1=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-1}\sum_{i=1}^{n}\alpha_i^{(1)}\boldsymbol{u}_i$$

Similarly, rewriting $\sum_{i=1}^{n}\alpha_i^{(r-1)}\beta_{ik}$ to $\alpha_k^{(r)}$, and from

$$\boldsymbol{x}_r=(\boldsymbol{A}-\mu_j\boldsymbol{I})^{-r}\sum_{i=1}^{n}\alpha_i^{(r)}\boldsymbol{u}_i, \quad r=1,2,... \tag{4.10}$$

$\boldsymbol{x}_r$ is given by

$$\boldsymbol{x}_r=\frac{1}{(\lambda_j-\mu_j)^r}\times \left\{\alpha_j^{(r)}\boldsymbol{u}_j+\sum_{\substack{i=1\\i\neq j}}^{n}\alpha_i^{(r)}\boldsymbol{u}_i(\lambda_j-\mu_j)^r/(\lambda_i-\mu_j)^r\right\} \tag{4.11}$$

The constant $1/(\lambda_j$-$\mu_j)^r$ can be eliminated by normalizing $\boldsymbol{x}_r$ at each iteration step. Therefore

$$\boldsymbol{x}_r=\alpha_j^{(r)}\boldsymbol{u}_j+\sum_{\substack{i=1\\i\neq j}}^{n}\alpha_i^{(r)}\boldsymbol{u}_i(\lambda_j-\mu_j)^r/(\lambda_i-\mu_j)^r \tag{4.12}$$

In general, $|(\lambda_j$-$\mu_j)/(\lambda_i$-$\mu_j)|<<1$, so Eq. (4.12) indicates that if $\alpha_j^{(r)}\neq 0$, the larger r becomes, the closer $\boldsymbol{x}_r$ approaches eigenvector $\alpha_j^{(r)}\boldsymbol{u}_j$.

- Initial vector and convergence criteria

  This subroutine normalize $\boldsymbol{x}_{r\text{-}1}$ at each step such that

$$\|\boldsymbol{x}_{r-1}\|_1=n^{3/2}u\|\boldsymbol{A}\| \tag{4.13}$$

and if $\boldsymbol{x}_r$ satisfies

$$\|\boldsymbol{x}_r\|_1\geq 1 \tag{4.14}$$

then $\boldsymbol{x}_r$ is accepted as an eigenvector, where u is the unit round off, and $\|\boldsymbol{A}\|$ is represented by

$$\|\boldsymbol{A}\|=\frac{2}{n}\sum_{i\geq j}|a_{ij}| \tag{4.15}$$

which is close to $\|\boldsymbol{A}\|_1$ and is easy to calculate. The reason that Eq. (4.15) can be used is as follows: from Eqs. (4.3) and (4.7).

$$(\boldsymbol{A}-\mu_j\boldsymbol{I})\boldsymbol{x}_r/\|\boldsymbol{x}_r\|_1=\boldsymbol{P}^{\mathrm{T}}\boldsymbol{Lx}_{r-1}/\|\boldsymbol{x}_r\|_1 \tag{4.16}$$

The right hand side of Eq. (4.16) is the residual vector when considering $\boldsymbol{x}_r / \|\boldsymbol{x}_r\|_1$ as an eigenvector.

Since $\boldsymbol{L}$ is a lower triangular matrix whose diagonal elements are 1 and absolute values of other off-diagonal elements are less than 1, and $\boldsymbol{P}$ is a permutation matrix, the norm of the vector is assumed not to be increased by the linear transformation $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{L}$. Therefore, if $\|\boldsymbol{x}_r\|_1 \geq 1$, the norm in the right hand side of Eq.(4.16) is very small, so there is no objection to accept $\boldsymbol{x}_n$ as an eigenvector. For the initial vector $\boldsymbol{x}_0$, a vector which consists of the continuous n elements

$$R_i = i\phi - [i\phi] \; i = 1,2,... \tag{4.17}$$

is used, where R_i's are not pseudo random numbers in a statistical sense but are increasingly uniformly distributed random numbers in the interval [0,1], and

$$\phi = \left(\sqrt{5} - 1\right) / 2 \tag{4.18}$$

By doing this, there is no need to alter the way of choosing an initial vector for multiple eigenvalues, and/or to give any perturbation to the eigenvalues, so that the computation becomes simpler. This subroutine initializes the first random number to be $R_1=\phi$ every time it is called in order to guarantee consistency with its computed results.

If five iterations of Eq, $\boldsymbol{x}_r$ is still not enough to satisfy the convergence criterion (4.14), this subroutine tries another five more iterations after relaxing the criterion by setting the coefficient of $\|\boldsymbol{A}\|$ in Eq. (4.13) to $10n^{3/2}u$.

If convergence is still not accomplished, the iteration is assumed to be non-convergent and the corresponding column of EV are all set to zero and ICON is given 15000.

- Orthogonalizing the eigenvectors
  All eigenvectors of a real symmetric matrix should be orthogonal to each other, but once the eigenvalues become close to each other, the orthogonal nature of the eigenvectors tend to collapse. Therefore, this subroutine, to insure the eigenvectors orthogonal, performs the following:
  first it examines to see if the eigenvalue μ_i to which the corresponding eigenvector is obtained and the eigenvalue at one step before, μ_{i-1}, satisfies the relationship

$$|\mu_i - \mu_{i-1}| \leq 10^{-3} \|\boldsymbol{A}\| \tag{4.19}$$

If the relationship is satisfied, the eigenvector $\boldsymbol{x}_i$ corresponding to the eigenvalue μ_i is modified to the eigenvector $\boldsymbol{x}_{i-1}$ corresponding to μ_{i-1} in such a way that

$$(\boldsymbol{x}_i, \boldsymbol{x}_{i-1}) = 0 \tag{4.20}$$

In a similar way, for a group of eigenvalues which successively satisfy the relationship (4.19), their corresponding eigenvectors are reorthogonalized.

B51-21-0302 BTRID, DBTRID

Reduction of a real symmetric band matrix to a real symmetric tridiagonal matrix (Rutishauser-Schwarz method)
CALL BTRID(A,N,NH,D,SD,ICON)

Function

A real symmetric band matrix ***A*** of order *n* and bandwidth *h* is reduced to a real symmetric tridiagonal matrix ***T*** by using the Rutishauser-Schwarz's orthogonal similarity transformation, such as $\boldsymbol{T} = \boldsymbol{Q}_S^{\mathrm{T}} \boldsymbol{A} \boldsymbol{Q}_S$, where $\boldsymbol{Q}_s$ is an orthogonal matrix, and also $0 \le h << n$.

Parameters

A	Input. Real symmetric band matrix ***A***. Its contents are destroyed an output. Compressed mode for a symmetric band matrix. One-dimensional array of size $n(h+1)-h(h+1)/2$
N	Input. Order *n* of the matrix ***A***.
NH	Input. Bandwidth *h*.
D	Output. Diagonal elements of the tridiagonal matrix. One-dimensional array of size *n*.
SD	Output. Subdiagonal elements of the tridiagonal matrix ***T***. One-dimensional array of size *n*. In the subroutine, only SD(2) to SD(N) are used and SD(1) is set to zero.
ICON ..	Output. Condition code. See Table BTRID-1.

Table BTRID-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	NH=0 or NH=1	No reduction performed.
30000	NH<0 or NH≥N	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH and MGSSL
  FORTRAN basic functions ... MIN0, ABS and SQRT

- When compared to the Householder method which reduces the matrix to a real symmetric tridiagonal matrix, the Rutishauser method used in this subroutine is better both in terms of the amount of storage and computations if the ratio of the bandwidth to the order number, $r=h/n$, is small. If the ratio *r* exceeds 1/6, the Householder method is better.

- Example
  This example computes eigenvalues by using subroutine BSCT1 after reduction of a real symmetric band matrix of order *n* and bandwidth *h* to a tridiagonal matrix under conditions $n \le 100$ and $h \le 10$.

```
C     **EXAMPLE**
      DIMENSION A(1100),D(100),SD(100),
     *          E(100),VW(300)
   10 READ(5,500) N,NH,M,EPST
      IF(N.EQ.0) STOP
      NN=(NH+1)*(N+N-NH)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N,NH,IABS(M)
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=MIN0(NH+1,I)+NE
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      CALL BTRID(A,N,NH,D,SD,ICON)
      WRITE(6,620)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,640) (I,D(I),SD(I),I=1,N)
      CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)
      WRITE(6,650)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      MM=IABS(M)
      WRITE(6,660) (I,E(I),I=1,MM)
      GO TO 10
  500 FORMAT(3I5,E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'/
     *       11X,'** ORDER =',I5,10X,'NH=',
     *       I3,'M=',I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0'/11X,'** TRIDIAGONAL ',
     *       'MATRIX')
  630 FORMAT(/11X,'** CONDITION CODE =',I5/)
  640 FORMAT(5X,I5,2E16.7)
  650 FORMAT('0'/11X,'** EIGENVALUES')
  660 FORMAT(5X,'E(',I3,')=',E15.7)
      END
```

Method

Although the Househo????ler method is often used for reduction of a real symmetric matrix to a tridiagonal matrix (see"Method" for the subroutine TRID1), if this method is applied to a band matrix, however, its non-zero elements spread over the matrix during the reduction, and therefore it cannot be used only in the range of band area. On the other hand, in the Rutishauser-Schwarz method, subdiagonal lines can be eliminated one by one from outside first by processing the band part of the band matrix ***A*** (of order *n*, band width *h*, and each element a_{ij}).

Consider lower part of matrix ***A*** including its diagonal elements since the matrix is symmetric.

First, in order to eliminate $a_{h+1,1}$, transform matrix ***A*** into $\boldsymbol{A}_1 = \boldsymbol{R}_1 \boldsymbol{A} \boldsymbol{R}_1^{\mathrm{T}}$, where $\boldsymbol{R}_1$ is an orthogonal matrix associated with the h-th and $(h+1)$-th rows of $\boldsymbol{A}$, and $\tan\theta_1 = a_{h+1,1}/a_{h,1}$.

$$R_1 = \begin{array}{c@{}c}
\begin{array}{cc} & h \quad h+1 \end{array} & \\
\left[\begin{array}{ccccccc}
1 & & & & & & \\
 & \ddots & & & & 0 & \\
 & & 1 & & & & \\
 & & & \cos\theta_1 \sin\theta_1 & & & \\
 & & & -\sin\theta_1 \cos\theta_1 & & & \\
 & & & & 1 & & \\
 & 0 & & & & \ddots & \\
 & & & & & & 1
\end{array}\right] & \begin{array}{l} \\ \\ \\ h \\ h+1 \\ \\ \\ \\ \end{array}
\end{array}$$

The element $a_{h+1.1}$ becomes zero by this operation, but another new non-zero element $a_{2h+1,h}$ is generated. In order to eliminate it, perform $\boldsymbol{A}_1=\boldsymbol{R}_2\boldsymbol{A}_1\boldsymbol{R}_2^{\mathrm{T}}$ by using the orthogonal matrix,

$$R_2 = \begin{array}{c@{}c}
\begin{array}{cc} & 2h \quad 2h+1 \end{array} & \\
\left[\begin{array}{ccccccc}
1 & & & & & & \\
 & \ddots & & & & 0 & \\
 & & 1 & & & & \\
 & & & \cos\theta_2 \sin\theta_2 & & & \\
 & & & -\sin\theta_2 \cos\theta_2 & & & \\
 & & & & 1 & & \\
 & 0 & & & & \ddots & \\
 & & & & & & 1
\end{array}\right] & \begin{array}{l} \\ \\ \\ 2h \\ 2h+1 \\ \\ \\ \\ \end{array}
\end{array}$$

associated with the $2h$-th and $(2h+1)$-th rows of the matrix, where $\tan\theta_2 = a_{2h+1,h}/a_{2h,h}$. By this operation, $a_{2h+1,h}$ becomes zero, but again another new non-zero element $a_{3h+1,2h}$ is generated. By using this similarity transformation (orthogonal similarity transformation) repeatedly by an appropriate orthogonal matrix, element $a_{h+1,1}$ can be eliminated entirely. To eliminate $a_{h+2,2}$ an appropriate orthogonal transformation is repeatedly performed to eliminate non-zero elements generated during the transformation. Then element $a_{h+2,2}$ can be entirely eliminated. By repeatedly performing this operation, all the most outward subdiagonal elements can be eliminated and the bandwidth is reduced to h-1.

The bandwidth can be reduced further by one by applying the same procedure as done to the original matrix to this newly produced matrix.

Fig. BTRID-1 shows the non-zero elements (indicated by $\times$ in the diagram) that are generated successively when eliminating the element $a_{h+1,1}$, i.e., a_{31} (indicated by * in the diagram) for n=10 and h=2 and the lines of influence of the orthogonal similarity transformation to eliminate the non-zero elements.

The number of multiplications necessary for eliminating the most outward subdiagonal elements of a matrix of bandwidth h is approximately $4n^2$, so that the number of necessary multiplications for a tridiagonalization is about $4hn^2$ (See Reference [12]). On the other hand, in Householder method the number of necessary multiplications for tridiagonalization of a real symmetric matrix of order n is $2n^3/3$. Therefore, for $r=h/n<1/6$, the Rutishauser method is better in respect to the number of multiplications carried out.

The orthogonal matrix $\boldsymbol{Q}_s$ used to reduce the original matrix $\boldsymbol{A}$ to a tridiagonal matrix ($\boldsymbol{T}=\boldsymbol{Q}_s\boldsymbol{A}\boldsymbol{Q}_s^{\mathrm{T}}$) can get denoted as a product of orthogonal matrices $\boldsymbol{R}_1,\boldsymbol{R}_2,...$ such as $\boldsymbol{Q}_s=\boldsymbol{R}_s...\boldsymbol{R}_2\boldsymbol{R}_1$.

However, since it is not a good idea to have the $\boldsymbol{Q}_s$ as an $n \times n$ matrix as far as amount of storage and computations are concerned, this subroutine does not carry out such an computation.

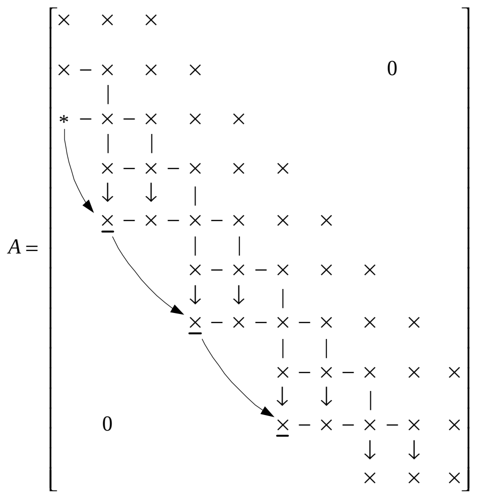


Fig. BTRID-1 Elimintation of elements by Rutishauser-Schwarz method

I11-81-1101 BYN, DBYN

Integer order Bessel function of the second kind $Y_n(x)$
CALL BYN(X,N,BY,ICON)

Function

This subroutine computes the integer order Bessel function of the second kind

$$Y_n(x)=\frac{2}{\pi}\left[J_n(x)\{\log(x/2)+\gamma\}\right]$$
$$-\frac{1}{\pi}\sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}(x/2)^{2k-n}-\frac{1}{\pi}\sum_{k=0}^{\infty}\frac{(-1)^k}{k!(n+k)!}$$
$$\cdot\left\{(x/2)^{2k+n}\left(\sum_{m=1}^{k}1/m+\sum_{m=1}^{n+k}1/m\right)\right\}$$

for $x>0$, by the recurrence formula. Where, $J_n(x)$ is the integer order Bessel function of the first kind, and γ denotes the Euler's constant and also the assumption $\sum_{m=1}^{0}1/m=0$ is made.

Parameters

X	Input. Independent variable x.
N	Input. Order n of $Y_n(x)$.
BY	Output. Function value $Y_n(x)$.
ICON ..	Output. Condition code. See Table BYN-1.

When N=0 or N=1, ICON is handled the same as in ICON for BY0 and BY1.

Table BYN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X≥t_{max}	BY is set to 0.0.
30000	X≤0	BY is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... BY0, BY1, UBJ0, UBJ1, MGSSL, and UTLIM
  FORTRAN basic function ... IABS, FLOAT, DSIN, DCOS, DLOG, and DSQRT
- Notes
  [Range of argument X]
  $0<X<t_{max}$
  If X becomes large enough, sin(x-π/4) and cos(x-π/4) , which are used in calculating $Y_0(x)$ and $Y_1(x)$, will lose accuracy. The limit is provided for that reason. (See (4.4) in the Method section of BY0 and BY1)
  When calculating $Y_o(x)$ and $Y_1,(x)$, use subroutines BY0 and BY1 instead.
- Example
  The following example generates a table of $Y_n(x)$ for range of x from 1 to 10 with increment 1 and for the range of N from 20 to 30 with increment 1.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 20 N=20,30
      DO 10 K=1,10
      X=K
      CALL BYN(X,N,BY,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,N,BY
      IF(ICON.NE.0) WRITE(6,620) X,N,BY,ICON
   10 CONTINUE
   20 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF BESSEL ',
     *'FUNCTION'///6X,'X',5X,'N',8X,
     *'YN(X)'/)
  610 FORMAT(' ',F8.2,I5,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     *E17.7,5X,'N=',I5,5X,'BY=',E17.7,5X,
     *'CONDITION=',I10)
      END
```

Method

Bessel function $Y_n(x)$ is calculated using the following recurrence formula.

$$Y_{k+1}(x)=\frac{2k}{x}Y_k(x)-Y_{k-1}(x),k=1,2,...,n-1 \qquad (4.1)$$

where, both $Y_0(x)$ and $Y_1(x)$ are calculated by using BY0 and BY1.

I11-83-0201 BYR, DBYR

Real order Bessel function of the second kind $Y_\nu(x)$
CALL BYR(X,V,BY,ICON)

Function

This subroutine evaluates a real order Bessel function of the second kind

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}$$

by using a modified series expansion and the τ method. In the above expression, $J_\nu(x)$ is the Bessel function of the first kind.

Parameters

X	Input. Independent variable x.
V	Input. Order ν of $Y_\nu(x)$.
BY	Output. Value of function $Y_\nu(x)$.
ICON ..	Output. Condition code. (See Table BYR-1.)

Table BYR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	• X=0.0 or the value of BY was large almost to overflow. • X≥t_{max}	The negative infinite value of the floating point expression is output to BY. BY is set as 0.0.
30000	X<0.0 or V<0.0	BY is set as 0.0.

Comments on use

- Subprograms used

  SSL II ... AMACH, AFMAX, MGSSL, ULMAX, and UTLIM

  FORTRAN basic function ... FLOAT, ALOG, AMAX1, ALGAMA, ABS, GAMMA, SQRT, SIN, and COS

- Notes

  X>0.0 and ν≥0.0 must be satisfied.

  To evaluate $Y_0(x)$ or $Y_1(x)$, it is better to use BY0 or BY1 respectively rather than this subroutine.

  If values $Y_\nu(x)$, $Y_{\nu+1}(x)$, $Y_{\nu+2}(x)$, ... $Y_{\nu+M}(x)$ are necessary at the same time, calculate $Y_\nu(x)$ and $Y_{\nu+1}(x)$ and with this subroutine , then use recurrence formula((4.2) in "Method") repeatedly to obtain the values of higher order as $Y_{\nu+2}(x)$, $Y_{\nu+3}(x)$, ..., $Y_{\nu+M}(x)$.

  If this subroutine is called repeatedly with the same value of ν for large values of x, the common procedure is bypassed to calculate the value of $Y_\nu(x)$ effectively.

- Example

  The following example calculates the value of $Y_\nu(x)$ for ν from 0.5 to 0.8 with increment 0.01 and for x from 1 to 10 with increment 1.

```
C     **EXAMPLE**
      DO 20 NV=50,80
      V=FLOAT(NV)/100.0
      DO 10 K=1,10
      X=FLOAT(K)
      CALL BYR(X,V,BY,ICON)
      WRITE(6,600) V,X,BY,ICON
   10 CONTINUE
   20 CONTINUE
      STOP
  600 FORMAT(' ',F8.3,F8.2,E17.7,I7)
      END
```

Method

A ν-order Bessel function of the second kind is defined using Bessel functions of the first kind $J_\nu(x)$ and $J_{-\nu}(x)$ as follows:

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \tag{4.1}$$

If the value of order ν is expressed as integer n, this is defined as the limit value for $\nu \to n$.

In this subroutine, the value of $Y_\nu(x)$ is calculated directly for $0 \le \nu \le 2.5$. For $\nu > 2.5$, if fraction part μ of ν is equal to or less than 0.5, values of $Y_{\mu+1}(x)$ and $Y_{\mu+2}(x)$ are calculated directly; and if μ is greater than 0.5, values of $Y_\mu(x)$ and $Y_{\mu+1}(x)$ are calculated directly. These values are used to calculate $Y_\nu(x)$ form the recurrence formula:

$$Y_{\nu+1}(x) = \frac{2\nu}{x} Y_\nu(x) - Y_{\nu-1}(x) \tag{4.2}$$

For $0 \le \nu \le 2.5$, calculation method of $Y_\nu(x)$ is explained below. The method varies depending on the value of x. For

$$x \le 3.66 \tag{4.3}$$

the calculation method for a small value of x is used; whereas, for

$$x > 3.66 \tag{4.4}$$

the calculation method for a large value of x is used.

The method for a small value of *x*

Series expansion of the following functions $J_\nu(x)$ and $J_{-\nu}(x)$ is used:

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \left\{ \frac{1}{\Gamma(1+\nu)} + \frac{-\frac{1}{4}x^2}{1!\Gamma(2+\nu)} \right.$$

$$\left. + \frac{\left(-\frac{1}{4}x^2\right)^2}{2!\Gamma(3+\nu)} + \cdots \right\} \tag{4.5}$$

$$J_{-\nu}(x) = \left(\frac{x}{2}\right)^{-\nu} \left\{ \frac{1}{\Gamma(1-\nu)} + \frac{-\frac{1}{4}x^2}{1!\Gamma(2-\nu)} \right.$$

$$\left. + \frac{\left(-\frac{1}{4}x^2\right)^2}{2!\Gamma(3-\nu)} + \cdots \right\} \tag{4.6}$$

$\phi_1(\nu,x)$ and $\phi_2(\nu,x)$ are defined as follows:

$$\left. \begin{aligned} \phi_1(\nu,x) &= \frac{\left(\frac{x}{2}\right)^{-\nu} - 1}{\nu} \\ \phi_2(\nu,x) &= \frac{1-\left(\frac{x}{2}\right)^{\nu}}{\nu} \end{aligned} \right\} \tag{4.7}$$

The calculation method varies depending on the interval of ν: $0\le\nu\le0.5$, $0.5<\nu\le1.5$, and $1.5<\nu\le2.5$. Explanations of the calculation method for $0\le\nu\le0.5$ follow.

The following is obtained from expressions (4.5), (4.6), and (4.7):

$$J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x) = \nu\sum_{k=0}^{\infty}\{A_k(\nu,x)+B_k(\nu,x)\} \tag{4.8}$$

where

$$\left. \begin{aligned} A_k(\nu,x) &= -\left(-\frac{1}{4}x^2\right)^k \\ &\cdot\left\{\frac{1}{k!\nu}\left(\frac{1}{\Gamma(k+1-\nu)} - \frac{\cos(\nu\pi)}{\Gamma(k+1-\nu)}\right)\right\} \\ B_k(\nu,x) &= -\left(-\frac{1}{4}x^2\right)^k \\ &\cdot\left\{\frac{1}{k!}\left(\frac{\phi_1(\nu,x)}{\Gamma(k+1-\nu)} + \frac{\phi_2(\nu,x)\cos(\nu\pi)}{\Gamma(k+1+\nu)}\right)\right\} \end{aligned} \right\} \tag{4.9}$$

If $A_k(\nu,x)$ and $B_k(\nu,x)$ are calculated as they are, cancellation occurs for $\nu \approx 0$. This is prevented as follows:

Since $\phi_1(\nu,x)$ and $\phi_2(\nu,x)$ have the same sign in $B_k(\nu,x)$,cancellation does not occur even when they are added; however, if $\phi_1(\nu,x)$ and $\phi_2(\nu,x)$ defined in expression (4.7) are calculated as they expressed in the right sides, cancellation occurs if $(x/2)^\nu$ is close to 1. To calculate $\phi_1(\nu,x)$ and $\phi_2(\nu,x)$ as accurate as binary rounding error, it is only necessary to obtain the best approximation of the following function $f(t)$ having the required accuracy for $-\log2\le t\le\log2$:

$$f(t) = \frac{e^t - 1}{t} = \frac{1}{1!} + \frac{t}{2!} + \frac{t^2}{3!} + \cdots \tag{4.10}$$

This is because $\phi_1(\nu,x)$ and $\phi_2(\nu,x)$ can be calculated with a high accuracy using the best approximation as follows:

$$\begin{aligned} \phi_1(\nu,x) &= -f\left(-\nu\log\left(\frac{x}{2}\right)\right)\log\left(\frac{x}{2}\right) \\ \phi_2(\nu,x) &= f\left(\nu\log\left(\frac{x}{2}\right)\right)\log\left(\frac{x}{2}\right) \end{aligned} \tag{4.11}$$

The following best approximation of function $f(t)$ is incorporated in this subroutine:

$$f(t) \approx \frac{2\sum_{k=0}^{M} p_k t^{2k}}{\sum_{k=0}^{N} q_k t^{2k} - t\sum_{k=0}^{M} p_k t^{2k}} \tag{4.12}$$

To calculate $(1/\Gamma(k+1-\nu)-\cos(\nu\pi)/\Gamma(k+1+\nu))/(k!\nu)$ for $A_k(\nu,x)$ in expression (4.9) without cancellation, it is only necessary to obtain the best approximation having the required accuracy.

In this case, since the following is satisfied for the part enclosed in braces in $A_k(\nu,x)$ of expression (4.9).

$$\tilde{A}_k(\nu) = \frac{1}{(k+\nu)(k-\nu)}\left[\tilde{A}_{k-1}(\nu) + \frac{1}{k!}\left\{\frac{1}{\Gamma(k-\nu)} + \frac{\cos(\nu\pi)}{\Gamma(k+\nu)}\right\}\right]$$
$$(k \ge 1) \tag{4.13}$$

it is only necessary to obtain an approximation of $\tilde{A}_0(\nu)$ for $0\le\nu\le0.5$. In this subroutine,

$$\tilde{A}_0(\nu) \approx (\nu - \nu_0)\sum_{k=0}^{M} p_k \nu^k \tag{4.14}$$

is used as the best approximation.

Values of $A_k(\nu,x)$ and $B_k(\nu,x)$ can be obtained without cancellation in this way, buy all values do not have the same sign; consequently, cancellation may occur in the addition to be processed in expression (4.8). Since function $Y_\nu(x)$ is an oscillating function with respect to x for constant ν and has a zero point, it is impossible to obtain the value of such a function with a relative accuracy. This means that the absolute accuracy must be used. (except for the case if the value of x is small and the value of $Y_\nu(x)$ takes a great negative value.) Examining the range for x in which the value of $Y_\nu(x)$ can be calculated with an absolute accuracy applicable in principle,

$x \le 3.66$ (4.15)

is found to be valid in this method. Since values of the terms comprising the sum in expression (4.8) become sufficiently small as the value of *k* becomes greater, it is only necessary to calculate a relatively small number of terms until the resultant value converges according to the required accuracy.

Therefore, the value of $Y_\nu(x)$ is obtained from

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} = \frac{\sum_{k=0}^{\infty}\{A_k(\nu,x) + B_k(\nu,x)\}}{g(\nu)} \tag{4.16}$$

using the best approximation

$$g(\nu) \approx \sum_{k=0}^{M} P_k \nu^{2k} \tag{4.17}$$

of

$$g(\nu) = \frac{\sin(\nu\pi)}{\nu} \tag{4.18}$$

For $\nu = 0$, the value of $Y_0(x)$ is calculated with the limit value obtained from expression (4.1) for $\nu \to 0$ as follows.

$$Y_0(x) = \frac{2}{\pi}\sum_{k=0}^{\infty}\frac{\left(-\frac{1}{4}x^2\right)^k}{(k!)^2}\left\{\gamma + \log\left(\frac{x}{2}\right) - \Phi_k\right\} \tag{4.19}$$

This is more efficient than the calculation of (4.16).
Where γ is the Euler's constant, $\Phi_0=1$, and

$$\Phi_k = \sum_{m=0}^{k}\frac{1}{m}(k \ge 1).$$

The calculation method for $0 \le \nu \le 0.5$ has been explained. Calculation methods for $0.5 < \nu \le 1.5$ and $1.5 < \nu \le 2.5$ are omitted because the same concept applies to these ranges.

The method for a large value of *x*

Bessel function of the second kind $Y_\nu(x)$ is given as the imaginary part of the Hankel function of the first kind

$$H_\nu^{(1)}(x) = J_\nu(x) + iY_\nu(x) \tag{4.20}$$

Where *i* is the imaginary unit $\left(i = \sqrt{-1}\right)$, $J_\nu(x)$ is the Bessel function of the first kind. And $f_\nu\left(\frac{1}{x}\right)$ is defined as follows:

$$f_\nu\left(\frac{1}{x}\right) = \left(\frac{\pi x}{2}\right)^{\frac{1}{2}} e^{-i\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right)} H_\nu^{(1)}(x) \tag{4.21}$$

where, $f_\nu\left(\frac{1}{x}\right)$ is an imaginary function, and assume whose real part is $P\left(\nu\frac{1}{x}\right)$ and whose imaginary part is $Q\left(\nu,\frac{1}{x}\right)$. In this case, $f_\nu\left(\frac{1}{x}\right)$ is expressed as:

$$f_\nu\left(\frac{1}{x}\right) = P\left(\nu,\frac{1}{x}\right) + iQ\left(\nu,\frac{1}{x}\right) \tag{4.22}$$

Using expressions (4.20), (4.21), and (4.22), $Y_\nu(x)$ is represented as :

$$Y_\nu(x) = \left(\frac{2}{\pi x}\right)^{\frac{1}{2}}\left\{P\left(\nu,\frac{1}{x}\right)\sin\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right) + Q\left(\nu,\frac{1}{x}\right)\cos\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right)\right\} \tag{4.23}$$

Let us obtain the approximation of $f_\nu\left(\frac{1}{x}\right)$ of (4.21) in the following paragraph. If the approximation is known, the real and imaginary parts of expression (4.22) are obtained as $P\left(\nu,\frac{1}{x}\right)$ and $Q\left(\nu,\frac{1}{x}\right)$. With these values, $Y_\nu(x)$ is obtained.

The Hankel function of the first kind $H_\nu^{(1)}(x)$ satisfies the following differential equation:

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} + \left(x^2 - \nu^2\right)w = 0 \tag{4.24}$$

Assuming $t = \frac{1}{x}$ and substituting $H_\nu^{(1)}(x)$ of (4.21) to (4.24), $f_\nu\left(\frac{1}{x}\right)$ or $f_\nu(t)$ satisfies the following differential equation.

$$t^2 f_\nu''(t) + 2(t-i)f(t) + \left(\frac{1}{4} - \nu^2\right)f_\nu(t) = 0 \tag{4.25}$$

Applying the τ method to this differential equation, the approximation of $f_\nu(t)$ is obtained for a small value of *t*. For this purpose, let us consider the following differential equation derived from (4.25) by adding the shifted ultraspherical polynomial on the orthogonal interval $[0,\eta]$ to the right side by multiplying τ.

$$t^2 f_{\nu m}''(t) + 2(t-i)f_{\nu m}'(t) + \left(\frac{1}{4} - \nu^2\right)f_{\nu m}(t) = \tau C_m^{*(\alpha)}\left(\frac{t}{\eta}\right) \tag{4.26}$$

This equation has the following polynomial of degree *m* as a particular solution:

$$f_{\nu m}(t) = -\tau\sum_{k=0}^{m}\frac{C_{mk}^{*(\alpha)}\sum_{l=0}^{k}a_l t^l}{2(k+1)a_{k+1}\eta^k} \tag{4.27}$$

where

$$\left.\begin{aligned} a_0 &= 1 \\ a_l &= i^l \frac{(4\nu^2-1^2)(4\nu^2-3^2)\cdots(4\nu^2-(2l-1)^2)}{l!8^l} \end{aligned}\right\} \quad (4.28)$$

$C^{*(\alpha)}_{mk}$ is the k-th order coefficient of $C^{*(\alpha)}_m(t)$. If the value of τ in the right side of (4.26) is sufficiently small, $f_{\nu m}(t)$ can be regarded as the approximation polynomial of $f_\nu(t)$.

Determining the value of τ (the value of τ decreases as the value of m increases) by the initial condition $f_{\nu m}(0)=1(f_\nu(t) \to 1$ for $t \to 0)$, we obtain the following approximation polynomial $f_{\nu m}(t)$ of $f_\nu(t)$ for $0 \le t \le \eta$:

$$f_{\nu m}(t) = \frac{\displaystyle\sum_{k=0}^{m} \frac{C^{*(\alpha)}_{mk} \sum_{l=0}^{k} a_l t^l}{(k+1)a_{k+1}\eta^k}}{\displaystyle\sum_{k=0}^{m} \frac{C^{*(\alpha)}_{mk}}{(k+1)a_{k+1}\eta^k}} \quad (4.29)$$

The value of α is assumed to be 1 based on the results of experiments conducted for the accuracy of $f_{\nu m}(t)$ with various values.

Since $P\left(\nu,\frac{1}{x}\right)$ and $Q\left(\nu,\frac{1}{x}\right)$ are the real and imaginary parts of $f_\nu\left(\frac{1}{x}\right)$, they can be expressed as

$$\begin{aligned} \begin{Bmatrix} P\left(\nu,\frac{1}{x}\right) \\ Q\left(\nu,\frac{1}{x}\right) \end{Bmatrix} &\approx \begin{Bmatrix} \mathrm{Re} \\ \mathrm{Im} \end{Bmatrix} \left(\frac{\displaystyle\sum_{k=0}^{m} \frac{C^{*(1)}_{mk} \sum_{l=0}^{k} a_l \left(\frac{1}{x}\right)^l}{(k+1)a_{k+1}\eta^k}}{\displaystyle\sum_{k=0}^{m} \frac{C^{*(1)}_{mk}}{(k+1)a_{k+1}\eta^k}} \right) \\ &= \begin{Bmatrix} \mathrm{Re} \\ \mathrm{Im} \end{Bmatrix} \left(\frac{\displaystyle\sum_{l=0}^{m} a_l \left(\frac{1}{x}\right)^l \sum_{k=l}^{m} \frac{C^{*(1)}_{mk}}{(k+1)a_{k+1}\eta^k}}{\displaystyle\sum_{k=0}^{m} \frac{C^{*(1)}_{mk}}{(k+1)a_{k+1}\eta^k}} \right) \end{aligned} \quad (4.30)$$

For efficient calculation formulas of $P\left(\nu,\frac{1}{x}\right)$ and $Q\left(\nu,\frac{1}{x}\right)$, these expressions are transformed to

$$\begin{Bmatrix} P\left(\nu,\frac{1}{x}\right) \\ Q\left(\nu,\frac{1}{x}\right) \end{Bmatrix} \approx \begin{Bmatrix} \mathrm{Re} \\ \mathrm{Im} \end{Bmatrix} \left(\frac{\sqrt{\frac{\pi}{2}} \displaystyle\sum_{l=0}^{m} \left(\frac{1}{x}\right)^l (-i)^l \frac{d_l}{W_l} \sum_{k=l}^{m} i^k W_{k+1}}{\displaystyle\sum_{k=0}^{m} i^k W_{k+1}} \right) \quad (4.31)$$

where

$$\left.\begin{aligned} W_0 &= 1 \\ W_k &= \prod_{n=0}^{k-1} \frac{e_n}{\left(n+\frac{1}{2}\right)^2 - \nu^2} \quad (k \ge 1) \end{aligned}\right\} \quad (4.32)$$

d_l and e_n are given as

$$\left.\begin{aligned} d_0 &= \sqrt{\frac{2}{\pi}} \\ d_l &= \sqrt{\frac{2}{\pi}} \frac{C^{*(1)}_{m,l-1}}{\eta^{l-1} l} \quad (l \ge 1) \end{aligned}\right\} \quad (4.33)$$

$$\left.\begin{aligned} e_0 &= 2C^{*(1)}_{m0} \\ e_n &= \frac{2nC^{*(1)}_{mn}}{\eta C^{*(1)}_{m,n-1}} \quad (n \ge 1) \end{aligned}\right\} \quad (4.34)$$

As explained at the beginning of explanations, this method is applied to the range of x>3.66; thus the calculation formula can be determined with a minimum value for the required accuracy assuming η=1/3.66. Since the value of m decreases in proportion to that of η, it is best to change the values of η and m depending on intervals of x to improve calculation efficiency for a large value of x. It is known that the values of $P\left(\nu,\frac{1}{x}\right)$ and $Q\left(\nu,\frac{1}{x}\right)$ can be efficiently calculated from the following:

For single precision:

$$\left.\begin{aligned} &\text{for } 3.66 < x < 10, \quad \eta = \frac{1}{3.66}, \quad m = 8 \\ &\text{for } x \ge 10, \quad \eta = \frac{1}{10}, \quad m = 5 \end{aligned}\right\} \quad (4.35)$$

For double precision:

$$\left.\begin{aligned} &\text{for } 3.66 < x < 10, \quad \eta = \frac{1}{3.66}, \quad m = 24 \\ &\text{for } x \ge 10, \quad \eta = \frac{1}{10}, \quad m = 15 \end{aligned}\right\} \quad (4.36)$$

In this subroutine, the constants d_l and e_n are tabulated in advance.

I11-81-0401 BY0, DBY0

Zero order Bessel function of the second kind $Y_0(x)$.
CALL BY0(X,BY,ICON)

Function

This subroutine computes the zero order Bessel function of the second kind $Y_0(x)$

$$Y_0(x)=\frac{2}{\pi}\left[J_0(x)\{\log(x/2)+\gamma\}\right.\\ \left.-\sum_{k=1}^{\infty}\frac{(-1)^k(x/2)^{2k}}{(k!)^2}\cdot\sum_{m=1}^{k}1/m\right]$$

(where $J_0(x)$: zero order Bessel function of the first kind and γ: Euler's constant)
by rational approximations and the asymptotic expansion.
Where, $x>0$.

Parameters

X	Input. Independent variable x.
BY	Output. Function value $Y_0(x)$.
ICON ..	Output. Condition code. See Table BY0-1.

Table BY0-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$X\geq t_{max}$	BY is set to 0.0.
30000	$X\leq 0$	BY is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... UBJ0,MGSSL, and UTLIM
  FORTRAN basic functions ... DSIN, DCOS, DLOG and DSQRT

- Notes
  [Range of argument X]
  $0<X<t_{max}$
  These limits are used because $\sin(x-\pi/4)$ and $\cos(x-\pi/4)$ lose accuracy if X becomes large. (See "Method".)

- Example
  The following example generates a table of $Y_0(x)$ from 1 to 100 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,100
       X=K
       CALL BY0(X,BY,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BY
       IF(ICON.NE.0) WRITE(6,620) X,BY,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'//6X,'X',9X,'Y0(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BY=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With x=8 as the boundary, the approximation formula used to calculate Bessel function $Y_0(x)$ changes.

- For $0<x\leq 8$
  The expansion of $Y_0(x)$ into power series

$$Y_0(x)=\frac{2}{\pi}\left[J_0(x)\{\log(x/2)+\gamma\}\right.\\ \left.-\sum_{k=1}^{\infty}\frac{(-1)^k(x/2)^{2k}}{(k!)^2}\sum_{m=1}^{k}1/m\right] \quad (4.1)$$

(where $J_0(x)$: Zero-order Bessel function of the first kind and γ: Euler's constant) is calculated using the following rational approximations.
Single precision:

$$Y_0(x)=\sum_{k=0}^{5}a_kx^{2k}\Big/\sum_{k=0}^{5}b_kx^{2k}+\frac{2}{\pi}J_0(x)\log(x) \quad (4.2)$$

Theoretical precision = 8.14 digits

Double precision:

$$Y_0(x)=\sum_{k=0}^{8}a_kx^{2k}\Big/\sum_{k=0}^{8}b_kx^{2k}+\frac{2}{\pi}J_0(x)\log(x) \quad (4.3)$$

Theoretical precision = 18.78 digits

- For $x>8$
  The asymptotic expansion of $Y_0(x)$

$$Y_0(x)=\sqrt{\frac{2}{\pi x}}\{P_0(x)\sin(x-\pi/4)\\ +Q_0(x)\cos(x-\pi/4)\} \quad (4.4)$$

is evaluated through use of the following approximate expressions of $P_1(x)$ and $Q_1(x)$
Single precision:

$$P_0(x)=\sum_{k=0}^{2}a_kz^{2k}\Big/\sum_{k=0}^{2}b_kz^{2k}, z=8/x \quad (4.5)$$

Theoretical precision = 10.66 digits

$$Q_0(x) = \sum_{k=0}^{1} c_k z^{2k+1} \Bigg/ \sum_{k=0}^{2} d_k z^{2k} , \quad z = 8/x \tag{4.6}$$

Theoretical precision = 9.58 digits

Double precision:

$$P_0(x) = \sum_{k=0}^{5} a_k z^{2k} \Bigg/ \sum_{k=0}^{5} b_k z^{2k} , \quad z = 8/x \tag{4.7}$$

Theoretical precision = 18.16 digits

$$Q_0(x) = \sum_{k=0}^{5} c_k z^{2k+1} \Bigg/ \sum_{k=0}^{5} d_k z^{2k} , \quad z = 8/x \tag{4.8}$$

Theoretical precision = 18.33 digits
For more information, see Reference [78] pp.141 - 149.

I11-81-0501 BY1, DBY1

First order Bessel function of the second kind $Y_1(x)$.
CALL BY1(X,BY,ICON)

Function

This subroutine computes the first order Bessel function of the second kind

$$Y_1(x) = 2/\pi\{J_1(x)(\log(x/2)+\gamma)-1/x\} \\ -1/\pi\left\{\sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{2k+1}}{k!(k+1)!}\left(\sum_{m=1}^{k}1/m+\sum_{m=1}^{k+1}1/m\right)\right\}$$

(where $J_1(x)$: first order Bessel function of the first kind and γ: Euler's constant) by rational approximations and the asymptotic expansion. Where, x>0.

Parameters

X	Input. Independent variable x.
BY	Output. Function value $Y_1(x)$.
ICON ..	Output. Condition code. See Table BY1-1.

Table BY1-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X≥t_{max}	BY=0.0
30000	X≤0	BY=0.0

Comments on use

- Subprograms used
  SSL II ... UBJ1, MGSSL and UTLIM
  FORTRAN basic function ... DSIN, DCOS, DLOG and DSQRT

- Notes
  The range of argument X
  0<X<t_{max}
  These limits are set because sin(x-3π/4) and cos(x-3π/4) can not be calculated accuracy if X becomes large. (See "Method".)

- Example
  The following example generates a table of $Y_1(x)$ from 1 to 100 with increment 1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,100
       X=K
       CALL BY1(X,BY,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,BY
       IF(ICON.NE.0) WRITE(6,620) X,BY,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF BESSEL ',
      *'FUNCTION'///6X,'X',9X,'Y1(X)'/)
   610 FORMAT(' ',F8.2, E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'BY=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Method

With x=8 as the boundary, the approximation formula used to calculate Bessel function $Y_1(x)$ changed.

- For 0<x≤8
  The power series expansion of $Y_1(x)$.

$$Y_1(x) = 2/\pi\{J_1(x)(\log(x/2)+\gamma)-1/x\} \\ -1/\pi\left\{\sum_{k=0}^{\infty}\frac{(-1)^k(x/2)^{2k+1}}{k!(k+1)!}\left(\sum_{m=1}^{k}1/m+\sum_{m=1}^{k+1}1/m\right)\right\} \tag{4.1}$$

(where $J_1(x)$: first order Bessel function of the first kind and γ: Euler's constant) is calculated using the following rational approximations.
Single precision:

$$Y_1(x) = \sum_{k=0}^{7}a_k x^{2k+1}/\sum_{k=0}^{2}b_k x^{2k} \\ +\frac{2}{\pi}\{J_1(x)\log(x)-1/x\} \tag{4.2}$$

Theoretical precision = 8.96 digits

Double precision:

$$Y_1(x) = \sum_{k=0}^{7}a_k x^{2k+1}/\sum_{k=0}^{8}b_k x^{2k} \\ +\frac{2}{\pi}\{J_1(x)\log(x)-1/x\} \tag{4.3}$$

Theoretical precision = 18.24 digits

- For x>8
  The asymptotic expansion of $Y_1(x)$

$$Y_1(x) = \sqrt{\frac{2}{\pi x}}\{P_1(x)\sin(x-3\pi/4) \\ +Q_1(x)\cos(x-3\pi/4)\} \tag{4.4}$$

is calculated through use of the following approximate expression $P_1(x)$ and $Q_1(x)$:
Single precision:

$$P_1(x) = \sum_{k=0}^{2}a_k z^{2k}/\sum_{k=0}^{2}b_k z^{2k}, z=8/x \tag{4.5}$$

Theoretical precision = 10.58 digits

$$Q_1(x) = \sum_{k=0}^{1} c_k z^{2k+1} \Big/ \sum_{k=0}^{2} d_k z^{2k}, z = 8/x \tag{4.6}$$

Theoretical precision = 9.48 digits

Double precision:

$$P_1(x) = \sum_{k=0}^{5} a_k z^{2k} \Big/ \sum_{k=0}^{5} b_k z^{2k}, z = 8/x \tag{4.7}$$

Theoretical precision = 18.11 digits

$$Q_1(x) = \sum_{k=0}^{5} c_k z^{2k+1} \Big/ \sum_{k=0}^{5} d_k z^{2k}, z = 8/x \tag{4.8}$$

Theoretical precision = 18.28 digits

For more information, see Reference [78] pp.141 - 149.

I11-82-1101 CBIN, DCBIN

Integer order modified Bessel function of the first kind $I_n(z)$ with complex variable
CALL CBIN(Z,N,ZBI,ICON)

Function

This subroutine computes integer order modified Bessel function of the first kind $I_n(z)$ with complex variable

$$I_n(z)=\left(\frac{1}{2}z\right)^n\sum_{k=0}^{\infty}\frac{\left(\frac{1}{4}z^2\right)^k}{k!(n+k)!}$$

by the power series expansion of the above form and recurrence formula.

Parameters

Z	Input. Independent variable z. Complex variable.
N	Input. Order n of $I_n(z)$.
ZBI	Output. Function value $I_n(z)$. Complex variable.
ICON ..	Output. Condition code. See Table CBIN-1.

Table CBIN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\|Re(z)\| \geq \log(fl_{max})$ or $\|Im(z)\| \geq \log(fl_{max})$	ZB1=(0.0,0.0)

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL, ULMAX
  FORTRAN basic functions ... REAL, AIMAG, ABS, IABS, FLOAT, CEXP, MAX0

- Notes
  The range of argument should, be $\left|Re(z)\right| \geq \log(fl_{max})$ and $\left|Im(z)\right| \geq \log(fl_{max})$.

  When a set of function values $I_n(z)$, $I_{n+1}(z)$, $I_{n+2}(z)$, ..., $I_{n+M}(z)$, is required at the same time, first, obtain $I_{n+M}(z)$ and $I_{n+M-1}(z)$ with this subroutine. Then the others can be obtained in sequence from high order to low order, that is, $I_{n+M-2}(z)$, $I_{n+M-3}(z)$, ..., $I_n(z)$, by using repeatedly the recurrence formula. Obtaining of the values in the reverse order, that is $I_{n+2}(z)$, $I_{n+3}(z)$, ..., $I_{n+M}(z)$, by the recurrence formula after obtaining $I_n(z)$ and $I_{n+1}(z)$ by this subroutine, must be avoided because of its unstableness.

- Example
  The value of $I_n(z)$ is computed for n=1,2, where z=10+5i.

```
C     **EXAMPLE**
      COMPLEX Z,ZBI
      Z=(10.0,5.0)
      DO 10 N=1,2
      CALL CBIN(Z,N,ZBI,ICON)
      WRITE(6,600) Z,N,ZBI,ICON
   10 CONTINUE
      STOP
  600 FORMAT(' ',2F8.2,I6,5X,2E17.7,I7)
      END
```

Method

If it is known beforehand that the absolute value of $I_n(z)$ will underflow, the following computation is not performed with setting the result as (0.0,0.0).

$I_n(z)$ is computed in different ways depending upon the value of z.

1) When $\left|\mathrm{Re}(z)\right|+\left|\mathrm{Im}(z)\right|\leq 1$,

   The power series expansion,

$$I_n(z)=\left(\frac{1}{2}z\right)^n\sum_{k=0}^{\infty}\frac{\left(\frac{1}{4}z^2\right)^k}{k!(n+k)!} \quad (4.1)$$

is evaluated until the k-th item becomes less than the round-off level of the first term.

2) When $\left|\mathrm{Re}(z)\right|+\left|\mathrm{Im}(z)\right|>1$, $\left|\mathrm{Re}(z)\right|\leq\log(fl_{max})$ and $\left|\mathrm{Im}(z)\right|\leq\log(fl_{max})$.

   The recurrence formula is used. Suppose m to be an appropriately large integer (which depends upon the required precision, z and n) and δ to be an appropriately small constant (10^{-38} in this subroutine).

   With the initial values.

   $G_{m+1}(z)=0$, $G_m(z)=\delta$

   repeat the recurrence equation,

$$G_{k-1}(z)=\frac{2k}{z}G_k(z)+G_{k+1}(z) \quad (4.2)$$

   for k=m, m-1,...,1.

   $I_n(z)$ can then be obtained from

$$I_n(z)\approx e^z G_n(z)\Big/\left(\sum_{k=0}^{m}\varepsilon_k G_k(z)\right) \quad (4.3)$$

   where

$$\varepsilon_k=\begin{cases}1 & (k=0)\\ 2 & (k\geq 1)\end{cases}$$

   Equation (4.3) can be used when Re(z)≥ 0.

   When Re(z)<0, cancellation will take place in the above computations. Therefore, using the relation $I_n(-z)=(-1)^n I_n(z)$, the problem can be reduced to the computation above.

   For detailed information, for example, on how to determine m, see Reference [81] and [83].

I11-82-1301 CBJN, DCBJN

Integer order Bessel function of the first kind $J_n(z)$ with complex variable
CALL CBJN(Z,N,ZBJ,ICON)

Function

This subroutine computes the integer order Bessel function of the first kind with complex variable

$$J_n(z)=\left(\frac{1}{2}z\right)^n\sum_{k=0}^{\infty}\frac{\left(-\frac{1}{4}z^2\right)^k}{k!(n+k)!}$$

by evaluating the power series expansion of the form above and recurrence formula.

Parameters

Z	Input. Independent variable z. Complex variable.
N	Input. Order n of $J_n(z)$.
ZBJ	Output. Value of function $J_n(z)$. Complex variable.
ICON ..	Output. Condition code. See Table CBJN-1.

Table CBJN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert\mathrm{Re}(z)\rvert > \log(fl_{\max})$ or $\lvert\mathrm{Im}(z)\rvert > \log(fl_{\max})$	ZBJ=(0.0,0.0)

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL, ULMAX
  FORTRAN basic functions ... REAL, AIMAG, ABS, IABS, MOD, FLOAT, CEXP, CONJG, MAX0, CMPLX

- Notes
  The range of argument z should be $\lvert\mathrm{Re}(z)\rvert \le \log(fl_{max})$ and $\lvert\mathrm{Im}(z)\rvert \le \log(fl_{max})$.
  When all the values of $J_n(z)$, $J_{n+1}(z)$, $J_{n+2}(z)$, ..., $J_{n+M}(z)$ are required at the same time, first obtain $J_{n+M}(z)$ and $J_{n+M-1}(z)$, by this routine. Then the others can be obtained in sequence from high order to low order, that is, $J_{n+M-2}(z)$, $J_{n+M-3}(z)$, ..., $J_n(z)$, by repeating the recurrence formula. Obtaining of values in the reverse order, that is, $J_{n+2}(z)$, $J_{n+3}(z)$, ..., $J_{n+M}(z)$, by the recurrence formula after obtaining $J_n(z)$ and $J_{n+1}(z)$ by this subroutine, is not recommended because of its unstableness.

- Example
  The value of $J_n(z)$ is computed for n=1 and n=2, where z=10+5i.

```
C      **EXAMPLE**
       COMPLEX Z,ZBJ
       Z=(10.0,5.0)
       DO 10 N=1,2
       CALL CBJN(Z,N,ZBJ,ICON)
       WRITE(6,600) Z,N,ZBJ,ICON
   10  CONTINUE
       STOP
  600  FORMAT(' ',2F8.2,I6,5X,2E17.7,I7)
       END
```

Method

If it is known beforehand that the absolute value of $J_n(z)$ will underflow the following computation is bypassed with the result as (0.0,0.0).

The method for computing $J_n(z)$ depends upon the value of z.

- When $\lvert\mathrm{Re}(z)\rvert+\lvert\mathrm{Im}(z)\rvert \le 1$,
  The power series expansion

$$J_n(z)=\left(\frac{1}{2}z\right)^n\sum_{k=0}^{\infty}\frac{\left(-\frac{1}{4}z^2\right)^k}{k!(n+k)!} \tag{4.1}$$

  is evaluated until the k-th term becomes less than the round-off level of the first term.

- When $\lvert\mathrm{Re}(z)\rvert+\lvert\mathrm{Im}(z)\rvert > 1$, $\lvert\mathrm{Re}(z)\rvert \le \log(fl_{max})$ and $\lvert\mathrm{Im}(z)\rvert \le \log(fl_{max})$
  The recurrence formula is used for the computation. Suppose m to be an appropriately large integer (which depends upon the required accuracy, z and n) and δ to be an appropriately small constant (10^{-38} in this subroutine).
  With the initial values,

  $F_{m+1}(z)=0$, $F_m(z)=\delta$

  and repeating the recurrence equation,

$$F_{k-1}(z)=\frac{2k}{z}F_k(z)-F_{k+1}(z) \tag{4.2}$$

  for $k=m$, $m-1$,...,1
  $J_n(z)$ can be obtained from

$$J_n(z)\approx e^{-iz}F_n(z)\Bigg/\left(\sum_{k=0}^{m}\varepsilon_k i^k F_k(z)\right) \tag{4.3}$$

  where

$$\varepsilon_k=\begin{cases}1 & (k=0)\\ 2 & (k\ge 1)\end{cases}$$

  Equation (4.3) can be used only when $0\le\arg z\le\pi$. When $-\pi<\arg z<0$ cancellation will take place. Therefore, using the relation $J_n(\bar{z})=\overline{J_n(z)}$ it is, reduced to be within the condition $0\le\arg z\le\pi$.

For detailed information, for example, on how to determine m, see References [81] and [83].

I11-84-0101 CBJR, DCBJR

Real order Bessel function of the first kind $J_v(z)$ with complex variable
CALL CBJR(Z,V,ZBJ,ICON)

Function

This subroutine computes the value of real order Bessel function of the first kind with complex variable z

$$J_v(z)=\left(\frac{1}{2}z\right)^v\sum_{k=0}^{\infty}\frac{\left(-\frac{1}{4}z^2\right)^k}{k\,!\Gamma(v+k+1)}$$

using the power series expansion (above expression) and the recurrence formula.

In the above expression, the value of $(1/2\cdot z)^v$ adopt the principal value. Though the principal value of $(1/2\cdot z)^v\equiv e^{v\log(z/2)}$ depends on how the principal value of log(z/2) is selected, it is determined by FORTRAN basic function CLOG. Usually, the principal value of log(z/2)=log $|z/2|$+ i *arg z*, $-\pi<arg\ z\le\pi$

Parameters

Z	Input. Independent variable z. (complex variable).
V	Input. Order v of $J_v(z)$ ($v\ge0$).
ZBJ	Output. Value of function $J_v(z)$. (complex variable).
ICON ..	Output. Condition code. See Table CBJR-1.

Table CBJR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert\mathrm{Re}(z)\rvert>\log(fl_{max})$ or $\lvert\mathrm{Im}(z)\rvert\le\log(fl_{max})$	Set ZBJ=(0.0,0.0)
30000	V<0	Set ZBJ=(0.0,0.0)

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH, ULMAX
  FORTRAN basic functions ... REAL, AIMAG, ABS, FLOAT, CEXP, CLOG, GAMMA, CONJG, AMAX1, CMPLX

- Notes
  $\lvert\mathrm{Re}(z)\rvert\le\log(fl_{max})$, $\lvert\mathrm{Im}(z)\rvert\le\log(fl_{max})$, and V≥0.
  When a set of function values $J_v(z)$, $J_{v+1}(z)$, $J_{v+2}(z)$, ..., $J_{v+\mathrm{M}}(z)$, is needed at the same time, $J_{v+1}(z)$ and $J_{v+M-1}(z)$ are computed with this subroutine, and next $J_{v+M-2}(z)$, $J_{v+M-3}(z)$, ..., $J_v(z)$ are computed by using the recurrence formula repeatedly, it should be avoided in computing $J_{v+2}(z)$, $J_{v+3}(z)$, ..., $J_{v+M}(z)$ by the recurrence formula, after computing $J_v(z)$ and $J_{v+1}(z)$ with this subroutine, in sequence from low order to high order.

- Example
  The following example generates a table of $J_v(z)$ at z=10+5i for the range of v from 0.1 to 10 with increment 0.1.

```
C      **EXAMPLE**
       COMPLEX Z,ZBJ
       Z=(10.0,5.0)
       DO 10 NV=1,100
       V=FLOAT(NV)/10.0
       CALL CBJR(Z,V,ZBJ,ICON)
       IF(ICON.EQ.0) WRITE (6,600) Z,V,ZBJ
    10 CONTINUE
       STOP
   600 FORMAT(' ',2F8.2,F10.3,5X,2E17.7)
       END
```

Method

When it is known the value of $J_v(z)$ will underflow, the following computations are bypassed and the result 0.0 is output.

The computation of $J_v(z)$ depends on z

- $\lvert\mathrm{Re}(z)\rvert+\lvert\mathrm{Im}(z)\rvert\le1$

With the power series expansion

$$J_v(z)=\left(\frac{1}{2}z\right)^v\sum_{k=0}^{\infty}\frac{\left(-\frac{1}{4}z^2\right)^k}{k\,!\Gamma(v+k+1)}\tag{4.1}$$

it is computed until the k-th term is less than the unit round-off in relative to the first term.

- $\lvert\mathrm{Re}(z)\rvert+\lvert\mathrm{Im}(z)\rvert>1$ and $\lvert\mathrm{Re}(z)\rvert\le\log(fl_{max})$ and $\lvert\mathrm{Im}(z)\rvert\le\log(fl_{max})$
  Recurrence formula is used.
  Let's suppose that m is a certain large integer (determined by z, v, and the desired precision), and that δ is set to a certain small constant (10^{-38}) and moreover that n and α are determined by

$$v=n+\alpha(n:\text{integer, }0\le\alpha<1)$$

Initial values

$$F_{\alpha+m+1}(z)=0,F_{\alpha+m}(z)=\delta$$

are set, and recurrence formula

$$F_{\alpha+k-1}(z)=\frac{2(\alpha+k)}{z}F_{\alpha+k}(z)-F_{\alpha+k+1}(z)\tag{4.2}$$

is repeatedly applied to k=m,m-1,...,1. Then the value of function $J_v(z)$ is obtained as

$$J_v(z) \approx \frac{1}{2}\left(\frac{z}{2}\right)^{\alpha} \frac{\Gamma(2\alpha+1)}{\Gamma(\alpha+1)} e^{-iz} F_{\alpha+n}(z) \Bigg/ \left(\sum_{k=0}^{m} \frac{(\alpha+k)\Gamma(2\alpha+k) i^k}{k!} F_{\alpha+k}(z) \right) \tag{4.3}$$

The above expression is suitable in the range of 0 ≤ *arg* *z* ≤ π. Since cancellation occurs in the above expression when -π≤*arg* *z*<0, the problem is returned to that of 0≤*arg* *z*≤π using the relation of $J_v(\bar{z}) = \overline{J_v(z)}$.

For the method of determining of *m* and other details, see References [81] and [83].

I11-82-1201 CBKN, DCBKN

Integer order modified Bessel function of the second kind $K_n(z)$ with complex variable.
CALL CBKN(Z,N,ZBK,ICON)

Function

This subroutine computes the value of integer order modified Bessel function of the second kind with complex variable z

$$
\begin{aligned}
K_n(z) &= K_{-n}(z) \\
&= (-1)^{n+1}\left\{\gamma + \log\left(\frac{z}{2}\right)\right\} I_n(z) \\
&\quad + \frac{(-1)^n}{2}\left(\frac{z}{2}\right)^n \sum_{k=0}^{\infty} \frac{\left(\frac{z^2}{4}\right)^k}{k!(n+k)!}\left(\Phi_k + \Phi_{k+n}\right) \\
&\quad + \frac{1}{2}\left(\frac{z}{2}\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!}\left(-\frac{z^2}{4}\right)^k
\end{aligned}
$$

by using the recurrence formula and the τ-method. Here, when n=0, the last term is zero, γ is the Euler's constant.
$I_n(z)$ is the modified Bessel function of the first kind, and

$$
\Phi_0 = 0
$$

$$
\Phi_k = \sum_{m=1}^{k} \frac{1}{m} \quad (k \geq 1)
$$

Parameters

Z	Input. Independent variable z. Complex variable.
N	Input. Order n of $K_n(z)$.
ZBK	Output. Value of function $K_n(z)$. Complex variable.
ICON ..	Output. Condition code. See Table CBKN-1.

Table CBKN-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	One of the following occurred • $\lvert\mathrm{Re}(z)\rvert > \log(fl_{max})$ • $\lvert\mathrm{Re}(z)\rvert < 0$ and $\lvert\mathrm{Im}(z)\rvert > \log(fl_{max})$ • $\lvert\mathrm{Re}(z)\rvert \geq 0$ and $\lvert\mathrm{Im}(z)\rvert \geq t_{max}$	ZBK=(0.0,0.0)
30000	$\lvert z\rvert = 0.0$	ZBK=(0.0,0.0)

Comments on use

- Subprograms used
  SSL II ..... AMACH, CBIN, MGSSL, ULMAX, UTLIM
  FORTRAN basic functions ... REAL, AIMAG, ABS, IABS, MOD, CSQRT, CEXP, CLOG, FLOAT, CMPLX

- Notes
  - $|z| \neq 0$
  - $|\mathrm{Re}(z)| \leq \log(fl_{max})$
  - When Re(z)>0,
    $|\mathrm{Im}(z)| < t_{max}$
    Otherwise the value of exp(-z) used in the computations cannot be correctly computed.
  - When Re(z)<0, $|\mathrm{Im}(z)| \leq \log(fl_{max})$ is required.

  When Re(z)≥0 and all the values of $K_n(z)$, $K_{n+1}(z)$, $K_{n+2}(z)$, ..., $K_{n+M}(z)$ are required at the same time, first obtain $K_n(z)$ and $K_{n+1}(z)$ by this routine. Then the others can be obtained in the order of $K_{n+2}(z)$, $K_{n+3}(z)$, ..., $K_{n+M}(z)$, by repeating the recurrence formula (See (4.1) in Method). When Re(z)<0, since this is not stable, this subroutine must be called for each required order.

- Example
  The value ov $K_1(1+2i)$ is computed.

```
C      **EXAMPLE**
       COMPLEX Z,ZBK
       Z=(1.0,2.0)
       N=1
       CALL CBKN(Z,N,ZBK,ICON)
       WRITE(6,600) Z,N,ZBK,ICON
       STOP
  600  FORMAT(' ',2F8.2,I6,5X,2E17.7,I7)
       END
```

Method

Since

$K_{-n}(z) = K_n(z)$

when n<0, it is reduced to the case n≥0.
The method for computing $K_n(z)$ depends on whether Re(z)≥0 or Re(z)<0.

- When $|\mathrm{Re}(z)| \geq 0$

  $K_n(z)$ can be computed by the recurrence formula,

$$
K_{n+1}(z) = \frac{2k}{z} K_k(z) + K_{k-1}(z) \qquad (4.1)
$$

$$
k = 1,2,\ldots,n-1
$$

with the starting values $K_0(z)$ and $K_1(z)$
Computational procedures for $K_0(z)$ and $K_1(z)$ depend on the value of z.

- For:
  Single precision: |Im(z)| < -2.25 Re(z) + 4.5 and
  Double precision: |Im(z)| < -4 Re(z) + 8
  $K_0(z)$ and $K_1(z)$ are computed as follows

$$K_0(z) = -\left\{\gamma + \log\left(\frac{z}{2}\right)\right\} I_0(z) + 2\sum_{k=1}^{\infty} \frac{I_{2k}(z)}{k} \qquad (4.2)$$

$$K_1(z) = \left(\frac{1}{z} - I_1(z)K_0(z)\right) \Big/ I_0(z) \qquad (4.3)$$

where, $I_k(z)$ is obtained by the recurrence formula.

- For:
  Single precision: |Im(z)| ≥ -2.5 Re(z) + 4.5 and
  Double precision: |Im(z)| ≥ -4 Re(z) + 8
  The τ-method is used. (See Reference [84]). In this method, $K_n(z)$ is obtained from

$$K_n(z) = \sqrt{\frac{\pi}{2z}} e^{-z} f_n\left(\frac{1}{z}\right) \qquad (4.4)$$

and $f_n(1/z)$ is approximated.
Assume t=1/z, then $f_n(t)$ satisfies

$$t^2 f_n''(t) + 2(t+1) f_n'(t) - \left(n^2 - \frac{1}{4}\right) f_n(t) = 0 \qquad (4.5)$$

Adding on the righthand side of (4.5) τ times a shifted Ultraspherical polynomial orthogonal over the interval [0,η], then

$$t^2 f_n''(t) + 2(t+1) f_n'(t) - \left(n^2 - \frac{1}{4}\right) f_n(t) = \tau C_m^{*(\alpha)}\left(\frac{t}{\eta}\right) \qquad (4.6)$$

where

$$C_m^{*(\alpha)}(t) = \sum_{i=0}^{m} C_{mi}^{*(\alpha)} t^i \qquad (4.7)$$

denotes the shifted Ultraspherical polynomial (when α=0, it is equivalent to the shifted Chebyshev polynomial and when α=0.5, it is equivalent to the shifted Legendre polynomial).
Equation (4.6) contains the solution of the following m-th degree polynomial

$$f_{nm}(t) = \tau \sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)} \sum_{i=0}^{k} a_i t^i}{(k+1)\, a_{k+1} \eta^k} \qquad (4.8)$$

where,

$$\left.\begin{aligned} a_0 &= 1 \\ a_k &= \frac{(4n^2 - 1^2)(4n^2 - 3^2)\cdots(4n^2 - (2k-1)^2)}{k!8^k} (k \geq 1) \end{aligned}\right\} \qquad (4.9)$$

If τ is determined from the initial condition $f_{nm}(0)$=1 (as $t \to 0, f_n(t) \to 1$) we can obtain

$$f_{nm}(t) = \frac{\displaystyle\sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)} \sum_{i=0}^{k} a_i t^i}{(k+1)a_{k+1}\eta^k}}{\displaystyle\sum_{k=0}^{m} \frac{C_{mk}^{*(\alpha)}}{(k+1)a_{k+1}\eta^k}} \qquad (4.10)$$

This equation contains α and η as unknowns. It has been seen when α=0.5 (the shifted Ultraspherical polynomial) and η=t, the highest accuracy can be obtained. (See Reference [84]).
In this case,

$$f_{nm}(t) = \frac{\displaystyle\sum_{k=0}^{m} \frac{P_{mk}^{*} \sum_{i=0}^{k} a_i t^i}{(k+1)a_{k+1}t^k}}{\displaystyle\sum_{k=0}^{m} \frac{P_{mk}^{*}}{(k+1)a_{k+1}t^k}} \qquad (4.11)$$

where P_{mk}^{*} are the coefficients of the shifted Ultraspherical polynomial

$$P_m(t) = \sum_{i=0}^{m} P_{mi}^{*} t^i \qquad (4.12)$$

By multiplying the denominator and numerator by t^m and expressing as powers of t.

$$f_{nm}(t) = \frac{\displaystyle\sum_{i=0}^{m} G_i(m,n) t^i}{\displaystyle\sum_{i=0}^{m} H_i(m,n) t^i} \qquad (4.13)$$

where,

$$G_i(m,n) = \sum_{k=0}^{i} \frac{P_{m,m-i+k}^{*} a_k}{(m+1-i+k)\, a_{m+1-i+k}} \qquad (4.14)$$

$$H_i(m,n) = \frac{P_{m,m-i}}{(m-i+1)\, a_{m-i+1}} \qquad (4.15)$$

Then

$$K_n(z) = \sqrt{\frac{1}{z}} e^{-z} \frac{\displaystyle\sum_{i=0}^{m} \tilde{G}_i(m,n)\left(\frac{1}{z}\right)^i}{\displaystyle\sum_{i=0}^{m} \tilde{H}_i(m,n)\left(\frac{1}{z}\right)^i} \qquad (4.16)$$

can be obtained as a computational expression for $K_n(z)$ from (4.4) and (4.13).
Where

$$\tilde{G}_i(m,n) = G_i(m,n) / G_m(m,n) \qquad (4.17)$$

$$\tilde{H}_i(m,n)=\sqrt{\frac{2}{\pi}}H_i(m,n)/G_m(m,n) \tag{4.18}$$

This subroutine sets m in case n equal 0 or 1 as follows in consideration of efficiency.

Single precision:

when $\lvert z\rvert \geq 17$	, m=3
otherwise	, m=7

Double precision:

when $(\mathrm{Re}(z))^2+0.425(\mathrm{Im}(z))^2 \geq 15^2$	, m=10
otherwise	, m=19

This subroutine contains a table in which constants $\overline{G}_i(m,n)$ and $\overline{H}_i(m,n)$ are stored in the data statement.

- When Re(z)<0
  The cut of $K_n(z)$ is selected on the negative real axis. Therefore, when Im(z)≥0, the relation
  $$K_n(z)=(-1)^n K_n(-z)-\pi i I_n(-z) \tag{4.19}$$
  when Im(z)<0, the relation
  $$K_n(z)=(-1)^n K_n(-z)-\pi i I_n(-z) \tag{4.20}$$

are used, where the value of $K_n(-z)$ can be obtained by using the calculation for Re(z)≥0 mentioned previously, and the value of $I_n(-z)$ can be obtained by subroutine CBIN. Thus the additional computations for $I_n(-z)$ are required when Re(z)<0, as is not the case when Re(z)>0.

B21-15-0202 CBLNC, DCBLNC

Balancing of a complex matrix
CALL CBLNC (ZA, K, N, DV, ICON)

Function

An *n*-order complex matrix ***A*** is balanced by the diagonal similar transformation shown in (1.1).

$$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D} \tag{1.1}$$

Balancing means to almost completely equalize the sum of norm of the *i*-th column (*i* = 1, ..., *n*) and that of the *i*-th row for the transfo rmed complex matrix. The norm of the elements is $\|z\|_1=|x|+|y|$ for complex number $z = x + i \cdot y$. ***D*** is a real diagonal matrix, and $n \geq 1$.

Parameters

ZA...	lnput. Complex matrix ***A***.
	Output. Balanced complex matrix $\tilde{\boldsymbol{A}}$. ZA is a two-dimensional array, ZA (K,N)
K...	Input. Adjustable dimension of array ZA.
N...	Input. Order *n* of complex matrix ***A*** and $\tilde{\boldsymbol{A}}$.
DV...	Output. Scaling factor (Diagonal elements of ***D***)
	One-dimensional array of size *n*.
ICON...	Output. Condition code.
	See Table CBLNC- 1 .

Table CBLNC-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	No balancing.
30000	N < 1 or K < N.	Bypassed

Comments on use

- Subprograms used
  SSL II ... IRADIX, MGSSL
  FORTRAN basic functions ... REAL, AIMAG, ABS

- Notes
  If there are large difference in magnitude of elements in a matrix, the precision of the computed eigen-values and eigenvectors on this matrix may not be computed accurately. This subroutine is used to avoid the adverse effects.
  When each elements of a matrix is nearly the same magnitude, this subroutine performs no transformation. Therefore this subroutine should not be executed .
  This subroutine omits balancing of the column (or row) and the corresponding row (or column) in which all the elements of a column or row except the diagonal elements are zero.
  When obtaining eigenvector *x* of matrix ***A*** , the back transformation of (3.1) must be applied to the eigenvector $\tilde{x}$ of matrix $\tilde{\boldsymbol{A}}$ balanced by this subroutine.

$$\boldsymbol{x} = \boldsymbol{D}\tilde{\boldsymbol{x}} \tag{3.1}$$

  The back transformation of (3.1) can be performed by subroutine CHBK2 (See the selection on CHBK2).

- Example
  After balancing the *n*-order complex matrix ***A***, it is transformed to the complex Hessemberg matrix by subroutine CHES2 and the eigenvalue is obtained by the subroutine CHSQR.
  When $n \leq 100$.

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100)
      DIMENSION DV(100),IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE (6,600) N
      DO 20 I=1,N
   20 WRITE(6,610)(I,J,ZA(I,J),J=1,N)
      CALL CBLNC(ZA,100,N,DV,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL CHES2(ZA,100,N,IP,ICON)
      CALL CHSQR(ZA,100,N,ZE,M,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,630) (ZE(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',
     *       5X,'N=',I3/)
  610 FORMAT(/2(5X,'A(',I3,',',I3,')=',
     *       2E15.7))
  620 FORMAT('0',20X,'ICON=',I5)
  630 FORMAT('0',5X,'EIGENVALUE'/'0',20X,
     *       'REAL',16X,'IMAG'/('0',15X,
     *       2(E15.7, 5X)))
      END
```

Method

An *n*-order complex matrix ***A*** is balanced by performing iterative diagonal similar transformation in (4.1).

$$\boldsymbol{A}_s = \boldsymbol{D}_s^{-1}\boldsymbol{A}_{s-1}\boldsymbol{D}_s \quad , s = 1,2,... \tag{4.1}$$

Where $\boldsymbol{A}_0 = \boldsymbol{A}$ and $\boldsymbol{D}_s$ is a real diagonal matrix expressed in (4.2) and *s* is the number of iterations.

$$D_s = \begin{bmatrix} d_1^{(s)} & & & 0 \\ & d_2^{(s)} & & \\ & & \ddots & \\ 0 & & & d_n^{(s)} \end{bmatrix} \tag{4.2}$$

In the balancing of (4.1), excluding the diagonal element, the sum of the magnitude of elements in the *i*-th row of $\boldsymbol{A}_s$ is made almost equal to that of the magnitude of elements in the *i*-th column.

Assuming $\boldsymbol{A}_s = \left(a_{ij}^{(s)}\right)$, balancing is performed such that

$$\sum_{\substack{k=1 \\ k\neq i}}^{n} \left\| a_{ik}^{(s)} \right\|_1 \approx \sum_{\substack{k=1 \\ k\neq i}}^{n} \left\| a_{ki}^{(s)} \right\|_1 \tag{4.3}$$

should be satisfied.

If $\boldsymbol{D}_{si}$ is defined as shown in (4.4), $\boldsymbol{D}_s$ of (4.2) can be expressed as (4.5).

$$\boldsymbol{D}_{si} = \begin{array}{c} \begin{array}{c} i \end{array} \\ \left[\begin{array}{ccccccc} 1 & & & & & & \\ & \ddots & & & & 0 & \\ & & 1 & & & & \\ & & & d_i^{(s)} & & & \\ & & & & 1 & & \\ & 0 & & & & \ddots & \\ & & & & & & 1 \end{array}\right] i \end{array} \tag{4.4}$$

$$\boldsymbol{D}_s = \boldsymbol{D}_{s1}\boldsymbol{D}_{s2} \cdots \boldsymbol{D}_{sn} \tag{4.5}$$

From (4.5), (4.1) can be transformed by sequentially performing transformation of (4.6) for i = l, 2, ..., n.

$$\boldsymbol{A}_{si} = \boldsymbol{D}_{si}^{-1} \boldsymbol{A}_{si-1} \boldsymbol{D}_{si} \tag{4.6}$$

where,

Diagonal element $a_i^{(s)}$ of $\boldsymbol{D}_{si}$ is defined so that the transformed *i*-th column and corresponding row satisfy (4.3). If they satisfy (4.3) before transformation, $d_i^{(s)} = 1$, that is $\boldsymbol{D}_{si} = \boldsymbol{I}$.

Iteration of (4.1) terminates when (4.3) is satisfied for all column and rows.

This subroutine executes this operation in the following steps:

1) The sum of norms of each element in the *i*-th column and row is computed excluding the diagonal element.

$$C = \sum_{\substack{k=1 \\ k\neq i}}^{n} \left\| a_{ki} \right\|_1 \tag{4.7}$$

$$R = \sum_{\substack{k=1 \\ k\neq i}}^{n} \left\| a_{ik} \right\|_1 \tag{4.8}$$

2) $d_i^{(s)}$ is defined as,

$$d_i^{(s)} = \rho^k \tag{4.9}$$

where $\rho = \begin{cases} 2 & \text{for binary digits} \\ 16 & \text{for hexadecimal digits} \end{cases}$

k is defined to satisfy the following condition.

$$R \cdot \rho > C \cdot \rho^{2k} \geq R/\rho \tag{4.10}$$

From (4.10), $k > 0$ when $C < R/\rho$ and $k \leq 0$ when $C \geq R/\rho$.

3) By the condition shown in (4.11), whether or not transformation is required is determined.

$$\left(C \cdot \rho^{2k} + R\right)/\rho^k < 0.95(C + R) \tag{4.11}$$

When (4.11) is satisfied, transformation is performed where $d_i^{(s)} = \rho^k$ and if not satisfied, transformation is bypassed.

4) When transformation has been performed for all columns and rows, the balancing process terminates.

Then, the diagonal elements of $\boldsymbol{D}$ shown in (4.12) are stored as the scaling factor in array DV.

$$\boldsymbol{D} = \boldsymbol{D}_1\boldsymbol{D}_2 \cdots \boldsymbol{D}_s \tag{4.12}$$

For details, see Reference [13] pp.315 - 326.

I11-82-1401 CBYN, DCBYN

Integer order Bessel function of the second kind $Y_n(z)$ with complex variable
CALL CBYN (Z, N, ZBY, ICON)

Function

This subroutine computes the value of integer order Bessel function of the second kind with complex variable.

$$
\begin{aligned}
Y_n(z) &= (-1)^n Y_{-n}(z) \\
&= \frac{2}{\pi}\left\{\gamma + \log\left(\frac{z}{2}\right)\right\} J_n(z) \\
&\quad - \frac{1}{\pi}\left(\frac{z}{2}\right)^n \sum_{k=0}^{\infty} \frac{\left(-\frac{z^2}{4}\right)^k}{k!(n+k)!}(\Phi_k + \Phi_{k+n}) \\
&\quad - \frac{1}{\pi}\left(\frac{z}{2}\right)^{-n} \sum_{k=0}^{n-1} \frac{(n-k-1)!}{k!}\left(\frac{z^2}{4}\right)^k
\end{aligned}
$$

by using the recurrence formula and the τ-method. In the definition when $n = 0$, the last item is zero, γ denotes the Euler's constant $J_n(z)$ is Bessel function of the second kind and

$$
\begin{aligned}
\Phi_0 &= 0 \\
\Phi_k &= \sum_{m=1}^{k} \frac{1}{m} \quad (k \geq 1)
\end{aligned}
$$

Parameters

Z ...	Input. Independent variable z. Complex variable.
N ...	Input. Order n of $Y_n(z)$.
ZBY ...	Output. Value of function $Y_n(z)$. Complex variable.
ICON ...	Output. Condition code. See Table CBYN-1.

Table CBYN-1 Condition codes

Code	Meaning	Proceccisng
0	No error	
20000	$\lvert\mathrm{Re}(z)\rvert > \log(fl_{\max})$ or $\lvert\mathrm{Im}(z)\rvert > \log(fl_{\max})$	ZBY = (0.0, 0.0)
30000	$\lvert z\rvert$ = 0.0	ZBY = (0.0, 0.0)

Comments on use

- Subprograms used
  SSL II ... AMACH, CBIN, CBKN, MGSSL, ULMAX, UTLIM
  FORTRAN basic functions ... REAL, AIMAG, IABS, CONJG, CMPLX, MOD

- Notes
  – $|z| \neq 0$
  – $|\mathrm{Re}(z)| \leq \log(fl_{max})$ and $|\mathrm{Im}(z)| \leq \log(fl_{max})$
  – When all the values of $Y_n(z)$, $Y_{n+1}(z)$, $Y_{n+2}(z)$, ..., $Y_{n+M}(z)$ are required at a time, the procedure to be mentioned below under Method is most recommendable.

- Example
  The value of $Y_1(1+2i)$ is computed.

```
C       **EXAMPLE**
        COMPLEX Z,ZBY
        Z=(1.0,2.0)
        N=1
        CALL CBYN(Z,N,ZBY,ICON)
        WRITE(6,600)Z,N,ZBY,ICON
        STOP
  600   FORMAT(' ',2F8.2,I6,5X,2E17.7,I7)
        END
```

Method

Because we know

$$Y_{-n}(z) = (-1)^n Y_n(z) \tag{4.1}$$

the computation for $n < 0$ can be reduced to that for $n > 0$

Also, because

$$Y_n(\bar{z}) = \overline{Y_n(z)} \tag{4.2}$$

when Im $(z) < 0$, this is reduced to that wen Im $(z) \geq 0$. $Y_n(z)$ is computed using the relation.

$$Y_n(z) = i^{n+1} I_n(-iz) - \frac{2}{\pi} i^n (-1)^n K_n(-iz) \quad , \; i = \sqrt{-1} \tag{4.3}$$

where, the value of $I_n(-iz)$ is computed by subroutine CBIN which uses the recurrence formula, and the value of $K_n(-iz)$ is computed by subroutine CBKN which uses the recurrence formula and τ-method.

When all the values of $Y_n(z)$, $Y_{n+1}(z)$, $Y_{n+2}(z)$, ..., $Y_{n+M}(z)$ are required at a time, they are obtained efficiently in the way indicated below. First the value of $I_{n+M}(-iz)$ and $I_{n+M-1}(-iz)$ are obtained by CBIN, and then repeating the recurrence formula, the values are obtained sequentially in the order of the highest value first until $I_n(-iz)$, is obtained. $K_n(-iz)$ and $K_{n+1}(-iz)$ are obtained by CBKN and then repeating the recurrence formula the values are obtained sequentially in the order of the lowest value first until $K_{n+M}(z)$ is obtained.

Then, by the relation in (4.3), the required computation is done.

B21-15-0101 CEIG2, DCEIG2

Eigenvalues and corresponding engenvectors of a complex matrix (QR method)
CALL CEIG2 (ZA, K, N, MODE, ZE, ZEV, VW, IVW, ICON)

Function

This subroutine computes the eigenvalues and the corresponding eigenvectors of an n-order complex matrix $\boldsymbol{A}$. The eigenvectors are normalized so that $\|x\|_2=1$. $n \geq 1$.

Parameters

ZA ...	Input. Complex matrix $\boldsymbol{A}$. ZA is a complex two-dimensional array, ZA (K, N) The contents of ZA are altered on output.
K ...	Input. Adjustable dimension ($\geq n$) of arrays ZA and ZEV.
N ...	Input. Order n of complex matrix $\boldsymbol{A}$
MODE ...	Input. Specifies whether or not balancing is required. When MODE = 1, balancing is omitted. When MODE $\neq$ 1, balancing is included.
ZE ...	Output. Eingenvalues. Complex two-dimensional array of size n.
ZEV ...	Output. Eigenvectors. The eigenvectors are stored in the rows corresponding to the eigenvalues. ZEV is a complex two-dimensional array, ZEV (K, N)
VW ...	Work area. One-dimensional array of size n.
IVW ...	Work area. One-dimensional array of size n.
ICON ...	Output. Condition code. See Table CEIG2-1.

Table CEIG2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	ZE (1) = ZA (1,1) ZEV (1) = (1.0, 0.0)
20000	The eigenvalues and eigenvectors could not be determined since reduction to trangular matrix was not possible.	Discontinued
30000	N < 1 or K < N.	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, CBLNC, CHES2, CSUM, CNRML, IRADIX, MGSSL
  FORTRAN basic functions ... ABS, REAL, AIMAG, AMAX1, CSQRT, SIGN, SQRT, CONJG

- Notes
  When the magnitude in each element of a complex matrix varies greatly, the precision of the results can be improved by valancing the matrix with subroutine CBLNC. When the magnitude in each element of a matrix is about the same, balancing will produce minimal improvement. In this state, by specifying MODE = 1, the balancing procedure should be skipped.
  This subroutine obtains all eigenvalues and eigenvectors of a complex matrix.
  When only eigenvalues are required, they must be obtained by subroutines CBLNC, CHES2 and CHSQR.
  When a subset of eigenvectors is required, they must be obtained by subroutines CBLNC, CHES2, CHSQR, CHVEC, CHBK2, CNRML.

- Example
  All eigenvalues and eigenvectors of an n-order complex matrix $\boldsymbol{A}$ are determined. $n \leq 100$.

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100),
     *        ZEV(100,100)
      DIMENSION IND(100),VW(100),IVW(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(I,J),J=1,N)
      CALL CEIG2(ZA,100,N,0,ZE,ZEV,VW,
     *           IVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      DO 30 I=1,N
   30 IND(I)=1
      CALL CEPRT(ZE,ZEV,100,N,IND,N)
      GO TO 10
  500 FORMAT(I3)
  510 FORMAT(4E15.7)
  600 FORMAT('0',5X,'ORIGINAL MATRIX',
     *5X,'N=',I3)
  610 FORMAT(/2(5X,'A(',I3,',',I3,')=',
     *2E15.7))
  620 FORMAT('0',5X,'ICON=',I5)
      END
```

In the above example, subroutine CEPRT prints all the eigenvalues and eigenvectors of a complex matrix. The contents are:

```
      SUBROUTINE CEPRT(ZE,ZEV,K,N,IND,M)
      COMPLEX ZE(M),ZEV(K,M)
      DIMENSION IND(M)
      WRITE(6,600)
      MM=0
      DO 20 J=1,M
      IF(IND(J).EQ.0) GO TO 20
      MM=MM+1
      ZE(MM)=ZE(J)
      DO 10 I=1,N
      ZEV(I,MM)=ZEV(I,J)
   10 CONTINUE
   20 CONTINUE
```

```
      IF(MM.EQ.0) GO TO 50
      DO 40 INT=1,MM,3
      LST=MIN0(INT+2,MM)
      WRITE(6,610) (J,J=INT,LST)
      WRITE(6,620) (ZE(J),J=INT,LST)
      DO 30 I=1,N
      WRITE(6,630) (ZEV(I,J),J=INT,LST)
   30 CONTINUE
   40 CONTINUE
   50 RETURN
  600 FORMAT('1',30X,'**EIGENVECTORS**')
  610 FORMAT('0',26X,I3,29X,I3,29X,I3)
  620 FORMAT('0',' EIGENVALUES',
     *3(2X,2E15.7))
  630 FORMAT(12X,3(2X,2E15.7))
      END
```

Method

This subroutine determines the eigenvalues and the correspondng eigenvectors of an n-order complex matrix $\boldsymbol{A}$.

The eigenvalues of an n-order complex matrix are determined as diagonal elements of upper triangle matrix $\boldsymbol{R}$ by processing the following three steps:

- An complex matrix $\boldsymbol{A}$ is balanced by the diagonal similar transformation,

$$\tilde{\boldsymbol{A}} = \boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{B} \qquad (4.1)$$

  where $\boldsymbol{B}$ is the diagonal matrix whose element is a scaling factor. For details, refer to subroutine CBLNC.

- The complex matrix $\tilde{\boldsymbol{A}}$ is reduced by the stabilized elementary transformation into the complex Hessemberg matrix $\boldsymbol{H}$.

$$\boldsymbol{H} = \boldsymbol{S}^{-1}\tilde{\boldsymbol{A}}\boldsymbol{S} \qquad (4.2)$$

  where $\boldsymbol{S}$ is obtained by the product of transformation matrices $\boldsymbol{S}_1$, $\boldsymbol{S}_2$, ..., $\boldsymbol{S}_{n-2}$,

$$\boldsymbol{S} = \boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2} \qquad (4.3)$$

  and each $\boldsymbol{S}_i$ can be obtained by permutation matrix $\boldsymbol{P}_i$ and elimination matrix $\boldsymbol{N}_i$ as

$$\boldsymbol{S}_i = \boldsymbol{P}_i\boldsymbol{N}_i^{-1}, i = 1,2,...,n-2 \qquad (4.4)$$

  For details, refer to subroutine CHES2.

- The complex Hessemberg matrix $\boldsymbol{H}$ is reduced by the complex QR method into the complex upper triangle matrix $\boldsymbol{R}$.

$$\boldsymbol{R} = \boldsymbol{Q}_R^*\boldsymbol{H}\boldsymbol{Q}_R \qquad (4.5)$$

  Where $\boldsymbol{Q}_R$ is an unitary matrix given by,

$$\boldsymbol{Q}_R = \boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q}_L \qquad (4.6)$$

which is the product of transformation matrix $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, ..., $\boldsymbol{Q}_L$ used in the complex QR method.

For details, refer to subroutine CHSQR.

The eigenvectors can be obtained as column vectors in matrix $\boldsymbol{X}$ obtained by (4.8) if matrix $\boldsymbol{F}$ which transforms upper triangle matrix $\boldsymbol{R}$ into a diagonal matrix $\boldsymbol{D}$ by a similarity transformation (4.7) is available.

$$\boldsymbol{D} = \boldsymbol{F}^{-1}\boldsymbol{R}\boldsymbol{F} \qquad (4.7)$$

$$\boldsymbol{X} = \boldsymbol{B}\boldsymbol{S}\boldsymbol{Q}_R\boldsymbol{F} \qquad (4.8)$$

To verify that column vectors of matrix $\boldsymbol{X}$ given by (4.8) are the eigenvectors of matrix $\boldsymbol{A}$, substitute (4.1), (4.2) and (4.5) to obtain (4.7).

$$\boldsymbol{D} = \boldsymbol{F}^{-1}\boldsymbol{Q}_R^*\boldsymbol{S}^{-1}\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{B}\boldsymbol{S}\boldsymbol{Q}_R\boldsymbol{F} = \boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X} \qquad (4.9)$$

If $\boldsymbol{BSQ_R}$ are represented as $\boldsymbol{Q}$, from (4.3) and (4.6)

$$\boldsymbol{Q} = \boldsymbol{B}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}\boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q}_L \qquad (4.10)$$

As shown in (4.10), $\boldsymbol{Q}$ can be computed by sequentially taking the product of the transformation matrices.

$\boldsymbol{F}$ can be determined as a unit upper triangular matrix. From (4.7),

$$\boldsymbol{F}\boldsymbol{D} = \boldsymbol{R}\boldsymbol{F} \qquad (4.11)$$

Let the elements of $\boldsymbol{D}$, $\boldsymbol{R}$, and $\boldsymbol{F}$ be represented as $\boldsymbol{D}$=diag(λ_i), $\boldsymbol{R}$=(γ_{ij}) and $\boldsymbol{F}$=(f_{ij}) respectively, then elements f_{ij} can be obtained from (4.11) for j=n, n-1, ..., 2 as follows

$$f_{ij} = \sum_{k=i+1}^{j} \gamma_{ik} f_{kj} \Big/ \left(\lambda_j - \lambda_i\right) \quad i = j-1, j-2, ..., 1 \qquad (4.12)$$

where,

$$\gamma_{ij} = 0 (i > j), \quad \gamma_{ii} = \lambda_i$$
$$f_{ij} = 0 (i > j), \quad f_{ii} = 1$$

If λ_i=λ_j , f_{ij} is obtained as follows.

$$f_{ij} = \sum_{k=i+1}^{j} \gamma_{ik} f_{kj} \Big/ \left(u\|\boldsymbol{A}\|_\infty\right) \qquad (4.13)$$

where u is a unit round-ff and $\|\boldsymbol{A}\|_\infty$ is a norm defined by

$$\|\boldsymbol{A}\|_\infty = \max_j \sum_{i=1}^{n} \|a_{ij}\|_1 \qquad (4.14)$$

where $\boldsymbol{A}$=(a_{ij}).

Therefore, norm $\|z\|_1$ for complex number $z = x + iy$ is defined by

$$\|z\|_1 = |x| + |y|$$

For details, see References [12] and [13] pp.372 - 395.

I11-11-0101 CELI1, DCELI1

Complete elliptic integral of the first kind $K(x)$
CALL CELIl (X, CELI, ICON)

Function

This subroutine computes the complete elliptic integral of the first kind

$$K(x) = \int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{1 - x\sin^2\theta}}$$

using an approximation formula for $0 \le x < 1$.

Parameter

X	Input. Independent variable x
CELI ...	Output. Function value $K(x)$.
ICON ...	Output. Condition code. See Table CELI1-1.

Table CELI1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	X < 0 or X ≥ 1	CELI is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... DLOG

- Example
  The following example generates a table of the function values from 0.00 to 1.00 with increment 0.01.

```
C       **EXAMPLE**
        WRITE(6,600)
        DO 10 K=1,100
        A=K-1
        X=A/100.0
        CALL CELI1(X,CELI,ICON)
        IF(ICON.EQ.0)WRITE(6,610)X,CELI
        IF(ICON.NE.0)WRITE(6,620)X,CELI,ICON
   10 CONTINUE
        STOP
  600 FORMAT('1','EXAMPLE OF COMPLETE ',
       *'ELLIPTIC INTEGRAL OF THE FIRST ',
       *'KIND'///6X,'X',9X,'K(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
       *E17.7,5X,'CELI=',E17.7,5X,
       *'CONDITION=',I10)
        END
```

Method

For $0 \le x < 1$, the complete elliptic integral of the first kind $K(x)$ is calculated using the following approximations.

- Single precision:

$$K(x) = \sum_{k=0}^{4} a_k t^k - \log(t) \sum_{k=0}^{4} b_k t^k \qquad (4.1)$$

where $t = 1 - x$

Theoretical precision = 7.87 digits

- Double precision:

$$K(x) = \sum_{k=0}^{10} a_k t^k - \log(t) \sum_{k=0}^{10} b_k t^k \qquad (4.2)$$

where $t = 1 - x$

Theoretical precision = 17.45 digits

For further information, see Reference [78] pp.150 ~ 154.

I11-11-0201 CELI2, DCELI2

Complete elliptic integral of the second kind $E(x)$
CALL CELI2 (X, CELI, ICON)

Function

This subroutine computes the complete elliptic integral of the second kind

$$E(x)=\int_0^{\frac{\pi}{2}}\sqrt{1-x\sin^2\theta}\,d\theta$$

using an approximation formula for $0 \le x \le 1$.

Parameters

X .... Input. Independent variable x.
CELI .... Output. Function value $E(x)$.
ICON ... Output. Condition code. See Table CELI2-1.

Table CELI2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	X < 0 or X > 1	CELI is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... DLOG

- Example
  The following example generates a table of the function values from 0.00 to 1.00 with increment 0.01.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       A=K-1
       X=A/100.0
       CALL CELI2(X,CELI,ICON)
       IF(ICON.EQ.0)WRITE(6,610)X,CELI
       IF(ICON.NE.0)WRITE(6,620)X,CELI,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF COMPLETE ',
      *'ELLIPTIC INTEGRAL OF THE SECOND ',
      *'KIND'///6X,'X',9X,'E(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'CELI=',E17.7,5X,
      *'CONDITION=',I10)
       END
```

Method

For $0 \le x \le 1$, the value of complete elliptic integral of the second kind $E(x)$ is calculated using the following approximations.

- Single precision:

$$E(x)=\sum_{k=0}^{4}a_k t^k-\log(t)\sum_{k=1}^{4}b_k t^k \qquad (4.1)$$

where $t = 1 - x$

Theoretical precision = 7.80 digits

- Double precision:

$$E(x)=\sum_{k=0}^{10}a_k t^k-\log(t)\sum_{k=1}^{10}b_k t^k \qquad (4.2)$$

where $t = 1 - x$

Theoretical precision = 17.42 digits

However, when x =1, $E(x)$ is set to 1.
For more information, see Reference [78] pp.150 ~ 154.

I11-51-0201 CFRI, DCFRI

Cosine Fresnel integral $C(x)$
CALL CFRI (X, CF, ICON)

Function

This subroutine computes Cosine Fresnel integral,

$$C(x)=\frac{1}{\sqrt{2\pi}}\int_0^x \frac{\cos(t)}{\sqrt{t}}dt=\int_0^{\sqrt{\frac{2}{\pi}x}}\cos\left(\frac{\pi}{2}t^2\right)dt$$

by series and asymptotic expansion, where $x \geq 0$.

Parameters

X	Input. Independent variable x .
CF	Output. Value of $C(x)$.
ICON..	Output. condition codes. See Table CFRI-1.

Table CFRI-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X ≥ t_{max}	CF = 0.5
30000	X<0	CF = 0.0

Comments on use

- Subprograms used
  SSL II ... MGSSL, UTLIM
  FORTRAN basic functions ... SIN, COS, and SQRT.

- Notes
  Teh valid range of parameter X are:
  $0 \leq X < t_{max}$
  This is provided because sin(x) and cos(x) lose their accuracy if X exceeds the above ranges.

- Example
  The following example generates a table of $C(x)$ from 0.0 to 100.0 with increment 1.0.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       X=K-1
       CALL CFRI(X,CF,ICON)
       IF(ICON.EQ.0)WRITE(6,610)X,CF
       IF(ICON.NE.0)WRITE(6,620)X,CF,ICON
   10 CONTINUE
       STOP
  600 FORMAT('1','EXAMPLE OF FRESNEL ',
     *'INTEGRAL'///6X,'X',9X,'C(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     *E17.7,5X,'C=',E17.7,5X,'CONDITION=',
     *I10)
       END
```

Method

Two different approximation formulas are used depending on the ranges of $x$divided at $x = 4$.

- For $0 \leq x < 4$
  The power series expansion of $C(x)$

$$C(x)=\sqrt{\frac{2}{\pi}x}\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!(4n+1)}x^{2n} \tag{4.1}$$

is calculated with the following approximation formulas:
Single precision:

$$C(x)=\sqrt{x}\sum_{k=0}^{7}a_k z^{2k},\ z=x/4 \tag{4.2}$$

Double precision:

$$C(x)=\sqrt{x}\sum_{k=0}^{12}a_k x^{2k} \tag{4.3}$$

- For $x \geq 4$
  The asymptotic expansion of $C(x)$

$$C(x)=\frac{1}{2}+\sin(x)P(x)+\cos(x)Q(x) \tag{4.4}$$

is calculated through use of the following approximate expression for $P(x)$ and $Q(x)$:
Single precision:

$$P(x)=\frac{2}{\sqrt{x}}\sum_{k=0}^{11}a_k z^k, z=4/x \tag{4.5}$$

$$Q(x)=\frac{2}{\sqrt{x}}\sum_{k=0}^{10}b_k z^{k+1}, z=4/x \tag{4.6}$$

Double precision:

$$P(x)=\frac{-1}{\sqrt{x}}\sum_{k=0}^{10}a_k z^k \Big/ \sum_{k=0}^{10}b_k z^k, z=4/x \tag{4.7}$$

$$Q(x)=\frac{1}{\sqrt{x}}\sum_{k=0}^{10}c_k z^{k+1} \Big/ \sum_{k=0}^{11}d_k z^k, z=4/x \tag{4.8}$$

F12-15-0101 CFT, DCFT

Multi-variate discrete complex Fourier transform (radix 8 and 2 FFT)
CALL CFT (A, B, N, M, ISN, ICON)

Function

When M -variate (where the dimension of each variable is $N1$, $N2$, ..., NM) complex time series data $\{x_{J1,...,JM}\}$ is given, this subroutine performs a discrete complex Fourier transform or its inverse transform using the Fast Fourier Transform (FFT) method. $N1$, $N2$, ..., NM must be equal to 2^l each (where l = 0 or positive integer), and $M \geq 1$.

- Fourier transform
  When $\{x_{J1,...,JM}\}$ is input, this subroutine determines $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ by performing the transform defined in (1.1).

$$N1 \cdots NM\alpha_{K1,...,KM} = \sum_{J1=0}^{N1-1} \cdots \sum_{JM=0}^{NM-1} x_{J1,...,JM}\,\omega_1^{-J1.K1} \cdots \omega_M^{-JM.KM}$$
$$,K1 = 0,1,..., N1-1$$
$$:$$
$$,KM = 0,1,..., NM-1 \qquad (1.1)$$
$$,\omega_1 = \exp(2\pi i/N1)$$
$$:$$
$$,\omega_M = \exp(2\pi i/NM)$$

- Inverse Fourier transform
  When $\{\alpha_{K1,...,KM}\}$ is input, this subroutine determines $\{x_{J1,...,JM}\}$ by performing the inverse transform defined in (1 .2) .

$$x_{J1,...,JM} = \sum_{K1=0}^{N1-1} \cdots \sum_{KM=0}^{NM-1} \alpha_{K1,...,KM} \cdot \omega_1^{J1.K1} \cdots \omega_M^{JM \cdot KM}$$
$$,J1 = 0,1, ..., N1-1$$
$$:$$
$$,JM = 0,1, ..., NM-1 \qquad (1.2)$$
$$,\omega_1 = \exp(2\pi i/N1)$$
$$:$$
$$,\omega_M = \exp(2\pi i/NM)$$

Parameters

A ..... Input. Real parts of $\{x_{J1,...,JM}\}$ or $\{\alpha_{K1,...,KM}\}$.
Output. Real parts of $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$.
M-dimensional array. (See "Notes".)

B ..... Input. Imaginary parts of $\{x_{J1,...,JM}\}$ or $\{\alpha_{K1,...,KM}\}$.
Output. Imaginay parts of $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$.
M-dimensional array.
(See "Notes".)

N ..... Input. The dimensions of the M-variate transform are specified as N (1) = $N1$, N (2) = $N2$, ..., N(M) = NM.
N is a one-dimensional array of size M.

M ... Input. Number (M) of variables.

ISN ... Input. Specifies normal or inverse transform ($\neq 0$).
Transform: ISN = +1
Inverse transform: ISN = –1
(See "Notes".)

ICON .. Output. Condition code.
See Table CFT-1.

Table CFT-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M < 1, ISN = 0 or either of N1, N2, ..., or NM is not 2^l (l = 0 or positive integer)	Aborted

Comments on use

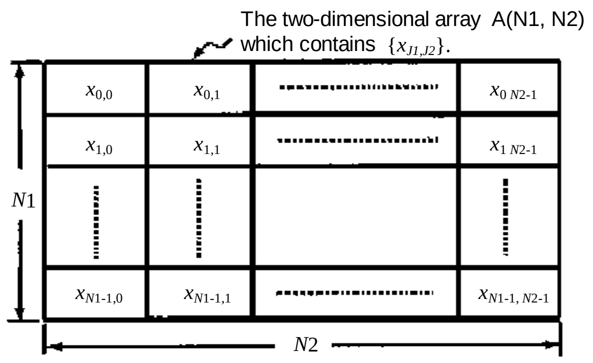
- Subprograms used
  SSL II ... CFTN, PNR, and MGSSL
  FORTRAN basic functions ... ATAN, ALOG, SQRT, SIN, and IABS

- Notes
  General definition of discrete complex Fourier transform:
  Multi-variate discrete complex Fourier transform and inverse Fourier transform are generally defined as:

$$\alpha_{K1,...,KM} = \frac{1}{N1 \cdots NM} \sum_{J1=0}^{N1-1} \cdots \sum_{JM=0}^{NM-1} x_{J1,...,JM}\,\omega_1^{-J1 \cdot K1} \cdots \omega_M^{-JM \cdot KM} \qquad (3.1)$$

$$x_{J1,...,JM} = \sum_{K1=0}^{N1-1} \cdots \sum_{KM=0}^{NM-1} \alpha_{K1,...,KM}\,\omega_1^{J1 \cdot K1} \cdots \omega_M^{JM \cdot KM} \qquad (3.2)$$

$K1,..., KM, J1,..., JM, \omega_1,...,\omega_M$ are defined in (1.1) and (1.2). This subroutine determines $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$ in place of $\{\alpha_{K1,...,JM}\}$ of (3.1) or $\{x_{J1,...,JM}\}$ of (3.2). Scaling of the resultant values is left to the user. Notice that a normal transform followed by an iverse transform returns the original data multipled by the value $N1 \cdots NM$.
Data storage:
User must store the real parts of input $\{x_{j1,...,JM}\}$ in M-dimensional array A as shown in Fig. CFT-1, and store the imaginary parts in M-dimensional array B in the same way. On output, this subroutine stores $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$ in this manner.



Example of a two-varitate transform (M=2)

Fig. CFT-1 Storage of $\{x_{J1,...,JM}\}$

In general, when performing M-variate -variate transform, if the data sequence is the same as given in FORTRAN for an M-dimensional array, parameters A and B can each be a one-dimensional array.
Specifying ISN:
ISN is used to specify normal or inverse transform. It is also used as follows: If the real and imaginary parts of $\{x_{J1,..., JM}\}$ or $\{\alpha_{K1,..., KM}\}$ are each stored with an interval I, the ISN parameter is speified as follows:

Transform : ISN = +I

Inverse transform: ISN = –I

In this case, the results of transform are also stored in intervals of I.

- Examples
  (a) 1-variable transform
  Complex time series data $\{x_{J1}\}$ of dimensional $N1$ is put, and the result $\{N1\alpha_{K1}\}$ is determined using this routine. In case of $Nl \leq 1024$ (= 2^{10})

```
C       **EXAMPLE**
        DIMENSION A(1024),B(1024)
        READ(5,500) N,(A(I),B(I),I=1,N)
        WRITE(6,600) N,(I,A(I),B(I),I=1,N)
        CALL CFT(A,B,N,1,1,ICON)
        WRITE(6,610) ICON
        IF(ICON.NE.0) STOP
        WRITE(6,620) (I,A(I),B(I),I=1,N)
        STOP
    500 FORMAT(I5/(2E20.7))
    600 FORMAT('0',10X,'INPUT DATA N=',I5/
       *(15X,I5,2E20.7))
    610 FORMAT('0',10X,'RESULT ICON=',I5)
    620 FORMAT(15X,I5,2E20.7)
        END
```

  (b) 2-variate transform
  Complex time series data $\{x_{J1,J2,J3}\}$ of dimension Nl, $N2$ and $N3$ is put, a Fourier transform is performed, and then by performing an inverse transform on the results, $\{x_{J1,J2,J3}\}$ is determined. Here $Nl = 2$, $N2 = 4$ and $N3 = 4$.

```
C       **EXAMPLE**
        DIMENSION A(2,4,4),B(2,4,4),N(3)
        DATA N/2,4,4/
        READ(5,500)(((A(I,J,K),B(I,J,K),
       *            I=1,2),J=1,4),K=1,4)
        WRITE(6,600)N,(((I,J,K,
       *            A(I,J,K),B(I,J,K),
       *            I=1,2),J=1,4),K=1,4)
C       NORMAL TRANSFORM
        CALL CFT(A,B,N,3,1,ICON)
        IF(ICON.NE.0)STOP
C       INVERSE TRANSFORM
        CALL CFT(A,B,N,3,-1,ICON)
        IF(ICON.NE.0)STOP
        NT=N(1)*N(2)*N(3)
        DO 10 I=1,2
        DO 10 J=1,4
        DO 10 K=1,4
        A(I,J,K)=A(I,J,K)/FLOAT(NT)
        B(I,J,K)=B(I,J,K)/FLOAT(NT)
     10 CONTINUE
        WRITE(6,610)(((I,J,K,
       *            A(I,J,K),B(I,J,K),
       *            I=1,2),J=1,4),K=1,4)
        STOP
    500 FORMAT(8E5.0)
    600 FORMAT('0',10X,'INPUT DATA',5X,
       *'(',I3,',',I3,',',I3,')'/
       *(15X,'(',I3,',',I3,',',I3,')',2E20.7))
    610 FORMAT('0',10X,'OUTPUT DATA'/
       *(15X,'(',I3,',',I3,',',I3,')',2E20.7))
        END
```

Method

This subroutine performs a multi-variate discrete complex Fourier transform using the radix 8 and 2 Fast Fourier Transform (FFT) method.

- Multi-variate transform
  The multi-variate transform defined in (3.1) can be reduced to simpler form by rearranging common terms.
  For example, the two-variate transform can be reduced to as shown in (4.1).

$$\alpha_{K1,K2} = \sum_{J1=0}^{N1-1} \omega_1^{-J1,K1} \sum_{J2=0}^{N2-1} x_{J1,J2} \omega_2^{-J2,K2} \quad (4.1)$$

In (4.1), the scaling factor $1/Nl \cdot N2$ is omitted. $\sum_{J2}$ of (4.1) is performed on Nl groups 1-variable transforms of dimension $N2$ with respect to $J1$. Then based on the results, $\sum_{J1}$ is performed on $N2$ groups l-variable transforms of dimension $N1$ with respect to $J2$.
In the same way, a mult-variate discrete complex Fourier transform is achieved by performing 1 -variable transforms on complex number groups in each variable.

In this routine, the 8 and 2 radix Fast Fourier Transform is used to perform 1 -variable transforms on each variable.

- Principles of the Fast-Fourier Transform (FFT) method 1-variable discrete complex Fourier transform is defined as

$$\alpha_k = \sum_{j=0}^{n-1} x_j \omega^{-jk} \quad ,k = 0,1,\ldots,n-1 \qquad ,\omega = \exp(2\pi i/n) \tag{4.2}$$

In (4.2), the scaling factor 1/ n is omitted.
If (4.2) is calculated directly, n^{2k} complex multiplications will be required. While, when (4.2) is calculated by the FFT method, if n can be factored into $r \cdot r$, the number of multiplications is reduced to the order of 2 nr by taking account of the characteristics of the exponential function ω^{-jk}. This is illustrated below.
Since k and j of (4.2) can be expressed as

$$\left.\begin{aligned} j &= j_0 + r \cdot j_1 \quad ,0 \le j_0 \le r-1,\ 0 \le j_1 \le r-1 \\ k &= k_0 + r \cdot k_1 \quad ,0 \le k_0 \le r-1,\ 0 \le k_1 \le r-1 \end{aligned}\right\} \tag{4.3}$$

when (4.3) is substituted in (4.2), it results

$$\alpha_{k_0+r\cdot k_1} = \sum_{j_0=0}^{r-1}\sum_{j_1=0}^{r-1} x_{j_0+r\cdot j_1} \cdot \exp\left\{-2\pi i \frac{(j_0 + r\cdot j_1)(k_0 + r\cdot k_1)}{r^2}\right\} \tag{4.4}$$

If the right side of (4.4) is re-organized according to j_0 and j_1 and common terms are rearranged, we obtain

$$\alpha_{k_0+r\cdot k_1} = \sum_{j_0=0}^{r-1} \exp\left(-2\pi i \frac{j_0 \cdot k_1}{r}\right) \cdot \exp\left\{-2\pi i \frac{j_0 \cdot k_1}{r^2}\right\} \sum_{j_1=0}^{r-1} x_{j_0+r\cdot j_1} \exp\left(-2\pi i \frac{j_1 \cdot k_0}{r}\right) \tag{4.5}$$

In calculating (4.5) , each of $\sum\limits_{J0}$ and $\sum\limits_{J1}$ performs r sets of elementary Fourier transforms of dimension r and $\exp\left(-2\pi i \frac{j_0 \cdot k_0}{r^2}\right)$ is the rotation factor for the results of $\sum\limits_{j1}$.
Therefore, the number of multiplications involved in the calculation of (4.5) is as shown in (4.6); when n is large, the calculation load decreases.

$$\begin{aligned} C_n &= r \cdot (r^2) + r \cdot (r^2) + (r-1)(r-1) \\ &= 2nr + (r-1)^2 \end{aligned} \tag{4.6}$$

If r is factored into smaller numbers, the calculation efficiency can be further increased.

See the specific example given in the section CFTN. For further information, refer to References [55], [56], and [57].

F12-11-0101 CFTM, DCFTM

Multi-variate discrete complex Fourier transform (mixed radix FFT)
CALL CFTM (A, B, N, M, ISN, ICON)

Function

When *M*-variate (where the dimenstion of each variable is *N*1, *N*2, ..., *NM*) complex time series data $\{x_{J1,...,JM}\}$ is given. This subroutine performs a discrete complex Fourier transforms or its inverse transform by using the Fast Fourier Transform (FFT). The dimension of each variable must be 1 or satisfy the following conditions:

- It must be expressed by a product of prime factors *p* (*p* = {4, 3, 5, 7,11,13,17,19,23,2}). (The same prime factor can be duplicated.)
- The maximum number of prime factors used must be eleven.
- The product of the square free factors (i.e., the remainder obtained when divided by the square factor) must be less than or equal to 210.
  Also $M \geq 1$.

- Fourier transform
  By inputting $\{x_{J1,...,JM}\}$ and performing the transform defined by (1.2), $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ is obtained.

$$\begin{aligned}
&N1 \cdots NM\alpha_{K1,...,KM} \\
&\quad = \sum_{J1=0}^{N1-1} \cdots \sum_{JM=0}^{NM-1} x_{J1,...,JM}\, \omega_1^{-J1 \cdot K1} \cdots \omega_M^{-JM \cdot KM} \\
&\quad , K1 = 0,1,...,N1-1 \\
&\qquad \vdots \\
&\quad , KM = 0,1,...,NM-1 \\
&\quad , \omega_1 = \exp(2\pi i/N1), \cdots, \omega_M = \exp(2\pi i/NM)
\end{aligned} \tag{1.2}$$

- Fourier inverse transform
  By inputting $\{\alpha_{K1,...,KM}\}$ and performing the transform defined by (1.3), $\{x_{J1,...,JM}\}$ is obtained.

$$\begin{aligned}
&x_{J1,...,JM} = \sum_{K1=0}^{N1-1} \cdots \sum_{KM=0}^{NM-1} \alpha_{K1,...,KM} \cdot \omega_1^{J1 \cdot K1} \cdots \omega_M^{JM \cdot KM} \\
&\quad , J1 = 0,1,\cdots,N1-1 \\
&\qquad \vdots \\
&\quad , JM = 0,1,\cdots,NM-1 \\
&\quad , \omega_1 = \exp(2\pi i/N1), \cdots, \omega_M = \exp(2\pi i/NM)
\end{aligned} \tag{1.3}$$

Parameters

A .... Input. Real part of $\{x_{J1,...,JM}\}$ or $\{\alpha_{K1,...,KM}\}$
Output. Real part of $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$
M-dimensional array.
See "Notes".

B .... Input. Imaginary part of $\{x_{J1,...,JM}\}$ or $\{\alpha_{K1,...,KM}\}$
Output. Imaginary part of $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$
M-dimensional array.
See "Notes".

N .... Input. Dimensions for the *M*-variate transform are given such as N(1) = *N*1, N(2) = *N*2 , ⋯, N(M) = *NM*.
One-dimensional array of size *M*.

M ...... Input. Number of variate: *M*

ISN ... Input. Either transform or inverse transform is specified ($\neq 0$) as follows:
for transform: ISN = +1
for inverse transform: ISN = –1
See "Notes".

ICON .. Output. Condition code
See Table CFTM-1.

Table CFTM-1 Condition codes

Code	Meaning		Processing
0	No error		
29100	Either of the dimensions, N1, ..., NM, has a prime factor γ that is no less than 23, and	the dimension N satisfies N (mod γ^2) =0	Bypassed
29200		The dimension N satisfies N (mod γ) = 0	
29300	Number of prime factors exceeds 11		
29400	Product of square free factors exceeds 210		
30000	M ≤ 0, ISN = 0 or one of the dimension ≤ 0		

Comments on use

- Subprograms used
  SSL II .... UCFTM and MGSSL
  FORTRAN basic functions ... ATAN, COS, SIN, SQRT, MOD and FLOAT

- Notes
  General definition of Fourier transform:
  The multi-variate discrete complex Fourier transform and its inverse transform are generally defined by (3.1) and (3.2).

$$\alpha_{K1,...,KM} = \frac{1}{N1 \cdots NM} \sum_{J1=0}^{N1-1} \cdots \sum_{JM=0}^{NM-1} x_{J1,...,JM}\, \omega_1^{-J1 \cdot K1} \cdots \omega_M^{-JM \cdot KM} \tag{3.1}$$

$$x_{J1,...,JM} = \sum_{K1=0}^{N1-1} \cdots \sum_{KM=0}^{NM-1} \alpha_{K1,...,KM}\, \omega_1^{J1 \cdot K1} \cdots \omega_M^{JM \cdot KM} \tag{3.2}$$

where definitions of $K1$, ..., KM, $J1$, ..., JM, and $\omega_1,...,\omega_M$ are given in (1.2) and (1.3).

The subroutine obtaines $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ or $\{x_{J1,...,JM}\}$ corresponding to the left-hand side of (3.1) and (3.2), respectively, and the user must scale the results, if necessary. If the transform and/ or inverse transform are executed without being scaled by calling the subroutine successively, each element of the input data is output and multiplied by $N1 \cdots NM$.
Determination of dimension and processing speed:
When determining the dimension in each variable, the conditions given in paragraph "Function" need to be satisfied, but if possible, it is desirable that the prime factors chosen are no larger than 5 (i.e., $p \le 5$). In this way, the processing speed is generally faster than the case where the prime factors are greater than 5 (i.e., $p > 5$). Table CFTM-2 lists all numbers up to 10000 which can be expressed only by using prime factors less than or equal to 5. Further, if the dimension is expressed to the power of 2 (2^l, $l \ge 0$ and also integer), the processing speed is faster if subroutine CFT is used.

Table CFTM-2 all numbers up to 10000

2	90	405	1215	2916	6075
3	96	432	1250	3000	6144
4	100	450	1280	3072	6250
5	108	480	1296	3125	6400
6	120	486	1350	3200	6480
8	125	500	1440	3240	6561
9	128	512	1458	3375	6750
10	135	540	1500	3456	6912
12	144	576	1536	3600	7200
15	150	600	1600	3645	7290
16	160	625	1620	3750	7500
18	162	640	1728	3840	7680
20	180	648	1800	3888	7776
24	192	675	1875	4000	8000
25	200	720	1920	4050	8100
27	216	729	1944	4096	8192
30	225	750	2000	4320	8640
32	240	768	2025	4374	8748
36	243	800	2048	4500	9000
40	250	810	2160	4608	9216
45	256	864	2187	4800	9375
48	270	900	2250	4860	9600
50	288	960	2304	5000	9720
54	300	972	2400	5120	10000
60	320	1000	2430	5184	
64	324	1024	2500	5400	
72	360	1080	2560	5625	
75	375	1125	2592	5760	
80	384	1152	2700	5832	
81	400	1200	2880	6000	

Data storing method:
All the real parts of the input $\{x_{J1,...,JM}\}$ are stored into the M-dimensional array A as shown in Fig. CFTM-1. The imaginary parts are stored likewise into the M-dimensional array B, as are the input $\{\alpha_{K1,...,KM}\}$ and the output $\{N1 \cdots NM\alpha_{K1,...,KM}\}$ and $\{x_{J1,...,JM}\}$

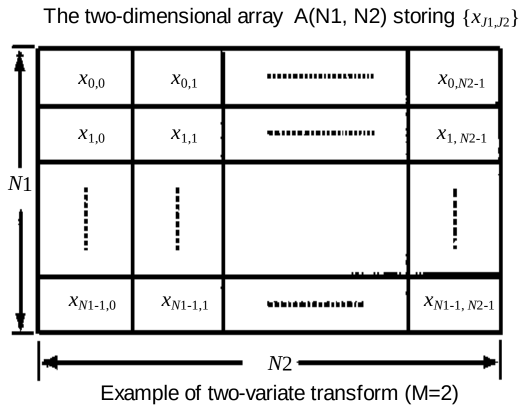


Fig. CFTM-1 Storing method of $\{x_{J1,...,JM}\}$

In general, when performing M-variate transform, if its data sequence is the same as given in FORTRAN for an M-dimensional array, parameters A and B can be each a one-dimensional array.
Giving the parameter ISN:
The parameter ISN specifies whether transform or inverse transform is performd, and it can also specify the interval I with which the real and imaginary parts of $\{x_{J1,...,JM}\}$ or $\{\alpha_{K1,...,KM}\}$ are stored in array A and B.

Transform: ISN = –I
Inverse transform: ISN = –I

The transformed results are also stored with the interval I.

- Example
  - (a) For a one-variable transform
    By inputting complex time series data $\{x_{J1}\}$ of dimension $N1$ and performing Fourier transform, $\{N1\alpha_{K1}\}$ is obtained.
    Here $N1 \le 1000$.

```
C     **EXAMPLE**
      DIMENSION A(1000),B(1000)
      READ(5,500) N,(A(I),B(I),I=1,N)
      WRITE(6,600) N,(I,A(I),B(I),I=1,N)
      CALL CFTM(A,B,N,1,1,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,620) (I,A(I),B(I),I=1,N)
      STOP
  500 FORMAT(I5/(2E20.7))
  600 FORMAT('0',10X,'INPUT DATA N=',I5/
     *      (15X,I5,2E20.7))
  610 FORMAT('0',10X,'RESULT   ICON=',I5)
  620 FORMAT('0',10X,'OUTPUT DATA'/
     *      (15X,I5,2E20.7))
      END
```

(b) For three-variate transform

By inputting complex time series data $\{x_{J1,J2,J3}\}$ of dimensions Nl , N2 and N3 , performing Fourier transform, and by using the results performing Fourier inverse transform, $\{x_{J1,J2,J3}\}$ is obtained. Here Nl = 5, N2 = 12 and N3 = 7.

```
C      **EXAMPLE**
       DIMENSION A(5,12,7),B(5,12,7),N(3)
       DATA N/5,12,7/
       READ(5,500) (((A(I,J,K),B(I,J,K),
      *             I=1,N(1)),J=1,N(2)),
      *             K=1,N(3))
       WRITE(6,600) N,(((I,J,K,A(I,J,K),
      *             B(I,J,K),I=1,N(1)),
      *             J=1,N(2)),K=1,N(3))
C      NORMAL TRANSFORM
       CALL CFTM(A,B,N,3,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0)STOP
C      INVERSE TRANSFORM
       CALL CFTM(A,B,N,3,-1,ICON)
       NT=N(1)*N(2)*N(3)
       DO 10 K=1,N(3)
       DO 10 J=1,N(2)
       DO 10 I=1,N(1)
       A(I,J,K)=A(I,J,K)/FLOAT(NT)
       B(I,J,K)=B(I,J,K)/FLOAT(NT)
    10 CONTINUE
       WRITE(6,620) (((I,J,K,A(I,J,K),
      *             B(I,J,K),I=1,N(1)),
      *             J=1,N(2)),K=1,N(3))
       STOP
   500 FORMAT(2E20.7)
   600 FORMAT('0',10X,'INPUT DATA',5X,
      *       '(',I3,',',I3,',',I3,')'/
      *      (15X,'(',I3,',',I3,',',I3,')',
      *       2E20.7))
   610 FORMAT('0',10X,'RESULT ICON=',I5)
   620 FORMAT('0',10X,'OUTPUT DATA'/
      *      (15X,'(',I3,',',I3,',',I3,')',
      *       2E20.7))
       END
```

Method

The multi-variate discrete complex Fourier transform is performed by using the mixed radix Fast Fourier Transform (FFT) with the prime factor p ($2 \le p \le 23$).

- Multi-variate transform

  The multi-variate transform defined in (1.2) can be reduced by rearranging common terms. For example, the two-variate transform can be reduced to as shown in (4.1)

$$N1 \cdot N2\alpha_{K1,K2} = \sum_{J1=0}^{N1-1} \omega_1^{-J1\cdot K1} \sum_{J2=0}^{N2-1} x_{J1\cdot J2}\, \omega_2^{-J2\cdot K2} \qquad (4.1)$$

In (4.1) $\sum_{J2}$ takes N1 sets of one-variable transforms of dimension N2 with respect to J1 , and for that result, $\sum_{J1}$ takes N2 sets of one-variable transforms of dimension N1 with respect to J2.

Similary, the multi-variable discrete complex Fourier transform is achieved by performing a multi-set of one-variable transforms in each variable. The subroutine applies the mixed radix Fast Fourier Transform with the prime factor p to perform one-variable transforms in each variable.

- Principle of mixed radix Fast Fourier Transform

  A one-variable discrete complex Fourier transform is defined as

$$\alpha_k = \sum_{j=0}^{n-1} x_j \omega^{-jk}, k = 0,1,\cdots,n-1 \qquad (4.2)$$
$$,\omega = \exp(2\pi i/n)$$

In (4.2), an ordinary scaling factor 1/ n is omitted. When calculating (4.2) directly, the multiplications of complex numbers are required as many as n^2.
If n is expressed by $n = r{\cdot}q$ with arbitrary factors r and q, and the characteristics of the exponential function ω^{-jk} are considered, then the number of multiplication is reduced to about n ($r + q$) . Let k and j in (4.2) be as follows:

$$\begin{aligned} k &= k_0 + k_1 \cdot q \quad ,0 \le k_0 \le q-1 \quad ,0 \le k_1 \le r-1 \\ j &= j_0 + j_1 \cdot r \quad ,0 \le j_0 \le r-1 \quad ,0 \le j_1 \le q-1 \end{aligned} \qquad (4.3)$$

Substituting (4.3) into (4.2),

$$\alpha_{k_0+k_1\cdot q} = \sum_{j_0=0}^{r-1}\sum_{j_1=0}^{q-1} x_{j_0+j_1\cdot r} \cdot \exp\left\{-2\pi j \frac{(k_0+k_1\cdot q)(j_0+j_1\cdot r)}{r\cdot q}\right\} \qquad (4.4)$$

Rearranging the right side of (4.4) with respect to j_0 and j_1 and putting the common terms together, (4.5) is obtained

$$\alpha_{k_0+k_1\cdot q} = \sum_{j_0=0}^{r-1} \exp\left\{-2\pi i \frac{k_1 j_0}{r}\right\} \exp\left\{-2\pi i \frac{k_0 j_0}{r}\right\} \cdot \sum_{j_1=0}^{q-1} \exp\left\{-2\pi i \frac{k_0 j_1}{q}\right\} x_{j_0+j_1\cdot r} \qquad (4.5)$$

In (4.5), $\sum_{j_1}$ takes r sets of transforms of dimension q with respect to j_0 , and $\sum_{j_0}$ takes q sets of transforms of dimension r with respect to j_1. The $\exp\{-2\pi i{\cdot}k_0 j_0/r\}$ is a rotation factor for the result of $\sum_{j_1}$. Therefore, the number of multiplications, C_n, done in (4.5) can be given in (4.6), and for a large n , it is smaller than n^2 which is the computation amount needed when (4.2) is calculated directly.

$$\begin{aligned} C_n &= r \cdot q^2 + q \cdot r^2 + (q-1)(r-1) \\ &= n(r+q+1) - (r+q) + 1 \end{aligned} \tag{4.6}$$

This type of transform is called a radix *r* and *q* Fast Fourier Transform. If *r* and *q* can be factored further to prime factors, the computation efficiency will be increased. The subroutine uses the mixed radix Fast Fourier Transform with prime factors of up to 23. For further details, refer to Reference [57].

F12-15-0202 CFTN, DCFTN

Discrete complex Fourier transforms (radix 8 and 2 FFT. reverse binary order output)
CALL CFTN (A, B, NT, N, NS, ISN, ICON)

Function

When one-variable complex time series data $\{x_j\}$ of dimension n is given. this subroutine performs discrete complex Fourier transforms or inverse transforms using the Fast Fourier Transform (FFT) method. The value n must be equal to 2^1(l = 0 or positive integer) .

- Fourier transform

  When $\{x_j\}$ is input, this subroutine determines $\{n\tilde{\alpha}_k\}$ by performing the transform defined in (1.1).

$$n\tilde{\alpha}_k = \sum_{j=0}^{n-1} x_j \omega^{-jk}, k = 0,1,...,n-1, \qquad (1.1)$$
$$\omega = \exp(2\pi i/n)$$

- Inverse Fourier transform

  When $\{\alpha_k\}$ is input, this subroutine determines $\{\tilde{x}_j\}$ by performing the inverse transform defined in (1.2).

$$\tilde{x}_j = \sum_{k=0}^{n-1} \alpha_k \omega^{jk} \quad , j = 0,1,...,n-1, \qquad (1.2)$$
$$\omega = \exp(2\pi i/n)$$

$\{\tilde{\alpha}_k\}$ and $\{\tilde{x}_j\}$ indicate that the transformed data is not in ascending order, since transform is performed in the area that the data was input. Let the results $\{\alpha_k\}$ and $\{x_j\}$ of the Fourier transform or inverse Fourier transform defined in (3.1) and (3.2) be ordered in ascending order, $\{\tilde{\alpha}_k\}$, $\{\tilde{x}_j\}$ are in reverse binary order.

Parameters

A	Input. Real parts of $\{x_j\}$ or $\{\alpha_k\}$. Output. Real parts of $\{n\tilde{\alpha}_k\}$ or $\{\tilde{x}_j\}$. One-dimensional array of size NT.
B......	Input. Imaginary part of $\{x_j\}$ or $\{\alpha_k\}$. Output. Imaginary parts of $\{n\tilde{\alpha}_k\}$ or $\{\tilde{x}_j\}$. One-dimensional array of size NT.
NT....	Input. The total number of data, including $\{x_j\}$ or $\{\alpha_k\}$, to be transformed (≥ N, NS) . Normally, NT = N is specified. (See "Notes".)
N	Input. Dimension n.
NS	Input. The interval of the consecutive data $\{x_j\}$ or $\{\alpha_k\}$ to be transformed of dimension n in the NT data (≥ 1 and ≤ NT) . Normally, NS = 1 is specified. (See "Notes".)
ISN ...	Input. Specifies normal or inverse transform (≠ 0). Transform : ISN = +1 Inverse transform: ISN = −1 (Refer to "Notes".)
ICON ..	Output. Condition code. See Table CFTN-1.

Table CFTN-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	ISN = 0, NS < 1, NT < N, NT < NS, or N ≠ 2^l (l = 0 or positive integer)	Bypassed

Comments on use

- Subprograms used

  SSL II ... MGSSL

  FORTRAN basic functions ... ATAN, ALOG, SQRT and SIN

- Notes

  General definition of discrete complex Fourier transform:

  Discrete complex Fourier transform and inverse Fourier transform are generally defined as:

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega^{-jk} \quad ,k = 0,1,...,n-1 \qquad (3.1)$$

and

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega^{jk}, j = 0,1,...,n-1 \qquad (3.2)$$

where

$$\omega = \exp(2\pi i/n)$$

This subroutine determines $\{n\tilde{\alpha}_k\}$ or $\{\tilde{x}_j\}$ in place of $\{\alpha_k\}$ of (3.1) or $\{x_j\}$ of (3.2). $\{\tilde{\alpha}_k\}$ and $\{\tilde{x}_j\}$ indicate that the transformed data is not in normal ascending order, since transform is performed in the area that data was input. That is, $\{n\tilde{\alpha}_k\}$ is in reverse binary order and its elements are multiplied by the value n compared to$\{\alpha_k\}$. $\{\tilde{x}_k\}$ is also in reverse binary order compared to $\{x_k\}$. Scaling and permutation of the result data are left to the user. Data can be permuted using subroutine PNR.

Refer to Example (a).

Use of this subroutine:

Usually, when performing a Fourier transform or an inverse Fourier transform, the subroutine CFT may be used. When normal and inverse transforms are performed successively, using this subroutine and subroutine CFTR together will increase efficiency. That is, at first $\{n\tilde{\alpha}_k\}$ is obtained by this routine. After a certain process is carried out, then processed $\{n\tilde{\alpha}_k\}$ can be performed inverse transform by subroutine CFTR without permutation. So, by the deletion of permutation, the processing speed can be reduced.

In subroutine CFTR, the Fourier transform and inverse Fourier transform defined in (3.3) or (3.4) are performed.

$$n\alpha_k = \sum_{j=0}^{n-1} \tilde{x}_j \omega^{-jk} \; , k = 0,1,\ldots,n-1 \tag{3.3}$$

$$x_j = \sum_{k=0}^{n-1} \tilde{\alpha}_k \omega^{jk} \; , j = 0,1,\ldots,n-1 \tag{3.4}$$

Refer to Example (b).

Multi-variate transform:

With this subroutine, multi-variable transforms are possible. For the 2-variate transform, the Fourier transform and inverse Fourier transform are defined as:

$$\alpha_{K1,K2} = \frac{1}{N1 \cdot N2} \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} x_{J1,J2}\, \omega_1^{-J1 \cdot K1} \cdot \omega_2^{-J2 \cdot K2} \tag{3.5}$$
$$, K1 = 0,\ldots,N1-1, K2 = 0,\ldots,N2-1$$

and

$$x_{J1,J2} = \sum_{K1=0}^{N1-1} \sum_{K2=0}^{N2-1} \alpha_{K1,K2}\, \omega_1^{J1 \cdot K1} \cdot \omega_2^{J2 \cdot K2} \tag{3,6}$$
$$, J1 = 0,\ldots,N1-1, J2 = 0,\ldots,N2-1$$

where $\omega_1 = \exp(2\pi i/N1), \omega_2 = \exp(2\pi i/N2)$

(3.5) can be reduced to simpler form by rearranging common terms as

$$\alpha_{K1,K2} = \frac{1}{N1 \cdot N2} \sum_{J1=0}^{N1-1} \omega_1^{-J1 \cdot K1} \sum_{J2=0}^{N2-1} x_{J1,J2} \omega_2^{-J2 \cdot K2} \tag{3.7}$$

The multi-variate transform of (3.7) is achieved by first performing $\sum_{J2}$ on Nl group 1-variable transforms of dimension $N2$ with respect to Jl , and then $\sum_{J1}$ is performed with the results on $N2$ groups l-variable transforms of dimension Nl with respect to $J2$. With calling this subroutine once, $\sum_{J2}$ on $N1$ groups 1-variable transforms of dimension $N2$ is performed. And with another calling, $\sum_{J1}$ on $N2$ groups l-variable transforms of dimension Nl is performed. This subroutine should be called as shown below concretely.

a) User must store $\{x_{J1,J2}\}$ in the two-dimensional array A and B as shown in Fig. CFTN-1.

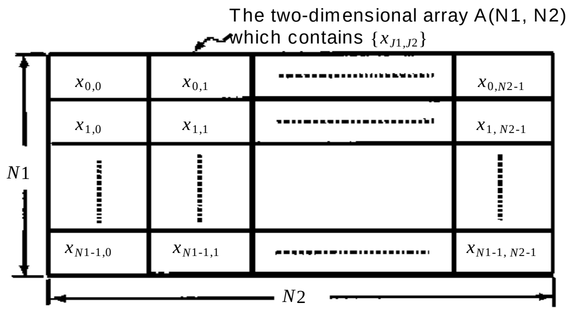


Fig. CFTN-1 Storage of $\{x_{J1,J2}\}$

b) This subroutine is called twice

```
    :
NT=N1*N2
NS=1
CALL CFTN(A,B,NT,N1,NS,1,ICON)
NS=N1
CALL CFTN(A,B,NT,N2,NS,1,ICON)
    :
```

Since the resultis is $\{N1 \cdot N2\tilde{\alpha}_{K1,K2}\}$, as with one-variable transforms, scaling and permutation of the data should be performed yhen necessary.

Data permutation can be done with subroutine PNR. The inverse transform defined in (3.6) can be processed similarly. Processing is possible for more than two variates. Refer to example (c) for threevariate applications.

Specifying ISN:

ISN is used to specify normal or inverse transform.

It is also used as follows:

If the real parts and imaginary parts of NT data are each stored in areas of size NT·I in intervals of I, the following specification is made.

Transform	: ISN = +I
Inverse transform	: ISN = –I

In this case, the results of transform are also stored in intervals of I.

- Examples

(a) 1-variable transform

Complex time series data $\{x_j\}$ of dimension n is put, and a Fourier transform is performed. The results are permuted using subroutine PNR and $\{n\alpha_k\}$ obtained. In case of $n \le 1024$ ($=2^{10}$).

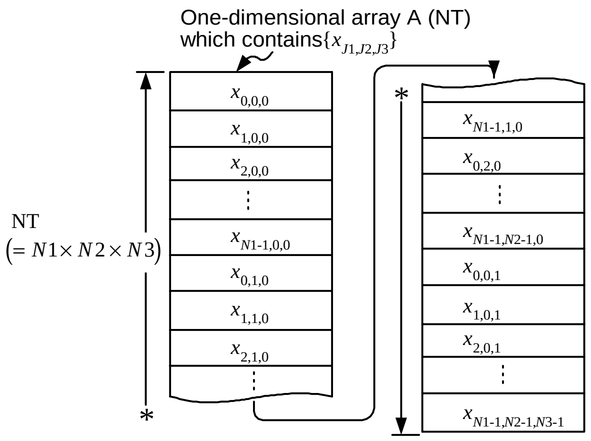
```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024)
       READ(5,500) N,(A(I),B(I),I=1,N)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
       CALL CFTN(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       CALL PNR(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       WRITE(6,620) (I,A(I),B(I),I=1,N)
       STOP
  500 FORMAT(I5 /(2E20.7))
  600 FORMAT('0',10X,'INPUT DATA N=',I5/
     *       /(15X,I5,2E20.7))
  610 FORMAT('0',10X,'RESULT ICON=',I5)
  620 FORMAT(15X,I5,2E20.7)
       END
```

(b) Successive transform/inverse transform
Complex time series data $\{x_j\}$ of dimension n is put, and this routine performs a Fourier transform to obtain $\{n\tilde{\alpha}_k\}$. Processing is done without permutation of the data, then the subroutine CFTR performs an inverse Fourier transform on the results.
In case of $n \le$ l024 (= 2^{10}) .

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024)
       READ(5,500) N,(A(I),B(I),I=1,N)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
C      NORMAL TRANSFORM
       CALL CFTN(A,B,N,N,1,1,ICON)
       IF(ICON.NE.0)STOP
          :
C      INVERSE TRANSFORM
       CALL CFTR(A,B,N,N,1,-1,ICON)
       IF(ICON .NE. 0)STOP
       DO 10 I=1,N
       A(I)=A(I)/FLOAT(N)
       B(I)=B(I)/FLOAT(N)
   10 CONTINUE
       WRITE(6,610) (I,A(I),B(I),I=1,N)
       STOP
  500 FORMAT(I5/(2E20.7))
  600 FORMAT('0',10X,'INPUT DATA N=',I5/
     *       /(15X,I5,2E20.7))
  610 FORMAT('0',10X,'OUTPUT DATA'/
     *       /(15X,I5 ,2E20.7))
       END
```

(c) 3-variate transform
Complex time series data $\{x_{J1,J2,J3}\}$ of dimension Nl , N2 and N3 is put, and this subroutine performs a Fourier transform to obtain $\{N1 \cdot N2 \cdot N3\tilde{\alpha}_{K1,K2,K3}\}$.
Processing is done without permulation of the data, then subroutine CFTR performs a Fourier inverse transform on the results.

In case of $N\mathrm{l}\cdot N2\cdot N3 \le 1024$ (= 2^{10}).
The data can be stored in a one-dimensional array as shown in Fig. CFTN-2.



Note:
When stored in this way, NS can be specified in the order of the three ca1ls --- 1, N1, $N1*N2$.

Fig. CFTN-2 Storage of $\{x_{J1,J2,J3}\}$

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024),N(3)
       READ(5,500) N
       NT=N(1)*N(2)*N(3)
       READ(5,510) (A(I),B(I),I=1,NT)
       WRITE(6,600) N,(I,A(I),B(I),I=1,NT)
C      NORMAL TRANSFORM
       NS=1
       DO 10 I=1,3
       CALL CFTN(A,B,NT,N(I),NS,l,ICON)
       IF(ICON.NE.0) STOP
       NS=NS *N(I)
   10 CONTINUE
             :
C      INVERSE TRANSFORM
       NS=1
       DO 20 I=1,3
       CALL CFTR(A,B,NT,N(I),NS,-1,ICON)
       IF(ICON.NE.0) STOP
       NS=NS*N(I)
   20 CONTINUE
C      NORMALIZATION
       DO 30 I=1,NT
       A(I)=A(I)/FLOAT(NT)
       B(I)=B(I)/FLOAT(NT)
   30 CONTINUE
       WRITE(6,610) (I,A(I),B(I),I=1,NT)
       STOP
  500 FORMAT(3I5)
  510 FORMAT(2E20.7)
  600 FORMAT('0',10X,'INPUT DATA N=',3I5/
     *       /(15X,I5,2E20.7))
  610 FORMAT('0,10X,'OUTPUT DATA'/
     *       /(15X,I5,2E20.7))
       END
```

Method

This subroutine performs discrete complex Fourier transforms using the radix 8 and 2 Fast Fourier Transform (FFT) method, or performs the inverse transforms. Refer to the section on subroutine CFT for the principles of FFT. In this section, a specific example in which n = 16 will be discussed. In this case, the Fourier transform is defined as

$$\alpha_k = \sum_{j=0}^{15} x_j \, \omega^{-jk} \quad , k = 0,1,...,15, \qquad (4.1)$$
$$\omega = \exp(2\pi i/16)$$

The scaling factor l/16 is omitted in (4.1). In this routine, n is factored into factors 8 and 2 (n =8 × 2). k and j can be expressed as

$$\begin{aligned} k &= k_0 + k_1 \cdot 8 \quad ,0 \le k_0 \le 7 \quad ,0 \le k_1 \le 1 \\ j &= j_0 + j_1 \cdot 2 \quad ,0 \le j_0 \le 1 \quad ,0 \le j_1 \le 7 \end{aligned} \qquad (4.2)$$

If (4.2) is substituted in (4.1) and common terms are rearranged, (4.3) results. Where, $\alpha(k_0 + k_1 \cdot 8) \equiv a_{k_0+k_1 \cdot 8}$

$$\begin{aligned} \alpha(k_0 + k_1 \cdot 8) = \sum_{j_0=0}^{1} \exp\left(-2\pi i \frac{j_0 k_1}{2}\right) \\ \cdot \exp\left(-2\pi i \frac{j_0 \cdot k_0}{16}\right) \sum_{j_1=0}^{7} \exp\left(-2\pi i \frac{j_1 k_0}{8}\right) \\ \cdot x(j_0 + j_1 \cdot 2) \end{aligned} \qquad (4.3)$$

In this routine, (4.3) is successively calculated. The results are stored in the same area that data was input. The procedure follows, see Fig. CFTN-3.

- Process l: $x(j_0 + k_0 \cdot 2) \Leftarrow$

$$\exp\left(-2\pi i \frac{j_0 k_0}{16}\right) \sum_{j_1=0}^{7} \exp\left(-2\pi i \frac{j_1 k_0}{8}\right) x(j_0 + j_1 \cdot 2) \qquad (4.4)$$

(4.4) is executed. The elementary Fourier transform of dimension 8 corresponding to $\sum_{j_1}$ are performed initially with respect to j_0=0 , that is, the data x_0, x_2, x_4, x_6, x_8, x_{10}, x_{12} and x_{14} are transformed.

Then the transform of dimension 8 is performed with respect to j_0=1, that is the data x_1, x_3, x_5, x_7, x_9, x_{11}, x_{13} and x_{15} are transformed.
The results are multiplied by the rotation factor

$$\exp\left(-2\pi i \frac{j_0 k_0}{16}\right)$$

This multiplication is not necessary for j_0=0 because the rotation factor is 1 when j_0=0. When j_0=1 , with the exception of k_0=0, the following rotation factors are multiplied together in order – $\xi^1,\xi^2,\xi^3,\xi^4,\xi^5,\xi^6$ and ξ^7, ξ=exp(−2πi/16). The above results are stored in $x(j_0+k_0 \cdot 2)$.

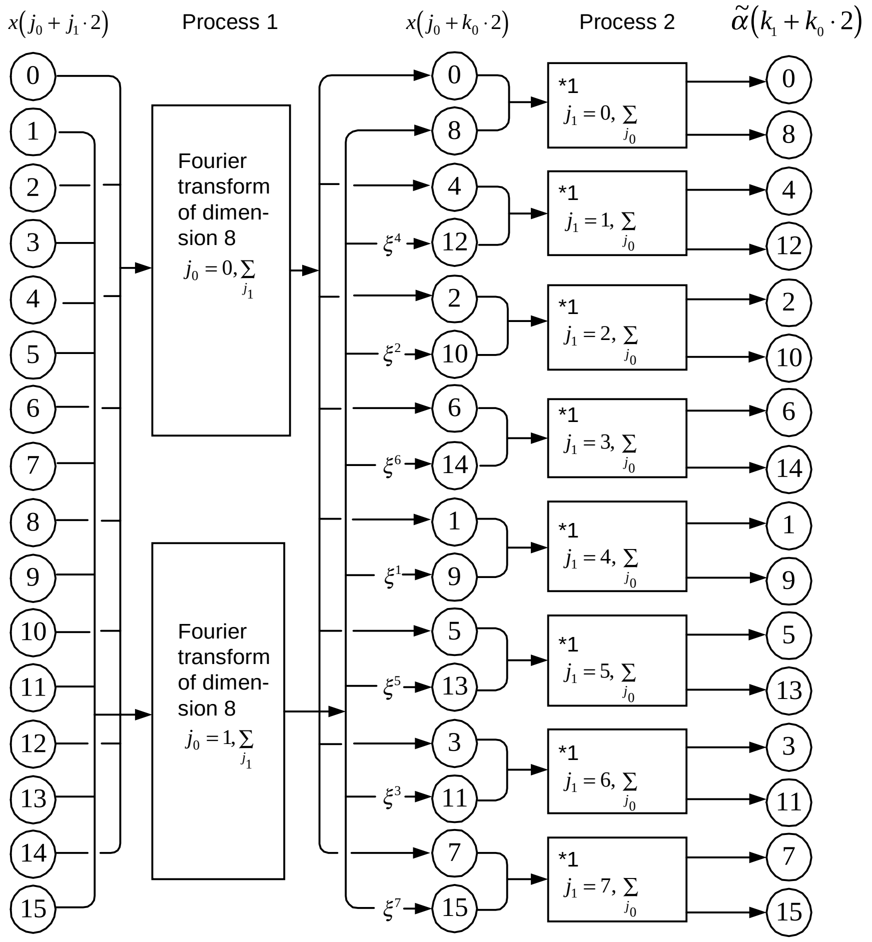
- Process 2: $x(k_1 + k_0 \cdot 2) \Leftarrow$

$$\sum_{j_0=0}^{1} \exp\left(-2\pi i \frac{j_0 k_1}{2}\right) x(j_0 + k_0 \cdot 2) \qquad (4.5)$$

is executed. With the results of process 1 , the Fourier transform of dimension 2 corresponding to $\sum_{j_0}$ is performed with respect to k_0=0, that is data x_0 and x_1 are transformed. Similarly, with respect to k_0=1,...,7 , the Fourier transform of dimension 2 is performed for each. It is not necessary to multiply these results by the rotation factor.The above results are stored in $x(k_1+k_0 \cdot 2)$.
Since the $x(k_1+k_0 \cdot 2)$ obtained in this way is in reverse digit order compared to the $\alpha(k_0+k_1 \cdot 8)$ to be obtained as the discrete complex Fourier transform of dimension 16, it is expressed as $\tilde{\alpha}(k_1 + k_0 \cdot 2)$. In this routine, since the results of elementary Fourier transform of dimension 8 are in reverse binary order, the final result $\tilde{\alpha}(k_1 + k_0 \cdot 2)$ is also in reverse binary order compared to $\alpha(k_0 + k_1 \cdot 8)$.

For further information, see References [55], [56], and [57].

$$\alpha(k_0 + k_1 \cdot 8) = \sum_{j_0=0}^{1} \exp\left(-2\pi i \frac{j_0 k_1}{2}\right) \exp\left(-2\pi i \frac{j_0 k_0}{16}\right) \sum_{j_1=0}^{7} x(j_0 + j_1 \cdot 2) \exp\left(-2\pi i \frac{j_1 k_0}{8}\right)$$



Note:
The circled numbers represent the order of the data.
$\xi = \exp(-2\pi i/16)$. Since the results of Fourier transforms of dimension 8 are in reverse binary order, $\tilde{\alpha}(k_1 + k_0 \cdot 2)$ is in reverse binary order against $\alpha(k_0 + k_1 \cdot 8)$.
*1 Complex Fourier transform of dimension 2.

Fig. CFTN-3 Flow chart of a complex Fourier transform of dimension 16 (reverse binary order output)

F12-15-0302 CFTR, DCFTR

Discrete complex Fourier transform (radix 8 and 2 FFT. reverse binary order input)
CALL CFTR (A, B, NT, N, NS, ISN, ICON)

Function

When the one-variable complex time series data $\{x_j\}$ of dimension n is given in reverse binary order as $\{\tilde{x}_j\}$, the discrete complex Fourier transform or its inverse transform is performed by using the Fast Fourier Transform (FFT). Here $n = 2^l$ (l = 0 or positive integer).

- Fourier transform
  $\{\tilde{x}_j\}$ is input and the transform defined by (1.1) is carried out to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} \tilde{x}_j \omega^{-jk}, k = 0,1,...,n-1 \qquad (1.1)$$
$$,\omega = \exp(2\pi i/n)$$

- Fourier inverse transform
  $\{\tilde{\alpha}_k\}$ is input and the transform defined by (1.2) is carried out to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \tilde{\alpha}_k \omega^{jk}, j = 0,1,...,n-1 \qquad (1.2)$$
$$,\omega = \exp(2\pi i/n)$$

The sequence of the transformed $\{n\alpha_k\}$ and $\{x_j\}$ are given in the normal sequence.

Parameters

A	Input. Real part of $\{\tilde{x}_j\}$ or $\{\tilde{\alpha}_k\}$. Output. Real part of $\{n\alpha_k\}$ or $\{x_j\}$. One-dimensional array of size NT.
B	Input. Imaginary part of $\{\tilde{x}_j\}$ or $\{n\alpha_k\}$. Output. Imaginary part of $\{n\alpha_k\}$ or $\{x_j\}$. One-dimensional array of size NT.
NT ...	Input. Total number of data in which $\{\tilde{x}_j\}$ or $\{\tilde{\alpha}_k\}$ to be transformed are contained. (NT ≥ N and NS) Normally, NT = N. See "Notes".
N ..	Input. Dimension: n
NS ..	Input. Interval of the consecutive data $\{\tilde{x}_j\}$ or $\{n\alpha_k\}$ to be transformed of dimension n in the NT data. (NS ≥ 1 and ≤ NT) Normally, NS = 1. See "Notes".
ISN ..	Input. Either transform or inverse transform is specified (≠ 0) as follows: for transform: ISN = +l for inverse transform: ISN = –l See "Notes".
ICON ..	Output. Condition code See Table CFTR-1 .

Table CFTR-1 Conditions codes

Code	Meaning	Processing
0	No error	
30000	ISN = 0. NS < 1, NT < N, NT < NS, or N≠2^l (l = 0 or positive integer)	By pessed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic functions ... ATAN, ALOG, SQRT and SIN

- Notes
  General definition of Fourier transform:
  The discrete complex Fourier transform and its inverse transform are generally defined as given in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega^{-jk} \quad ,k = 0,1,...,n-1 \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega^{jk} \quad , j = 0,1,..., n-1 \qquad (3.2)$$
$$,\omega = \exp(2\pi i/n)$$

The subroutine performs transform for $\{\tilde{x}_j\}$ or $\{\tilde{\alpha}_k\}$ whose sequence is reverse binary order. The $\{\tilde{x}_j\}$ and $\{n\alpha_k\}$ correspond to $\{x_j\}$ and $\{n\alpha_k\}$ in the right-hand side of (3.1) and (3.2), respectively. The transformed results, $\{n\alpha_k\}$ or $\{x_j\}$, are obtained each corresponding to $\{n\alpha_k\}$ and $\{x_j\}$ in the left-hand side of (3.1) and (3.2), respectively. That is, the obtained result sequence is normal order and the elements of $\{n\alpha_k\}$ are given multiplied by n .

The normalization of the results must be carried out by the user, if necessary.

Use of the subroutine:

For Fourier transform or inverse Fourier transform, the subroutine CFT is normally used, but if the transform and inverse transform are to be executed one after another, the subroutine CFTN and CFTR described in this section should be used for better efficiency.

Refer to "Comments on use" of subroutine CFTN.

Application of multi-variate transform:

This subroutine, CFTR, can be applied to a multi-variate transform. For exarnple, a two-variate Fourier transform and its inverse transform are defined as follows:

$$\alpha_{K1,K2} = \frac{1}{N1\cdot N2}\sum_{J1=0}^{N1-1}\sum_{J2=0}^{N2-1} x_{J1,J2}\,\omega_1^{-J1\cdot K1}\omega_2^{-J2\cdot K2} \qquad (3.5)$$
$$,K1 = 0,1,...,N1-1, \quad K2 = 0,1,...,N2-1$$

$$x_{J1,J2} = \sum_{K1=0}^{N1-1}\sum_{K2=0}^{N2-1}\alpha_{K1,K2}\,\omega_1^{J1\cdot K1}\omega_2^{J2\cdot K2} \qquad (3.6)$$
$$, J1 = 0,1,...,N1-1, J2 = 0,1,...,N2-1$$

where $\omega_1 = \exp(2\pi i/N1), \omega_2 = \exp(2\pi i/N2)$
Eq. (3.5) can be rewritten to

$$\alpha_{K1,K2} = \frac{1}{N1\cdot N2}\sum_{J1=0}^{N1-1}\omega_1^{-J1\cdot K1}\sum_{J2=0}^{N2-1}x_{J1,J2}\,\omega_2^{-J2\cdot K2} \qquad (3.7)$$

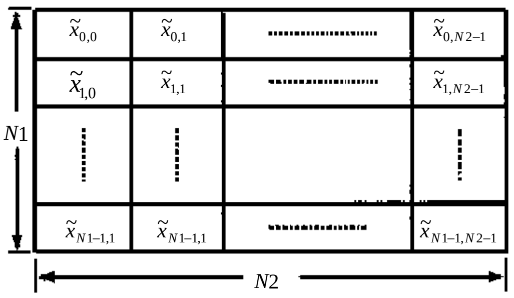
The two-variate transform (3.7) above is achieved such that Nl sets of one-variable transforms of dimension $N2$ are performed with respect to $J1$, and for that result. $N2$ sets of one-variable transforms of dimension $N1$ performed with respect to $J2$.
The subroutine is capable of performing $N1$ sets of one-variable transforms of dimesnsion $N2$ simultaneously once it is called. In this way computational load is reduced compared to the case where the one-variable transforms of dimension $N2$ are calculated one by one, so that a multi-variate transform can be accomplished efficiently. In practice, specify as shown below and then call the subroutine.

a) $\{\tilde{x}_{J1,J2}\}$ are stored in the two-dimensional array A and B as illustrated in Fig. CFTR-1.
b) the subroutine is called twice as follows:

```
       :
   NT=N1*N2
   NS=1
   CALL CFTR(A,B,NT,N1,NS,1,ICON)
   NS=N1
   CALL CFTR(A,B,NT,N2,NS,1,ICON)
       :
```

The obtained result is $\{N1\cdot N2\alpha_{K1,K2}\}$, so normalization must be done by the user, if necessary, in the same way as for a one-variable transform.
The inverse transform defined in (3.6) can be similarly performed. Transforms with more than two variate are also possible and example (b) shows a case of three variates.
Giving ISN:
The parameter ISN specifies whether transform or inverse transform is performed. and can be also used for the following case. That is, when the real and imaginay parts of NT number of $\{\tilde{x}_j\}$ or $\{\tilde{\alpha}_k\}$ are stored in the area of size NT·I with an interval I between each other, the ISN is specified as follows:
for transform: ISN = +I
for inverse transform: ISN = –I
The transformed results are stored also with an interval I.

Two dimensional array, A (N1. N2), which stores $\{\tilde{x}_{J1,J2}\}$



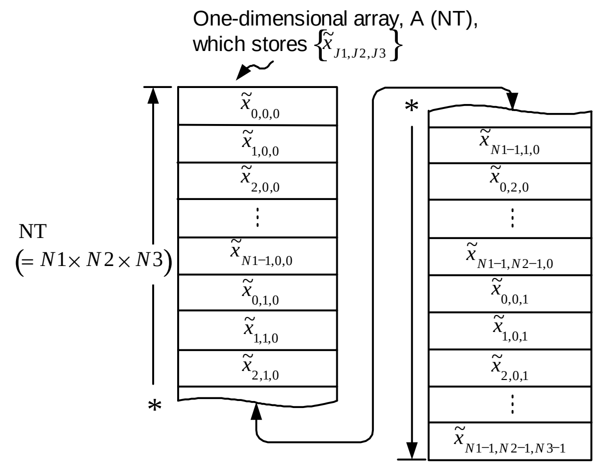
Note:
Array A contains the real parts of $\{\tilde{x}\}$. The imaginary parts are stored likewise in the two-dimensional array B (N1, N2).

Fig. CFTR-1 Storing method

- Example
  (a) One-variable transform
  The complex time series data $\{x_j\}$ of dimension n are input, and permuted by subroutine PNR in reverse binary order.
  The result $\{\tilde{x}_j\}$ is subjected to Fourier transform by the subroutine to obtain $\{n\alpha_k\}$.
  Here $n \le 1024$ (= 2^{10}).

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024)
       READ(5,500) N,(A(I),B(I),I=1,N)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
       CALL PNR(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       CALL CFTR(A,B,N,N,1,1,ICON)
       WRITE(6,620) (I,A(I),B(I),I=1 ,N)
       STOP
  500 FORMAT(I5/(2E20.7))
  600 FORMAT('0',10X,'INPUT DATA N=',I5/
     *       /(15X,I5 ,2E20.7))
  610 FORMAT('0',10X,'RESULT ICON=',I5)
  620 FORMAT('0',10X,'OUTPUT DATA'//
     *       (15X,I5 ,2E20.7))
       END
```

  (b) Three-variate transform
  The three-variate complex time series data $\{x_{J1,J2,J3}\}$ of dimension $N1$, $N2$ and $N3$ are input and permuted by subroutine PNR in reverse binary order.
  The results $\{\tilde{x}_{J1,J2,J3}\}$ are subjected to Fourier transform by the subroutine to obtain $\{N1\cdot N2\alpha_{K1,K2}\}$.
  The data $\{x_{J1,J2,J3}\}$ may be stored in a one-dimensional array as illustrated in Fig.CFTR-2.



Note
Array A contains the real part of $\{\tilde{x}\}$. The imaginary parts are contained likewise in the one-dimensional array B (NT). Parameter NS, when contained in this way, is given 1, $N1$ and $N1\cdot N2$ for each of the three times the subroutine is called.

Fig. CFTR-2 Storage of $\{\tilde{x}_{J1,J2,J3}\}$

```
C     **EXAMPLE**
      DIMENSION A(1024),B(1024),N(3)
      READ(5,500) N
      NT=N(1)*N(2)*N(3)
      READ(5,510) (A(I),B(I),I=1,NT)
      WRITE(6,600) N,(I,A(I),B(I),I=1,NT)
      NS=1
      DO 10 I=1,3
      CALL PNR(A,B,NT,N(I),NS,1,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) STOP
      CALL CFTR(A,B,NT,N(I),NS,1,ICON)
      NS=NS*N(I)
   10 CONTINUE
      WRITE(6,620) (I,A(I),B(I),I=1,NT)
      STOP
  500 FORMAT(3I5)
  510 FORMAT(2E20.7)
  600 FORMAT('0',10X,'INPUT DATA N=',3I5/
     *       /(15X,I5 ,2E20.7))
  610 FORMAT('0',10X,'RESULT ICON=',I5)
  620 FORMAT('0',10X,'OUTPUT DATA'//
     *       (15X,I5 ,2E20.7))
      END
```

Method

The discrete complex Fourier transform is performed by using the radix 8 and 2 Fast Fourier Transform (FFT) . The input data is to be given in reverse binary order. For the principle of the Fast Fourier Transform, refer to the section of subroutine CFT.

In this section, the transform is explaind for n = 16.

$$\alpha_k = \sum_{j=0}^{15} x_j \omega^{-jk}, k = 0,1,...,15, \qquad \omega = \exp(2\pi i/16) \tag{4.1}$$

where the scaling factor 1/16 is omitted. The subroutine factors n by using 2 and 8 ($n = 2 \times 8$).

k and j are expressed as follows:

$$\begin{aligned} k &= k_0\cdot 2 + k_1,\ 0 \le k_0 \le 7,\ 0 \le k_1 \le 1 \\ j &= j_0\cdot 8 + j_1,\ 0 \le j_0 \le 1,\ 0 \le j_1 \le 7 \end{aligned} \tag{4.2}$$

Substituting (4.2) into (4.1) and rearranging the common terms, (4.3) is obtained.

$$\begin{aligned} \alpha(k_0\cdot 2 + k_1) = &\sum_{j_1=0}^{7} \exp\left\{-2\pi j \frac{j_1 k_0}{8}\right\} \\ &\cdot \exp\left\{-2\pi i \frac{j_1 k_1}{16}\right\} \\ &\cdot \sum_{j_0=0}^{1} \exp\left\{-2\pi i \frac{j_0 k_1}{2}\right\} x(j_0\cdot 8 + j_1) \end{aligned} \tag{4.3}$$

where $\alpha(k_0\cdot 2 + k_1) \equiv \alpha_{k_0\cdot 2 + k_1}$

The subroutine is given its input data in reverse binary order, i.e., $\tilde{x}(j_0 + j_1\cdot 2)$ is given instead of $x(j_0\cdot 8 + j_1)$, then Eq. (4.3) is calculated successively. The results are stored in the same area.

The procedure is explained below, in conjunction with Fig. CFTR-3.

- Step 1: $\tilde{x}(k_1 + j_1\cdot 2) \Leftarrow$

$$\sum_{j_0=0}^{1} \exp\left\{-2\pi i \frac{j_0 k_1}{2}\right\} \tilde{x}(j_0 + j_1\cdot 2) \tag{4.4}$$

First of all, Fourier transforms of dimension 2 are performed by $\sum_{j_0}$ with respect to j_0=0, that is, transform is done for $\tilde{x}_0$ and $\tilde{x}_1$. Similarly, the Fourier transforms of dimension 2 are performed with respect to j_0=1 to 7 and the results are stored in $\tilde{x}(k_1 + j_1\cdot 2)$.

- Step 2: $\tilde{x}(k_1 + k_0\cdot 2) \Leftarrow$

$$\sum_{j_1=0}^{7} \exp\left\{-2\pi i \frac{j_1 k_0}{8}\right\} \exp\left\{-2\pi i \frac{j_1 k_1}{16}\right\} \cdot \tilde{x}(k_1 + j_1\cdot 2) \tag{4.5}$$

The results obtained at step 1 are multiplied by the rotation factors, $\exp\left\{-2\pi i \frac{j_1 k_1}{16}\right\}$ and Fourier transforms of dimension 8 are performed by $\sum_{j_1}$ with respect to k_0=0 and k_0=1.

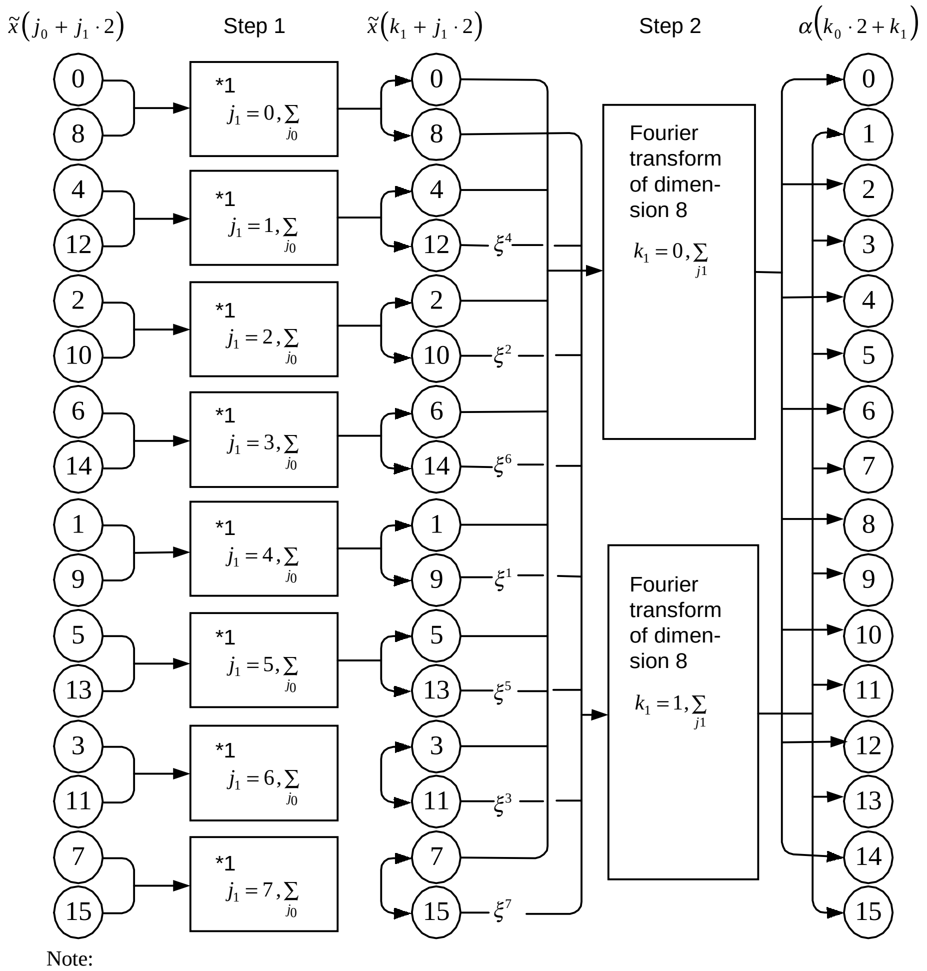
When k_0=0, the all rotation factors are 1, so no

multiplication is necessary, and the transform is done for $\tilde{x}_0, \tilde{x}_2, \tilde{x}_4, \tilde{x}_6, \tilde{x}_8, \tilde{x}_{10}, \tilde{x}_{12}$ and $\tilde{x}_{14}$ when k_1=1 the transform is done after $\tilde{x}_1, \tilde{x}_3, \tilde{x}_5, \tilde{x}_7, \tilde{x}_9, \tilde{x}_{11}, \tilde{x}_{13}, \tilde{x}_{15}$ are multiplied by the rotation factor. The rotation factors are ξ^0, ξ^1, ξ^2, ξ^3, ξ^4, ξ^5, ξ^6 and ξ^7, where
$\xi = \exp(-2\pi i/16)$. Since each of the input data sequence for these two sets of Fourier transforms of dimension 8 is in reverse binary order, each of the transformed results is permuted in the normal order and stored in $\tilde{x}(k_1 + k_0 \cdot 2)$.

The obtained $\tilde{x}(k_1 + k_0 \cdot 2)$ agrees with $\tilde{\alpha}(k_1 + k_0 \cdot 2)$ which is the result of the "discrete complex Fourier transform of dimension 16".

For further information, see References [55], [56], and [57].

$$\alpha(k_0 \cdot 2 + k_1) = \sum_{j_1=0}^{7} \exp\left\{-2\pi i \frac{j_1 k_0}{8}\right\} \exp\left\{-2\pi i \frac{j_1 k_1}{16}\right\} \sum_{j_0=0}^{1} \exp\left\{-2\pi i \frac{j_0 k_1}{2}\right\} \tilde{x}(j_0 + j_1 \cdot 2)$$



Note:
Numbers inside ○ marks indicate the data sequence, and $\xi = \exp(-2\pi i/16)$.
*1: Complex Fourier transform of dimension 2

Fig. CFTR-3 Flow chart of a complex Fourier transform of dimension 16
(reverse binary order input)

A11-40-0101 CGSBM, DCGSBM

Storage mode conversion of matrices (real general to real symmetric band)
CALL CGSBM (AG, K, N, ASB, NH, ICON)

Functions

This subroutine converts an $n \times n$ real symmetric band matrix with band width h stored in the general mode into one stored in the compressed mode for symmetric band matrix, where $n > h \geq 0$.

Parameters

AG	Input. The symmetric band matrix stored in the general mode. Two-dimensional array, AG (K, N). (See "Comments on Use.")
K	Input. The adjustable dimension ($\geq$ N) of array AG.
N	Input. The order (n) of the matrix.
ASB ...	Output. The Symmetric band matrix stored in the compressed mode. One-dimensional array of size $n(h+1)-h(h+1)/2$.
NH	Input. Band width h of the matrix.
ICON ..	Output. Condition code. (See Table CGSBM-1.)

Table CGSBM-1 Condition codes

Code	Meaning	Processing
0	No error.	
30000	NH < 0, N ≤ NH, or K<N	By passed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... MAX0

- Notes
  Storing method of the symmetric band matrix in the general mode:
  Only the elements of the lower band and the diagonal portions can be stored in array AG. The elements of the lower band portion are copied into those of the upper band portion in this subroutine.

- Saving the storage area:
  If the contents of array AG need not be retained, the storage area can be saved using an EQUIVALENCE statement as follows:
  EQUIVALENCE (AG (1,1), ASB (1)) (See "Example" for details.)

- Example
  Given an $n \times n$ positive-definite symmetric band matrix with band width h in the general mode, this example converts it into one stored in the compressed mode for symmetric band matrix, then performs decomposition. Subroutine SBDL is used for the decomposition, whereas this subroutine and subroutine CSBGM are used to convert the mode, where $n \leq 100$ and $h \leq 20$.

```
C      **EXAMPLE**
       DIMENSION AG(100,100),ASB(1890)
       EQUIVALENCE(AG(1,1),ASB(1))
    10 READ(5,500) N,NH
       IF(N.EQ.0)STOP
       K=100
       READ(5,510) ((AG(I,J),I=1,N),J=1,N)
       WRITE(6,600) N,NH,((I,J,AG(I,J),
      *             I=1,N),J=1,N)
       CALL CGSBM(AG,K,N,ASB,NH,ICON)
       WRITE(6,610) ICON
       IF(ICON.EQ.30000) GO TO 10
       EPSZ=0.0
       CALL SBDL(ASB,N,NH,EPSZ,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) GO TO 10
       CALL CSBGM(ASB,N,NH,AG,K,ICON)
       WRITE(6,630) N,NH,((I,J,AG(I,J),
      *             J=1,N),I=1,N)
       GO TO 10
   500 FORMAT(2I5)
   510 FORMAT(4E15.7)
   600 FORMAT(//10X,'** INPUT MATRIX **'/
      *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
      *(2X,4('(',I3,',',I3,')',E17.8)))
   610 FORMAT('1'/10X,'CGSBM ICON=',I5)
   620 FORMAT(/10X,'SBDL ICON=',I5)
   630 FORMAT('1'//10X,'DECOMPOSED MATRIX'/
      *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
      *(2X,4('(',I3,',',I3,')',E17.8)))
       END
```

Method

A real symmetric band matrix stored in a two-dimensional array AG in the general mode is processed as follows to be a symmetric band matrix in a one-dimensional array in the compressed mode.

The elements of the lower band portion are moved to the upper band portion using the diagonal as the axis of symmetry.

$$\mathrm{AG(I,J)} \rightarrow \mathrm{AG(J,I)},\ \mathrm{I} - h \leq \mathrm{J} \leq \mathrm{I} - 1$$

The elements of the diagonal and upper band AG (I,J) are moved to the ASB, beginning from column 1 of AG, as follows:

Elements in the general mode	Matrix Elements	Elements in the compressed mode

$$\mathrm{AG\ (I,\ J)} \longrightarrow a_{ij} \longrightarrow \mathrm{ASB\ \ (J\ (J-1)/2+I)}$$
$$,\ \mathrm{I} = 1, 2, \cdots, \mathrm{J}\ \ ,\ \ \mathrm{J} = 1, 2, \cdots, h+1$$
$$\mathrm{AG\ (I,\ J)} \longrightarrow a_{ij} \longrightarrow \mathrm{ASB}\ (h\mathrm{J} - h\ (h+1)/2+\mathrm{I})$$
$$,\ \mathrm{I} = \mathrm{J}-h, \mathrm{J}-h+1, \cdots, \mathrm{J}\ ,\ \mathrm{J} = h+2, h+3, \cdots,\mathrm{N}$$

A11-10-0101 CGSM, DCGSM

Storage mode conversion of matrices (real general to real symmetric)
CALL CGSM (AG, K, N, AS, ICON)

Functions

This subroutine converts an $n \times n$ real symmetric matrix stored in the general mode into a symmetric matrix stored in the compressed mode. $n \geq 1$.

Parameters

AG ..	Input. The symmetric matrix stored in the general mode. AG is a two-dimensional array, AG (K, N). (See "Comments on use".)
K ..	Input. The adjustable dimension (≥ N) of array AG.
N ...	Input. The order n of the matrix.
AS ...	Output. The symmetric matrix stored in the compressed mode. AS is a one-dimensional array of size n (n +1)/2.
ICON ...	Output. Condition codes. Refer to Table CGSM-1.

Table CGSM-1 Condition code

Code	Meaning	Processing
0	No error	
30000	N < 1 or K < N	By passed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... None

- Notes
  Storage method of the symmetric matrix in the general mode:
  Only the elements of the diagonal and lower triangular portions need be given to array AG. The subroutine copies the lower triangular portion to the upper triangular portion.
  If there is no need to keep the contents on the array AG, more atorage can be saved by using the EQUIVALENCE statement as follow;

  EQUIVALENCE (AG(1,1), AS(1))

  Refer to the example shown below.

- Example
  Given an $n \times n$ positive-definite symmetric matrix in the general mode, the inverse matrix is obtained by subroutines SLDL and LDIV as shown in the example.
  In this case, the required mode conversion is performed by subroutines CGSM and CSGM.
  Here $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(5050)
      EQUIVALENCE(A(1,1),B(1))
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      K=100
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),
     *             I=1,N)
      CALL CGSM(A,K,N,B,ICON)
      WRITE(6,610) ICON
      IF(ICON.EQ.30000) GOTO 10
      EPSZ=0.0
      CALL SLDL(B,N,EPSZ,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GOTO 10
      CALL LDIV(B,N,ICON)
      WRITE(6,630) ICON
      CALL CSGM(B,N,A,K,ICON)
      WRITE(6,640) ICON
      WRITE(6,650)N,((I,J,A(I,J),J=1,N),
     *            I=1,N)
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'//10X,'** INPUT MATRIX **'/
     *10X,'ORDER=',I5/(2X,
     *4('(',I3,',',I3,')',E17.8)))
  610 FORMAT(10X,'CGSM ICON=',I5)
  620 FORMAT(10X,'SLDL ICON=',I5)
  630 FORMAT(10X,'LDIV ICON=',I5)
  640 FORMAT(10X,'CSGM ICON=',I5)
  650 FORMAT('1'//10X,'** INVERSE ',
     *'MATRIX **'/10X,'ORDER=',I5/(2X,
     *4('(',I3,',',I3,')',E17.8)))
      END
```

Methods

This subroutine converts an $n \times n$ real symmetric matrix stored in a two-dimensional array AG in the general mode to in a one-dimensional array in the compressed mode through the following procedures.

- With the diagonal as the axis of symmetry, the elements of the lower triangular portion are transferred to the upper triangular portion.

  AG(I,J) $\rightarrow$ AG(J,I), J<I

- The diagonal and upper triangular elements AG (I,J) is transferred to the J (J-1)/2 + I position in AS.
  Here, J ≥ I. Transfer begins with the first column of AG and continues column by column.
  The correspondence between locations is shown below, where NT = n (n + 1)/2 .

Elements in the general mode		Elements of the matrix		Elements in the compressed mode
AG (1, 1)	$\rightarrow$	a_{11}	$\rightarrow$	AS (1)
AG (1, 2)	$\rightarrow$	a_{21}	$\rightarrow$	AS (2)
AG (2, 2)	$\rightarrow$	a_{22}	$\rightarrow$	AS (3)
:		:		:
AG (I, J)	$\rightarrow$	a_{ji}	$\rightarrow$	AS (J (J – 1)/2+I)
:		:		:
AG (N-1, N)	$\rightarrow$	a_{nn-1}	$\rightarrow$	AS (NT – 1)
AG (N, N)	$\rightarrow$	a_{nn}	$\rightarrow$	AS (NT)

B21-15-0602 CHBK2, DCHBK2

Back transformation of the eigenvectors of a complex Hessenberg matrix to the eigenvectors of a complex matrix
CALL CHBK2 (ZEV, K, N, IND, M, ZP, IP, DV, ICON)

Function

This subroutine back-transforms m eigenvectors of an n -order Hessenberg matrix $\boldsymbol{H}$ to eigenvectors of a complex matrix $\boldsymbol{A}$. $\boldsymbol{H}$ is assumed to be obtained from $\boldsymbol{A}$ using the stabilized elementary similarity transformation method. No eigenvectors of complex $\boldsymbol{A}$ are normalized. $n \geq 1$.

Parameters

ZEV ...	Input. m eigenvectors of complex Hessenberg matrix $\boldsymbol{H}$. Output. m_l eigenvectors of complex matrix $\boldsymbol{A}$. The m_l indicates the number of IND elements whose value is 1. ZEV is a two-dimensional array, ZEV (K,M)
K ..	Input. Adjustable dimension of arrays ZEV and ZP. ($\geq n$).
N ..	Input. Order n of complex matrices $\boldsymbol{A}$ and $\boldsymbol{H}$.
IND ...	Input. IND(J)=1 implies that the eigenvector corresponding to the j-th eigenvalue of ZEV is to be back-transformed. IND(J)=0 implies that the eigenvector corresponding to the j-th eigenvalue of ZEV is not to be back-transformed. IND is one-dimensional array of size M . See "Comments on use"
M ...	Input. Total number of eigenvalues corres-ponding to eigenvectors of complex Hessenberg matrix $\boldsymbol{H}$.
ZP ...	Input. Information necessary for a transfor-mation matrix ($\boldsymbol{A}$ to $\boldsymbol{H}$). ZP (K,N) is a two-dimensional complex array. See "Comments on use".
IP ...	Input. Information necessary for a transformation matrix ($\boldsymbol{A}$ to $\boldsymbol{H}$) . IP is a one-dimensional array of size n. See "Comments on use".
DV ...	Input. Scaling factor applied to balance complex matrix $\boldsymbol{A}$. DV is a one-dimensional array of size n. If balancing of complex matrix $\boldsymbol{A}$ was not performed, DV=0.0 can be specified. Therefore, DV does not necessarily have to be a one-dimensional array. See "Comments on use".
ICON ...	Output. Condition code. See Table CHBK2-l

Table CHBK2-1 Condition Codes

Code	Meaning	Processing
0	No error	
10000	N=1	ZEV(1,1)=(1.0,0.0)
30000	N <M,M < 1 or K < N	By passed

Comments on use

- Subroutines used
  SSL II ... MGSSL
  FORTRAN basic function ... None

- Notes
  After subroutine CHVEC is executed, parameters ZEV, IND and M can be used as input parameters for this subroutine.
  Parameters ZA and IP for subroutine CHES2 correspond to parameters ZP and IP for this subroutine and can be used as input parameters for this subroutine. For information about the contents of parameters ZP and IP, refer to the section on CHES2.
  For information about the contents of scaling factor DV, refer to the section on CBLNC.

- Example
  Eigenvectors and eigenvalues of an n -order complex matrix are calculated by using the following subroutines:

CBLNC	Balancing of a complex matrix
CHES2	Reduction to a complex Hessenberg matrix
CHSQR	Determination of eigenvalues of a complex Hessenberg matrix
CHVEC	Determination of eigenvectors of a complex Hessenberg matrix.
CHBK2	Back-transformation of eigenvectors of a complex Hessenberg matrix to eigenvectors of a complex matrix
CNRML	Nomalization of eigenvectors of a complex matrix.

  However eigenvectors are calculated in the order in which eigenvalues are determined. When $n \leq 100$:

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100),
     *ZAW(100,101),ZEV(100,100)
      DIMENSION IND(100),DV(100),IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(I,J),J=1,N)
      CALL CBLNC(ZA,100,N,DV,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL CHES2(ZA,100,N,IP,ICON)
```

```
    DO 30 J=1,N
    IM=MIN0(J+1,N)
    DO 30 I=1,IM
 30 ZAW(I,J)=ZA(I,J)
    CALL CHSQR(ZAW,100,N,ZE,M,ICON)
    WRITE(6,620) ICON
    IF(ICON.GE.20000) GO TO 10
    DO 40 I=1,M
 40 IND(I)=1
    CALL CHVEC(ZA,100,N,ZE,IND,M,
   *           ZEV,ZAW,ICON)
    WRITE(6,620) ICON
    IF(ICON.GE.20000) GO TO 10
    CALL CHBK2(ZEV,100,N,IND,M,ZA,IP,
   *           DV,ICON)
    CALL CNRML(ZEV,100,N,M,2,ICON)
    CALL CEPRT(ZE,ZEV,100,N,IND,M)
    GO TO 10
500 FORMAT(I5)
510 FORMAT(4E15.7)
600 FORMAT('1',10X,'ORIGINAL MATRIX'
   *//11X,'ORDER=',I5)
610 FORMAT(/2(5X,'A(',I3,',',I3,')=',
   *2E15.7))
620 FORMAT(/11X,'CONDITION CODE=',I5/)
    END
```

In this example, subroutine CEPRT is used to print the eigenvalues and corresponding eigenvectors of a complex matrix. For further information see the example of CEIG2.

Method

This subroutine back-transforms the eigenvectors of an *n*-order complex Hessenberg matrix ***H*** to the eigenvectors of balanced complex matrix ***A*** and then back-transforms the resultant eigenvectors to eigenvectors of an original complex matrix ***A***.

An complex matrix ***A*** is balanced by using the diagonal similarity transformation method shown in (4.1). The balanced complex matrix is reduced to a complex Hessenberg matrix ***H*** by using the (*n* – 2) stabilized elementary similarity transformations as shown in (4.2).

$$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D} \tag{4.1}$$

$$\boldsymbol{H} = \boldsymbol{S}_{n-2}^{-1} \cdots \boldsymbol{S}_2^{-1}\boldsymbol{S}_1^{-1}\tilde{\boldsymbol{A}}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2} \tag{4.2}$$

Where ***D*** is a diagonal matrix and $\boldsymbol{S}_i$ is represented by permutation matrix $\boldsymbol{P}_i$ and elimination matrix $\boldsymbol{N}_i$ as shown in (4.3).

$$\boldsymbol{S}_i = \boldsymbol{P}_i\boldsymbol{N}_i^{-1} \quad i = 1,2,...,n-2 \tag{4.3}$$

Let eigenvalues and eigenvectors of ***H*** by λ and ***y*** respectively and then obtain

$$\boldsymbol{H}\boldsymbol{y} = \lambda\boldsymbol{y} \tag{4.4}$$

From (4.1) and (4.2), (4.4) becomes:

$$\boldsymbol{S}_{n-2}^{-1} \cdots \boldsymbol{S}_2^{-1}\boldsymbol{S}_1^{-1}\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}\boldsymbol{y} = \lambda\boldsymbol{y} \tag{4.5}$$

If both sides of (4.5) are premultiplied by $\boldsymbol{D}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}$.

$$\boldsymbol{A}\boldsymbol{D}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}\boldsymbol{y} = \lambda\boldsymbol{D}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}\boldsymbol{y} \tag{4.6}$$

results and eigenvector ***x*** of ***A*** becomes:

$$\boldsymbol{x} = \boldsymbol{D}\boldsymbol{S}_1\boldsymbol{S}_2 \cdots \boldsymbol{S}_{n-2}\boldsymbol{y} \tag{4.7}$$

x is calculated as shown in (4.8) and (4.9) starting with $\boldsymbol{y}=\boldsymbol{x}_{n-1}$.

$$\boldsymbol{x}_i = \boldsymbol{S}_i\boldsymbol{x}_{i+1} = \boldsymbol{P}_i\boldsymbol{N}_i^{-1}\boldsymbol{x}_{i+1},\ i = n-2,...,2,1 \tag{4.8}$$

$$\boldsymbol{x} = \boldsymbol{D}\boldsymbol{x}_1 \tag{4.9}$$

For further information about stabilized elementary similarity transformation and balancing, see the section on CBLNC and CHES2.

For details see Reference [13] pp.339 - 358.

B21-15-0302 CHES2, DCHES2

Reduction of a complex matrix to a complex Hessenberg matrix (stabilized elementary similarity transformation)
CALL CHES2 (ZA, K, N, IP, ICON)

Function

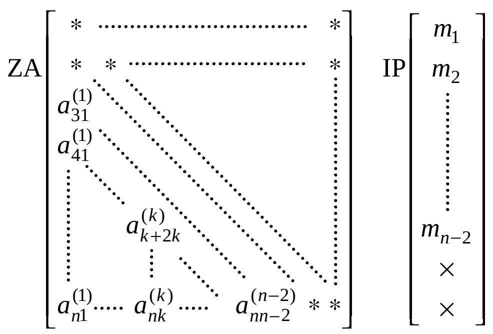
This subroutine reduces an n-order complex matrix $\boldsymbol{A}$ to a complex Hessenberg matrix $\boldsymbol{H}$ using the stabilized elementary similarity transformation method (Gaussian elimination method with partial pivoting) .

$$\boldsymbol{H} = \boldsymbol{S}^{-1}\boldsymbol{AS}$$

where $\boldsymbol{S}$ is a transformation matrix.
$n \geq 1$.

Parameters

ZA ...	Input. Complex matrix $\boldsymbol{A}$. Output. Complex Hessenberg matrix $\boldsymbol{H}$ and transformation matrix. See Figure CHES2-1. ZA is a complex two-dimentional array, ZA (K, N).
K ..	Input. Adjustable dimension of array ZA. ($\geq n$)
N ...	Input. Order n of complex matrix $\boldsymbol{A}$.
IP ...	Output. Information required by permutation matrix $\boldsymbol{S}$. (See Fig. CHES2-1.) IP is a one-dimensional array of size n.
ICON...	Output. See Table CHES2-1.



Note:
The section indicated with ∗ is the Hessenberg matrix. The rest contains some information for the transformation matrix. × indicates a work area.

Fig. CHES2-1 Array ZA and IP after transformation

Table CHES2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N =1 or N = 2	No transformation
30000	K < N or N < 1	Bypassed

Comments on use

- Subroutines used
  SSL II ... CSUM, AMACH, MGSSL
  FQRTRAN basic functions ... REAL, AIMAG, ABS, AMAX1

- Notes
  Output arrays ZA and IP are required to determine the eigenvectors of matrix $\boldsymbol{A}$.
  The precision of eigenvalues is determined in the complex Hessenberg transformation process. For that reason this subroutine has been implemented so that complex Hessenberg matrices can be determined as accurately as possible. However, if a complex matrix contains very large and very small eigenvalues, the precision of smaller eigenvalues is liable to be more affected by the reduction process – some smaller eigenvalues are difficult to determine precisely.

- Example
  This example transforms an n -order complex matrix to a complex Hessenberg matrix and determines its eigenvalues through use of the subroutine CHSQR.
  $n \leq 100$

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100)
      DIMENSION IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(1,J),J=1,N)
      CALL CHES2(ZA,100,N,IP,ICON)
      WRITE(6,620)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,610) ((I,J,ZA(I,J),
     *             J=1,N),I=1,N)
      CALL CHSQR(ZA,100,N,ZE,M,ICON)
      WRITE(6,640)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,650) (I,ZE(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',10X,'ORIGINAL MATRIX'
     * //11X,'ORDER=',I5/)
  610 FORMAT(/2(5X,'A(',I3,',',I3,
     * ')=',2E15.7))
  620 FORMAT('0'//11X,
     * 'HESSENBERG MATRIX')
  630 FORMAT(/11X,'CONDITION CODE=',I5/)
  640 FORMAT('0'/11X,'EIGENVALUES')
  650 FORMAT(5X,'E(',I3,')=',2E15.7)
      END
```

Method

An n-order complex matrix $\boldsymbol{A}$ is reduced to a complex Hessenberg matrix $\boldsymbol{H}$ through n-2 iterations of the stabilized elementary similarity transformation method.

$$\boldsymbol{A}_k = \boldsymbol{S}_k^{-1}\boldsymbol{A}_{k-1}\boldsymbol{S}_k,\ k = 1,2,...,n-2 \qquad (4.1)$$

where $\boldsymbol{A}_0=\boldsymbol{A}$.

When transformation is completed $\boldsymbol{A}_{n-2}$ is in the complex Hessenberg matrix.

The k-th transformation is performed according to the following procedure:

Let $\boldsymbol{A}_{k-1} = \left(a_{ij}^{(k-1)}\right)$

1) The elements $a_{ik}^{(k-1)}$, i=k+1,..., n in the k-th column are searched for the element of the maximum norm. The following is assumed as the norm of complex number $z = x + iy$:

   $\|z\|_1 = |x| + |y|$

   If a $a_{m_k k}^{(k-1)}$ is the element of the maximum norm, number m_k is stored as the k-th element of IP.
2) If $m_k = k$+1 is satisfied the next step is executed immediately.
   If $m_k > k$+1 is satisfied the k-th and subsequent columns elements in the m_k-th row are exchanged by the elements in the (k + 1)-th row of $\boldsymbol{A}_{k-1}$
3) Using $a_{k+1,k}^{(k-1)}$ as a pivot, all elements of the (k+2)-th and subsequent rows in the k-th column are eliminated through use of:

$$a_{ik}^{(k)} = -a_{ik}^{(k-1)} \big/ a_{k+1,k}^{(k-1)},\quad i = k+2,...,n \qquad (4.2)$$

   and

$$\tilde{a}_{ij}^{(k)} = a_{ij}^{(k-1)} + a_{ik}^{(k)}a_{k+1,j}^{(k-1)},\quad \begin{array}{l} i = k+2,...,n \\ j = k+1,...,n \end{array} \qquad (4.3)$$

4) If m_k=k+1 is satisfied the next step is executed.
   If $m_k > k$+1 is satisfied, all elements in the (k +1)-th column are exchanged by the m_k-th column.
5) At this step, each element in the (k +1)-th and subsequent columns is assumed as $\tilde{a}_{ij}^{(k)}$.

   All elements in the (k +1)-th column are modified as follows:

$$a_{ik+1}^{(k)} = \tilde{a}_{ik+1}^{(k)} - \sum_{j=k+2}^{n} \tilde{a}_{ij}^{(k)}a_{jk}^{(k)},\quad i = 1,2,...,n \qquad (4.4)$$

The row exchanging at the second step is performed by premultiplying $\boldsymbol{A}_{k-1}$ by permutation matrix $\boldsymbol{P}_k$ shown in Fig. CHES2-2. The elimination of elements at the third step performed by premultiplying $\boldsymbol{P}_k$ by elimination matrix $\boldsymbol{N}_k$ shown in Fig. CHES2-3. Therefore,

$$\boldsymbol{S}_k^{-1} = \boldsymbol{N}_k\boldsymbol{P}_k,\ \boldsymbol{S}_k = \boldsymbol{P}_k\boldsymbol{N}_k^{-1} \qquad (4.5)$$

result.

Where, $\boldsymbol{P}_k^{-1}$ is equal to $\boldsymbol{P}_k$, and $\boldsymbol{N}_k^{-1}$ is obtained by reversing all signs of non-diagonal elements of $\boldsymbol{N}_k$.

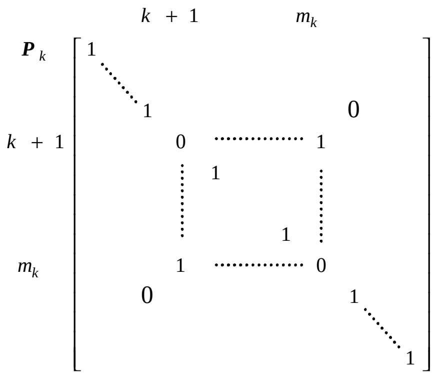
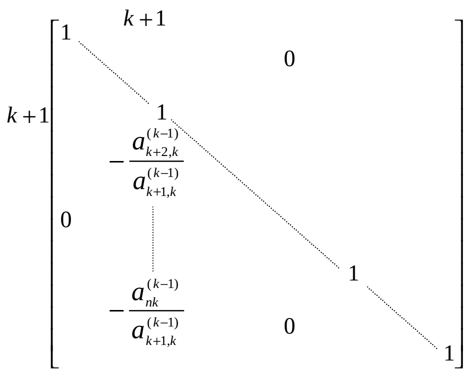


Fig. CHES2-2 Permutation matrix $\boldsymbol{P}_k$



$k+1$
$k+1$
1
0
1
$-\dfrac{a_{k+2,k}^{(k-1)}}{a_{k+1,k}^{(k-1)}}$
0
1
$-\dfrac{a_{nk}^{(k-1)}}{a_{k+1,k}^{(k-1)}}$
0
1

Note:

$a_{ij}^{(k-1)}$ is an element obtained after P_k is multiplied by N_k.

Fig. CHES2-3 Elimination matrix N_k

Information m_k necessary for $\boldsymbol{P}_k$ is stored as the k-th element of IP and all elements of the (k +2)-th and subsequent rows in the (k +1)-th columun of $\boldsymbol{N}_k$ are stored as $a_{ij}^{(k)}$ of the (k +2)-th and subsequent rows in the k-th column of $\boldsymbol{A}$.

If n = 2 or n = 1 is satisfied, no transformantion is performed.

For further information see Reference [13] pp.339 - 358.

B21-15-0402 CHSQR, DCHSQR

Eigenvalues of a complex Hessenberg matrix (QR method)
CALL CHSQR (ZA, K, N, ZE, M, ICON)

Function

This subroutine determines all eigenvalues of an n -order complex Hessenberg matrix $\boldsymbol{A}$ by using the QR method. $n \geq 1$.

Parameters

ZA ...	Input. Complex Hessenberg matrix $\boldsymbol{A}$. The contents of $\boldsymbol{A}$ are altered on output. ZA is a two-dimensional complex array, ZA (K,N)
K ..	Input. Adjustable dimension of array ZA.($\geq n$)
N ...	Input. Order n of the complex Hessenberg matrix $\boldsymbol{A}$.
ZE ...	Output. Eigenvalues. The J-th eigenvalue is ZE (J) (J = 1 ,2,...M) . ZE is a one-dimensional complex array of size n.
M ...	Output. Number of determined eigenvalues.
ICON ...	Output. Condition code. See Table CHSQR-1.

Table CHSQR-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	ZE (1) = ZA (1, 1)
15000	Any of the eigenvalues could not be determined.	The number of eigenvalues that were obtained is set to M.
20000	No eigenvalues could be determined.	M is set to zero.
30000	K < N or N < 1	Bypassed

Comments on use

- Subroutines used

  SSL II ... AMACH, MGSSL

  FORTRAN basic functions ... REAL, AIMAG, CONJG, ABS, SIGN, AMAXl, SQRT, CSQRT

- Notes

  Normally, this subroutine is used to determine all eigenvalues after CHES2 has been executed.

  If eigenvectors are also needed array ZA should be copied onto another area before this subroutine is called.

- Example

  This example reduces an n -order complex matrix to a complex Hessenberg matrix through use of CHES2 and then determines its eigenvalues. $n \leq 100$.

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100)
      DIMENSION IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(I,J),J=1,N)
      CALL CHES2(ZA,100,N,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      CALL CHSQR(ZA,100,N,ZE,M,ICON)
      WRITE(6,630)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,640) (I,ZE(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',10X,
     * '** ORIGINAL MATRIX'//11X,
     * '** ORDER=',I5/)
  610 FORMAT(/2(5X,'A(',I3,',',I3,
     * ')=',2E15.7))
  620 FORMAT(/11X,'** CONDITION CODE=',
     * I5/)
  630 FORMAT('0'/11X,'** EIGENVALUES')
  640 FORMAT(5X,'E(',I3,')=',2E15.7)
      END
```

Method

In the QR method, the diagonal elements become eigenvalues by making the lower subdiagonal elements of complex Hessenberg matrix $\boldsymbol{A}$ converge to zero. To accomplish this the unitary similarity transformation shown in (4.1) is applied repeatedly.

$$\boldsymbol{A}_{s+1} = \boldsymbol{Q}_s^* \boldsymbol{A}_s \boldsymbol{Q}_s \ , \ s = 1,2,... \tag{4.1}$$

where $\boldsymbol{A}_1=\boldsymbol{A}$.

$\boldsymbol{Q}_s$ is a unitary matrix which is uniquely determined in the QR decomposition shown in (4.2).

$$\boldsymbol{A}_s = \boldsymbol{Q}_s \boldsymbol{R}_s \tag{4.2}$$

where $\boldsymbol{R}_s$ is an upper triangular matrix whose diagonal elements are positive real numbers.

To improve the rate of convergence, QR decomposition is normally applid to origin-shifted matrix ($\boldsymbol{A}_s$ - $k_s\boldsymbol{I}$) instead of $\boldsymbol{A}_s$. k_s is the origin shift.

$\boldsymbol{Q}_s$ and $\boldsymbol{R}_s$ are obtained from (4.3) instead of (4.2).

$$\boldsymbol{A}_s - k_s \boldsymbol{I} = \boldsymbol{Q}_s \boldsymbol{R}_s \tag{4.3}$$

The process for the complex QR methods is described below.

Let $\boldsymbol{A}_s = \left(a_{ij}^{(s)}\right), \boldsymbol{A} = \left(a_{ij}\right)$.

1) (4.4) determines whether there are elements which can be regarded as relative zero among lower subdiagonal elements $a_{n,n-1}^{(s)}, \ldots, a_{21}^{(s)}$ of $\boldsymbol{A}_s$.

$$\left\|a_{l,l-1}^{(s)}\right\|_1 < u\|\boldsymbol{A}\| \quad ,l = n,n-1,\ldots,2 \qquad (4.4)$$

u is the unit round-off. $\|\ \|_1$ is a norm defined for complex number z= $x + iy$ as:

$$\|z\|_1 = |x| + |y| \qquad (4.5)$$

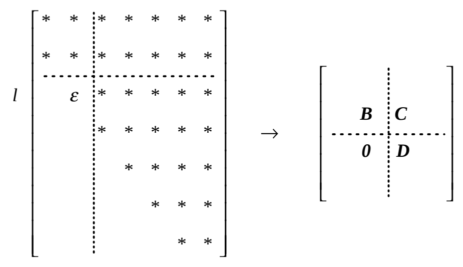
$\|\boldsymbol{A}\|$ is a norm defined as:

$$\|A\| = \max_j \sum_{i=1}^{n} \left\|a_{ij}\right\|_1 \qquad (4.6)$$

$a_{l,l-1}^{(s)}$ is relative zero if it satisfies (4.4). If it does not satisfy (4.4), step 2) is performed.

(a) If $l = n$, $a_{nn}^{(s)}$ is adopted as an eigenvalue, order n of the matrix is reduced to $n - 1$, and the process returns to step 1).
If n = 0, the process is terminated.

(b) When $2 \le l \le n - 1$, the matrix is split as shown in Fig. CHSQR-1, and the process proceeds to step 2) assuming submatrix $\boldsymbol{D}$ as $\boldsymbol{A}_s$.



Note: Element ε is regarded as zero.

Fig. CHSQR-1 A direct sum of submatrices for a Hessenberg matrix

2) Origin shift k_s corresponds to the eigenvalue of lowest norm $\|\ \|_1$ in the lowest 2×2 principle submatrix of $\boldsymbol{A}_s$. $\boldsymbol{A}_s - k_s\boldsymbol{I}$ is obtained by subtracting k_s from diagonal elements of $\boldsymbol{A}_s$.

3) By applying two-dimensional Givens unitary transformation $\boldsymbol{P}_l^*$ (for l =1, 2, . .., n –1) to $\boldsymbol{A}_s - k_s\boldsymbol{I}$ is transformed to upper triangular matrix $\boldsymbol{R}_s$:

$$\boldsymbol{P}_{n-1}^*\boldsymbol{P}_{n-2}^* \cdots \boldsymbol{P}_2^*\boldsymbol{P}_1^*\left(\boldsymbol{A}_s - k_s\boldsymbol{I}\right) = \boldsymbol{R}_s \qquad (4.7)$$

Therefore ,

$$\boldsymbol{Q}_s = \boldsymbol{P}_1\boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-1} \qquad (4.8).$$

Unitary matrix $\boldsymbol{P}_l$ is determined by using the l -th diagonal element x of matrix $\boldsymbol{A}_s$ - $k_s\boldsymbol{I}$ and its subdiagonal element y as shown below:

$$\boldsymbol{P}_l = \begin{array}{c} \\ \left[\begin{array}{cccccc} 1 & & & & & \\ & \ddots & & & 0 & \\ & & 1 & & & \\ & & & \bar{c} & -\bar{s} & & \\ & & & s & c & & \\ & & & & & 1 & \\ & 0 & & & & & \ddots \\ & & & & & & 1 \end{array}\right] \end{array} \begin{array}{l} \\ \\ \\ l \\ l+1 \\ \\ \\ \\ \end{array}$$

with columns l, $l+1$ labelled above.

where :

$$c = \overline{x}/r \qquad (4.9)$$
$$c = y/r \qquad (4.10)$$
$$r = \sqrt{|x|^2 + |y|^2} \qquad (4.11)$$

4) After two-dimensional unitary transformation $\boldsymbol{P}_l$ is applied to $\boldsymbol{R}_s$, $\boldsymbol{A}_{s+1}$ is obtained by adding origin shift k_s to diagonal elements.

$$\boldsymbol{A}_{s+1} = \boldsymbol{R}_s\boldsymbol{P}_1\boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-1} + k_s\boldsymbol{I} \qquad (4.12)$$

This subroutine executes step 3) in combination with step 4) to increasse computational speed and reduce storage requirements.

For further information see References [12] and [13] pp.372-395.

B21-15-0502 CHVEC,DCHVEC

Eigenvectors of a complex Hessenberg matrix (Inverse iteration method)
CALL CHVEC (ZA, K, N, ZE, IND, M, ZEV, ZAW, ICON)

Function

This subroutine determines eigenvector ***x*** which corresponds to selected eigenvalue μ of an n -order complex Hessenberg matrix by using the inverse iteration method. However no eigenvectors are normalized. $n \geq 1$.

Parameters

ZA ...	Input. Complex Hessenberg matrix ***A***. ZA is a two-dimensional complex array, ZA (K, N).
K ...	Input. Adjustable dimension of arrays ZA, EV and ZAW.
N ...	Input. Order n of the complex Hessenberg matrix ***A***.
ZE ...	Input. Eigenvalue μ. The j-th eigenvalue μ_j is stored in ZE (j). ZE is a one-dimensional complex array of size M.
IND ...	Input. Indicates whether or not eigenvectors are to be determined. If the eigenvector corresponding to the j-th eigenvalue μ_j is to be determined, IND (j) = 1 must be specified. If it is not to be determined IND (j) = 0 must be specified. IND is a one-dimensional array of size M.
M ..	Input. Total number of eigenvalues stored in array ZE ($\leq n$)
ZEV ...	Output. Eigenvectors are stored in columns of ZEV. ZEV is a two-dimensional complex array, ZEV (K, MK). MK indicates the number of eigenvectors to be determined. See "Comments on use".
ZAW ...	Work area. ZAW is a two-dimensional complex array, ZAW (K, N + 1) .
ICON ...	Output. condition code. See Table CHVEC- 1 .

Table CHVEC-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	ZEV (1, 1) = (1.0, 0.0)
15000	An eigenvector corresponding to a specified eigenvalue could not be determined.	IND information about the eigenvector that could not be determined is set to zero.
20000	No eigenvectors could be determined.	All IND information are set to zero.
30000	M < 1, N < M or K < N	Bypassed

Comments on use

- Subroutines used

  SSL II ... AMACH, CSUM, MGSSL

  FORTRAN basic functions ... REAL, AIMAG, ABS, MIN0, AMAXl, FLOAT, SQRT

- Notes

  The number of eigenvectors (MK) indicates the number of IND elements whose value is 1.

  Since IND elements are set to zero if any eigenvector cannot be determined, MK indicates the number of eigenvectors which does not exceed the number of eigenvectors specified before computation.

  The eigenvalues used by this subroutine can be obtained by the CHSQR subroutine.

  When they are determined by CHSQR, the parameters ZE and M can be used as input parameters for this subroutine.

  This subroutine can be used to determine eigenvectors of complex Hessenberg matrices only.

  When selected eigenvectors of a complex Hessenberg matrix are to be determined:

  (a) The complex matrix is first reduced into a complex Hessenberg matrix by the subroutine CHES2.

  (b) The eigenvalues of the complex matrix are determined by the subroutine CHSQR.

  (c) The eigenvectors of the complex Hessenberg matrix are determined by this subroutine.

  (d) The above eigenvectors are back-transformed to eigenvectors of the complex matrix by the subroutne CHBK2.

  However the subroutne CEIG2 should be utilized to determine all eigenvalues and corresponding eigenvectors of a complex matrix for convenience.

  This subroutine can be used to determine some of the eigenvectors of a complex matrix according to the procedure described above. Therefore, the eigenvectors are not normalized. If necessary, they must be normalized by the subroutine CNRML subsequently.

  When the subroutines CHBK2 and CNRML are used, the parameters IND, M and ZEV can be used as input parameters for the subroutines CHBK2 and CNRML.

- Example

  This example determines eigenvalues of an n-order complex matrix through use of the subroutines CHES2 and CHSQR and obtains eigenvectors through use of the subroutines CHVEC and CHBK2.

  The obtained eigenvectors are normalized by the subroutine CNRML ($\|\boldsymbol{x}\|_2$=1). $n \leq 100$.

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZAW(100,101),
     *        ZE(100),ZEV(100,100)
      DIMENSION IND(100),IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(I,J),J=1,N)
      CALL CHES2(ZA,100,N,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      DO 30 J=1,N
      IM=MIN0(J+1,N)
      DO 30 I=1,IM
   30 ZAW(I,J)=ZA(I,J)
      CALL CHSQR(ZAW,100,N,ZE,M,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      DO 40 I=1,M
   40 IND(I)=1
      CALL CHVEC(ZA,100,N,ZE,IND,M,ZEV,
     *           ZAW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL CHBK2(ZEV,100,N,IND,M,ZA,
     *           IP,0.0,ICON)
      CALL CNRML(ZEV,100,N,IND,M,2,ICON)
      CALL CEPRT(ZE,ZEV,100,N,IND,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',
     *       5X,'N=',I3)
  610 FORMAT(/2(5X,'A(',I3,',',I3,')=',
     *       2E15.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

The subroutine CEPRT used in this example prints the eigenvalues and eigenvectors of the complex matrix. For further information see an example of subroutine CEIG2.

Method

The inverse iteration method is used to determine eigenvector $\boldsymbol{x}$ corresponding to eigenvalue λ of an n - order complex Hessenberg matrix.

In the inverse iteration method, if matrix $\boldsymbol{A}$ and approximation μ_j for eigenvalue λ_j of $\boldsymbol{A}$ are given, an appropriate initial vector $\boldsymbol{x}_0$ is used to solve equation (4.1) iteratively.

When convergence conditions have been satisfied, $\boldsymbol{x}_r$ is determined as the resultant eigenvector.

$$(\boldsymbol{A}-\mu_j\boldsymbol{I})\boldsymbol{x}_r = \boldsymbol{x}_{r-1} \quad ,r=1,2,... \tag{4.1}$$

Now let the eigenvalue of n-order matrix $\boldsymbol{A}$ be λ_i (i=1,2,...,n), and the eigenvector corresponding to λ_i be $\boldsymbol{u}_i$.

Since the appropriate initial vector $\boldsymbol{x}_0$ can be expressed by the linear combination of eigenvector $\boldsymbol{u}_i$ of matrix $\boldsymbol{A}$ as shown in (4.2), $\boldsymbol{x}_r$ can be written as shown in (4.3) if all eigenvalues λ_i are different.

$$\boldsymbol{x}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{u}_i \tag{4.2}$$

$$\boldsymbol{x}_r = 1/(\lambda_j-\mu_j)^r \cdot\left[\alpha_j\mu_j + \sum_{\substack{i=1\\ i\neq j}}^{n} \alpha_i \boldsymbol{u}_i (\lambda_j-\mu_j)^r/(\lambda_i-\mu_i)^r\right] \tag{4.3}$$

Since

$$\left|(\lambda_j-\mu_j)/(\lambda_i-\mu_i)\right| \langle\langle 1.0$$

is established if i is not equal to j, (4.3) indicates that if $\alpha_j \neq 0$, $\boldsymbol{x}_r$ tends rapidly to approach a multiple of $\boldsymbol{u}_j$ as r grows greater.

The system of linear equations shown in (4.1) is solved by using (4.4) after decomposition of $(\boldsymbol{A}-\mu_j\boldsymbol{I})$ to a lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{LUx}_r = \boldsymbol{Px}_{r-1} \tag{4.4}$$

where $\boldsymbol{P}$ is the permutation matrix used for pivoting (4.4) can be solved as follows:

$$\boldsymbol{Ly}_{r-1} = \boldsymbol{Px}_{r-1} \qquad \text{(Forward substitution)} \tag{4.5}$$

$$\boldsymbol{Ux}_r = \boldsymbol{y}_{r-1} \qquad \text{(Backward substitution)} \tag{4.6}$$

Since any vector may be used for initial vector $\boldsymbol{x}_0$, $\boldsymbol{x}_0$ may be given such that $\boldsymbol{y}_0$ of (4.7) has a form such as $\boldsymbol{y}_0=(1,1,...,1)^{\mathrm{T}}$

$$\boldsymbol{y}_0 = \boldsymbol{L}^{-1}\boldsymbol{Px}_0 \tag{4.7}$$

Therefore, for the first iteration, the forward substitution in (4.5) can be omitted. In general, the eigenvectors can be obtained by repeating forward substitution and backeard substitution for the second and following iterations.

This subroutine uses the inverse iteration method as described below:

- Selection of the initial vector
  $\boldsymbol{y}_0$ in (4.8) is used for the initial vector.

  $$\boldsymbol{y}_0 = \mathrm{EPS1}\cdot\boldsymbol{r} \tag{4.8}$$

  where vector $\boldsymbol{r}$ is defined as shown in (4.9) and $\boldsymbol{\Phi}$ is a golden section ratio shown in (4.10).

$$r_k = k\boldsymbol{\Phi} - [k\boldsymbol{\Phi}] \quad ,k=1,2,... \tag{4.9}$$

$$\boldsymbol{\Phi} = (\sqrt{5}+1)/2 \tag{4.10}$$

EPS1 can be expressed as shown in (4.11).

$$\mathrm{EPS1} = u\cdot\|\boldsymbol{A}\| \tag{4.11}$$

where u is the unit round-off.

Norm $\|\boldsymbol{A}\|$ can be expressed as shown in (4.12).

$$\|\boldsymbol{A}\| = \max_j \cdot \sum_{i=1}^{n} \|a_{ij}\|_1 \qquad (4.\ 12)$$

For complex number $z = x + iy$, norm $\|z\|$ is assumed as shown in (4.13).

$$\|z\|_1 = |x| + |y| \qquad (4.\ 13)$$

- Normalization of the iterated vector
  Let

$$\boldsymbol{x}_r = (x_{r1}, x_{r2}, ..., x_{rn})^{\mathrm{T}}$$

then the iterated vector is normalized as shown in (4.14) for $r \geq 1$.

$$\|\boldsymbol{x}_r\|_1 = \sum_{i=1}^{n} \|x_{ri}\|_1 = \mathrm{EPS1} \qquad (4.14)$$

- Method for convergence criterion
  After backward substitution, $\boldsymbol{x}_r$ (before normalization) is tested by (4.15) to determine whether the eigenvectors have been accepted.

$$\|\boldsymbol{x}_r\|_1 \geq 0.1 / \sqrt{n} \qquad (4.15)$$

If (4.15) is satisfied, $\boldsymbol{x}_r$ is accepted as the eigenvectors.
If it is not satisfied, normalization described in second step is repeated up to five times. When convergence is not successful, the eigenvector is ignored and the next eigenvector is processed.

- Measures to be taken when eigenvalues have multiple roots or close roots
  Since this subroutine produces initial vector $\boldsymbol{y}_0$ using random numbers, you need not take special measures when an eigenvalue has multiple roots or close roots.
  To make a reproduction of numerical solutions, the random numbers are initialized each time this subroutine is called.

For further information see Reference [13] pp.418-439.

C22-15-0101 CJART, DCJART

Zeros of a polynomial with complex coefficients (Jarratt method)
CALL CJART (ZA, N, Z, ICON)

Function

This subroutine finds zeros of a polynomial with complex coefficients

$$a_0 z^n + a_1 z^{n-1} + \cdots + a_n = 0 \qquad (1.1)$$

(a_i : complex number, $|a_0| \neq 0$)

by the Jarratt method.

Parameters

ZA ..... Input. Coefficients of the equation.
ZA is a complex one-dimensional array of size $n + 1$, Where ZA (1) = a_0, ZA (2) = a_1, ... , ZA(N+1)=a_n , The contents of ZA are altered on output.

N ..... Input. Order n of the equation.
Output. Number of roots that were obtained. (See "Comments on use").

Z ... Output. n roots.
Z is a complex one-dimensional array of size n. The roots are output in Z (1), Z (2), ... in the order that they were obtained.
Thus. if N roots are obtained, they are stored in Z (1) to Z (N).

ICON ... Output. Condition code. See Table CJART-1.

Table CJART-l Condition codes

Code	Meaning	Processing
0	No error	
10000	All n roots could not be obtained.	The number of roots which were obtained is output to the parameter N and the roots are output to Z(1) through Z(N).
30000	n < 1 or | a_0 |=0.	Bypassed

Comments on use

Subprogram used
SSL II ... AMACH, CQDR, UCJAR, and MGSSL
FORTRAN basic functions ... CMPLX, SQRT, CABS, CONJG, AIMAG, REAL and ABS.

- Notes
For arrays ZA and Z, COMPLX must be declared in a program which calls this subroutine.
When n is 1 or 2, instead of Jarratt's method, the root formula is used.
An n-th degree polynomial equation has n roots, however it is possible, though rare, that all of these roots can not be found. The user must check the parameters ICON and N on output to confirm that all of roots were found.

- Example
The degree n and complex coefficients a_i are input, and the roots are calculated. $1 \le n \le 50$.

```
C      **EXAMPLE**
       DIMENSION ZA(51),Z(50)
       COMPLEX ZA,Z
       READ(5,500) N
       N1=N+1
       READ(5,510) (ZA(I),I=1,N1)
       DO 10 I=1,N1
       K=I-1
   10  WRITE(6,600) K,ZA(I)
       CALL CJART(ZA,N,Z,ICON)
       WRITE(6,610) N,ICON
       IF(ICON.EQ.30000) STOP
       WRITE(6,620) (I,Z(I),I=1,N)
       STOP
  500  FORMAT(I2)
  510  FORMAT(2F10.0)
  600  FORMAT(10X,'A(',I2,')=',2E20.8)
  610  FORMAT(10X,'N=',I2,5X,'ICON=',I5)
  620  FORMAT(10X,'Z(',I2,')=',2E20.8)
       END
```

Method

This subroutine uses a slightly modified version of Garside-Jarratt-Mack method (an iterative method). Let the polynomial equation be

$$f(z) \equiv a_0 z^n + a_1 z^{n-1} + \cdots + a_n = 0 \qquad (4.1)$$

then its roots match those of

$$1/F(z) \equiv f(z)/f'(z) = 0 \qquad (4.2)$$

This subroutine makes use of the faster convergence property of (4.2) since it has only simple roots.

- The iterative formula
Suppose that z_1, z_2, and z_3 are three initial values to a root and they satisfy $|f(a_3)| \le |f(a_2)| \le |f(a_1)|$. Then the following three methods are used as iterative formulas. In the following, let $F_i \equiv F(z_i)$ for i = l, 2, and 3.

Method 1:
Near a root of a n-th degree polynomial $f(z)$, we can write

$$1/F(z) \approx (z-a)/(b+cz) \qquad (4.3)$$

where a, b, and c are constants
Thus, a new root can be obtained by choosing a such that (4.3) is true for z_1, z_2, and z_3. Namely, a is obtained as

$$a = z_3 + \frac{(z_2 - z_3)(z_3 - z_1)(F_2 - F_1)}{(z_3 - z_2)(F_2 - F_1) + (z_1 - z_2)(F_3 - F_2)} \qquad (4.4)$$

Method 2 (Newton’s method):
The following *a*’ can be used as a new root.

$$a' = z_3 - 1/F_3 \qquad (4.5)$$

Method 3 (modified Newton’s method):
Suppose that .

$$|f(z_3)| < \sqrt{u_1|a_n|} \qquad (4.6)$$

is satisfied, where $u_1 = \sqrt{u}$ and *u* is the round-off unit. Let *m* be the rough multiplicity of current root, then we can write $m \approx (z_3 - a)F_3$ and the following

$$a'' = z_3 - m/F_3 \qquad (4.7)$$

Usually, Method 1 and Method 2 are both used, and the result with the smallest adjustment is used as the new approximate root a_4. The Method 3 is used to increase the speed of convergence.

- Choosing initial values

$$|a_n/a_0|^{1/n} = 5w \qquad (4.8)$$

With (4.8), initial values z_1, z_2, z_3 are setected according to

$$iw, -w + iw, 2iw \qquad (4.9)$$

They are ordered such that $|f(z_3)| \le |f(z_2)| \le |f(z_1)|$
If convergence does not occur after 50 iterations, new initial values are selected as follows:

$$-3w + 2iw, -2w + 2iw, -3w + 3iw$$

When a root α is obtained, the degree of the equation is reduced by setting.

$$g(z) = f(z)/(z - \alpha)$$

Using $g(z)$ as the new $f(z)$, *w* is recalculated and

$$-w \pm iw, -w \pm 2iw, \overline{\alpha} \qquad (4.10)$$

are used as initial values for solving $f(z)=0$. The sign is selected to match the sign of the imaginary part of $\overline{\alpha}$.

- Preventing overflows
  To prevent overflow when evaluating $f(z)$ and $f'(z)$, the coefficients of $f(z)$ are normalized by dividing them with the arithmetic mean of their absolute values.

- Convergence criterion
  If

$$|f(z_3)| \le 10nu \sum_{i=0}^{n} \left|a_i z_3^{n-i}\right| \qquad (4.11)$$

is satisfied, z_3 is then taken as a root of $f(z)$.

For further information, see References [26] and [27].

A22-15-0202 CLU, DCLU

LU-decomposition of a complex general matrix (Crout's method)
CALL CLU (ZA, K, N, EPSZ, IP, IS, ZVW, ICON)

Function

An $n \times n$ complex general matrix $\boldsymbol{A}$ is LU-decomposed using the Crout's method

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{1.1}$$

Where $\boldsymbol{P}$ is a permutation matrix which performs the row exchanged required in partial pivoting, $\boldsymbol{L}$ is a lower triangular matrix, and $\boldsymbol{U}$ is a unit upper triangular matrix. $n \geq 1$.

Parameter

ZA	Input. Matrix $\boldsymbol{A}$ Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$. See Fig. CLU-1. ZA is a complex two-dimensional array, ZA(K, N).
K	Input. Adjustables dimension of array ZA (≥ N).
N	Input. Order n of matrix $\boldsymbol{A}$.
EPSZ ..	Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ (≥0.0). When EPSZ is 0.0, a standard value is used. (See Notes.)
IP....	Output. The transposition vector which indicates the history of row exchanging that occurred in partial pivoting. IP is a one-dimensional array of size n. (See to Notes.) .
IS....	Output. Information for obtaining the determinant of matrix $\boldsymbol{A}$. If the n calculated diagonal elements of array ZA are multiplied by IS, the determinant is obtained.
ZVW ...	Work area. ZVW is a complex one-dimensional array of size n.
ICON ..	Output. Condition code. See Table CLU-1.

Table CLU-l Condition codes

Code	Meaning	Processing
0	No error	
20000	Either all of the elements of some row in matrix A were zero or the pivot became relatively zero. It is highly probable that the matrix is singular.	Discontinued
30000	K < N, N < 1 or EPSZ < 0.0	By passed

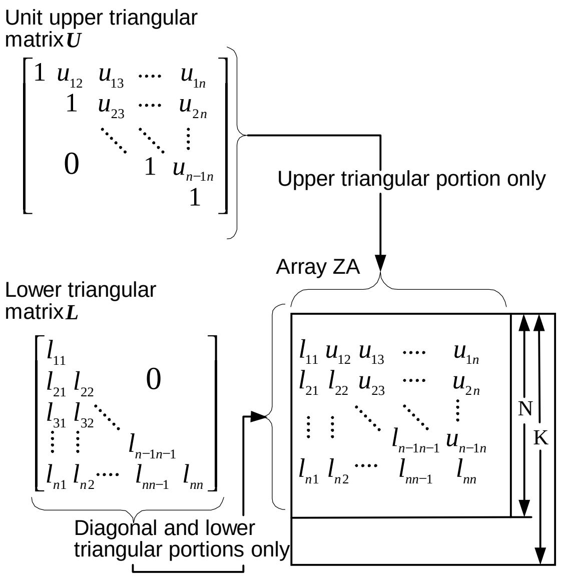

Unit upper triangular matrix $\boldsymbol{U}$
$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & \ddots & \vdots \\ & 0 & & 1 & u_{n-1n} \\ & & & & 1 \end{bmatrix}$
Upper triangular portion only
Array ZA
Lower triangular matrix $\boldsymbol{L}$
$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & 0 & \\ l_{31} & l_{32} & \ddots & & \\ \vdots & \vdots & & l_{n-1n-1} & \\ l_{n1} & l_{n2} & \cdots & l_{nn-1} & l_{nn} \end{bmatrix}$
$\begin{matrix} l_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ & & & l_{n-1n-1} & u_{n-1n} \\ l_{n1} & l_{n2} & \cdots & l_{nn-1} & l_{nn} \end{matrix}$
N
K
Diagonal and lower triangular portions only

Fig. CLU-1 Storage of factors $\boldsymbol{L}$ and $\boldsymbol{U}$ in array ZA

Comments on use

- Subprograms used
  SSL II ..... AMACH, CSUM, MGSSL
  FORTRAN basic functions ..... REAL, AIMAG, ABS

- Notes
  If EPSZ is set to 10^{-s}, this value has the following meaning: while performing the LU-decomposition by Crout's method, if the loss over s significant digits occured for both real and imaginaty parts of the pivot, the LU-decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero. Let u be the unit round-off, then the standard value of EPSZ is l6u. If the processing is to proceed at a lower pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.
  The transposition vector corresponds to the permutation matrix $\boldsymbol{P}$ of LU decomposition in partial pivoting. In this subroutine, with partial pivoting, the elements of array ZA are actually exchanged. In other words, if the I-th row has been selected as the pivotal row in the J-th stage (J = 1 , ..., n) of decomposition, the elements has the I-th and J-th rows of array ZA are exchanged. The history of this exchange is recorded in IP by storing I in IP(J). A system of linear equations can be solved by calling subroutine CLUX following this subroutine. However, instead of calling these two subroutines, subroutine LCX is usually called to solve such equations in one step.

- Example
  An $n \times n$ complex matrix is LU decomposed. $n \leq 100$.

```
C      **EXAMPLE**
       DIMENSION ZA(100,100),ZVW(100),IP(100)
       COMPLEX ZA,ZVW,ZDET
       READ(5,500) N
       IF(N.LE.0) STOP
       READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
       WRITE(6,600) N,((I,J,ZA(I,J),J=1,N),
      *             I=1,N)
       CALL CLU(ZA,100,N,0.0,IP,IS,ZVW,ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) STOP
       ZDET=CMPLX(FLOAT(IS),0.0)
       DO 10 I=1,N
       ZDET=ZDET*ZA(I,I)
   10  CONTINUE
       WRITE(6,620) (I,IP(I),I=1,N)
       WRITE(6,630) ((I,J,ZA(I,J),J=1,N),
      *I=1,N)
       WRITE(6,640) ZDET
       STOP
  500  FORMAT(I5)
  510  FORMAT(5(2F8.3))
  600  FORMAT('1'//10X,'**INPUT MATRIX **',
      * /12X,'ORDER=',I5/(5X,
      * 2('(',I3,',',I3,')',2E15.8,2X)))
  610  FORMAT('0',10X,'CONDITION CODE =',I5)
  620  FORMAT('0',10X,'TRANSPOSITION VECTOR',
      * /(10X,7('(',I3,')',I5,5X)))
  630  FORMAT('0',10X,'OUTPUT MATRIX',
      * /(5X,2('(',I3,',',I3,')',2E15.8,2X)))
  640  FORMAT('0',10X, 'DETERMINANT OF ',
      * 'MATRIX',2E20.8)
       END
```

Method

- Crout's method

  Generally, in exchanging rows using partial pivoting, an $n \times n$ non-singular complex general matrix $\boldsymbol{A}$ can be decomposed into a lower triangular matrix $\boldsymbol{L}$ and a unit upper triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{PA} = \boldsymbol{LU} \qquad (4.1)$$

$\boldsymbol{P}$ is the permutation matirx which performs the row-exchanging required in partial pivoting. The Crout's method is one method to obtain the elements of $\boldsymbol{L}$ and $\boldsymbol{U}$. This subroutine obtains values in the j-th column of $\boldsymbol{L}$ and j-th column of $\boldsymbol{U}$ in the order (j = l, ... , n) using the following equations.

$$u_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right) \Bigg/ l_{ii} \quad , i = 1, \ldots, j-1 \qquad (4.2)$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \quad , i = j, \ldots, n \qquad (4.3)$$

where, $\boldsymbol{A}$ = (a_{ij}) , $\boldsymbol{L}$ = (l_{ij}) and $\boldsymbol{U}$ = (u_{ij}) Actually, using partial pivoting, rows are exchanged. The Crout's method is a variation of the Gaussian elimination method. The same caluculations are involved, but the seqnece is differenct. In the Crout's method, the elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ are calculated at the same time using equations (4.2) and (4.3). By increasing the precision of the inner products in this step, the effects of rounding errors are minimized.

- Partial pivoting

  When the matrix $\boldsymbol{A}$ is given as

$$\boldsymbol{A} = \begin{bmatrix} 0.0 + i \cdot 0.0 & 1.0 + i \cdot 0.0 \\ 1.0 + i \cdot 0.0 & 0.0 + i \cdot 0.0 \end{bmatrix}$$

Though this matrix is numerically stable, LU decomposition can not be performed. In this state, even if a matrix is numerically stable, large errors would occur if LU decomposition were directly computed. To avoid these errors, partial pivoting with row equalibration is used. For more details, see References [1], [3], and [4].

A22-15-0602 CLUIV, DCLUIV

The inverse of a complex general matrix decomposed into the factors *L* and *U*
CALL CLUIV (ZFA, K, N, IP, ICON)

Function

The inverse $\boldsymbol{A}^{-1}$ of an $n \times n$ complex general matrix $\boldsymbol{A}$ given in decomposed form $\boldsymbol{PA} = \boldsymbol{LU}$ is computed.

$$\boldsymbol{A}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$$

Where $\boldsymbol{L}$ and $\boldsymbol{U}$ are respectively an $n \times n$ lower triangular and a unit upper triangular matrices and $\boldsymbol{P}$ is a permutation matrix which performs the row exchanges in partial pivoting for LU decomposition. $n \geq 1$.

Parameters

ZFA	Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ Output. Inverse $\boldsymbol{A}^{-1}$. ZFA is a two-dimensional array, FA (K, N). See Fig. CLUIV-1.
K	Input. Adjustable dimension of array ZFA (≥N).
N	Input. Order n of the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.
IP	Input. Transposition vector which indicates the history of row exchanges in partial pivoting. One-dimensional array of size n
ICOM ..	Output. Condition code. See Table CLUIV-1.

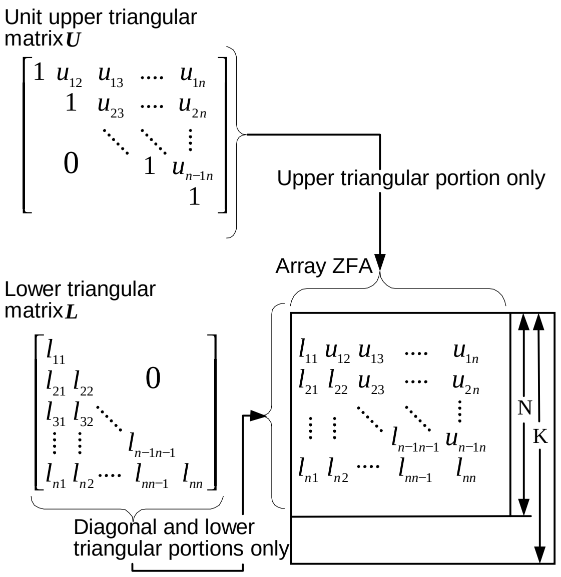

Unit upper triangular matrix $\boldsymbol{U}$
$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & \ddots & \vdots \\ & 0 & & 1 & u_{n-1n} \\ & & & & 1 \end{bmatrix}$
Upper triangular portion only
Array ZFA
Lower triangular matrix $\boldsymbol{L}$
$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & 0 & \\ l_{31} & l_{32} & \ddots & \\ \vdots & \vdots & l_{n-1n-1} & \\ l_{n1} & l_{n2} \cdots & l_{nn-1} & l_{nn} \end{bmatrix}$
l_{11} u_{12} u_{13} $\cdots$ u_{1n}
l_{21} l_{22} u_{23} $\cdots$ u_{2n}
l_{n-1n-1} u_{n-1n}
l_{n1} l_{n2} $\cdots$ l_{nn-1} l_{nn}
N
K
Diagonal and lower triangular portions only

Fig. CLUIV-1 Storage of the elements of *L* and *U* is array ZFA

Table CLUIV-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Matrix was singular.	Discontinued
30000	K < N or N < 1 or there was an error in IP.	Bypassed

Comments on use

- Subprogram used
  SSL II ..... CSUM, MGSSL
  FORTRAN basic function ..... None.

- Notes
  Prior to calling this subroutine, LU-decomposed matrix must be obtained by subroutine CLU and must be input as the parameters ZFA and IP to be used for this subroutine.
  The subrotine LCX should be used for solving a system of linear equatons. Obtaining the solution by first computing the inverse requires more steps of calculation, so subroutine CLUIV should be used only when the inverse is inevitable.
  The transposition vector corresponds to the permutation matrix $\boldsymbol{P}$ of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

when performing LU decomposition with partial pivoting, refer to Notes of the subroutine CLU.

- Example
  The inverse of an $n \times n$ complex general matrix iscomputed. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION ZA(100,100),ZVW(100),IP(100)
      COMPLEX ZA,ZVW
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,ZA(I,J),J=1,N),
     *             I=1,N)
      CALL CLU(ZA,100,N,0.0,IP,IS,ZVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      CALL CLUIV(ZA,100,N,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,630) ((I,J,ZA(I,J),I=1,N),
     *             J=1,N)
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT(//11X,'**INPUT MATRIX**'/12X,
     *'ORDER=',I5/(2X,4('(',I3,',',I3,')',
     *2E16.8)))
  610 FORMAT('0',10X,'CONDITION CODE(CLU)=',
     *I5)
  620 FORMAT('0',10X,'CONDITION ',
     *'CODE(CLUIV)=',I5)
  630 FORMAT('0',10X,'**INVERSE MATRIX**',
     */(2X,4('(',I3,',',I3,')',2E16.8)))
      END
```

Method

The subroutine computes the inverse of an $n \times n$ complex general matrix $\boldsymbol{A}$, giving the LU-decomposed matrices $\boldsymbol{L}$, $\boldsymbol{U}$ and the permutation matrix $\boldsymbol{P}$ which indicates row exchanges in partial pivoting.

Since

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{4.1}$$

then, the inverse $\boldsymbol{A}^{-1}$ can be represented using (4.1) as follows: the inverse of $\boldsymbol{L}$ and $\boldsymbol{U}$ are computed and then the inverse $\boldsymbol{A}^{-1}$ is computed as (4.2).

$$\boldsymbol{A}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P} \tag{4.2}$$

$\boldsymbol{L}$ and $\boldsymbol{U}$ are as shown in Eq. (4.3) for the following explanation.

$$\boldsymbol{L} = (\boldsymbol{l_{ij}}), \boldsymbol{U} = (\boldsymbol{u_{ij}}) \tag{4.3}$$

- Calculating $\boldsymbol{L}^{-1}$

Since the inverse $\boldsymbol{L}^{-1}$ of a lower triangular matrix $\boldsymbol{L}$ is also a lower triangular matrix, if we represent $\boldsymbol{L}^{-1}$ by

$$\boldsymbol{L}^{-1} = \left(\tilde{l}_{ij}\right) \tag{4.4}$$

then Eq. (4.5) is obtained based on the relation $\boldsymbol{LL}^{-1}$=$\boldsymbol{I}$.

$$\sum_{k=1}^{n} l_{ik}\tilde{l}_{kj} = \delta_{ij}, \qquad \delta_{ij}\begin{cases} =1 & ,i=j \\ =0 & ,i \neq j \end{cases} \tag{4.5}$$

(4.5) is rewritten as

$$\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} + l_{ii}\tilde{l}_{ij} = \delta_{ij}$$

and the elements $\tilde{l}_{ij}$ of the j-th column (j=1,...,) of the matrix $\boldsymbol{L}^{-1}$ are obtained as follows:

$$\begin{aligned} \tilde{l}_{ij} &= \left(-\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj}\right)\Big/ l_{ii} \quad ,i = j+1,...,n \\ \tilde{l}_{jj} &= 1/l_{jj} \end{aligned} \tag{4.6}$$

where, $l_{ii} \neq 0 (i = j,...,n)$

- Calculating $\boldsymbol{U}^{-1}$

Since the inverse $\boldsymbol{U}^{-1}$ of a unit upper triangular matrix $\boldsymbol{U}$ is also a unit upper triangular matrix, if we represent $\boldsymbol{U}^{-1}$ by

$$\boldsymbol{U}^{-1} = \left(\tilde{u}_{ij}\right) \tag{4.7}$$

then Eq. (4.8) is obtained based on the relation $\boldsymbol{U}\boldsymbol{U}^{-1}$=$\boldsymbol{I}$.

$$\sum_{k=1}^{n} u_{ik}\tilde{u}_{kj} = \delta_{ij} \qquad \delta_{ij}\begin{cases} =1 & ,i=j \\ =0 & ,i \neq j \end{cases} \tag{4.8}$$

Since u_{ij} = 1,(4.8) can be rewritten

$$\tilde{u}_{ij} + \sum_{k=i+1}^{j} u_{ik}\tilde{u}_{kj} = \delta_{ij}$$

Considering $\tilde{u}_{jj}$ =1, the elements $\tilde{u}_{jj}$ of the j-th column (j=n,..., 2) of $\boldsymbol{U}^{-1}$ are obtained as follows:

$$\tilde{u}_{ij} = -u_{ij} - \sum_{k=i+1}^{j-1} u_{ik}\tilde{u}_{kj} \quad ,i = j-1,...,1 \tag{4.9}$$

- Calculating $\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$

Let the product of matrices $\boldsymbol{U}^{-1}$ and $\boldsymbol{L}^{-1}$ be $\boldsymbol{B}$, then its elements are obtained by

$$b_{ij} = \sum_{k=j}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \quad ,i=1,..., j-1$$

$$b_{ij} = \sum_{k=i}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \quad ,i= j,...,n$$

Considering $\tilde{u}_{ii}$ =1, the element b_{ij} of the j-th column (j=1, ...,n) of $\boldsymbol{B}$ are obtained by

$$\begin{aligned} b_{ij} &= \sum_{k=j}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \quad ,i=1,..., j-1 \\ b_{ij} &= \tilde{l}_{ij} + \sum_{k=i+1}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \quad ,i= j,...,n \end{aligned} \tag{4.10}$$

Next, matrix $\boldsymbol{B}$ is multiplied by the permutation matrix to obtain the inverse $\boldsymbol{A}^{-1}$. Actually however, based on the values of the transposition vector $\boldsymbol{IP}$, the elements of $\boldsymbol{A}^{-1}$ are obtained simply by exchanging the column in the matrix $\boldsymbol{B}$. The precison of the inner products in (4.6), (4.9) and (4.10) has been raised to minimize the effect of rounding errors. For more invormation, see Reference [1].

A22-15-0302 CLUX, DCLUX

A system of linear equations with a complex general matrix decomposed into the factors ***L*** and ***U***
CALL CLUX (ZB, ZFA, K, N, ISW, IP, ICON)

Function

This subroutine solves a system of linear equations.

$$\boldsymbol{LUx} = \boldsymbol{Pb} \qquad (1.1)$$

L and ***U*** are, respectively, the $n \times n$ lower triangular and unit upper triangular matrices, ***P*** is a permutation matrix which performs row exchange with partial pivoting for LU decomposition of the coefficient matrix, ***b*** is an *n*-dimensional complex constant vector, and ***x*** is the *n*-dimensional solution vector. Also, one of the following equations can be solved instead of (1.1).

$$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad (1.2)$$
$$\boldsymbol{Uz} = \boldsymbol{b} \qquad (1.3)$$

Parameters

ZB .... Input. Constant vector ***b***.
Output. One of the solution vectors ***x***, ***y***, or ***z***.
ZB is a complex one-dimensional array of size *n*.

ZFA .... Input. Matrix ***L*** and matrix ***U***.
Refer to Fig. CLUX-1.
ZFA is a complex two-dimensional array, ZFA (K, N).

K .... Input. Adjustable dimension of the array ZFA (≥N).

N .... Input. The order *n* of the matrices ***L*** and ***U***.

ISW .... Input. Control information
ISW = 1 ... ***x*** is obtained.
ISW = 2 ... ***y*** is obtained.
ISW = 3 ... ***z*** is obtained.

IP .... Input. The transposition vector which indicates the history of row exchange in partial pivoting. IP is a one-dimensional array of size *n*. (See Notes of subroutine CLU.)

ICON .. Output. Condition code
See Table CLUX-1

Table CLUX- 1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The coefficient matrix was singular.	Aborted
30000	K<N, N<1, ISW = 1, 2, 3, or there was an error in IP.	Aborted

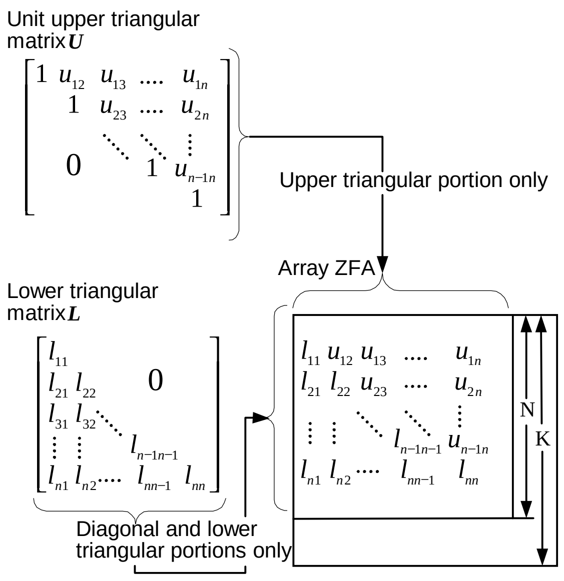

Unit upper triangular matrix $\boldsymbol{U}$
$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & \ddots & \vdots \\ & 0 & & 1 & u_{n-1n} \\ & & & & 1 \end{bmatrix}$
Upper triangular portion only
Array ZFA
Lower triangular matrix $\boldsymbol{L}$
$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & 0 & \\ l_{31} & l_{32} & \ddots & \\ \vdots & \vdots & & l_{n-1n-1} \\ l_{n1} & l_{n2} \cdots & l_{nn-1} & l_{nn} \end{bmatrix}$
$l_{11}\ u_{12}\ u_{13}\ \cdots\ u_{1n}$
$l_{21}\ l_{22}\ u_{23}\ \cdots\ u_{2n}$
$l_{n-1n-1}\ u_{n-1n}$
$l_{n1}\ l_{n2} \cdots\ l_{nn-1}\ l_{nn}$
N
K
Diagonal and lower triangular portions only

Fig. CLUX-1 Storage of the elements of ***L*** and ***U*** in the array ZFA

Comments on use

- Subprograms used
  SSL II ..... CSUM, MGSSL
  FORTRAN basic function ..... None

- Notes
  A system of linear equations can be solved by first calling the subroutine CLU to decompose the coefficient into ***L*** and ***U***and then by calling this routine. However, instead of calling these two routines, subroutine LCX is usually called to solve such equations in one step.

- Example
  A system of linear equations is solved by first using subroutine CLU to decompose the $n \times n$ coefficient matrix into ***L*** and ***U***. $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION ZA(100,100),ZB(100),
     *    ZVW(100),IP(100)
      COMPLEX ZA,ZB,ZVW
      READ(5,500) N
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      READ(5,510) (ZB(I),I=1,N)
      WRITE(6,600) N,((I,J,ZA(I,J),J=1,N),
     *             I=1,N)
      WRITE(6,610) (I,ZB(I),I=1,N)
      CALL CLU(ZA,100,N,0.0,IP,IS,ZVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      CALL CLUX(ZB,ZA,100,N,1,IP,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,640) (I,ZB(I),I=1,N)
      STOP
```

```
500 FORMAT(I5)
510 FORMAT(5(2F8.3))
600 FORMAT(///10X,'**COMPLEX MATRIX **',
   */12X,'ORDER=',I5/(5X,2('(',I3,',',I3,
   *')',2E15.8,2X)))
610 FORMAT('0',10X,'CONSTANT VECTOR',
   */(5X,3('(',I3,')',2E15.8,2X)))
620 FORMAT('0',10X,'CONDITION ',
   *'CODE(CLU)=',I5)
630 FORMAT('0',10X,'CONDITION ',
   *'CODE(CLUX)=',I5)
640 FORMAT('0',10X,'SOLUTION VECTOR',
   */(5X,3('(',I3,')',2E15.8,2X)))
    END
```

Method
A system of linear equations

$$\boldsymbol{LUx} = \boldsymbol{Pb} \tag{4.1}$$

can be solved by solving the two equations:

$$\boldsymbol{Ly} = \boldsymbol{Pb} \tag{4.2}$$
$$\boldsymbol{Ux} = \boldsymbol{y} \tag{4.3}$$

- Solving $\boldsymbol{Ly} = \boldsymbol{Pb}$ (forward substition)
  $\boldsymbol{Ly} = \boldsymbol{Pb}$ can be serially solved using equation (4.4).

$$y_i = \left(b'_i - \sum_{k=1}^{i-1} l_{ik} y_k \right) \Big/ l_{ii} \quad , i = 1,...,n \tag{4.4}$$

where, $\boldsymbol{L}=(l_{ij})$,$\boldsymbol{y}^T=(y_1, ..., y_n)$,$(\boldsymbol{Pb})^T=(b'_1, ..., b'_n)$.

- Solving $\boldsymbol{Ux} = \boldsymbol{y}$ (backward substitution)
  $\boldsymbol{Ux} = \boldsymbol{y}$ can be serially solved using equations

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k \quad , i = n,...,1 \tag{4.5}$$

where, $\boldsymbol{U}=(u_{ij})$,$\boldsymbol{x}^T = (x_1, ..., x_n)$

The precision of the inner products in (4.4) and (4.5) has heen raised to minimize the effect of rounding errors. For more information, see References [1], [3], and [4].

B21-15-0702 CNRML, DCNRML

Normalization of eigenvectors of a complex matrix
CALL CNRML (ZEV, K, N, IND, M, MODE, ICON)

Function

This subroutine normalizes m eigenvectors $\boldsymbol{x_i}$ of an n-order complex matrix using either (1.1) or (1.2).

$$\boldsymbol{y}_i = \boldsymbol{x}_i \Big/ \|\boldsymbol{x}_i\|_\infty \qquad (1.1)$$

$$\boldsymbol{y_i} = \boldsymbol{x_i} \Big/ \|\boldsymbol{x_i}\|_2 \qquad (1.2)$$

$n \geq 1$.

Parameters

ZEV ... Input. m eigenvectors $\boldsymbol{x}_i$ (i= l, 2, ..., m).
Eigenvectors $\boldsymbol{x}_i$ are stored in columns of ZEV.
See "Comments on use".
Output. m normalized eigenvectors $\boldsymbol{y}_i$.
Normalized eigenvectors $\boldsymbol{y}_i$ are stored into the corresponding eigenvector $\boldsymbol{x}_i$.
ZEV is a complex two-dimensional array.
ZEV(K, M).

K ... Input. Adjustable dimension of array ZEV ($\geq n$).

N ... Input. Order n of the complex matrix.

IND ... Input. Specifies eigenvectors to be normalized.
If IND (j) = 1 is specified the eigenvector corresponding to the j-th eigenvalue is normalized. If IND (j)=0 is specified the eigenvector corresponding to the j-th eigenvalue is not normalized.
IND is a one-dimensional array of size M.
See "Comments on use".

M ... Input. Size of array IND. See Notes.

MODE ...Input. Indicates the method of normalization.
When MODE=1 ... (1.1) is used.
When MODE=2 ... (1.2) is used.

ICON ... Output. Condition code.
See Table CNRML-1.

Table CNRML-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	ZEV(1, 1) = (1.0. 0.0)
30000	N<M, M<1, K<N, MODE was neither 1 or 2, or IND specification was invalid.	Bypassed

Comments on use

- Subroutines used
  SSL 11 ... MGSSL
  FORTRAN basic functions ... REAL, AIMAG, AMAXl, SQRT

- Notes
  When the CHVEC or CHBK2 subroutine is called before this subroutine, parameters ZEV, IND and M can be used as input parameters for this subroutine.

- Example
  This example normalizes the eigenvectors of an n-order complex matrix obtained by the subroutine CEIG2 so that the resultant eigenvectors ean be $\|\boldsymbol{x}\|_\infty$ =1.
  $n \leq 100$.

```
C     **EXAMPLE**
      COMPLEX ZA(100,100),ZE(100),
     *        ZEV(100,100)
      DIMENSION IND(100),VW(100),IVW(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,ZA(I,J),J=1,N)
      CALL CEIG2(ZA,100,N,1,ZE,ZEV,
     *           VW,IVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GT.10000) GO TO 10
      DO 30 I=1,N
   30 IND(I)=1
      CALL CNRML(ZE,ZEV,100,N,IND,N,1,ICON)
      CALL CEPRT(ZE,ZEV,100,N,IND,N)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',
     *       5X,'N=',I3/)
  610 FORMAT(/2(5X,'A(',I3,',',I3,')=',
     *       2E15.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

The subroutine CEPRT used in this example, prints the eigenvalues and corresponding eigenvectors of a complex matrix.
For further information see an example of using the subroutine CEIG2.

Method

Given m eigenvectors $\boldsymbol{x}_i$ (i =1, ..., m) ofan n-order complex vector, normalized eigenvectors $\boldsymbol{y}_i$ are computed. Where $\boldsymbol{x}_i = (\boldsymbol{x}_{1i}, \ldots, \boldsymbol{x}_{ni})^T$

When MODE=1 is specified. each vector $\boldsymbol{x}_i$ is normalized such that the maximum absolute value among the elements of each vector becomes 1 .

$$\boldsymbol{y_i} = \boldsymbol{x_i} \big/ \|\boldsymbol{x_i}\|_\infty \, , \ \|\boldsymbol{x_i}\|_\infty = \max_k |\boldsymbol{x_{ki}}| \qquad (4.1)$$

When MODE=2 is specified, each vector $\boldsymbol{x}_i$ is normalized such that the sum of the square of absolute values corresponding to its elements is 1.

$$\boldsymbol{y_i} = \boldsymbol{x_i} \big/ \|\boldsymbol{x_i}\|_{\boldsymbol{2}} \, , \ \|\boldsymbol{x_i}\|_2 = \sqrt{\sum_{k=1}^{n} |\boldsymbol{x_{ki}}|^2} \qquad (4.2)$$

I11-41-0201 COSI, DCOSI

Cosine integral $C_i(x)$
CALL COSI (X, CI, ICON)

Function

This subroutine computes cosine integral

$$C_i(x) = -\int_x^\infty \frac{\cos(t)}{t}dt$$

by series and asymptotic expansions, where $x \neq 0$.
If $x < 0$, cosine integral $C_i(x)$ is assumed to take a principal value.

Parameters

X	Input. Independent variable x.
CI	Output. Value of $C_i(x)$.
ICON ...	Output. Condition codes. See Table COSI-1

Table COSI-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert X\rvert \geq t_{max}$	CI = 0.0
30000	X=0	CI = 0.0

Comments on use

- Subprograms used
  SSL II ... MGSSL, UTLIM
  FORTRAN basic functions . .. ABS, SIN, COS, and ALOG

- Notes
  The valid ranges of parameter X are:
  $\lvert X\rvert < t_{max}$
  This is provided because sin (x) and cos (x) lose their acduracy, if |X| exceeds the above ranges.

- Example
  The following example generates a table of $C_i(x)$ from 0.1 to l0.0 with increment 0.1.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,100
       X=FLOAT(K)/10.0
       CALL COSI(X,CI,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,CI
       IF(ICON.NE.0) WRITE(6,620) X,CI,ICON
    10 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF COSINE ',
      *'INTEGRAL FUNCTION'///6X,'X',9X,
      *'CI(X)'/)
   610 FORMAT(' ',F8.2,E17.7)
   620 FORMAT(' ','** ERROR **',5X,'X=',
      *E17.7,5X,'CI=',E17.7,5X,'CONDITION=',
      *I10)
       END
```

Methods

Two different approximation formulas are used depending on the ranges of x divided at $x = \pm 4$.
Since $C_i(x)=C_i(-x)$, the following discussion is limited to the case in which $x > 0$.

- For $0 < x < 4$
  The power series expansion of $C_i(x)$,

$$C_i(x) = \gamma + \log(x) + \sum_{n=1}^{\infty}\frac{(-1)^n x^{2n}}{(2n)!2n} \quad (4.1)$$

is calculated, with the following approximation formulas:
Single precision:

$$C_i(x) = \log(x) + \sum_{k=0}^{7} a_k z^{2k} \quad ,z = x/4 \quad (4.2)$$

Double precision:

$$C_i(x) = \log(x) + \sum_{k=0}^{12} a_k x^{2k} \quad (4.3)$$

- For $x \geq 4$
  The asymptotic expansion of

$$C_i(x) = -\{P(x)\sin(x) + Q(x)\cos(x)\}/x \quad (4.4)$$

is calculated through use of the following approximate expression for $\boldsymbol{P}(x)$ and $\boldsymbol{Q}(x)$:
Single precision:

$$P(x) = \sum_{k=0}^{11} a_k z^k, z = 4/x \quad (4.5)$$

$$Q(x) = \sum_{k=0}^{11} b_k z^k, z = 4/x \quad (4.6)$$

Double precision:

$$P(x) = \sum_{k=0}^{11} a_k z^k \Big/ \sum_{k=0}^{11} b_k z^k, z = 4/x \quad (4.7)$$

$$Q(x) = -\sum_{k=0}^{10} c_k z^{k+1} \Big/ \sum_{k=0}^{11} d_k z^k, z = 4/x \quad (4.8)$$

C21-15-0101 CQDR, DCQDR

Zeros of a quadratic equation with complex coefficients
CALL CQDR (Z0, Z1, Z2, Z,ICON)

Function

This subroutine finds zeros of a quadratic equation with complex coefficients:

$$a_0 z^2 + a_1 z + a_2 = 0 (|a_0| \neq 0)$$

Parameters

Z0, Zl, Z2 ..Input. Coefficients of the quadratic equation. Complex variables where Z0 = a_0, Zl = a_1 and Z2 = a_2

Z ..... Output. Roots of the quadratic equation. Z is a complex one-dimensional array of size 2.

ICON .. Output. Condition code.

Table CQDR-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	$\|a_0\|$ = 0.0	$-a_2/a_1$ is stored in Z (1). Z(2) may be invalid.
30000	$\|a_0\|$ = 0.0 and $\|a_1\|$ = 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II ...MGSSL
  FORTRAN basic functions ... CSQRT, REAL, AIMAG, and ABS

- Example.
  Complex coefficients are entered and the roots *z* are determined.

```
C      **EXAMPLE**
       DIMENSION Z(2)
       COMPLEX Z0,Z1,Z2,Z
       READ(5,500) Z0,Z1,Z2
       CALL CQDR(Z0,Z1,Z2,Z,ICON)
       WRITE(6,600) ICON,Z0,Z1,Z2
       IF(ICON.EQ.30000) STOP
       WRITE(6,610) (Z(I),I=1,2)
       STOP
   500 FORMAT(6F10.0)
   600 FORMAT(10X,'ICON=',I5/10X,'A=',2E15.6/
      *       (12X,2E15.6))
   610 FORMAT(10X,'Z=',2E15.6/12X,2E15.6)
       END
```

Method

The roots of complex quadratic equation $a_0z^2 + a_1z + a_2 = 0$ can be obtained from

$$-P_1 + \sqrt{P_1^2 - 4P_2}, -P_1 - \sqrt{P_1^2 - 4P_2}$$

where $P_1 = a_1/a_0$ and $P_2 = a_2/a_0$

If $|P_1^2| >> |4P_2|$, there will be potentially large loss of accuracy in either $-P_1 + \sqrt{P_1^2 - 4P_2}$ or $-P_1 - \sqrt{P_1^2 - 4P_2}$. In order to avoid this situation, root formulas with rationalized numerators, as shown below, are used in calculations.

Let $D = P_1^2 - 4P_2$

When $|D| = 0$

$z_1 = -P_1/2$
$z_2 = -P_1/2$

When $|D| \neq 0$

let the real part of P_1 be x and the imaginary part be y.
for $x > 0$

$$\begin{aligned} z_1 &= \left(-P_1 - \sqrt{D}\right)/2 \\ z_2 &= -2P_2/\left(P_1 + \sqrt{D}\right) \end{aligned} \tag{4.1}$$

for $x < 0$

$$\begin{aligned} z_1 &= 2P_2/\left(-P_1 + \sqrt{D}\right) \\ z_2 &= \left(-P_1 + \sqrt{D}\right)/2 \end{aligned} \tag{4.2}$$

for $x = 0$

$$\left.\begin{aligned} z_1 &= \left(-P_1 - \sqrt{D}\right)/2 \\ z_2 &= -2P_2/\left(P_1 + \sqrt{D}\right) \end{aligned}\right\} y \geq 0$$
$$\left.\begin{aligned} z_1 &= 2P_2/\left(-P_1 + \sqrt{D}\right) \\ z_2 &= \left(-P_1 + \sqrt{D}\right)/2 \end{aligned}\right\} y < 0 \tag{4.3}$$

In the calculation of discriminant $D = P_1^2 - 4P_2$ if $|P_1|$ is very large, P_1^2 may cause an overflow. In order to avoid this situation, the real and imaginary parts of P_1 are tested to see if they are greater than 10^{35}. The above methods are used when those conditions are not satisfied, otherwise the following used.

$$\sqrt{D} = P_1\sqrt{1 - 4P_2/P_1/P_1}$$

A11-40-0201 CSBGM, DCSBGM

Storage mode conversion of matrices (real symmetric band to real general)
CALL CSBGM (ASB, N, NH, AG, K, ICON)

Function

This subroutine converts an $n \times n$ real symmetric band matrix with band width h stored in the compressed mode into that stored in the general mode, where $n > h \geq 0$.

Parameters

ASB	Input. The symmetric band matrix stored in the compressed mode One-dimensional array of size $n(h+1)-h(h+1)/2$.
N.......	Input. The order n of the matrix.
NH.....	Input. The band width h of the matrix.
AG.....	Output. The symmetric band matrix stored in the general mode Two-dimensional array, AG (K, N). (See "Comments on Use.")
K	Input. The adjustable dimension (≥ N) of array AG.
ICON .	Output. Condition code. (See Table CSBGM-1.)

Table CSBGM-1 Condition codes

Code	Meaning	Processing
0	No error.	
30000	NH < 0, N ≤ NH, or K < N.	Bypassed.

Comments on use

- Subprograms used
  SSLII ... MGSSL
  FORTRAN basic function ... None

- Notes
  Storing method of the symmetric band matrix in the general mode:
  The symmetric band matrix in the general mode transformed by this subroutine contains not only the lower band and diagonal portions but also the upper band portion and other elements (zero elements).

  Saving the storage area:
  If there is no need to keep the contents on the array ASB, more storage area can be saved by using the EQUIVALENCE statement as follows.;
  EQUIVALENCE (ASB (1), AG (1, 1))
  (See "Example" for details.)

- Example
  Given an $n \times n$ real symmetric band matrix with band width h , the inverse matrix is obtained using subroutines ALU and LUIV, where mode conversion is performed by this subroutine, where $n \leq 100$ and $h \leq 20$.

```
C      **EXAMPLE**
       DIMENSION ASB(1890),AG(100,100),
      *          VW(100),IP(100)
       EQUIVALENCE(ASB(1),AG(1,1))
   10  READ(5,500) N,NH
       IF(N.EQ.0) STOP
       NT=(NH+1)*(N+N-NH)/2
       READ(5,510) (ASB(I),I=1,NT)
       K=100
       CALL CSBGM(ASB,N,NH,AG,K,ICON)
       WRITE(6,600) ICON
       IF(ICON.EQ.30000) GO TO 10
       WRITE(6,610) N,NH,((I,J,AG(I,J),
      *             J=1,N),I=1,N)
       EPSZ=0.0
       CALL ALU(AG,K,N,EPSZ,IP,IS,VW,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) GO TO 10
       CALL LUIV(AG,K,N,IP,ICON)
       WRITE(6,630) ICON
       WRITE(6,640) N,NH,((I,J,AG(I,J),
      *             J=1,N),I=1,N)
       GO TO 10
  500  FORMAT(2I5)
  510  FORMAT(4E15.7)
  600  FORMAT('1'/10X,'CSBGM ICON=',I5)
  610  FORMAT(//10X,'** INPUT MATRIX **'/
      *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
      *(2X,4('(',I3,',',I3,')',E17.8)))
  620  FORMAT(/10X,'ALU ICON=',I5)
  630  FORMAT(/10X,'LUIV ICON=',I5)
  640  FORMAT('1'//10X,'** INVERSE ',
      *'MATRIX **'/10X,'ORDER=',I5,5X,
      *'BANDWIDTH=',I5/(2X,4('(',I3,',',I3,
      *')',E17.8)))
       END
```

Method

A real symmetric band matrix stored in a one-dimensional array ASB in the compressed mode is converted as follows to be a symmetric band matrix in a two-dimensional array in the general mode :

The elements of the ASB are moved to the diagonal and upper triangular portions of the AG in descending element number order, beginning from the last element (of column n).

The correspondence between locations is shown below.

Elements in the compressed mode	Matrix elements	Elements in the general mode

$$\mathrm{ASB}(h\mathrm{J}-h(h+1)/2+\mathrm{I}) \rightarrow a_{ij} \rightarrow \mathrm{AG(I,J)}$$
$$\mathrm{I}=\mathrm{J},\mathrm{J}-1,\ldots,\mathrm{J}-h \quad ,\mathrm{J}=\mathrm{N},\mathrm{N}-1,\ldots,h+2$$

$$\mathrm{ASB}(\mathrm{J}(\mathrm{J}-1)/2+\mathrm{I}) \rightarrow a_{ij} \rightarrow \mathrm{AG(I,J)}$$
$$\mathrm{I}=\mathrm{J},\mathrm{J}-1,\ldots,1 \quad ,\mathrm{J}=h+1,h,\ldots,1$$

The lower triangular portion is copied by the upper triangular portion using the diagonal as the axis of symmetry so as to be AG (I, J) = AG(J, I), where I>J.

A11-50-0201 CSBSM, DCSBSM

Storage mode conversion of matrices (real symmetric band to real symmetric)
CALL CSBSM (ASB, N, NH, AS, ICON)

Function

This subroutine converts an $n \times n$ symetric band matrix with band width h stored in the compressed mode for symmetric band matrix into that stored in the compressed mode for symmetric matrix, where $n>h\geq 0$.

Parameters

ASB	Input. The symmetric band matrix stored in the compressed mode for symmetric band matrix. One-dimensional array of size $n(h+1)-h(h+1)/2$.
N	Input.The order n of the matrix.
NH	Input. Band width h of the symmetric band matrix.
AS	Output. The symmetric band matrix stored in the compressed mode for symmetric matrix. One-dimensional array of size $n(n+1)/2$. (See “Comments on Use.”)
ICON	Output. Condition code. (See Table CSBSM-I.)

Table CSBSM-1 Condition codes

Code	Meaning	Processing
0	No error.	
30000	NH < 0 or NH ≥ N.	Bypassed.

Comments on use

- Subprograms used
  SSL II ...MGSSL
  FORTRAN basic function ... None

- Notes
  Saving the storage area:
  If there is no need to keep the contents on the array ASB, more storage can be saved by using the EQUIVALENCE statement as follows ;

  EQUIVALANCE (ASB (1), AS (1))
  (See “Example” for details.)

- Example
  Given an $n \times n$ positive-definite symmetric band matrix with band width h stored in the compressed mode, for symmetric band matrix the inverse matrix is obtained using subroutines SLDL and LDIV, where the mode conversion is performed by this subroutihe. Where $n \leq 100$ and $h \leq 20$.

```
C     **EXAMPLE**
      DIMENSION AS(5050),ASB(1890)
      EQUIVALENCE(AS(1),ASB(1))
   10 READ(5,500) N,NH
      IF(N.EQ.0) STOP
      NS=(N+1)*N/2
      NT=(NH+1)*(N+N-NH)/2
      READ(5,510) (ASB(I),I=1,NT)
      CALL CSBSM(ASB,N,NH,AS,ICON)
      WRITE(6,600) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,610) N,NH,(I,AS(I),I=1,NS)
      EPSZ=0.0
      CALL SLDL(AS,N,EPSZ,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL LDIV(AS,N,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,640) N,NH,(I,AS(I),I=1,NS)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'/10X,'CSBSM ICON=',I5)
  610 FORMAT(//10X,'** INPUT MATRIX **'/
     *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
     *(2X,5('(',I4,')',E17.8)))
  620 FORMAT(/10X,'SLDL ICON=',I5)
  630 FORMAT(/10X,'LDIV ICON=',I5)
  640 FORMAT('1'//10X,'** INVERSE ',
     * 'MATRIX **'/
     * 10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
     * (2X,5('(',I4,')',E17.8)))
      END
```

Method

A real symmetric band matrix stored in a one-dimensional array ASB in the compressed mode for symmetric band matrix is converted as follows to be stored in one-dimensional array AS in the symmetric matrix compressed mode for symmetric matrix. The elements are moved in column wise, where zero elements are stored outside the band portion. The correspondence between locations other than these zero elements are shown below.

Elements in the compressed mode for symmetric band matrix	Matrix elements	Elements in the compressed mode for symmetric matrix

$$\mathrm{ASB}\big(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J}\big) \rightarrow a_{ij} \rightarrow \mathrm{AS}(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J})$$
$$,\mathrm{J}=1,2,...,\mathrm{I} \quad ,\mathrm{I}=1,2,...,h+1$$

$$\mathrm{ASB}\big(\mathrm{I}\cdot h+\mathrm{J}-h(h+1)/2\big) \rightarrow a_{ij} \rightarrow \mathrm{AS}(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J}))$$
$$,\mathrm{J}=\mathrm{I},\mathrm{I}-1,...,\mathrm{I}-h \quad ,\mathrm{I}=\mathrm{N},\mathrm{N}-1,...,h+2$$

A11-10-0201 CSGM, DCSGM

Storage mode conversion of matrices (real symmetric to real general)
CALL CSGM (AS, N, AG, K, ICON)

Function

This subroutine converts an $n \times n$ real symmetric matrix stored in the compressed mode into a symmetric matrix stored in the general mode. $n \geq 1$.

Parameters

AS ...	Input. The symmetric matrix stored in the compressed mode. AS is a one-dimensional array of size $n(n+1)/2$.
N ...	Input. The order n of the matrix.
AG ...	Output. The symmetric matrix stored in the general mode. AG is a two dimensional array, AG (K,N).
K ...	Input. The adjustable dimension (≥ N) of array AG
ICON..	Output. Condition code. See Table CSGM-1.

Table CSGM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N < 1 or K < N	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... none

- Note
  The symmetric matrix in the general mode:
  The symmetric matrix in the general mode transformed by the subroutine contains not only the lower trianguler portion and the diagnoal portion but also the upper triangular portion.
  Saving the storage area:
  If there is no need to keep the contents on the array AS, more storage area can be saved by using the EQUIVALENCE statement as follows;

  EQUIVALENCE (AS(1), AG(1,1))

  Refer to the example shown below.

- Example
  Given an $n \times n$ real symmetric matrix in the compressed mode, the inverse matrix is obtained by subroutines ALU and LUIV as shown in the example. In this case, the required mode conversion is performed by this subroutine. Here, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100,100),
     *          VW(100),IP(100)
      EQUIVALENCE (A(1),B(1,1))
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      K=100
      CALL CSGM(A,N,B,K,ICON)
      WRITE(6,600) ICON
      IF(ICON.EQ.30000) GOTO 10
      WRITE(6,610) N,((I,J,B(I,J),
     *             J=1,N),I=1,N)
      EPSZ=0.0
      CALL ALU(B,K,N,EPSZ,IP,IS,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GOTO 10
      CALL LUIV(B,K,N,IP,ICON)
      WRITE(6,630) ICON
      WRITE(6,640) N,((I,J,B(I,J),
     *             J=1,N),I=1,N)
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'/10X,'CSGM ICON=',I5)
  610 FORMAT(//10X,'** INPUT MATRIX **'/
     *10X,'ORDER=',I5/
     *(2X,4('(',I3,',',I3,')',E17.8)))
  620 FORMAT(/10X,'ALU ICON=',I5)
  630 FORMAT(/10X,'LUIV ICON=',I5)
  640 FORMAT('1'//10X,'** INVERSE ',
     *'MATRIX **'/10X,'ORDER=',I5/
     *(2X,4('(',I3,',',I3,')',E17.8)))
      END
```

Method

This subroutine converts an $n \times n$ real symmetric matrix stored in a one-dimensional array AS in the compressed mode to in a two-dimensional array in the general mode acccording to the following procedures.

- The elements stored in AS are transferred to the diagonal and upper triangular portions serially from the largest address, i.e. the n-th column.
  The correspondence between locations is shown below, where NT = $n(n+1)/2$.

Elements in compressed mode		Elements of matrix		Elements in general mode
AS(NT)	→	a_{nn}	→	AG(N,N)
AS(NT-1)	→	a_{nn-1}	→	AG(N-1,N)
.		.		.
.		.		.
.		.		.
AS(I(I-1)/2+J)	→	a_{ij}	→	AG(J,I)
.		.		.
.		.		.
.		.		.
AS(2)	→	a_{21}	→	AG(1,2)
AS(1)	→	a_{11}	→	AG(1,1)

- With the diagonal as the axis of symmetry, the elements of the upper triangular portion are transferred to the lower triangular portion so that AG(I,J) = AG(J,I). Here, I>J.

A11-50-0101 CSSBM, DCSSBM

Storage conversion of matrices (real symmetric to real symmetric band)
CALL CSSBM (AS, N, ASB, NH, ICON)

Function

This subroutine converts an $n \times n$ real symmetric band matrix with band width h stored in compressed mode for symmetric matrix into one stored in compressed mode for symmetric band matrix, where $n \times h \geq 1$.

Parameters

AS	Input. The symmetric band matrix stored in compressed mode for symmetric matrix. One-dimensional array of size $n(n+1)/2$.
N	Input. The order n of the matrix.
ASB ...	Output. The symmetric band matrix stored in compressed mode for symmetric band matrix One-dimensional array of size $n(h+1) - h(h+1)/2$.
NH ...	Input. Band width h of the symmetric band matrix.
ICON ..	Output. Condition code. (See Table CSSBM-1).

Table CSSBM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	NH < 0 or NH ≤ N.	Bypassed.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... None.

- Notes
  Saving the storage area:
  If there is no need to keep the contents on the array AS, more storage area can be saved by using the EQUIVALENCE statement as follows;

  EQUIVALENCE (AS (1), ASB (1))
  (See "Example" for details.)

- Example
  Given an $n \times n$ positive-definite symmetric band matrix with band width h stored in compressed mode for symmetric band matrix, the LDL^T decomposition is performed using subroutine SBDL in the compressed mode for symmetric band matrix, where the mode conversion is performed by this subroutine.
  Where $n \leq 100$ and $h \leq 20$.

```
C     **EXAMPLE**
      DIMENSION AS(5050),ASB(1890)
      EQUIVALENCE(AS(1),ASB(1))
   10 READ(5,500) N,NH
      IF(N.EQ.0) STOP
      NT=(N+1)*N/2
      READ(5,510) (AS(I),I=1,NT)
      WRITE(6,600) N,NH,(I,AS(I),I=1,NT)
      CALL CSSBM(AS,N,ASB,NH,ICON)
      WRITE(6,610) ICON
      IF(ICON.EQ.30000) GO TO 10
      EPSZ=0.0
      CALL SBDL(ASB,N,NH,EPSZ,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL CSBSM(ASB,N,NH,AS,ICON)
      WRITE(6,630) N,NH,(I,AS(I),I=1,NT)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT(//10X,'** INPUT MATRIX **'/
     *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
     *(2X,5('(',I4,')',E17.8)))
  610 FORMAT('1'//10X,'CSSBM ICON=',I5)
  620 FORMAT(10X,'SBDL ICON=',I5)
  630 FORMAT('1'/10X,'DECOMPOSED MATRIX'/
     *10X,'ORDER=',I5,5X,'BANDWIDTH=',I5/
     *(2X,5('(',I4,')',E17.8)))
      END
```

Method

A real symmetric band matrix stored in one-dimensional, array AS in the compressed mode for symmetric matrix is converted as follows to be stored in one-dimensional array ASB in the compressed mode for symmetric band matrix. The elements are moved in column wise as follows:

Elements in the compressed mode for symmetric matrix	Matrix elements	Elements in the compressed mode for symmetric band matrix

$$\mathrm{AS}(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J}) \rightarrow a_{ij} \rightarrow \mathrm{ASB}(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J})$$
$$,\mathrm{J}=1,2,...,\mathrm{I} \quad ,\mathrm{I}=1,2,...,h+1$$

$$\mathrm{AS}(\mathrm{I}(\mathrm{I}-1)/2+\mathrm{J}) \rightarrow a_{ij} \rightarrow$$
$$\mathrm{ASB}(\mathrm{I}\cdot h+\mathrm{J}-h(h+1)/2)$$
$$,\mathrm{J}=\mathrm{I}-h,\mathrm{I}-h+1,...,\mathrm{I}$$
$$,\mathrm{I}=h+2,h+3,...,\mathrm{N}$$

C23-15-0101 CTSDM, DCTSDM

Zero of a complex function (Muller's method)
CALL CTSDM (Z, ZFUN, ISW, EPS, ETA, M, ICON)

Function

This subroutine finds a zero of a complex function

$$f(z) = 0$$

by Muller's method.
An initial approximation to the zero must be given.

Parameters

Z	Input. An initial value to the zero to be obtained.
ZFUN ..	Input. The name of the complex function subprogram which evaluates the function $f(z)$.
ISW ...	Input. Control information. The user assigns one value from ISW= 1,2 and 3 to define the convergence criterion. When ISW=1, z_i becomes the root if it satisfies the condition $\lvert f(z_i)\rvert \le$ EPS: Criterion I. When ISW=2, z_i becomes the root if it satisfies $\lvert z_i - z_{i-1}\rvert \le$ ETA · $\lvert z_i\rvert$: Criterion II. When ISW=3, z_i becomes the root if either of the criteria described above are satisfied.
EPS ...	Input. The tolerance used in Criteria I. (See parameter ISW.) EPS ≥ 0.0.
ETA	Inpuf. The tolerance used in Criteria II. (See parameter ISW.) ETA ≥ 0.0.
M	Input. The upper limit of iterations (> 0) (See Notes.) Output. The number of iterations executed.
ICON ...	Output. Condition code. See Table CTSDM-1.

Table CTSDM-1 Condition codes

Code	Meaning	Processing
1	Z_j satisfied Criteria I. (See parameter ISW.)	Normal return
2	Z_j satisfied Criteria II. (See apraameter ISW.)	Normal return
10	The iteration was repeated m times (M=-m)	Normal return
11	Although M=-m on input the iteration stopped because $\lvert f(z_j)\rvert=0$ was satisfied during iterations and z_j was taken as a root.	Normal return
12	Although M=-m on input the iteration stopped because $\lvert z_j-z_{j-1}\rvert \le u\cdot\lvert z_j\rvert$was satisfied during iteration and z_j was taken as a root.	Normal return
10000	Specified criterion was not met during the specified number of iteration.	The last z_j was returned in parameter Z.
20000	A certain difficulty occurred, so iteration could not be continued. (See notes.)	Bypassed
30000	Error(s) in parameter setting. When M>0, 1 ISW=1 and EPS<0, or 2 ISW=2 and ETA<0, or 3 ISW=3 and EPS<0 and/or ETA<0. or when M = 0, or ISW≠1, 2 or 3.	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH, AFMAX and AFMIN
  FORTRAN basic functions ... CABS, CMPLX, EXP, ALOG, CSQRT, SQRT, REAL and AIMAG

- Notes
  The complex function subprogram associated with argument ZFUN must be declared by the EXTERNAL statement in the calling program.
  Iterations are repeated *m* times unconditionally, when M is set equal to M = -*m* (*m* > 0). However, the iteration will stop during iterations when the lefthand side of the Criteria I is equal to 0.0, or the lefthand side of the Criteria II is smaller than or equal to the round-off level.

- Example
  One root of $f(z) = e^z-i$ is obtained with initial value $z_0 = 0$.

```
C     **EXAMPLE**
      COMPLEX Z,ZFEXP
      EXTERNAL ZFEXP
      Z=CMPLX(0.0,0.0)
      ISW=3
      EPS=0.0
      ETA=1.0E-6
      M=100
      CALL CTSDM(Z,ZFEXP,ISW,EPS,ETA,M,
     *ICON)
      WRITE(6,600) ICON,M,Z
      STOP
  600 FORMAT(10X,'ICON=',I5/13X,'M=',I5/
     *       13X,'Z=',2E15.7)
      END
      FUNCTION ZFEXP(Z)
      COMPLEX Z,ZFEXP
      ZFEXP=CEXP(Z)-CMPLX(0.0,1.0)
      RETURN
      END
```

Method

The subroutine uses Muller's method. The method uses a interpolating polynomial $P(z)$ of degree two which approximates $f(z)$ near a root. This algorithm has the following features.

- Derivatives of $f(z)$ are not required.
- The function is evaluated only once at each iteration.
- The order of convergence is 1.84 (for a single root).

Muller method

Let α be a root of $f(z)$ and let three values z_{i-2},z_{i-1} and z_i be approximations to the root (See later for initial values z_1, z_2,and z_3). According to Newton's interpolation formula of degree two, $f(z)$ is approximated around the three values as follows:

$$\begin{aligned} P(z) = {} & f_i + f[z_i, z_{i-1}](z - z_i) \\ & + f[z_i, z_{i-1}, z_{i-2}](z - z_i)(z - z_{i-1}) \end{aligned} \qquad (4.1)$$

where f_i=$f(z_i)$ and $f[z_i,z_{i-1}]$,$f[z_i, z_{i-1},z_{i-2}]$ are the first and the second order divided differences of .$f(z)$ respectively and defined as follows.

$$\begin{aligned} f[z_i, z_{i-1}] &= \frac{f_i - f_{i-1}}{z_i - z_{i-1}} \\ f[z_i, z_{i-1}, z_{i-2}] &= \frac{f[z_i, z_{i-1}] - f[z_{i-1}, z_{i-2}]}{z_i - z_{i-2}} \end{aligned} \qquad (4.2)$$

$P(z)$ = 0 is then solved and the two roots are

$$\begin{aligned} z &= z_i - \frac{2f_i}{\omega \pm \{\omega^2 - 4 f_i f[z_i, z_{i-1}, z_{i-2}]\}^{1/2}} \\ \omega &= f[z_i, z_{i-1}] + (z_i - z_{i-1}) f[z_i, z_{i-1}, z_{i-2}] \end{aligned} \qquad (4.3)$$

Of these two roots of $P(z)$= 0, the root closer to z_i is taken as the next approximation z_{i+1}. In (4.1) when the term of z^2 is zero, that is when $f[z_i,z_{i-1},z_{i-2}]$ = 0, in stead of (4.3) the following formula is used.

$$\begin{aligned} z &= z_i - \frac{f_i}{f[z_i, z_{i-1}]} \\ &= z_i - \frac{z_i - z_{i-1}}{f_i - f_{i-1}} f_i \end{aligned} \qquad (4.4)$$

In (4.1), wheh the both terms of z and z^2 are null, $P(z)$ reduces to a constant and defeats the Muller's algrorithm (See later Considerations of Algorithm.)

- Initial values. z_1, $z_2$and z_3

  Let the initial value set by the user in input parameter Z be z when $|z| \neq 0$.

$$\begin{cases} z_1 = 0.9z \\ z_2 = 1.1z \\ z_3 = z \end{cases}$$

  when $|z|\neq0$

$$\begin{cases} z_1 = -1.0 \\ z_2 = 1.0 \\ z_3 = 0.0 \end{cases}$$

- When $f(z_{i-2}) = f(z_{i-1}) = f(z_i)$

  In this case, Muller's method will fail to continue iterations because both terms of z^2 and z in (4.1) vanish. The subroutine perturbs z_{i-2}, z_{i-1} and z_i so that the subroutine may get out of the situation $f(z_{i-2})$ =.$f(z_{i-1}) = f(z_i)$

$$\begin{aligned} z'_{i-2} &= (1 + p^n) z_{i-2} \\ z'_{i-1} &= (1 + p^n) z_{i-1} \\ z'_i &= (1 + p^n) z_i \end{aligned}$$

  where $p = -u^{-1/10}$, and u is the round-off unit, and n is the number of times of perturbation.

  If the perturbation continues more than five times, this subroutine will terminate unsuccessfully with ICON =20000.

Convergence criterion

Two condition are considered.

Condition I

When z_i satisfies $|f(z_i)| \leq$ EPS. it is taken as a root.

Condition II

When z_i satisfies $|z_i - z_{i-1}| \leq \mathrm{ETA} \cdot |z_i|$, it is taken as a root. When the root is a multiple root or very close to another root, ETA must be set sufficiently large. When ETA < u (u is the round-off unit), the subroutine will increase ETA as equal to u.

For further details, see Reference [32]

B51-30-0201 ECHEB, DECHEB

Evaluation of a Chebyshev series
CALL ECHEB (A, B, C, N, V, F, ICON)

Function

Given an *n*-terms Chebyshev series $f(x)$ defined on interval $[a, b]$

$$f(x)=\sum_{k=0}^{n-1}{}' c_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \qquad (1.1)$$

this subroutine obtains value $f(v)$ at arbitrary point v in the interval.

Symbol Σ' denotes the initial term only is taken with factor 1/2.

$a\neq b$, $v\in[a,b]$ and $n \ge 1$.

Parameters

A.......... Input. Lower limit a of the interval for the Chebyshev series.

B.......... Input. Upper limit b of the interval for the Chebyshev series.

C........... Input. Coefficients $\{c_k\}$.
C is a one dimensional array of size n.
Coefficients are stored as shown below:
C(1) = c_0, C(2)=c_1, ..., C(N)= c_{n-1}

N.......... Input. Number of terms n of the Chebyshev series.

V.......... Input. Point v which lies in the interal $[a, b]$.

F........... Output. Value $f(v)$ of the Chebyshev series.

ICON... Output. Condition code. See Table ECHEB-1.

Table ECHEB-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	One of the following occurred: 1 N<1 2 A = B 3 $v\notin$ [a, b]	Bypassed

Comments on use

- Subroutine used
  SSL II ... MGSSL
  FORTRAN basic function ... None

- Notes
  This subroutine obtains value $f(v)$ of a Chebyshev series. The subroutine FCHEB can be utilized for Chebyshev series expansion of arbitrary smooth function $f(x)$.

- Example
  This example uses the subroutine FCHEB for series expansion of the sine function

$$f(x)=\sin x\left(=\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{(2n+1)!}\right)$$

in the interval $[0, \pi]$.

Precision requirements ... Absolute error $5\cdot 10^{-5}$.

By using the resultant expanded coefficients this subroutine evaluate Chebyshev series at 32 Chebyshev points as follows:

$$x_j=\frac{\pi}{2}+\frac{\pi}{2}\cos\frac{j}{32}\pi\left(=\pi\cos^2\frac{j}{64}\pi\right)$$
$$,j=0,1,...,31$$

This example also evaluates and prints its error.

```
C      **EXAMPLE**
       DIMENSION C(257),TAB(255)
       EXTERNAL FUN
       EPSA=5.0E-5
       EPSR=0.0
       NMIN=9
       NMAX=257
       A=0.0
       B=ATAN(1.0)*4.0
       CALL FCHEB(A,B,FUN,EPSA,EPSR,NMIN,
      *           NMAX,C,N,ERR,TAB,ICON)
       IF(ICON.NE.0) GOTO 20
       WRITE(6,600) N,ERR,ICON
       WRITE(6,601) (C(K),K=1,N)
       WRITE(6,602)
       X=A
       NS=32
       H=ATAN(1.0)*2.0/FLOAT(NS)
       DO 10 J=1,NS
       X=B*COS(H*FLOAT(J-1))**2
       CALL ECHEB(A,B,C,N,X,Y,ICON)
       IF(ICON.NE.0) GOTO 20
       ERROR=FUN(X)-Y
       WRITE(6,603) X,Y,ERROR
    10 CONTINUE
       STOP
    20 WRITE(6,604) ICON
       STOP
   600 FORMAT('0',3X,'EXPANSION OF',
      1' FUNCTION FUN(X)',3X,'N=',I4,3X,
      2'ERROR=',E13.3,3X,'ICON=',I6)
   601 FORMAT (/(5E15.5))
   602 FORMAT ('0',10X,'X',7X,'EVALUATION',
      18X,'ERROR'/)
   603 FORMAT(1X,3E15.5)
   604 FORMAT('0',5X,'CONDITION CODE',I8)
       END
      FUNCTION FUN(X)
      REAL*8 SUM,XP,TERM
      EPS=AMACH(EPS)
      SUM=X
      P=X*X
      XP=X*P
      XN=-6.0
      N=3
   10 TERM=XP/XN
      SUM=SUM+TERM
      IF(DABS(TERM).LE.EPS) GO TO 20
      N=N+2
      XP=XP*P
      XN=-XN*FLOAT(N)*FLOAT(N-1)
      GOTO 10
   20 FUN=SUM
      RETURN
      END
```

Method

The value of an n -terms Chebyshev series (1.1) is obtained by the backward recurrence formula.

- Backward recurrence formula

  To obtain the value $f(v)$ of Chebyshev series

$$f(x)=\sum_{k=0}^{n-1}{}' c_k T_k(x)$$

at the interval [-1,1], the following adjoint sequence $\{b_k\}$ is effective.

If $\{b_k\}$ is defined as:

$$\begin{aligned} b_{n+2} &= b_{n+1} = 0 \\ b_k &= 2vb_{k+1} - b_{k+2} + c_k, k = n, n-1, \ldots \end{aligned} \quad (4.1)$$

the value $f(v)$ of the Chebyshev series can be obtained by the following expression:

$$f(v) = (b_0 - b_2)/2 \quad (4.2)$$

This subroutine first transforms variable $v\in$ $[a, b]$ to variable $s\in$ [-1,1].

$$s = \frac{2v - (b+a)}{b-a} \quad (4.3)$$

Then it obtains Chebyshev series value $f(v)$ by using (4.1) and (4.2).

The number of multiplications required to evaluate the value of an n-terms Chebyshev series is approximately n.

E51-10-0201 ECOSP, DECOSP

Evaluation of a cosine series
CALL ECOSP (TH, A, N, V, F, ICON)

Function

Given an n-terms cosine series with period $2T$

$$f(t)=\frac{1}{2}a_0+\sum_{k=1}^{n-1}a_k\cos\frac{\pi}{T}kt \tag{1.1}$$

this subroutine obtains the value $f(v)$ for arbitrary point v. $T>0$ and $n \geq 1$.

Parameters

TH.....	Input. Half period T of the cosine series.
A.......	Input. Coefficients $\{a_k\}$. A is a one-dimensional array of size N. Each coefficient is stored as shown below: A(1) = a_0, A(2) = a_1, ..., A(N) = a_{n-1}
N.......	Input. Number of terms n of the cosine series.
V.......	Input. Point v.
F........	Output. Value $f(v)$ of the cosine series.
ICON...	Output. Condition code. See Table ECOSP-1.

Table ECOSP-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either the two occurred. 1 N<1 2 TH ≤ 0	Bypassed

Comments on use

- Subroutine used

  SSL II ... MGSSL

  FORTRAN basic functions ... COS and ATAN

- Notes

  This subroutine obtains value $f(v)$ of a cosine series.

  The FCOSF subroutine can be utilized when smooth even function $f(t)$ with period $2T$ is subject to cosine series expansion.

- Example

  This example integrates an odd function with parameter ω

$$F(x)=\int_0^x\frac{\sin\omega t}{\sqrt{1+\sin^2\omega t}}dt \tag{3.1}$$
$$,\omega=\frac{\pi}{4},\frac{\pi}{2},\pi,2\pi,4\pi$$

when $x= h, 2h, ..., 10h$. However $h = \pi/(10\omega)$.

Since this integrand is an odd function with period $2\pi/\omega$ it is expanded at first in a sine series by the subroutine FSINF according to the required precision of $\varepsilon_a = \varepsilon_r = 5 \cdot 10^{-5}$:

$$f(t)\approx\sum_{k=0}^{n-1}b_k\sin k\omega t \tag{3.2}$$

By integrating (3.2) termwise it can be obtained that

$$F(x)=\sum_{k=0}^{n-1}{}'a_k\cos k\omega x,$$
$$a_k=-\frac{b_k}{k\omega},k=1,2,...,n-1 \tag{3.3}$$

However the initial term a_0 is determined to satisfy $F(0) = 0$.

At this stage the subroutine ECOSP can be called to obtain an approximate value for indefinite integral (3.1) when $x = h, 2h, ..., 10h$.

Since this analytical solution is

$$F(x)=\frac{1}{\omega}\left(\frac{\pi}{4}-\sin^{-1}\left(\cos\omega x/\sqrt{2}\right)\right)$$

it is compared with the result obtained by this subroutine.

```
C      **EXAMPLE**
       DIMENSION A(257),TAB(127)
       EXTERNAL FUN
       COMMON W,PI
       PI=ATAN(1.0)*4.0
       EPSA=0.5E-04
       EPSR=EPSA
       NMIN=0
       NMAX=257
       W=PI*0.25
       DO 1 I=1,5
       TH=PI/W
C      EXPANSION OF INTEGRAND
       CALL FSINF(TH,FUN,EPSA,EPSR,NMIN,
      *           NMAX,A,N,ERR,TAB,ICON)
       IF(ICON.GT.10000) GO TO 10
       WRITE(6,600) N,ERR,ICON,W
       WRITE(6,601) (A(K),K=1,N)
C      TERMWISE INTEGRATION
       DO 2 K=2,N
       A(K)=-A(K)/(FLOAT(K-1)*W)
     2 CONTINUE
C      EVALUATION OF COSINE SERIES
       CALL ECOSP(TH,A,N,0.0,P,ICON)
       IF(ICON.NE.0) GO TO 10
       A(1)=-P*2.0
       WRITE(6,610)
       H=TH*0.1
       DO 3 K=1,10
       X=H*FLOAT(K)
       CALL ECOSP(TH,A,N,X,P,ICON)
       IF(ICON.NE.0) GO TO 10
       Q=TRFN(X)
```

```
    WRITE(6,611) X,P,Q
  3 CONTINUE
    W=W+W
  1 CONTINUE
    STOP
 10 WRITE(6,620) ICON
    STOP
600 FORMAT('0',5X,'EXPANSION OF',
   *' INTEGRAND',3X,'N=',I4,5X,'ERR=',
   *E15.5,5X,'ICON=',I5,5X,'W=',E15.5)
601 FORMAT(/(5E15.5))
610 FORMAT('0',5X,'EVALUATION OF',
   *' COSINE SERIES'/'0',6X,
   *'AUGUMENT',7X,'COMPUTED',11X,'TRUE')
611 FORMAT(3E15.5)
620 FORMAT('0',5X,'CONDITION CODE',I6)
    END
    FUNCTION FUN(T)
    COMMON W
    X=W*T
    P=SIN(X)
    FUN=P/SQRT(P*P+1.0)
    RETURN
    END
    FUNCTION TRFN(T)
    COMMON W,PI
    TRFN=(PI*.25-ASIN(COS(W*T)*
   *     SQRT(0.5)))/W
    RETURN
    END
```

Method

This subroutine obtains a value of an n-terms cosine series (1.1) by using a backward recurrence formula.

Upon choosing

$$\theta = \frac{\pi}{T} t$$

$f(t) = g\ (\theta)$ leads to a cosine series with peritd 2π.

Therefore the value of $g(\theta)$ can be obtained when

$$\theta = \frac{\pi}{T} v$$

is satisfied. It can be calculated efficiently throuhg use of the backward recurrence formula (4.1).

$$\begin{aligned} &b_{n+2} = b_{n+1} = 0 \\ &b_k = 2\cos\theta \cdot b_{k+1} - b_{k+2} + a_k \\ &k = n, n-1, \ldots, 1, 0 \end{aligned} \tag{4.1}$$

The value of an n-terms cosine series can be obtained by (4.2).

$$f(v) = g(\theta) = (b_0 - b_1)/2 \tag{4.2}$$

The number of multiplications required to evaluate the value of an n-terms cosine series is approximately n.

The cosine function is evaluated only once.

B21-11-0101 EIG1, DEIG1

Eigenvalues and corresponding eigenvectors of a real matrix (double QR method)
CALL EIGl (A, K, N, MODE, ER, EI, EV, VW, ICON)

Function

All eigenvalues and corresponding eigenvectors of an *n*-order real matrix ***A*** are determined. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$. $n \geq 1$.

Parameters

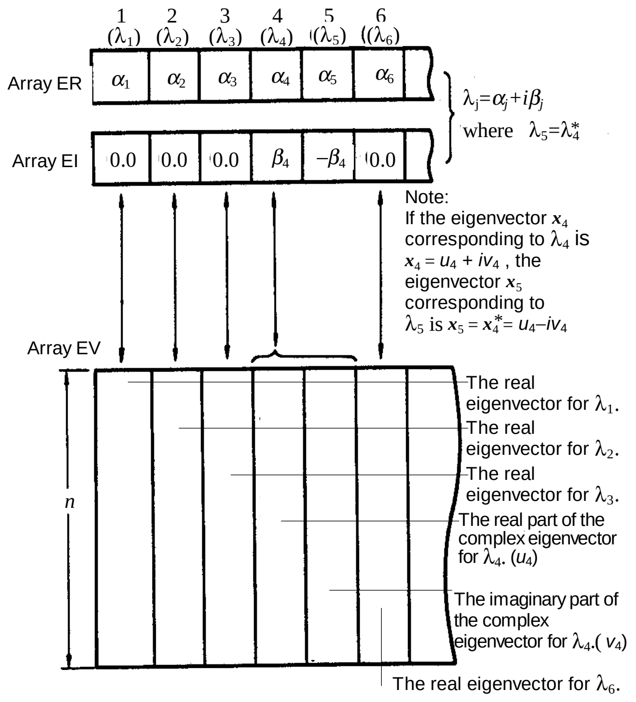
A	Input. Real matrix ***A***. A is a two-dimensional array, A (K, N). The contents of A are altered on output.
K	Input. The adjustable dimension of arrays A and EV.
N	Input. Order *n* of ***A***.
MODE ...	Input. Specifies balancing. MODE=1... Balancing is omitted. MODE≠1... Balancing is included.
ER,EI	Output. Eigenvalues. Eigenvalues are divided into their real and imaginary parts. The real part is stored in ER, and the imaginary part is stored in EI. If the jth eigenvalues is complex, the (j+1)th eigenvalues is its complex conjugate (refer to Fig. EIG1-1). ER and EI are one-dimensional arrays of size *n*.
EV	Output. Eigenvectors. Eigenvectors are stored in the columns which correspond to their eigenvalues. If an eigenvalue is complex, its eigenvector is also complex; such eigenvectors are stored as shown in Fig. EIG1-1. For details see “Comments on use”. EV is a two-dimensional array, EV (K,N).
VW	Work area. Used during balancing and reducing matrix ***A*** to a real Hessenberg matrix. VW is a one-dimensional array of size *n*.
ICON	Output. Condition code. Refer to Table EIG1-1.

Table EIG1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	ER (1) = A (1, 1), EV (1,1) = 1.0
20000	Eigenvalues and eigenvectors could not be determined since reduction to a triangular matrix was not possible.	Discontinued
30000	N < 1 or K < N	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, BLNC, IRADIX, HES1, and MGSSL
  FORTRAN basic functions ... ABS, SQRT, SIGN, and DSQRT.



1 2 3 4 5 6
(λ1) (λ2) (λ3) (λ4) (λ5) (λ6)
Array ER
α1 α2 α3 α4 α5 α6
λj=αj+iβj
where λ5=λ4*
Array EI
0.0 0.0 0.0 β4 −β4 0.0
Note:
If the eigenvector x4 corresponding to λ4 is x4 = u4 + iv4 , the eigenvector x5 corresponding to λ5 is x5 = x4*= u4−iv4
Array EV
n
The real eigenvector for λ1.
The real eigenvector for λ2.
The real eigenvector for λ3.
The real part of the complex eigenvector for λ4. (u4)
The imaginary part of the complex eigenvector for λ4.(v4)
The real eigenvector for λ6.

Fig. EIG1-1 Corresponding between eigenvalues and eigenvectors

- Notes
  Complex eigenvalues and corresponding eigenvectors
  In general, real matrices have real eigenvalues and complex eigenvalues. Complex eigenvalues become pairs of conjugate complex eigenvalucs. In this routine, if the *j*th eigenvalue (λ_j) is a complex eigenvalue, λ and λ^*_j are stored in ER and EI in the following way.

$$\begin{aligned}\lambda_j &= \mathrm{ER(J)} + i\cdot\mathrm{EI(J)} \quad (i\text{: imaginary unit})\\ \lambda_j^* &= \mathrm{ER(J+1)} + i\cdot\mathrm{EI(J+1)}\\ &= \mathrm{ER(J)} - i\cdot\mathrm{EI(J)}\end{aligned}$$

If eigenvector $\boldsymbol{x}_j$ which corresponds to λ_j becomes a complex vector.

$$\boldsymbol{x}_j = \boldsymbol{u}_j + i\boldsymbol{v}_j$$

Then. eigenvector $\boldsymbol{x}_j^*$ which corresponds to λ_j^* becomes

$$\boldsymbol{x}_j^* = \boldsymbol{u}_j + i\boldsymbol{v}_j$$

Therefore if real part u_j and imaginary part v_j of $\boldsymbol{x}$ are obtained, $\boldsymbol{x}$ can easily be determined. Consequently, in this subroutine, only eigenvector $\boldsymbol{x}_j$ which corresponds to λ_j is calculated. Real part u_j of $\boldsymbol{x}_j$ is stored in the Jth column of EV, and imaginary part v_j is stored in the (J+1)th column. Refer to Fig. EIG1-1. If the magnitude of each element in a real matrix varies greatly, the precision of the result can be improved by balancing the matrix with subroutine BLNC. Balancing will produce minimal improvement if the magnitude of each element in a matrix is about the same and should be skipped by setting MODE=1. This subrotine is used to obtaine all eigenvalues and corresponding eigenvectors of a real matrix. If all the eigenvalues of a real matrix are desired, BLNC, HES1 and HSQR should be used. If a subset of the eigenvectors of a real matrix is desired, BLNC, HES1, HSQR, HVEC, and HBK1 should be used.

- Example

  All eigenvalues and corresponding eigenvectors of a real matrix $\boldsymbol{A}$ of order n are determined. $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),ER(100),EI(100),
     * EV(100,100),VW(100),IND(100)
   10 CONTINUE
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610)(I,J,A(I,J),J=1,N)
      CALL EIG1(A,100,N,0,ER,EI,EV,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      IND(1)=0
      CALL EPRT(ER,EI,EV,IND,100,N,N)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('0',5X,'ORIGINAL MATRIX', 5X,
     * 'N=',I3)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     * E15.7))
  620 FORMAT('0',5X,'ICON=',I5)
      END
```

In the above example, the subroutine EPRT prints all eigenvalues and corresponding eigenvectors of a real matrix. The contents of EPRT are:

```
      SUBROUTINE EPRT(ER,EI,EV,IND,K,N,M)
      DIMENSION ER(M),EI(M),EV(K,M),IND(M)
      WRITE(6,600)
      IF(IABS(IND(1)).EQ.1) GO TO 40
      J=0
      DO 30 I=1,M
      IF(J.EQ.0) GO TO 10
      IND(I)=0
      J=0
      GO TO 30
   10 IF(EI(I).NE.0.0) GO TO 20
      IND(I)=1
      GO TO 30
   20 IND(I)=-1
      J=1
   30 CONTINUE
   40 MM=0
      DO 50 I=1,M
      IF(IND(I).EQ.0) GO TO 50
      MM=MM+1
      ER(MM)=ER(I)
      IND(MM)=IND(I)
      IF(EI(I).EQ.0.0) GO TO 50
      MM=MM+1
      ER(MM)=EI(I)
      IND(MM)=IND(I+1)
   50 CONTINUE
      KAI=(MM-1)/5+1
      LST=0
      DO 70 L=1, KAI
      INT=LST+1
      LST=LST+5
      IF(LST.GT.MM) LST=MM
      WRITE(6,610) (J,J=INT,LST)
      WRITE(6,620) (ER(J),J=INT,LST)
      WRITE(6,630) (IND(J),J=INT,LST)
      DO 60 I=1,N
      WRITE(6,640) I,(EV(I,J),J=INT,LST)
   60 CONTINUE
   70 CONTINUE
      RETURN
  600 FORMAT('1',10X,'**EIGENVECTORS**')
  610 FORMAT('0',5I20)
  620 FORMAT('0',1X,'EIGENVALUE',5E20.8)
  630 FORMAT(2X,'IND',6X,I10,4I20)
  640 FORMAT(6X,I5,5E20.8)
      END
```

Method

All eigenvalues and corresponding eigenvectors of an n-order real matrix $\boldsymbol{A}$ are determined as follows: Using the following two-stage transformation process, the eigenvalues of an n-order real matrix are determined by obtaining the diagonal elements of upper triangular matrix $\boldsymbol{R}$.

- Reduction of real matrix $\boldsymbol{A}$ to real Hessenberg matrix $\boldsymbol{H}$ using the Householder method.

$$\boldsymbol{H} = \boldsymbol{Q}_H^{\mathrm{T}} \boldsymbol{A} \boldsymbol{Q}_H \tag{4.1}$$

  $\boldsymbol{Q}_H$ is an orthogonal matrix which is the product of the transformation matrices $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, ..., $\boldsymbol{P}_{n-2}$ used in the Householder method.

$$\boldsymbol{Q}_H = \boldsymbol{P}_1 \cdot \boldsymbol{P}_2 \cdot \ldots \cdot \boldsymbol{P}_{n-2} \tag{4.2}$$

  For a description of the Householder method, refer to HESl.

- Reduction of a real Hessenberg matrix $\boldsymbol{H}$ to an upper triangular matrix $\boldsymbol{R}$ using the double QR method.

$$\boldsymbol{R} = \boldsymbol{Q}_R^{\mathrm{T}} \boldsymbol{H} \boldsymbol{Q}_R \tag{4.3}$$

$\boldsymbol{Q}_R$ is an orthogonal matrix which is the product of the transformation matrices $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, ..., $\boldsymbol{Q}_s$ used in the double QR method.

$$\boldsymbol{Q}_R = \boldsymbol{Q}_1 \cdot \boldsymbol{Q}_2 \cdot \ldots \cdot \boldsymbol{Q}_s \tag{4.4}$$

For further information on the double QR method, refer to HSQR. If matrix $\boldsymbol{F}$, which transforms upper triangular matrix $\boldsymbol{R}$ to diagonal matrix $\boldsymbol{D}$, in the similarity transformation of (4.5) is available, eigenvectors can be obtained as the column vectors of matrix $\boldsymbol{X}$ in (4.6).

$$\boldsymbol{D} = \boldsymbol{F}^{-1} \boldsymbol{R} \boldsymbol{F} \tag{4.5}$$
$$\boldsymbol{X} = \boldsymbol{Q}_H \boldsymbol{Q}_R \boldsymbol{F} \tag{4.6}$$

To verify that the column vectors of amtrix $\boldsymbol{X}$ in (4.6) are the eigenvectors of real matrix $\boldsymbol{A}$, substitute (4.1) and (4.3) in (4.5) to obtain (4.7).

$$\boldsymbol{D} = \boldsymbol{X}^{-1} \boldsymbol{A} \boldsymbol{X} = \boldsymbol{F}^{-1} \boldsymbol{Q}_R^{\mathrm{T}} \boldsymbol{Q}_H^{\mathrm{T}} \boldsymbol{A} \boldsymbol{Q}_H \boldsymbol{Q}_R \boldsymbol{F} \tag{4.7}$$

If $\boldsymbol{Q}_H \boldsymbol{Q}_R$ are represented as $\boldsymbol{Q}$, from (4.2) and (4.4), then

$$\boldsymbol{Q} = \boldsymbol{P}_1 \cdot \boldsymbol{P}_2 \cdot \ldots \cdot \boldsymbol{P}_{n-2} \cdot \boldsymbol{Q}_1 \cdot \boldsymbol{Q}_2 \cdot \ldots \cdot \boldsymbol{Q}_s \tag{4.8}$$

As shown in (4.8), $\boldsymbol{Q}$ can be computed by repeatedly taking the product of the transformation matrices. $\boldsymbol{X}$ can be obtained by taking the product of $\boldsymbol{Q}$ from (4.8) and matrix $\boldsymbol{F}$. Transformation matrix $\boldsymbol{F}$ can be determined as an upper triangular matrix as follows: From (4.5),

$$\boldsymbol{F}\boldsymbol{D} = \boldsymbol{R}\boldsymbol{F} \tag{4.9}$$

Let the elements of $\boldsymbol{D}$, $\boldsymbol{R}$, and $\boldsymbol{F}$ be represented as $\boldsymbol{D} = \mathrm{diag}(\lambda_i)$, $\boldsymbol{R} = (r_{ij})$, $\boldsymbol{F} = (f_{ij})$, then elements f_{ij} are obtained from (4.9) as in (4.10) for $j = n, n-1, ..., 2$.

$$f_{ij} = \sum_{k=i+1}^{j} r_{ik} f_{kj} \big/ \left(\lambda_j - \lambda_i\right) \left(i = j-1, j-2, \ldots, 1\right) \tag{4.10}$$

where, $r_{ij} = 0$ $(i>j)$, $r_{ii} = \lambda_i$
$f_{ij} = 0$ $(i>j)$, $f_{ii} = 1$

If $\lambda_i = \lambda_j$, f_{ij} is obtained as

$$f_{ij} = \sum_{k=i+1}^{j} r_{ik} f_{kj} \big/ u \|\boldsymbol{R}\|_\infty \tag{4.11}$$

where u is the unit round-off
The above computations are performed only when all the eigenvalues are real. However, if real matrix $\boldsymbol{A}$ has complex eigenvalues, $\boldsymbol{R}$ cannot become a complete triangular matrix. The elements of matrix $\boldsymbol{R}$ which has complex eigenvalues λ_{l-1} and $\lambda_l (= \lambda_{l-1}^*)$ are shown in Fig. EIG1-2.

$$\begin{bmatrix} r_{12} & r_{12} & \cdots & r_{1\,l-1} & r_{1l} & \cdots & r_{1n} \\ & r_{22} & & \vdots & \vdots & & \vdots \\ & & \ddots & & & & \\ & & & r_{l-1\,l-1} & r_{l-1\,l} & \cdots & r_{l-1\,n} \\ & & & r_{l\,l-1} & r_{ll} & \cdots & r_{ln} \\ & & & & & \ddots & \vdots \\ & & & & & & r_{nn} \end{bmatrix}$$

Fig. EIG1-2 Quasi triangular matrix $\boldsymbol{R}$ with a complex eigenvalue (λ_{l-1}, λ_l)

In this case, the procedure to obtain f_{ij} becomes complicated. When, in the course of computation f_{ij} in (4.10), if a complex eigenvalue λ_i $(i = l)$ is encountered, f_{lj} and f_{l-1j} are obtained by solving a system of linear equations (4.12).

$$\left.\begin{aligned} \left(r_{l-1l-1} - \lambda_j\right) f_{l-1j} + r_{l-1l} f_{lj} &= -\sum_{k=l+1}^{j} r_{l-1k} f_{kj} \\ r_{ll-1} f_{l-1j} + \left(r_{ll} - \lambda_j\right) f_{lj} &= -\sum_{k=l+1}^{j} r_{lk} f_{kj} \end{aligned}\right\} \tag{4.12}$$

In addition, the lth and $(l-1)$th column elements of $\boldsymbol{F}$ becomes complex vectors. However, since λ_l and λ_{l-1} are conjugates, their complex vectors are also conjugates of each other. Since, if one of them is determined the other need not be computed, only f_{il-1} $(i=l,l-1,...,1)$ is obtained corresponding to λ_{l-1} in this routine. f_{il-1} is determined as follows:
That is, for $i=l$ and $l-1$, f_{il-1} is determined from (4.13) and for $i=l-2$, $l-3$, ...1, f_{il-1} is determined from (4.10).

$$\begin{aligned} f_{ll-1} &= 1 \\ f_{l-1l-1} &= \begin{cases} -r_{l-1l} / \left(r_{l-1l-1} - \lambda_{l-1}\right) & \left(r_{ll-1} \le r_{l-1l}\right) \\ -\left(r_{ll} - \lambda_{l-1}\right) / r_{ll-1} & \left(r_{ll-1} > r_{l-1l}\right) \end{cases} \end{aligned} \tag{4.13}$$

On computation of f_{ij-1} in (4.10), if λ_i and λ_{i-1} are furthermore a pair of complex conjugate eigenvalues, corresponding f_{il-1} and f_{i-1l-1} are obtained from (4.12).
The real part of f_{il-1} is stored in the $(l-1)$th column of $\boldsymbol{F}$ and the imaginary part of f_{il-1} is stored in the lth column.

From matrix ***F*** thus obtained and transformation matrix ***Q***, eigenvectors ***x*** of brief description of this procedure follows:

$$\boldsymbol{X} = \boldsymbol{QF} \qquad (4.14)$$

Where, ***X*** is normalized such that $\|\boldsymbol{x}\|_2 = 1$.

Also, in this routine, a real matrix is balanced using BLNC before it is reduced to a Hessenberg matrix unless MODE=1.

For further information, see References [12] and [13] pp 372-395.

E51-20-0201 ESINP, DESINP

Evaluation of a sine series
CALL ESINP (TH, B, N, V, F, ICON)

Function

Given an n-terms sine series with period $2T$

$$f(t)=\sum_{k=0}^{n-1} b_k \sin\frac{\pi}{T}kt, \quad b_0=0 \tag{1.1}$$

this subroutine obtains value $f(v)$ for arbitrary point v.
$T > 0$ and $n \geq 1$.

Parameters

TH.....	Input. Half period T for the sine series.
B.....	Input. Coefficients $\{b_k\}$. B is a one-dimensional array of size N. Each coefficient is stored as shown below: B(1)=0.0, B(2)=b_1, ..., B(N)=b_{n-1}
N.....	Input. Number of terms n of the sine series.
V.....	Input. Arbitrary point v.
F.....	Output. Value $f(v)$ of the sine series.
ICON...	Output. Condition code. See Table ESINP-1.

Table ESINP-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either the tow occurred: 1 N < 1 2 TH ≤ 0	Bypassed

Comments on use

- Subroutine used
  SSL II ... MGSSL
  FORTRAN basic functions ... COS, ATAN and SIN

- Notes
  This subroutine obtains value $f(v)$ of a sine series. The subroutine FSINF should be utilized when smooth odd function $f(t)$ with period $2T$ is subject to sine series expansion.

- Example
  This example integrates an even function having parameter ω.

$$F(x)=\int_0^x \frac{\cos\omega t}{\sqrt{1+\cos^2\omega t}}dt$$
$$,\omega=\frac{\pi}{4},\frac{\pi}{2},\pi,2\pi,4\pi \tag{3.1}$$

when $x = h, 2h, ..., 10h$. However $h = \pi/(10\omega)$. Since this integrand is an even function with period $2\pi/\omega$ it is expanded at first in a cosine series shown in (3.2) by the subroutine FCOSF according to the required precision of $\varepsilon_a = \varepsilon_r = 10^{-5}$.

$$f(t)\approx\frac{1}{2}a_0+\sum_{k=1}^{n-1}a_k\cos k\omega t \tag{3.2}$$

Upon integrating (3.2) termwise it can be obtained that

$$F(x)=\sum_{k=0}^{n-1}b_k\sin k\omega x,$$
$$b_0=0, b_k=\frac{a_k}{k\omega}, k=1,2,...,n-1 \tag{3.3}$$

At this time the subroutine ESINP is called to obtain the approximate value of indefinite integration (3.3) upon choosing $x = h, 2h, ..., 10h$.
Since this analytical solution is

$$F(x)=\frac{1}{\omega}\sin^{-1}\left(\frac{\sin\omega t}{\sqrt{2}}\right) \tag{3.4}$$

it is compared with the result obtained by this subroutine.

```
C     **EXAMPLE**
      DIMENSION A(257),TAB(127)
      EXTERNAL FUN
      COMMON W
      PI=ATAN(1.0)*4.0
      EPSA=0.5E-04
      EPSR=EPSA
      NMIN=0
      NMAX=257
      W=PI*0.25
      DO 1 I=1,5
      TH=PI/W
C     EXPANSION OF INTEGRAND
      CALL FCOSF(TH,FUN,EPSA,EPSR,NMIN,
     *           NMAX,A,N,ERR,TAB,ICON)
      IF(ICON.GT.10000) GO TO 10
      WRITE(6,600) N,ERR,ICON,W
      WRITE(6,601) (A(K),K=1,N)
C     TERMWISE INTEGRATION
      DO 2 K=2,N
      A(K)=A(K)/(FLOAT(K-1)*W)
    2 CONTINUE
C     EVALUATION OF SINE SERIES
      WRITE(6,610)
      H=TH*0.1
      DO 3 K=1,10
      X=H*FLOAT(K)
      CALL ESINP(TH,A,N,X,P,ICON)
      IF(ICON.NE.0) GO TO 10
      Q=TRFN(X)
      WRITE(6,611) X,P,Q
    3 CONTINUE
      W=W+W
    1 CONTINUE
      STOP
   10 WRITE(6,620) ICON
      STOP
  600 FORMAT('0',5X,'EXPANSION OF',
     *' INTEGRAND',3X,'N=',I4,5X,'ERR=',
     *E15.5,5X,'ICON=',I5,5X,'W=',E15.5)
```

```
601 FORMAT(/(5E15.5))
610 FORMAT('0',5X,'EVALUATION OF',
   *' SINE SERIES'/'0',6X,
   *'AUGUMENT',7X,'COMPUTED',11X,'TRUE')
611 FORMAT(3E15.5)
620 FORMAT('0',5X,'CONDITION CODE',I6)
    END
    FUNCTION FUN(T)
    COMMON W
    X=W*T
    P=COS(X)
    FUN=P/SQRT(P*P+1.0)
    RETURN
    END
    FUNCTION TRFN(T)
    COMMON W
    TRFN=ASIN(SIN(W*T)*SQRT(0.5))/W
    RETURN
    END
```

Method

This subroutine obtains a value of an *n*-terms sine series shown in (1.1) by using the backward recurrence formula.

Upon choosing $\theta = \frac{\pi}{T} t$, $f(t)=g(\theta)$ leads to a sine series with period 2π.

Therefore $g(\theta)$ can be obtained when $\theta = \frac{\pi}{T} v$ is satisfied – it can be calculated efficiently by the following backward recurrence formula:

$$\begin{aligned} & c_{n+2} = c_{n+1} = 0 \\ & c_k = 2\cos\theta \cdot c_{k+1} - c_{k+2} + b_k \\ & k = n, n-1, \ldots, 1 \end{aligned} \tag{4.1}$$

Therefore the value of an arbitrary point in the sine series can be obtained by (4.2).

$$f(v) = g(\theta) = c_1 \sin\theta \tag{4.2}$$

The number of multiplications required for evaluation of an *n*-terms sine series is approximately *n*. The cosine and sine functions are evaluated only once respectively.

I11-31-0101 EXPI, DEXPI

Exponential integrals $E_i(x)$, $\overline{E}_i\,(x)$
CALL EXPI (X, EI, ICON)

Function

This subroutine computes the exponential integrals $E_i(x)$ and $\overline{E}_i\,(x)$ defined as follows using an approximation formula.

For $x < 0$:

$$E_i(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}\,dt = \int_{-\infty}^{x} \frac{e^{t}}{t}\,dt$$

For $x > 0$:

$$\overline{E}_i(x) = \mathrm{P.V.}\int_{\infty}^{-x} \frac{e^{-t}}{t}\,dt = \mathrm{P.V.}\int_{-\infty}^{x} \frac{e^{t}}{t}\,dt$$

P.V. means that the principal value is taken at t=0.
Where, $x \neq 0$.

Parameters

X.....	Input. Independent variable x.
EI.....	Output. Function value $E_i(x)$ or $\overline{E}_i\,(x)$.
ICON.....	Output. Condition code. See Table EXPI-1.

Table EXPI-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	X > log(fl_{max}) or X < log(fl_{max})	EI is set to fl_{max} or EI is set to 0.0.
30000	X=0	EI is set to 0.0.

Comments on use

- Subprograms used
  SSL II ... AFMAX, MGSSL and ULMAX
  FORTRAN basic functions ... EXP, ALOG, and ABS

- Notes
  [Range of argument]

$$|X| \leq \log(fl_{\max})$$

If $|X|$ exceeds the above limit, $E_i(x)$ and $\overline{E}_i\,(x)$ respectively cause underflow or overflow in calculating e^x. The limitation is made so that these difficulties can be avoided beforehand in the subroutine.

$X \neq 0$
$E_i(x)$ and $\overline{E}_i\,(x)$ are undefined for x=0.

- Example
  The following example generates a table of the function values from 0.01 to 0.50 with increment 0.01.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,50
       X=FLOAT(K)/100.0
       CALL EXPI(X,EI,ICON)
       IF(ICON.EQ.0) WRITE(6,610)X,EI
       IF(ICON.NE.0) WRITE(6,620)X,EI,ICON
   10 CONTINUE
       STOP
  600 FORMAT('1','EXAMPLE OF EXPONENTIAL ',
     * 'INTEGRAL FUNCTION'///
     * 6X,'X',9X,'EI(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'EI=',E17.7,5X,'CONDITION=',
     * I10)
       END
```

Method

The approximation formulas to compute $E_i(x)$ and $\overline{E}_i\,(x)$, differs depending on the following ranges of x.
[-174.673, -4), [-4, -1), [-1, 0), (0, 6], (6, 12], (12, 24], (24, 174.673]. In the following, $s=|x|$, $t=1/|x|$.

- For $\log(fl_{max}) \leq x < -4$
  Single precision:

$$E_i(x) = -te^{x}\left(1 + \sum_{k=0}^{3} a_k t^{k+1} \Big/ \sum_{k=0}^{3} b_k t^{k}\right) \qquad (4.1)$$

Theoretical precision = 9.09 digits

Double precision:

$$E_i(x) = -te^{x}\left(1 + \sum_{k=0}^{8} a_k t^{k+1} \Big/ \sum_{k=0}^{8} b_k t^{k}\right) \qquad (4.2)$$

Theoretical precision = 18.88 digits

- For $-4 \leq x < -1$
  Single precision:

$$E_i(x) = -\,e^{x}\sum_{k=0}^{4} a_k t^{k} \Big/ \sum_{k=0}^{4} b_k t^{k} \qquad (4.3)$$

Theoretical precision = 10.04 digits
Double precision:

$$E_i(x) = -e^x \sum_{k=0}^{8} a_k t^k \bigg/ \sum_{k=0}^{8} b_k t^k \qquad (4.4)$$

Theoretical precision = 19.20 digits

- For $-1 \le x < 0$
  Single precision:

$$E_i(x) = \log(s) - \sum_{k=0}^{3} a_k s^k \bigg/ \sum_{k=0}^{3} b_k s^k \qquad (4.5)$$

Theoretical precision = 10.86 digits

Double precision:

$$E_i(x) = \log(s) - \sum_{k=0}^{5} a_k s^k \bigg/ \sum_{k=0}^{5} b_k s^k \qquad (4.6)$$

Theoretical precision = 18.54 digits

- For $0 < x \le 6$
  Single precision:

$$\overline{E}_i(x) = \log(x/x_0) + (x - x_0) \sum_{k=0}^{5} a_k z^k \bigg/ \sum_{k=0}^{5} b_k z^k \qquad (4.7)$$

Theoretical precision = 9.94 digits
Where $z = x/6$, x_0=0.37250 7410
Double precision:

$$\overline{E}_i(x) = \log(x/x_0) + (x - x_0) \sum_{k=0}^{9} a_k z^k \bigg/ \sum_{k=0}^{9} b_k z^k \qquad (4.8)$$

Theoretical precision = 20.76 digits

Where $z = x/6$, x_0=0.37250 7410 81366 63446
- For $6 < x \le 12$

  Single precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ a_0 + \frac{b_0 |}{|a_1 + x} + \frac{b_1 |}{|a_2 + x} + \frac{b_2 |}{|a_3 + x} + \frac{b_3 |}{|a_4 + x} \right\} \qquad (4.9)$$

Theoretical precision = 8.77 digits
Double precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ a_0 + \frac{b_0 |}{|a_1 + x} + \frac{b_1 |}{|a_2 + x} + \dots + \frac{b_8 |}{|a_9 + x} \right\} \qquad (4.10)$$

Theoretical precision = 19.21 digits

- For $12 < x \le 24$
  Single precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ a_0 + \frac{b_0 |}{|a_1 + x} + \frac{b_1 |}{|a_2 + x} + \frac{b_2 |}{|a_3 + x} + \frac{b_3 |}{|a_4 + x} \right\} \qquad (4.11)$$

Theoretical precision = 9.45 digits
Double precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ a_0 + \frac{b_0 |}{|a_1 + x} + \frac{b_1 |}{|a_2 + x} + \dots + \frac{b_8 |}{|a_9 + x} \right\} \qquad (4.12)$$

Theoretical precision = 19.22 digits

- For $24 < x \le \log(fl_{max})$
  Single precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ 1 + \frac{1}{x}\left(a_0 + \frac{b_0 |}{|a_1 + x} + \frac{b_1 |}{|a_2 + x} \right) \right\} \qquad (4.13)$$

Theoretical precision = 8.96 digits
Double precision:

$$\overline{E}_i(x) = \frac{e^x}{x}\left\{ 1 + \frac{1}{x}\left(a_0 + \frac{b_0 |}{|a_1 + x} + \dots + \frac{b_8 |}{|a_9 + x} \right) \right\} \qquad (4.13)$$

Theoretical precision = 18.11 digits

For more information, see Reference [79] and [80].

E51-30-0101 FCHEB, DFCHEB

Chebyshev series expansion of a real function (Fast cosine transform, function input)
CALL FCHEB (A, B, FUN, EPSA, EPSR, NMIN, NMAX, C, N, ERR, TAB, ICON)

Function

Given a smooth function $f(x)$ on the interval $[a, b]$ and required accuracy ε_a and ε_r, this subroutine performs Chebyshev series expansion of the function and determines n coeffieients $\{c_k\}$ which satisfy (1.1).

$$\left| f(x) - \sum_{k=0}^{n-1}{}' c_k T_k\left(\frac{2x-(b+a)}{b-a}\right)\right| \le \max\{\varepsilon_a, \varepsilon_r \cdot \|f(x)\|\} \tag{1.1}$$

Symbol Σ' denotes to make sum but the initial term only is multipled by factor 1/2. $\|f\|$ can be defined as (1.3) by using function values taken at sample points x_j (shown in (1.2)) on the interval $[a, b]$.

$$x_j = \frac{a+b}{2} + \frac{b-a}{2}\cos\frac{\pi}{n-1}j\ ,\quad j = 0,1,...,n-1 \tag{1.2}$$

$$\|f\| = \max_{0\le j\le n-1}\left|f(x_j)\right| \tag{1.3}$$

Where $a \ne b$ and $\varepsilon_a \ge 0$ and $\varepsilon_r \ge 0$.

Parameters

A.............	Input. Lower limit a of the interval.
B.............	Input. Upper limit b of the interval.
FUN........	Input. Name of the function subprogram which calculates the function $f(x)$ to be expanded. See Notes.
EPSA......	Input. The absolute error tolerance ε_a.
EPSR......	Input. The relative error tolerance ε_r. Lower limit.
NMIN.....	Input. Lower limit of terms of Chebyshev series (≥0). The value of NMIN should be specified as (power of 2) + 1. The default value is 9. See Notes.
NMAX...	Input. Upper limit of terms of Chebyshev series (≥NMIN). The value of NMAX should be specified as (power of 2) + 1. The default value is 257. See Notes.
C.............	Output. Coefficients $\{c_k\}$. Each coefficient is stored as shows below: C(1) = c_0, C(2) = c_1, ..., C(N) = c_{n-1} One-dimensional array of size NMAX.
N.............	Output. Number of terms n of the Chebyshev series (≥5) The value of N takes as (power of 2) + 1.
ERR........	Output. Estimate of the absolute error of the Chebyshev series.
TAB........	Output. The trigonometric function table used for series expansion is stored. One-dimensional array whose size is greater than 3 and equal to the following: •If $a \ne 0$ ..... (NMAX-3)/2. •If $a = 0$ ..... NMAX-2. See Notes.
ICON......	Output. Condition code. See Table FCHEB-1.

Table FCHEB-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The required accuracy was not satisfied due to rounding-off errors. The required accuracy was far too high.	C contains resultant coefficients. The accuracy of the series is the maximum attainable.
20000	The required accuracy was not satisfied through the number of terms of the series has reached the upper limit.	Bypassed. C contains coefficients obtained at that time. ERR contains an estimate of absolute error obtained at that time.
30000	One of the following conditions occurred: 1 A = B 2 EPSA < 0.0 3 EPSR < 0.0 4 NMIN < 0 5 NMAX < NMIN	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH, UTABT and UCOSM
  FORTRAN basic functions ... ABS, AMAX1, AMIN1 and FLOAT

- Notes
  – The function subprogram specified by the FUN parameter must be defined as a subprogram having independent variable x only as the argument. The program calling this subroutine must include an EXTERNAL statement specifying the corresponding function name. If a given function contains auxiliary variable, they must be declared by a COMMON statement, which is used to establish an interface with the calling program.

  – Use of the trigonometric function talbe
  The trigonometric function table is produced only once even when this subroutine is called repeatedly. If it does not contain the

trigonometric function values necessary for series expansion, only the necessary values are calculated and added. Therefore, this subroutine should be called without making a change in the trigonometric function table – without changing the contents of TAB.

– This subroutine normally changes the interval of variable from [*a*, *b*] to [-1, 1] and expands function *f*(*x*) base on the chebyshev polynomials. When the end point *a* of the interval is zero, this subroutine expands function *f*(*x*) base on the shifted Chebyshev polynomials to avoid making an error in calculation (loss of significant digits) while making a change of the variable.
  However, the coefficients $\{c_k\}$ used the shifted Chebyshev polynomials are the same as those used the Chebyshev polynomials.
  On such reason, the size of trigonometric function table used internally is (NMAX-3)/2 in case of using the Chebyshev polynomials, on the other hand, (NMAX-2) in case of using the shifted Chebyshev polynomials.

– If the value of NMIN or NMAX is not equal to power of 2 plus one, the following value is assumed: (maximum power of 2 not exceeding that value) + 1 However, NMAX = 5 is assumed if NMAX is less than 5.

– The degree of error decrement depends on the smoothness of *f*(*x*) and the width of interval [*a*, *b*] as the number of terms *n* increases. When *f*(*x*) is an analytical function, the error decreases depending on the exponential function order O (r^n), $0 < r < 1$. When *f*(*x*) has up to *k*-th continuous derivatives, the error decreases depending on the the rational function order O ($\left|\frac{a-b}{n}\right|^k$). When $k = 0$ or $k = 1$, the error cannot be estimated accurately because the number of terms to be expanded in a series increases. Therefore the function to be processed by this subroutine should have, at least, up to 2nd continuous derivatives.

– Accuracy of the series
  This subroutine tries to obtain a Chebyshev series which satisfies (1.1) when ε_a and ε_r are given, $\varepsilon_r = 0$ means that *f*(*x*) is expanded in a Chebyshev series with its absolute error within ε_a. Similarly $\varepsilon_a = 0$ means that *f*(*x*) is expanded in a Chebyshev series with its relative error within ε_r. This purpose is sometimes obstructed by unexpected characteristics of *f*(*x*) or an unexpected value of ε_a or ε_r. For example, when ε_a or ε_r is extremely small in comparison with computational error in function evaluations, the effect of rounding-off errors becomes grater, so it is no use to continue the computation even though the number of terms subject to series expansion has not reached the upper limit. In this case, the processing is bypassed after a condition code of 10000 is set to ICON. At this time, the accuracy of the Chebyshev series becomes the attainable limit for the computer used. The Chebyshev series sometimes does not converge within NMAX evaluations. In this case, the coefficient value is an approximation obtained so far and is not always accurate. The processing is bypassed after a condition code of 20000 is set to ICON.
  To determine the accuracy of Chebyshev series, this subroutine always sets an estimate of absolute error in ERR.

– When inverse transform is required for Chebyshev series expansion, the subroutine FCOST should be utilized for cosine transform. The inverse transform mean to obtain the function value.

$$f(x_j) = \sum_{k=0}^{n-1}{}'' c_k T_k\left(\frac{2x_j(b+a)}{b-a}\right) \quad , j = 0,1,\ldots,n-1$$

  for a sample point x_j, which lies in the interval [*a*, *b*] indicated by (1.2). Where symbol Σ" denotes the initial and last terms are multiplied by factor 1/2. Therefore only the coefficient c_{n-1} of the last term must be doubled before the FCOST subroutine is called.
  During inverse transform, the contents of TAB must be kept intact because the trigonometric function table produced by this subroutine is used by the subroutine FCOST. See Example 2.

• Examples
  Example 1 expands exponential function

$$f(x) = e^x\left(= \sum_{n=0}^{\infty}\frac{x^n}{n!}\right)$$

  on the interval [-2, 2] in Chebyshev polynomials $\{T_k(x/2)\}$ according to the required accuracy of $\varepsilon_a = 0$ and $\varepsilon_r = 5\cdot10^{-5}$.
  NMIN = 9 and NMAX = 257 is assumed.

```
C     EXAMPLE 1
      DIMENSION C (257), TAB (127)
      EXTERNAL FUN
      EPSA=0.0
      EPSR=5.0E-5
      NMIN=9
      NMAX=257
      A=-2.0
      B=2.0
      CALL FCHEB(A,B,FUN,EPSA,EPSR,
     *           NMIN,NMAX,C,N,ERR,TAB,ICON)
```

```
      IF(ICON.GT.10000) GOTO 10
      WRITE(6,600) N,ERR,ICON
      WRITE(6,601) (C(I),I=1,N)
      STOP
   10 WRITE(6, 602) ICON
      STOP
  600 FORMAT('0',5X,'EXPANSION OF',
     *' FUNCTION FUN(X)',3X,'N=',I4,
     *5X,'ERR=',E15.5,5X,'ICON=',I5)
  601 FORMAT(/(5E15.5))
  602 FORMAT('0',5X,'CONDITION CODE',I8)
      END
      FUNCTION FUN(X)
      REAL*8 SUM,XP,TERM
      EPS=AMACH(EPS)
      SUM=1.0
      XP=X
      XN=1.0
      N=1
   10 TERM=XP/XN
      SUM=SUM+TERM
      IF(DABS(TERM).LE.
     1   DABS(SUM)*EPS) GOTO 20
      N=N+1
      XP=XP*X
      XN=XN*FLOAT(N)
      GOTO 10
   20 FUN=SUM
      RETURN
      END
```

Example 2 Inverse transform for Chebyshev series expansion
Example 2 expands since function

$$f(x) = \sin x \left(= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \right)$$

on the interval [0, π] in Chebyshev polynomials $\{T_k(x/2)\}$ according to the required accuracy of $\varepsilon_a = 5\cdot10^{-5}$ and $\varepsilon_r = 0$.
NMIN = 9 and NMAX = 513 is assumed.
Example 2 then checks the values taken by the Chebyshev series at n sample points.

$$x_j = \pi/2 + \pi/2 \cos\frac{\pi}{n-1} j \quad , j = 0,1,...,n-1$$

through use of the subroutine FCOST.

```
C     EXAMPLE 2
      DIMENSION C(513),TAB(511)
      EXTERNAL FUN
      EPSA=5.0E-5
      EPSR=0.0
      NMIN=9
      NMAX=513
      A=0.0
      B=ATAN(1.0)*4.0
      CALL FCHEB(A,B,FUN,EPSA,EPSR,
     *           NMIN,NMAX,C,N,ERR,TAB,ICON)
      IF(ICON.GT.10000) GOTO 10
      WRITE(6,600) N,ERR,ICON
      C(N)=C(N)*2.0
      NM1=N-1
      WRITE(6,601) (C(I),I=1,N)
      CALL FCOST(C,NM1,TAB,ICON)
      WRITE(6,602)
      WRITE(6,601) (C(I),I=1,N)
      STOP
   10 WRITE(6,603) ICON
      STOP
  600 FORMAT('0',5X,'EXPANSION OF',
     *' FUNCTION FUN(X)',3X,'N=',I4,
     *5X,'ERR=',E15.5,5X,'ICON=',I5)
  601 FORMAT(/(5E15.5))
  602 FORMAT('0',5X,'INVERSE TRANS',
     *'FORM BY ''FCOST''')
  603 FORMAT('0',5X,'CONDITION CODE',I8)
      END
      FUNCTION FUN(X)
      REAL*8 SUM,XP,TERM
      EPS=AMACH(EPS)
      SUM=X
      P=X*X
      XP=X*P
      XN=-6.0
      N=3
   10 TERM=XP/XN
      SUM=SUM+TERM
      IF(DABS(TERM).LE.EPS) GOTO 20
      N=N+2
      XP=XP*P
      XN=-XN*FLOAT(N)*FLOAT(N-1)
      GOTO 10
   20 FUN=SUM
      RETURN
      END
```

Method
This subroutine uses an extended Clenshaw-Curtis method for Chebyshev series expansion enhanced the speed by using fast cosine transform.

- Chebyshev series expansion method
  For simplicity, the domain of $f(x)$ subject to Chebyshev series expansion is assumed to be [-1, 1].
  The Chebyshev series expansion of $f(x)$ can be expressed as:

$$f(x) = \sum_{k=0}^{\infty}{}' c_k T_k(x) \qquad (4.1)$$

$$c_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} dx \qquad (4.2)$$

Upon choosing $x = \cos\theta$, (4.2) consequently gives

$$c_k = \frac{2}{\pi} \int_0^{\pi} f(\cos\theta) \cos k\theta \, d\theta \qquad (4.3)$$

Therefore, c_k is regarded as coefficients for the cosine series of the continuous even function $f(\cos\theta)$ with period 2π.
To obtain coefficients (4.2) for Chebyshev series expansion through use of the Gauss-Chebyshev numerical integration formula means to determine fourier coefficients (4.3) for even functions based on the midpoint rule. The Gaussian integral formula is the best

in the sense that it can be utilized to integrate correctly a polynomial of degree up to 2*n*-3 based on *n*-1 sample points. The trapezoidal rule for (4.3) can be utilized to integrate correctly a polynomial of degree up to 2*n*-3 based on *n* sample points. That is, the trapezoidal rule is almost the best.

Since input to this subroutine is a function, this subroutine obtains a Chebyshev series consisting of terms whose number is specified by doubling the number of sample points according to the required accuracy. Therefore this subroutine uses the best trapezoidal rule for this purpose. The resultant *n*-terms Chebyshev series satisfies interpolatory condition

$$f(x_j) = \sum_{k=0}^{n-1}{}'' c_k T_k(x_j) \qquad (4.4)$$

at *n* sample points shown below:

$$x_j = \cos\frac{\pi}{n-1} j,\ j = 0,1,...,n-1$$

In order to keep unity of the Chebyshev series notations, the last coefficient of (4.4) is multiplied by factor 1/2, and express the series as (4.1).

Based on the error evaluation of series expansion described below, this subroutine determines the number of terms *n* which satisfies

$$\left| f(x) - \sum_{k=0}^{n-1}{}' c_k T_k(x) \right| < \max\{\varepsilon_a, \varepsilon_r \|f\|\} \qquad (4.5)$$

on the interval [-1, 1] as well as series coefficients $\{c_k\}$ by using the Fast Fourier Transform (FFT) method.

Coefficients $\{ c_k^p \}$ for terms *n* ($=n_p+1$, $n_p=2^p$) are determined by the trapezoidal rule for as shown in (4.6):

$$c_k^p = \sum_{j=0}^{n_p}{}'' f\left(\cos\frac{\pi}{n_p} j\right)\cos\frac{\pi}{n_p} kj,\ k = 0,1,...,n_p \qquad (4.6)$$

However the ordinary scaling factor $2/n_p$ is omitted.

Coefficients $\{ c_k^{p+1} \}$ for terms $n = n_{p+1}+1$ whose number is doubled can efficiently be obtained by making a good use of complementary relation between the midpoint rule and the trapezoidal rule. That is, function *f*(*x*) be sampled at each midpoint of sample points at which $\{ c_k^p \}$ are determined as shown in (4.7):

$$f\left(\cos\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right),\ j = 0,1,...,n_p-1 \qquad (4.7)$$

The discreate cosine transform (using the midpoint rule) can be defined as shown in (4.8):

$$\tilde{c}_k^p = \sum_{j=0}^{n_p-1} f\left(\cos\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right)\cos\frac{\pi}{n_p}k\left(j+\frac{1}{2}\right) \qquad (4.8)$$
$$,k = 0,1,....,n_p-1$$

Then coefficients $\{ c_k^{p+1} \}$ can be determined by the recurrence formula as show in (4.9):

$$\left.\begin{array}{ll} c_k^{p+1} & = c_k^p + \tilde{c}_k^p \\ c_{n_{p+1}-k}^{p+1} & = c_k^p - \tilde{c}_k^p \\ c_{n_p}^{p+1} & = c_{n_p}^p \end{array}\right\} ,k = 0,1,...,n_p-1 \qquad (4.9)$$

As described above the coefficients $\{ c_k^p \}$ of discrete Chebyshev series.

$$f(x) \approx \sum_{k=0}^{n_p}{}'' c_k^p T_k(x) \qquad (4.10)$$

by which *f*(*x*) is expanded can be determined by increasing a value of *p* gradually based on (4.6), (4.8) and (4.9). When the convergence criterion is satisfied each coefficient $\{ c_k^p \}$ is normalized by multipling by factor $2/n_p$.

- Error evaluation for Chebyshev series

  The following relationship exists between the coefficients $\{c_k\}$ associated with Chebyshev series expansion shown in (4.1) and the coefficients $\{ c_k^p \}$ associated with discreate Chebyshev series expansion at the *p*-th stage shown in (4.10):

$$c_k^p = c_k + \sum_{m=1}^{\infty}\left(c_{2mn_p-k} + c_{2mn_p+k}\right) \qquad (4.11)$$
$$,k = 0,1,...,n_{p-1}$$

From (4.11), the error evalution formula for the *p*-stage Chebyshev series is introduced as

$$\left| f(x) - \sum_{k=0}^{n_p}{}'' c_k^p T_k(x) \right| \le 2\sum_{k=n_p+1}^{\infty}|c_k| \qquad (4.12)$$

If *f*(*x*) is analytical function on the interval, its series coefficients $\{c_k\}$ decreases according to exponential function order O (r^k), 0 < *r* < 1 as subscript *k* increases. As a result *r* can be estimated based on the *p*-th stage discrete Chebyshev series coefficients c_k^p.

Let c_k^p to be Ar^k (*A*: Constant). Since *k* is at most n_p, $r^{n_p/4}$ can be obtained based on the ratio of the coefficient corresponding to the last n_p to that corresponding to $3/4n_p$. This subroutine estimates *r* by using not only the c_{n_p} -th and $3/4n_p$-th but also the (n_p -1)-th and ($3/4n_p$ -1)-th coefficients in order to avoid a state that c_{n_p} or $c_{3/4n_p}$ becomes zero by accident as shown below:

$$r = \min\left\{\left(\frac{\left|c^p_{n_p-1}\right| + \frac{1}{2}\left|c^p_{n_p}\right|}{\left|c^p_{\frac{3}{4}n_p-1}\right| + \left|c^p_{\frac{3}{4}n_p}\right|}\right)^{4/n_p}, 0.99\right\}$$

The upper limit of r is defined as 0.99 because the convergence rate of the series become slow and Chebyshev series expansion of $f(x)$ is virtually impossible when r is greater than 0.99.

By using the resultant r, the error e_p at the p-th stage can be estimated by (4.12) and (4.13).

$$e_p = \frac{r}{1-r}\left(\left|c^p_{n_p-1}\right| + \frac{1}{2}\left|c^p_{n_p}\right|\right) \qquad (4.13)$$

The last coefficient $c^p_{n_p}$ only is multipled by factor 1/2 because (4.13) is a discreate Chebyshev series using a trapezoidal rule.

- Computational process

Step 1: Initialization

1) Initialization of the trigonometric function table
Three cosine function values are determined in reverse binary order corresponding to equispaced three sample points in the interval [0, $\pi/2$]. Each entry of the trigonometric function table can be used as a sample point during sampling of $f(x)$.
2) Initial Chebyshev series expansion
This subroutine determines c^2_0, c^2_1, c^2_2, c^2_3, and c^2_4, as a result of 5-terms Chebyshev series expansion (upon choosing n_p=4 in (4.6)). It also determines the norm $\|f\|$ of $f(x)$ based on (1.3).

Step 2: Convergence criterion
If n_p+1 ≤ NMIN is satisfied this subroutine bypasses a convergence test and unconditionally executes step 3.
If n_p+1 > NMIN is satisfied this subroutine performs a convergence test as described below:

– This subroutine estimates a computational error limit ρ

$$\rho = \sqrt{n_p}\,\|f\|(2u) \qquad (4.14)$$

where u is the unit round off, and a tolerance ε for convergence test as

$$\varepsilon = \max\left\{\varepsilon_a, \varepsilon_r \|f\|\right\} \qquad (4.15)$$

– If the coefficients $c^p_{n_{p-1}}$ and $c^p_{n_p}$ of the last two terms at the p-th stage have been lost significant digits, that is, if the coefficients satisfy (4.16).

$$\left|c^p_{n_p-1}\right| + \frac{1}{2}\left|c^p_{n_p}\right| < \rho \qquad (4.16)$$

this subroutine assumes that the series has

This is because the accuracy of computation can not increase any longer.
If $\rho < \varepsilon$ is satisfied this subroutine terminates normally after a condition code of 0 is set to ICON.
If $\rho \geq \varepsilon$ is satisfied this subroutine terminates abnormally after a condition code of 10000 is set to ICON, assuming that the required accuracy of ε_a or ε_r is too small.

– If coefficients of the last two terms consist of significant digits, this subroutine estimates absolute error e_p based on (4.13).
When $e_p < \varepsilon$ is satisfied this subroutine terminates normally after a condition code of 0 is set to ICON.
When $e_p < \varepsilon$ is not satisfied but $2n_p$+1 ≤ NMAX is satisfied, this subroutine unconditionally executes Step 3.
When $e_p < \varepsilon$ is not satisfied but $2n_p$+1 > NMAX is satisfied, this subroutine terminates abnormally after a condition code of 20000 is set to ICON, assuming that the required accuracy is not satisfied when the number of terms to be expanded in a Chebyshev series reaches the upper limit.

$$e_p < \varepsilon \qquad (4.17)$$

Note that resultant coefficients are normalized whether this subroutine terminates normally or abnormally.

Step 3: Sampling of $f(x)$ and calculating the norm of $f(x)$
At each midpoint of n_p+1 sample points which have previously determined, this subroutine samples $f(x)$ in reverse binary order as shown in (4.18):

$$f\left(\frac{a+b}{2} + \frac{b-a}{2}\cos\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right) \qquad (4.18)$$
$$, j = 0,1,\ldots,n_p-1$$

However, in case of a = 0, this subroutine samples $f(x)$ to avoid loss of significant digits while changing its variables as shown in (4.19):

$$f\left(b\cos^2\frac{\pi}{2n_p}\left(j+\frac{1}{2}\right)\right) \qquad (4.19)$$
$$, j = 0,1,\ldots,n_p-1$$

At this time, this subroutine adds a new trigonometric function table entry and determines the norm of $f(x)$ by using (1.3).

Step 4: Discrete cosine transform (using the midpoint rule)
This subroutine performs discrete cosine transform using the Fast Fourier Transform (FFT) method to determine $\{ \tilde{c}_k^p \}$ for sample points obtained by Step 3.

Step 5: Computing $\{ c_k^{p+1} \}$

By using $\{ c_k^p \}$ obtained previously and $\{ \tilde{c}_k^p \}$ obtained by Step 4, this subroutine determines of $2n_p+1$ terms based on the recurrence formula shown in (4.9).
Then, this subroutine executes Step 2 after it increases a value of p by one.

Step 3 and 4 consumes most of the time required to execute this subroutine. The number of multiplications required to determine coefficients for an n-terms Chebyshev series is approximately $n\log_2 n$ except for that required for sampling of $f(x)$.

For further information, see Reference [59]. For detailed information about discrete cosine transform, see an explanation of the subroutines FCOST and FCOSM.

E51-10-0101 FCOSF, DFCOSF

Cosine series expansion of an even function (Fast cosine transform)
CALL FCOSF (TH, FUN, EPSA, EPSR, NMIN, NMAX, A, N, ERR, TAB, ICON)

Function

This subroutine performs cosine series expansion of a smooth even function $f(t)$ with period $2T$ according to the required accuracy of ε_a and ε_r. It determines n coefficients $\{a_k\}$ which satisfy

$$\left| f(t) - \sum_{k=0}^{n-1}{}' a_k \cos\frac{\pi}{T}kt \right| \le \max\{\varepsilon_a, \varepsilon_r \|f\|\} \tag{1.1}$$

where symbol Σ’ denotes to make sum but the initial term only is multiplied by factor 1/2. The norm $\|f\|$ of $f(t)$ is defined as shown in (1.3) by using function values taken at sample points shown in (1.2) within the half period $[0, T]$.

$$t_j = \frac{T}{n-1}j \quad , j = 0,1,...,n-1 \tag{1.2}$$

$$\|f\| \le \max_{0 \le j \le n-1} |f(t_j)| \tag{1.3}$$

Where $T > 0$, $\varepsilon_a \ge 0$, $\varepsilon_r \ge 0$.

Parameters

TH.....	Input. Half period T of the function $f(t)$.
FUN....	Input. Name of the function subprogram with calculates $f(t)$ to be expanded in a cosine series. See an example of using this subroutine.
EPSA...	Input. The absolute error tolerance ε_a.
EPSR...	Input. The relative error tolerance ε_r.
NMIN...	Input. Lower limit of terms of cosine series (≥0). NMIN should be taken a value such as (power of 2) + 1. The default value is 9. See Notes.
NMAX...	Input. Upper limit of terms of cosine series. (NMAX ≥ NMIN). NMAX should be taken a value such as (power of 2) + 1. The default value is 257. See Notes.
A.....	Output. Coefficient $\{a_k\}$. One-dimensional array of size NMAX. Each coefficient is stored as shown below: A(1) = a_0, A(2) = a_1, ..., A(N) = a_{n-1}
N.....	Output. Number of terms n of the cosine series (≥5). N takes a value such as (power of 2) + 1.
ERR....	Output. Estimate of the absolute error of the series.
TAB....	Output. TAB contains a trigonometric function table used for series expansion. One-dimensional array whose size is greater than 3 and equal to (NMAX-3)/2.
ICON...	Output. Condition code. See Table FCOSF-1.

Table FCOSF-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The required accuracy was not satisfied due to rounding-off errors. The required accuracy is too high.	A contains resultant coefficients. The accuracy of the series is the maximum attainable.
20000	The required accuracy was not satisfied though the number of terms of the series has reached the upper limit.	Bypassed. A contains resultant coefficients and ERR contains an estimate of absolute error.
30000	One of the following cases occurred: 1 TH ≤ 0 2 EPSA < 0.0 3 EPSR < 0.0 4 NMIN < 0 5 NMAX < NMIN	Bypassed

Comments on use

- Subroutines used
  SSL II ... MGSSL, AMACH, UTABT, UCOSM and UNIFC
  FORTRAN basic functions ... ABS, AMAX1, AMIN1 and FLOAT

- Notes
  - The function subprogram specified by the FUN parameter must be a subprogram defined at the interval [0, T] having independent variable t only as the argument.
    The name must be declared by the EXTERNAL statement in the program which calls this subroutine. If the function contains auxiliary variable, they must be declared by a COMMON statement to establish an interface with the calling program. See Example of using this subroutine.

  - Use of the trigonometric function table
    When this subroutine is repeatedly called, the trigonometric function table is produced only once. A new trigonometric function table entry is made on an as-required basis. Therefore the contents of TAB must be kept intact when this subroutine is called subsequently.

  - If NMIN or NMAX does not take a value such as (power of 2) + 1, this subroutine assumes the maximum value of (power of 2) + 1 which does not exceed that value. Howeger NMAX = 5 is assumed if NMAN < 5 is satisfied.

– The degree of error decrement greatly depends on the smoothness of $f(t)$ in the open interval $(-\infty,\infty)$ as the number of terms n increases. If $f(t)$ is an analytical periodic function, the error decreases according to exponential function order O (r^n), $0 < r < 1$. If it has up to k-th continuous derivatives, the error decreases according to rational function order O (n^{-k}).
When k=0 or k=1, an estimate of absolute error is not always accurate because the number of terms to be expanded increases greatly.
Therefore, the function used by this subroutine should have, at least, up to 2nd continuous derivatives.

– Accuracy of the series
This subroutine determines a cosine series which satisfies (1.1) according to the required accuracy of ε_a and ε_r. If $\varepsilon_r = 0$ is specified, this subroutine expands $f(t)$ in a cosine series within the required accuracy of absolute error ε_a.
Similarly $\varepsilon_a = 0$ is specified, this subroutine expands $f(t)$ in a cosine series within the required accuracy of relative error ε_r. However cosine series expansion is not always successful depending on the specification of ε_a and ε_r. For example, when ε_a or ε_r is too small in comparison with computational error of $f(t)$, the effect of rounding-off errors becomes greater on the computational result even if the number of terms to be expanded does not reach the upper limit.
In such a case, this subroutine abnormally terminates after a condition code of 10000 is set to ICON. At this time, the accuracy of the cosine series becomes the attainable limit for the computer used.
The number of terms to be expanded in a cosine series sometimes does not converge within NMAX evaluations depending on the characteristics of $f(t)$.
In such a case, this subroutine abnormally terminates after a condition code of 20000 is set to ICON. Each coefficient is an approximation obtained so far, and is not accurate.
To determine the accuracy of cosine series this subroutine always set an estimate of absolute error in ERR.

– When inverse transform is attempted by the subroutine FCOST, the coefficient a_{n-1} of the last term must be doubled in advance.
Note that the content of TAB must be kept intact whether normal or inverse transform is attempted. See Example 2.

– When $f(t)$ is only a periodical function, this subroutine can be used to perform cosine series expansion for even function as $(f(t) + f(-t))/2$.

– If $f(t)$ has no period and is absolutely integrable, its theoretical cosine transform can be defined as shown in (3.1):

$$F(\omega)=\int_0^\infty f(t)\cos\omega t dt \tag{3.1}$$

If $f(t)$ is dampted according to order of O (e^{-at}) ($a < 0$), an approximation of the Fourier integral can be obtained as described below:
Assume that $|f(t)|$ can be ignored on the interval $[T,\infty)$ when T is sufficiently large. By defining T which satisfies (3.2)

$$|f(t)| < u\ , t \geq T \tag{3.2}$$

where u is the unit round off.
This subroutine can be used to determine cosine series coefficients $\{a_k\}$ for $f(t)$, assuming that $f(t)$ is a function with period $2T$.
Since $\{a_k\}$ can be expressed as

$$a_k = \frac{2}{T}\int_0^T f(t)\cos\frac{\pi}{T}ktdt \tag{3.3}$$

(3.4) can be established based on (3.1) and (3.2).

$$F\left(\frac{\pi}{T}k\right) \approx \frac{T}{2}a_k \quad , k = 0,1,...,n-1 \tag{3.4}$$

Based on this relationship this subroutine can calculate an approximation of cosine transform shown in (3.1) by using discrete cosine transform.
When inverse transform

$$f(t)=\frac{2}{\pi}\int_0^\infty F(\omega)\cos\omega t d\omega \tag{3.5}$$

is to be calculated, the subroutine FCOST can be called for n pieces of data as follows:

$$\frac{2}{T}F\left(\frac{\pi}{T}k\right) \quad , k = 0,1,...,n-2$$
$$\frac{4}{T}F\left(\frac{\pi}{T}(n-1)\right)$$

The subroutine FCOST can obtain an approximation of:

$$f\left(\frac{T}{n-1}j\right) \quad , j = 0,1,...,n-1$$

See Example 2.

- Examples

Example 1

This example expands the following even function with period 2π having auxiliary variable *p*

$$f(t) = \frac{1-p^2}{1-2p\cos t + p^2}$$

in a cosine series according to the required accuracy of $\varepsilon_a = 5\cdot10^{-5}$ and $\varepsilon_r = 5\cdot10^{-5}$.
Where NMIN = 9 and NMAX = 257 are assumed.
The theoretical cosine series expansion of $f(t)$ is as follows:

$$f(t) = 1 + 2\sum_{k=1}^{\infty} p^k \cos kt$$

This example prints cosine series coefficients when p = 1/4, 1/2 and 3/4.

```
C      **EXAMPLE**
       DIMENSION A(257),TAB(127)
       EXTERNAL FUN
       COMMON P
       TH=ATAN(1.0)*4.0
       EPSA=0.5E-4
       EPSR=EPSA
       NMIN=9
       NMAX=257
       P=0.25
     1 CALL FCOSF(TH,FUN,EPSA,EPSR,
      *NMIN,NMAX,A,N,ERR,TAB,ICON)
       IF(ICON.GT.10000) GO TO 10
       WRITE(6,600) N,ERR,ICON,P
       WRITE(6,601) (A(I),I=1,N)
       P=P+0.25
       IF(P.LT.1.0) GO TO 1
       STOP
    10 WRITE(6,602) ICON
       STOP
   600 FORMAT('0',5X,'EXPANSION OF',
      *' FUNCTION FUN(T)',3X,'N=',I4,
      *5X,'ERR=',E15.5,5X,'ICON=',I16,5X,
      *'P=',E15.5)
   601 FORMAT(/(5E15.5))
   602 FORMAT('0',5X,'CONDITION CODE',I8)
       END
       FUNCTION FUN(T)
       COMMON P
       FUN=(1.0-P*P)/(1.0-2.0*P*COS(T)+P*P)
       RETURN
       END
```

Example 2

Cosine transform and inverse transform

This example transforms even function

$$F(\omega) = \int_0^{\infty} \operatorname{sech}\frac{\pi}{2}x \cos\omega x \, dx$$

in a cosine series according to the required accuracy of $\varepsilon_a = 5\cdot10^{-5}$ and $\varepsilon_r = 5\cdot10^{-5}$ and compares the results with analytical solution $F(\omega) = \operatorname{sech}\omega$.

Then, this example performs inverse transform of the function by using the subroutine FCOST and check the accuracy of the results.

```
C      **EXAMPLE**
       DIMENSION A(257),TAB(127),
      *          ARG(257),T(257)
       EXTERNAL FUN
       COMMON HP
       HP=ATAN(1.0)*2.0
       TH=ALOG(1.0/AMACH(TH))/HP
       EPSA=0.5E-04
       EPSR=EPSA
       NMIN=9
       NMAX=257
C      COSINE TRANSFORM
       CALL FCOSF(TH,FUN,EPSA,EPSR,
      *NMIN,NMAX,A,N,ERR,TAB,ICON)
       IF(ICON.GT.10000) GO TO 10
       TQ=TH*0.5
       H=(HP+HP)/TH
       DO 1 K=1,N
       ARG(K)=H*FLOAT(K-1)
       A(K)=A(K)*TQ
       T(K)=TRFN(ARG(K))
     1 CONTINUE
       WRITE(6,600) N,ERR,ICON
       WRITE(6,610)
       WRITE(6,601) (ARG(K),A(K),T(K),K=1,N)
C      INVERSE TRANSFORM
       Q=1.0/TQ
       DO 2 K=1,N
       A(K)=A(K)*Q
     2 CONTINUE
       A(N)=A(N)*2.0
       NM1=N-1
       CALL FCOST(A,NM1,TAB,ICON)
       IF(ICON.NE.0) GO TO 10
       H=TH/FLOAT(NM1)
       DO 3 K=1,N
       ARG(K)=H*FLOAT(K-1)
       T(K)=FUN(ARG(K))
     3 CONTINUE
       WRITE(6,620)
       WRITE(6,610)
       WRITE(6,601) (ARG(K),A(K),T(K),K=1,N)
       STOP
    10 WRITE(6,602) ICON
       STOP
   600 FORMAT('0',5X,'CHECK THE COSINE',
      *' TRANSFORM OF FUNCTION FUN(T)',
      *3X,'N=',I4,5X,'ERR=',E15.5,5X,
      *'ICON=',I5)
   610 FORMAT('0',6X,'ARGUMENT',7X,
      *'COMPUTED',11X,'TURE')
   620 FORMAT('0',5X,'CHECK THE INVERSE'
      *,' TRANSFORM')
   601 FORMAT(/(3E15.5))
   602 FORMAT('0',5X,'CONDITION CODE',I6)
       END
       FUNCTION FUN(T)
       COMMON HP
       FUN=1.0/COSH(T*HP)
       RETURN
       END
       FUNCTION TRFN(W)
       TRFN=1.0/COSH(W)
       RETURN
       END
```

Method

This subroutine applies discrete fast cosine transform (based on the trapezoidal rule) to cosine transform for entry of functions.

- Cosine series expansion

  For simplicity, an even function $f(t)$ with a period of 2π. The function can be expanded in a cosine series as shown below:

$$f(t)=\frac{1}{2}a_0+\sum_{k=1}^{\infty}a_k\cos kt \tag{4.1}$$

$$a_k=\frac{2}{\pi}\int_0^{\pi}f(x)\cos ktdt \tag{4.2}$$

  This subroutine uses the trapezoidal rule to compute (4.2) by dividing the closed interval $[0, \pi]$ equally. By using resultant coefficients $\{a_k\}$, this subroutine approximates (4.1) by finite number of terms.

  If this integrand is smooth, the number of terms is doubled as far as the required accuracy of ε_a and ε_r is satisfied. If sampling is sufficient, (4.3) will be satisfied.

$$\left|f(t)-\sum_{k=0}^{n-1}{}''a_k\cos kt\right|<\max\{\varepsilon_a,\varepsilon_r\|f\|\} \tag{4.3}$$

  where n indicates the number of samples (power of 2+1).

  Where symbol Σ'' denotes to make sum but the initial and last terms only are multipled by factor 1/2.

  The resultant trigonometric polynomial is a trigonometric interpolation polynominal in which each breakpoint used by the trapezpidal rule is an interpolation point as shown below:

$$f\left(\frac{\pi}{n-1}j\right)=\sum_{k=0}^{n-1}{}''a_k\cos\frac{\pi}{n-1}kj \tag{4.4}$$
$$,j=0,1,...,n-1$$

  The cosine series expansion is explained in detail below.

  Assume that coefficients obtained by the trapezoidal rule using n sample points ($n = n_p+1$, $n_p = 2^p$) are

$$a_k^p=\sum_{j=0}^{n_p}{}''f\left(\frac{\pi}{n_p}j\right)\cos\frac{\pi}{n_p}kj \quad ,k=0,1,...,n_p \tag{4.5}$$

  Where the ordinary scaling factor $2/n_p$ is omitted from (4.5).

  When the number of terms is doubled as $n_{p+1} = 2n_p$ each coefficient can efficiently be determined by making a good use of complementary relation between the trapezoidal rule and the midpoint rule.

At each midpoint between sample points used by the trapezoidal rule (4.5), $f(t)$ can be sampled as shown below:

$$f\left(\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right) \quad ,j=0,1,...,n_p-1 \tag{4.6}$$

Discrete cosine transform (using the midpoint rule) for (4.6) can be defined as shown below:

$$\tilde{a}_k^p=\sum_{j=0}^{n_{p-1}}f\left(\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right)\cos\frac{\pi}{n_p}k\left(j+\frac{1}{2}\right) \tag{4.7}$$
$$,k=0,1,...,n_p-1$$

Since (4.8) is satisfied at this stage { a_k^{p+1} } can be determined.

$$\left.\begin{aligned}a_k^{p+1}&=a_k^p+\tilde{a}_k^p\\ a_{n_{p+1}-k}^{p+1}&=a_k^p-\tilde{a}_k^p\\ a_{n_p}^{p+1}&=a_{n_p}^p\end{aligned}\right\},k=0,1,...,n_p-1 \tag{4.8}$$

By using this recurrence formula for { a_k^p }, $f(t)$ can be expanded in a cosine series of higher degree while the number of terms is doubled as far as the required accuracy is satisfied. Then { a_k^p } is normalized by multipling by factor $2/n_p$. Note that the coefficient of the last term is multipled by factor $1/n_p$ so that all cosine series can be expressed in the same manner as shown in (4.1).

- Error evaluation for cosine series

  The following relationship exists between the theoretical cosine coefficients a_k of $f(t)$ and discrete cosine coefficient { a_k^p } taken at the p-th stage:

$$a_k^p=a_k+\sum_{m=1}^{\infty}\left(a_{2mn_p-k}+a_{2mn_p+k}\right) \tag{4.9}$$
$$,k=0,1,...,n_p$$

This results from (4.2) and (4.5) as well as orthogonality of trigonometric functions. The error evaluation for a cosine series at the p-th stage

$$\left|f(t)-\sum_{k=0}^{n_p}{}''a_k^p\cos kt\right|\le 2\sum_{k=n_p+1}^{\infty}|a_k| \tag{4.10}$$

can be deduced from (4.9). If $f(t)$ is an analytical periodic function its series coefficients $\{a_k\}$ decrease according to exponential function order O (r^k) $(0 < r < 1)$ as k increases. The r can be estimated from a discreate cosine coefficient at the p-th stage. Let $a_k^p = Ar^k$ (A: constant). Since k is at most n_p, $r^{n_p/4}$ can be estimated from the ratio of the coefficient of the last term n_p to the coefficient of term $3/4n_p$. This subroutine does not allow the two coefficients to be zero by accident. Therefore it uses the (n_p-1)-th and ($3/4n_p$-1)-th coefficients together with those coefficients to estimate a value of r as shown below.

$$r = \min\left\{\left(\frac{\left|a_{n_p-1}^p\right| + \frac{1}{2}\left|a_{n_p}^p\right|}{\left|a_{\frac{3}{4}n_p-1}^p\right| + \left|a_{\frac{3}{4}n_p}^p\right|}\right)^{4/n_p}, 0.99\right\}$$

If r is greater than 0.99, this subroutine cannot actually expand *f(t)* in a cosine series because the convergence rate of the series becomes weak.

By using the resultant r, the p-th stage error

$$e_p = \frac{r}{1-r}\left(\left|a_{n_p-1}^p\right| + \frac{1}{2}\left|a_{n_p}^p\right|\right) \qquad (4.11)$$

can be estimated from (4.10).

The last coefficient $a_{n_p}^p$ only is multipled by factor 1/2 since this coefficient is a discrete cosine coefficient using the trapezoidal rule.

- Computational process

  Step 1: Initialization

  – Initialization of Trigonometric function table

  At three points which divides in interval $[0, \pi/2]$ equally, three values for the cosine function is obtained in reverse binary order. The trigonometric function table is not initialized if this subroutine is called in advance. The trigonometric function table is used for discrete cosine transform.

  – Initial cosine series expansion

  This subroutine performs 5-terms cosine series expansion whose n_p is 4(p=2) in (4.5) and calculates a_0^2, a_1^2, a_2^2, a_3^2, a_4^2. At this time it also obtains $\|f\|$ based on the norm definition shown in (1.3).

Step 2: Convergence criterion

If $n_p + 1 \le$ NMIN is satisfied this subroutine does not perform a convergence test and executes step 3.

If $n_p + 1 >$ NMIN is satisfied this subroutine performs a convergence test as described below:

This subroutine estimates computational error limit

$$\rho = \sqrt{n_p}\,\|f\|(2u) \qquad (4.12)$$

where u is the unit round off, and a tolerance for convergence test as

$$\varepsilon = \max\{\varepsilon_a, \varepsilon_r\|f\|\} \qquad (4.13)$$

If the last two terms at the p-th stage have been lost significant digits, that is, if the coefficients satisfy (4.14).

$$\left|a_{n_p-1}^p\right| + \frac{1}{2}\left|a_{n_p}^p\right| < \rho \qquad (4.14)$$

the computational accuracy cannot be increased even if this computation continues. Therefore this subroutine replaces the absolute error e_p of the cosine series by the computational error ρ, assuming that the cosine series is converged. If $\rho < \varepsilon$ is satisfied this subroutine sets a condition code of 0 to ICON. If $\rho \ge \varepsilon$ is satisfied this subroutine sets a condition code of 10000 to ICON assuming that ε_a or ε_r is relatively smaller than unit round off u

If (4.14) is not satisfied, this subroutine estimates error e_p based on (4.11).

If $e_p < \varepsilon$ is satisfied this subroutine sets a condition code of 0 to ICON and terminates normally. If $e_p < \varepsilon$ is not satisfied but $2n_p + 1 \le$ NMAX is satisfied, this subroutine immediately executes Step 3. Otherwise this subroutine sets a condition code of 20000 to ICON and terminates abnormally assuming that the required accuracy is not satisfied even when the number of terms to be expanded is reached the upper limit.

Note that each coefficient is normalized whether this subroutine terminates normally or abnormally.

Step 3: Calculation of sample points

Sample points to be used for sampling of *f(t)* at the p-th stage can be expressed as follows:

$$t_j = \frac{\pi}{n_p}\left(j + \frac{1}{2}\right) \quad , j = 0,1,\ldots,n_p - 1$$

They can be obtained in reverse binary order through use of the recurrence formula shown below:

$$\begin{aligned} & t_0 = \pi/2^{p+1} \\ & t_{(2j+1)2^{p-l-1}} = t_{j2^{p-l}} + 2^{-p+l}\pi \qquad (4.16) \\ & j = 0,1,\ldots,2^l - 1, l = 0,1,\ldots,p-1 \end{aligned}$$

where $n_p = 2^p$.

Step 4: Sampling of $f(t)$ and calculation of the norm

This subroutine obtains values of $f(t)$ for n sample points based on (4.16). and overwrites them on the sample points.

It also calculates norm $\|f\|$ based on the norm definition shown in (1.3).

Step 5: Trigonometric function table creation
This subroutine produces the trigonometric function table required by step 6. The trigonometric function table is not recalculated each time this subroutine is called.

Step 6: Discrete cosine transform (using the midpoint rule)
For sample points obtained by Step 4, this subroutine performs discrete cosine transform using the Fast Fourier Transform (FFT) method to determine $\{\tilde{a}_k^p\}$.

Step 7: Calculation of $\{a_k^{p+1}\}$
This subroutine combines $\{a_k^p\}$ obtained previously with by using (4.8) to obtain the coefficients $\{a_k^{p+1}\}$ of the discrete cosine series consisting of $2n_p + 1$ terms.
Then, this subroutine executes Step 2 after it increases a value of p by one.

Step 4 and 6 consume most of the time required to execute this subroutine.
Then number of multiplications required to determine the coefficients of a cosine series consisting of n terms is about $n\log_2 n$.

To save storage this subroutine overwrites sample points, samples and expansion coefficients onto a one-dimensional array A.

For further information, see Reference [59].
For detailed information about discrete cosine transfom, see an explanation of the subroutines FCOST and FCOSM.

F11-11-0201 FCOSM, DFCOSM

Discrete cosine transform (midpoint rule, radix 2 FFT)
CALL FCOSM (A, N, ISN, TAB, ICON)

Function

Given n same point $\{x_{j+1/2}\}$,

$$x_{j+1/2} = x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right) \quad , j = 0,1,...,n-1 \qquad (1.1)$$

by equally dividing the half period of the even function with period 2π, a discrete cosine transform or its inverse transform based on the midpoint rule is performed by using the Fast Fourier Transform (FFT).

Here $n = 2^l$ (l = 0 or positive integer).

- Cosine transform

  By inputting $\{x_{j+1/2}\}$ and performing the transform defined in (1.2), the Fourier coefficients $\{n/2 \cdot a_k\}$ are obtained

$$\frac{n}{2}a_k = \sum_{j=0}^{n=1} x_{j+1/2} \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right) \quad ,k = 0,1,...,n-1 \qquad (1.2)$$

- Cosine inverse transform

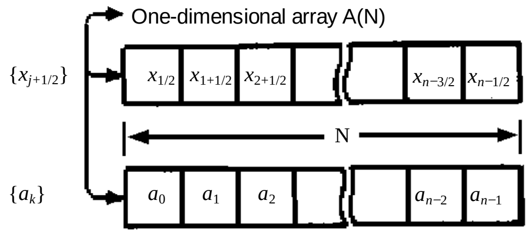
  By inputting $\{a_k\}$ and performing the transform defined in (1.3), the values of the Fourier series $\{x_{j+1/2}\}$ are obtained.

$$x_{j+1/2} = \sum_{k=0}^{n-1}{}' a_k \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right) \quad , j = 0,1,...,n-1 \qquad (1.3)$$

where Σ' denotes the first term only is taken with factor 1/2.

Parameters

A.....	Input. $\{x_{j+1/2}\}$ or $\{a_k\}$. Output. $\{n/2 \cdot a_k\}$ or $\{x_{j+1/2}\}$. One-dimensional array of size n. See Fig. FCOSM-1.
N.....	Input. Sample point number n.



Note: { $\frac{n}{2}a_k$ } is handled the same way as for $\{a_k\}$.

Fig. FCOSM-1 Data storing method

ISN...	Input. Transform or inverse transform is indicated. For transform: ISN = +1 For inverse transform: ISN = -1
TAB.....	Output. Trigonometric function table used in the transform is stored. One-dimensional array of size n-1. See “Notes”.
ICON..	Output. Condition code. See Table FCOSM-1.

Table FCOSM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	ISN ≠ 1, ISN ≠ -1 or N ≠ 2^l (l = 0 or positive integer)	Aborted.

Comments on use

- Subprograms used

  SSL II ... UPNR2, UTABT, UCOSM and MGSSL

  FORTRAN basic functions ... SQRT and MAX0

- Notes

  General definition of Fourier transform:

  The discrete cosine transform and its inverse transform based on the midpoint rule are generally defined by (3.1) and (3.2).

$$a_k = \frac{2}{n}\sum_{j=0}^{n-1} x_{j+1/2} \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right), \quad k = 0,1,...,n-1 \qquad (3.1)$$

$$x_{j+1/2} = \sum_{k=0}^{n-1}{}' a_k \cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right), \quad j = 0,1,...,n-1 \qquad (3.2)$$

The subroutine obtains $\{n/2 \cdot a_k\}$ and $\{x_{j+1/2}\}$ which correspond to the left-hand side of (3.1) and (3.2), respectively, and the user has to scale the results, if necessary.

Use of the trigonometric function table:

When the subroutine is called successively for transforms of a fixed dimension, the trigonometric function table is calculated and created only once. Therefore, when calling the subroutine subsequently, the contents of the parameter TAB must be kept intact.

Even when the dimension differs, the trigonometric function table need not be changed. A new trigonometic function table entry can be made on an as-required basis.

- Example

  By inputting n sample points $\{x_{j+1/2}\}$, performing the transform by the subroutine and scaling the results, the discrete Fourier coefficients $\{a_k\}$ are obtained. By performing the inverse transform after that, $\{x_{j+1/2}\}$ are obtained.

  Here $n \le 512$.

```
C     **EXAMPLE**
      DIMENSION X(512),TAB(511)
C     COSINE TRANSFORM
      ISN=1
      READ(5,500) N,(X(I),I=1,N)
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,N)
      CALL FCOSM(X,N,ISN,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
C     NORMALIZE
      CN=2.0/FLOAT(N)
      DO 10 K=1,N
      X(K)=X(K)*CN
   10 CONTINUE
      WRITE(6,602) ISN
      WRITE(6,601) (X(I),I=1,N)
C     COSINE INVERSE TRANSFORM
      ISN=-1
      CALL FCOSM(X,N,ISN,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602) ISN
      WRITE(6,601) (X(I),I=1,N)
   20 WRITE(6,603) ICON
      STOP
  500 FORMAT(I5/(6F12.0))
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA',5X,
     *       'ISN=',I2)
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

Method

The discrete cosine transform based on the midpoint rule of dimension n $(=2^l, l = 0, 1, ...)$ is performed by using the radix 2 Fast Fourier Transform (FFT).

The transform based on the midpoint rule can be accomplished by considering the even function $x(t)$ to be a complex valued function and performing the discrete complex Fourier transform for the even function based on the midpoint rule of dimension $2n$. In this case, however, it is known that the use of the characteristics of the complex transform permits efficient transform.

For the complex value function $x(t)$ with period 2π, the discrete Fourier transform based on the midpoint rule of dimension $2n$ $(= 2^{l+1}, l = 0, 1, 2, ...)$ is defined as

$$2n\alpha_k=\sum_{j=0}^{2n-1} x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)\exp\left(-\frac{\pi i}{n}k\left(j+\frac{1}{2}\right)\right),$$
$$k=0,1,...,2n-1 \tag{4.1}$$

The basic idea of the FFT is to accomplish the objective transform by repeating the elementary discrete Fourier transform by repeating the elementary discrete Fourier transform of small dimension (i.e., if the radix is 2, the dimension is 2). In other words, considering sample data which consist of $(j+1)$ th element and subsequent ones with the interval $\tilde{n}_p$, $(= 2^{l+1-p}, p = 1, 2, ...)$ and defining the transform

$$x^p(j,k)=\sum_{r=0}^{n_p-1} x\left(\frac{\pi}{n}\left(r\tilde{n}_p+j+\frac{1}{2}\right)\right)$$
$$\cdot\exp\left(-\frac{2\pi i}{n_p}\left(r+\left(j+\frac{1}{2}\right)\Big/\tilde{n}_p\right)k\right), \tag{4.2}$$
$$j=0,1,...,\tilde{n}_p-1, k=0,1,...,n_p-1$$

of dimension $n_p = 2^p$, then the transform (4.2) can be satisfied with the FFT algorithm (4.3) of radix 2.

Initial value

$$x^0(j,0)=x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right), j=0,1,...,2n-1$$
$$x^p(j,k)=x^{p-1}(j,k)+x^{p-1}(j+\tilde{n}_p,k)$$
$$x^p(j,k+n_{p-1})=\left\{x^{p-1}(j,k)-x^{p-1}(j+\tilde{n}_p,k)\right\}$$
$$\cdot\exp\left(-\frac{\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right) \tag{4.3}$$
$$j=0,1,...,\tilde{n}_p-1, k=0,1,...,n_{p-1}-1$$
$$p=1,2,...,l+1$$

The values obtained in the final step of these recurrence equations are the discrete Fourier coefficients (4.4).

$$2n\alpha_k=x^{l+1}(0,k), k=0,1,...,2n-1 \tag{4.4}$$

If $x(t)$ is an even function, the fact that $x(t)$ is real, i.e.,

$x(t) = \bar{x}(t)$ ($\bar{x}(t)$ is a complex conjugate of $x(t)$),

and symmetric, i.e.,

$$x(2\pi-t)=\bar{x}(t)$$

affects the intermediate results $\{x^p(j,k)\}$ of the FFT as follows:

Real property: $x^p(j, 0)$ is real

$$x^p(\tilde{n}_p-j-1,k)=\bar{x}^p(j,k),$$
$$j=0,1,...,\tilde{n}_p-1, k=0,1,...,n_p-1$$

Symmetric property:

$$x^p\left(j, n_p - k\right) = \bar{x}^p\left(j,k\right)\exp\left(-\frac{2\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right) \qquad (4.5)$$
$$j = 0,1,...,\tilde{n}_p - 1, k = 1,2,...,n_p - 1$$

On the other hand, the relationship

a_k=2α_k , k=0,1,...,n-1

can be satisfied between the complex Fourier coefficients {α_k} for the even function $x(t)$ and the Fourier coefficients defined by the following discrete cosine transform

$$a_k = \frac{2}{n}\sum_{j=0}^{n-1} x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)\cos\frac{\pi}{n}k\left(j+\frac{1}{2}\right)$$
$$k = 0,1,...,n-1$$

Therefore by using the characteristic (4.5) in the FFT algorithm (4.3) as well as the relationship above, the number of computations and the memory using them. That is, the range of j and k used in (4.3) are halved, so the FFT algorithm of radix 2 for the discrete cosine transform (4.6) can be represented by

$$x^p\left(j,k\right) = x^{p-1}\left(j,k\right) + \bar{x}^{p-1}\left(\tilde{n}_p - j - 1, k\right)$$
$$x^p\left(j, n_{p-1} - k\right) = \left\{\bar{x}^{p-1}\left(j,k\right) - x^{p-1}\left(\tilde{n}_p - j - 1, k\right)\right\}$$
$$\cdot \exp\left(-\frac{\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right) \qquad (4.7)$$
$$j = 0,1,...,\frac{\tilde{n}_p}{2} - 1, k = 0,1,...,\frac{n_{p-1}}{2} - 1$$
$$p = 1,2,...,l$$

Normally, the area to store the intermediate results {$x^p(j, k)$} needs one-dimensional array of size n, but if the input data is permuted in advance by reverse binary transformation, the above computation can be carried out by using the same area in which the data was input. The number of real number multiplications necessary for the cosine transform of dimension n is about $n\log_2 n$.

- Transform procedure in the subroutine
  - (a) The necessary trigonometric function table (size n-1) for the transform is created in reverse binary order.
  - (b) The sample points {$x(\pi/n(j+1/2))$} (size n) are permuted in reverse binary order.
  - (c) The FFT algorithm (4.7), which takes into consideration the symmetric property of the input data is performed in the same area to obtain the Foruier coefficients {a_k} in normal order.

- Inverse transform procedure
  - (a) The necessary trigonometric function table for the inverse transform procedure.
    This table is the same one as used in the transform procedure.
  - (b) By inputting the n Fourier coefficients {a_k} for the even function {$x(\pi/n(j+1/2))$}, and tracing backwards the recurrence equations (4.7) with respect to p, the function values {$x(\pi/n(j+1/2))$} are obtained in reverse binary order.
  - (c) By permuting the obtained n data by reverse binary transformation, the function values {$x(\pi/n(j+1/2))$} are obtained in normal order.
    For further details, refer to Reference [58].

F11-11-0101 FCOST, DFCOST

Discrete cosine transform (Trapezoidal rule, radix 2 FFT)
CALL FCOST (A, N, TAB, ICON)

Function

Given n+1 sample points $\{x_j\}$, by equally dividing the half period of the even function with period 2π into n,

$$x_j = x\left(\frac{\pi}{n} j\right) \quad , j = 0,1,...,n \qquad (1.1)$$

a discrete cosine transform or its inverse transform based on the trapezoidal rule is performed by using the Fast Fourier Transform (FFT). Here, $n = 2^l$ (l = 0 or positive integer).

- Cosine transform
  By inputting $\{x_j\}$ and performing the transform defined in (1.2), the Fourier coefficients $\{n/2 \cdot a_k\}$ are obtained.

$$\frac{n}{2} a_k = \sum_{j=0}^{n}{}'' x_j \cos\frac{\pi}{n} kj \quad , k = 0,1,...,n \qquad (1.2)$$

where Σ" denotes both the first and last terms of the sum are taken with factor 1/2.

- Cosine inverse transform
  By inputting $\{a_k\}$ and performing the transform defined in (1.3), the values of the Fourier series $\{x_j\}$ are obtained

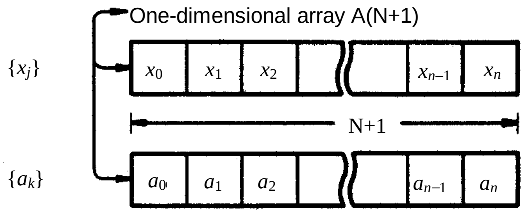
$$x_j = \sum_{j=0}^{n}{}'' a_k \cos\frac{\pi}{n} kj \quad , j = 0,1,...,n \qquad (1.3)$$

Parameters

A.....	Input. $\{x_j\}$ or $\{a_k\}$ Output. $\{n/2 \cdot a_k\}$ or $\{x_j\}$ One-dimensional array. See Fig. FCOST-1.
N.....	Input. Sample point number-1
TAB...	Output. Trigonometric function table used by the transform is stored. One-dimensional array of size n/2-1. See "Notes".
ICON..	Output. Condition code. See Table FCOST-1.

Table FCOST-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N ≠ 2^l (l: 0 or positive integer)	Bypassed



Note: $\{\frac{n}{2} a_k\}$ is handled in the same way as for $\{a_k\}$.

Fig. FCOST-1 Data storing method

Comments on use

- Subprograms used
  SSL II ... UPNR2, UTABT, UCOSM and MGSSL
  FORTRAN basic functions ... SQRT and MAX0

- Notes
  General definition of Fourier transform:
  The discrete cosine transform and its inverse transform based on the trapezoidal rule are generally defined by (3.1) and (3.2).

$$a_k = \frac{2}{n}\sum_{j=0}^{n}{}'' x_j \cos\frac{\pi}{n} kj \quad , k = 0,1,...,n \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n}{}'' a_k \cos\frac{\pi}{n} kj \quad , j = 0,1,...,n \qquad (3.2)$$

The subroutine obtains $\{n/2 \cdot a_k\}$ and $\{x_j\}$ which correspond to the left-hand side of (3.1) and (3.2), respectively and the user has to scale the results, if necessary.
Calculating trigonometric polynomial:
When obtaining values of the n-th order trigonometric polynomial

$$x(t) = \frac{1}{2} a_0 + a_1 \cos t + ... + a_n \cos nt \qquad (3.3)$$

at $x\left(\frac{\pi}{n} j\right), j = 0,1,...,n,$ by using the inverse transform, the highest order coefficient a_n must be doubled in advance. Refer to example (b).
Use of the trigonometric function table:
When the subroutine is called successively for transforms of a fixed dimension, the trigonometric function table is calculated and created only once. Therefore, when calling the subroutine subsequently, the contents of parameter TAB must be kept intact.
Even when the dimension differs, the trigonometric function table need not be changed. A new trigonometric function table entry can be made on an as-required basis.

- Example

(a) By inputting n+1 sample points $\{x_j\}$, performing the transform by the subroutine and scaling the results, the discrete Fourier coefficients $\{a_k\}$ are obtained. By performing its inverse transform after that, $\{x_j\}$ are obtained.

Here $n \leq 512$.

```
C     **EXAMPLE**
      DIMENSION X(513),TAB(255)
      READ(5,500) N
      NP1=N+1
      READ(5,501) (X(I),I=1,NP1)
C     COSINE TRANSFORM
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,NP1)
      CALL FCOST(X,N,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
C     NORMALIZE
      CN=2.0/FLOAT (N)
      DO 10 K=1,NP1
      X(K)=X(K)*CN
   10 CONTINUE
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,NP1)
C     COSINE INVERSE TRANSFORM
      CALL FCOST(X,N,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,NP1)
   20 WRITE(6,603) ICON
      STOP
  500 FORMAT(I5)
  501 FORMAT(6F12.0)
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA')
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

(b) By inputting cosine coefficients $\{a_k\}$, the values $\{x(\pi j/n)\}$ of the n-th order trigonometric polynomial

$$x(t) = \frac{1}{2}a_0 + a_1 \cos t + ... + a_{n-1}\cos(n-1)t$$
$$+ a_n \cos nt$$

at the sample points $\{\pi j/n\}$ are obtained. The coefficient of the last term must be doubled before it is input.

Here $n \leq 512$.

```
C     **EXAMPLE**
      DIMENSION A(513),TAB(255)
      READ(5,500) N
      NP1=N+1
      READ(5,501) (A(K),K=1,NP1)
      WRITE(6,600) N
      WRITE(6,601) (A(K),K=1,NP1)
      A(NP1)=A(NP1)*2.0
      CALL FCOST(A,N,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602)
      WRITE(6,601) (A(K),K=1,NP1)
   20 WRITE (6,603) ICON
      STOP
  500 FORMAT(I5)
  501 FORMAT(6F12.0)
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA')
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

Method

The discrete cosine transform based on the trapezoidal rule of dimension n+1 (=2^l+1, l = 0, 1, ...) is performed by using the radix 2 Fast Fourier Transform (FFT).

The transform based on the trapezoidal rule can be accomplished efficiently by using the transform based on the midpoint rule. The subroutine uses this method.

Dividing equally into n_p (=2p, p = 0,1, ...) the half period [0, π] of an even function $x(t)$ with period 2π, the discrete cosine transform based on the trapezoidal rule is defined as

$$a_k^p = \sum_{j=0}^{n_p}{}'' x\left(\frac{\pi}{n_p} j\right)\cos\frac{\pi}{n_p}kj \quad ,k = 0,1,...,n_p \qquad (4.1)$$

Also the discrete cosine transform based on the midpoint rule of dimension n_p is defined as

$$\hat{a}_k^p = \sum_{j=0}^{n_p-1} x\left(\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right)\cos\frac{\pi}{n_p}k\left(j+\frac{1}{2}\right),$$
$$k = 0,1,...,n_p - 1 \qquad (4.2)$$

In Eqs. (4.1) and (4.2), however, the ordinary scaling factor $2/n_p$ is omitted.

Doubling the division number to n_{p+1}, the $\{a_k^{p+1}\}$ can be expressed as follows by using $\{ a_k^p \}$ and $\{ \hat{a}_k^p \}$ both of which have half as many dimension as $\{ \hat{a}_k^{p+1} \}$.

$$\left.\begin{aligned} a_k^{p+1} &= a_k^p + \hat{a}_k^p \\ a_{n_{p+1}-k}^{p+1} &= a_k^p - \hat{a}_k^p \end{aligned}\right\} k = 0,1,...,n_p - 1 \qquad (4.3)$$
$$a_{np}^{p+1} = a_{np}^p$$

The equations shown in (4.3) can be considered as recurrence equations with respect to p to obtain { a_k^{p+1} } from { a_k^p }. Therefore, when performing transform with division number 2^l, if the initial condition { a_k^1 ; k = 0, 1, 2} is given and the discrete cosine transform based on the midpoint rule of dimension 2^p is performed each at the p-th stage (p = 1, 2, ..., l -1), then the discrete cosine transform series { a_k^p , k = 0, 1, ..., n_p} (p = 1, 2, ..., l) based on the trapezoidal rule can be obtained by using equations (4.3).

The number of multiplications of real numbers executed is about $n\log_2 n$, which means the calculation method described above to obtain { a_k^l } is fast.

Procedural steps taken in the subroutine
Taking the first n points out of the sequenced n+1 sample points x_0, x_1, ..., x_n and permuting them in reverse binary order representing x(0), x(1), ..., x(n). In this way, all the necessary data for the recurrence equations (4.3) can be gained successively.

Next, the trigonometric function table (size n/2-1) necessary for the transform is made in reverse binary order corresponding to the sample point numbers. Finishing with the preparatory processing, the cosine transform is performed as follows:

(a) Initialization
{ a_k^1 , k = 0, 1, 2} with p=1 are obtained by using x(0), x(1), x(n) and stored in x(.), respectively.

(b) Cosine transform based on the midpoint rule of dimension 2^p.
By inputting n_p sample points, $x(n_p)$, $x(n_p+1)$, ..., $x(n_{p+1}-1)$, and performing the cosine transform based on the midpoint rule, { a_k^p) are obtained in the same area. For the cosine transform based on the midpoint rule, see method for subroutine FCOSM.

(c) Calculation of recurrence equations with respect to { a_k^p }
All of the intermediate results are overwritten, so no supplementary work area is needed. That is, the four elements, a_k^p , $a_{n_p-k}^p$, $\hat{a}_k^p$ and $\hat{a}_{n_p-k}^p$, at the (p+1) th step are calculated according to the recurrence equations by using the four elements, a_k^{p+1}, $a_{n_p-k}^{p+1}$, $a_{n_p+k}^{p+1}$ and $a_{n_{p+1}-k}^{p+1}$ at the p-th stage, and are stored in the same positions corresponding to the ones at the p-th stage.
Repeating stages (b) and (c) with p = 1, 2, ..., l-1, {a_k} can be obtained.

For further details, see Reference [58].

E51-20-0101 FSINF, DFSINF

Sine series expansion of an odd function (Fast sine transform)
CALL FSINF (TH, FUN, EPSA, EPSR, NMIN, NMAX, B, N, ERR, TAB, ICON)

Function

This subroutine performs sine series expansion of a smooth odd function $f(t)$ with period $2T$ according to the required accuracy of ε_a and ε_r. It determines n coefficients $\{b_k\}$ which satisfy

$$\left| f(t) - \sum_{k=0}^{n-1} b_k \sin \frac{\pi}{T} kt \right| \le \max\{\varepsilon_a, \varepsilon_r \|f\|\} \tag{1.1}$$

Since $\{b_k\}$ contains trivial coefficient $b_0 = 0$, the number of coefficients to be expanded actually is n-1. The norm $\|f\|$ of $f(t)$ is defined as shown in (1.3) by using function values taken at sample points shown in (1.2) within the half period $[0, T]$.

$$t_j = \frac{T}{n-1} j, \quad j = 0,1,...,n-1 \tag{1.2}$$

$$\|f\| = \max_{0 \le j \le n-1} \left| f(t_j) \right| \tag{1.3}$$

Where $T > 0$, $\varepsilon_a \ge 0$, $\varepsilon_r \ge 0$.

Parameters

TH.....	Input. Half period T of the function $f(t)$.
FUN....	Input. Name of the function subprogram which calculates $f(t)$ to be expanded in a sine series. See Example of using this subroutine.
EPSA...	Input. The absolute error tolerance ε_a.
EPSR...	Input. The relative error tolerance ε_r.
NMIN...	Input. Lower limit of terms of sine series (>0). NMIN should be taken a value such as power of 2. The default value is 8. See Notes.
NMAX...	Input. Upper limit of terms of sine series. (NMAX > NMIN). NMAX should be taken a value such as power of 2. The default value is 256. See Notes.
B.....	Output. Coefficients $\{b_k\}$. One-dimensional array of size NMAX. Each coefficient is stored as shown below: B(1)=b_0, B(2)=b_1, ..., B(N)=b_{n-1},
N.....	Output. Number of terms n of the sine series (≥4). N takes a value such as power of 2.
ERR....	Output. Estimate of the absolute error of the series.
TAB....	Output. TAB contains a trigonometric function table used for series expansion. One-dimensional array whose size is greater than 3 and equal to NMAX/2-1.
ICON...	Output. Condition code. See Table FSINF-1.

Table FSINF-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The required accuracy was not satisfied due to rounding-off errors. The required accuracy is too high.	B contains resultant coefficients. The accuracy of the series is the maximum attainable.
20000	The required accuracy was not satisfied though the number of terms of the series has reached the upper limit.	Bypassed. B contains resultant coefficients and ERR contains an estimate of absolute error.
30000	One of the following cases occurred: 1 TH ≤ 0 2 EPSA < 0.0 3 EPSR < 0.0 4 NMIN < 0 5 NMAX < NMIN	Bypassed

Comments on use

- Subroutines used
  SSL II ... MGSSL, AMACH, UTABT, USINM and UNIFC
  FORTRAN basic functions ... ABS, AMAX1, AMIN1 and FLOAT

- Notes
  The function subprogram specified by the FUN parameter must be a subprogram defined at the interval $[0, T]$ having independent variable t only as the argument.
  The name must be declared by the EXTERNAL statement in the program which calls this subroutine.
  If the function contains auxiliary variable, they must be declared by a COMMON statement to establish an interface with the calling program.
  See Example of using this subroutine.

  Use of the trigonometric function table
  When this subroutine is repeatedly called, the trigonometric function table is produced only once. A new trigonometric function table entry is made on an as-required basis. Therefore the contents of TAB must be kept intact when this subroutine is called subsequently.

  If NMIN of NMAX does not take a value such as power of 2, this subroutine assumes the maximum value of power of 2 which does not exceed that value. However NMAX = 4 is assumed if NMAX < 4 is satisfied.

The degree of error decerement greatly depends on the smoothness of *f*(*t*) in the open interval (-∞, ∞) as the number of terms *n* increases. If *f*(*t*) is an analytical periodic function, the error decreases according to exponential function order O (r^n) ($0 < r < 1$).
If it has up to *k*-th continuous derivatives, the error decreases according to rational function order O(n^{-k}). When $k = 0$ or $k = 1$, an estimate of absolute error is not always accurate because the number of terms to be expanded increases greatly. Therefore, the function used by this subroutine should have, at least, up to 2-nd continuous derivatives.

Accuracy of the series
This subroutine determines a sine series which satisfies (1.1) according to the required accuracy of ε_a and ε_r.
If $\varepsilon_r = 0$ is specified, this subroutine expands *f*(*t*) in a sine series within the required accuracy of absolute error ε_a.
Similarly $\varepsilon_a = 0$ is specified, this subroutine expands *f*(*t*) in a sine series within the required accuracy of relative error ε_r.
However sine series expansion is not always successful depending on the specification of ε_a and ε_r. For example, when ε_a or ε_r is too small in comparison with computational error of *f*(*t*), the effect of rounding-off errors becomes greater on the computational result even if the number of terms to be expanded does not reach the upper limit.
In such a case, this subroutine abnormally terminates after a condition code of 10000 is set to ICON. At this time, the accuracy of the sine series becomes the attainable limit for the computer used. The number of terms to be expanded in a sine series sometimes does not converge within NMAX evaluations depending on the characteristics of *f*(*t*). In such a case, this subroutine abnormally terminates after a condition code of 20000 is set to ICON. Each coefficient is an approximation obtained so far, and is not accurate. To determine the accuracy of sine series, this subroutine always set an estimate of absolute error in ERR.

Any inverse transform can be attempted by the subroutine FSINT. Note that the contents of TAB must be kept intact whether normal or inverse transform is attempted. See Example 2.
When *f*(*t*) is only a periodical function, this subroutine can be used to perform sine series expansion for odd function as (*f*(*t*) - *f*(-*t*))/2.
If *f*(*t*) has no period and is absolutely integrable, its theoretical sine transform can be defined as shown in (3.1):

$$F(\omega) = \int_0^\infty f(t) \sin \omega t dt \qquad (3.1)$$

If *f(t)* is damped according to order of O (e^{-at}) ($a > 0$), an approximation of the Fourier integral can be obtained as described below:
Assume that $|f(t)|$ can be ignored on the interval [*T*, ∞) when *T* is sufficiently large.
By defining *T* which satisfies (3.2).

$$|f(t)| < u, t \geq T \qquad (3.2)$$

Where *u* is the unit round off.
This subroutine can be used to determine sine series coefficients $\{b_k\}$ for *f(t)*, assuming that *f(t)* is a function with period 2*T*.
Since $\{b_k\}$ can be expressed as

$$b_k = \frac{2}{T} \int_0^T f(t) \sin \frac{\pi}{T} ktdt \qquad (3.3)$$

(3.4) can be established based on (3.1) and (3.2).

$$F\left(\frac{\pi}{T}k\right) \approx \frac{T}{2} a_k, k = 0,1,...,n-1 \qquad (3.4)$$

Based on this relationship this subroutine can calculate an approximation of sine transform shown in (3.1) by using discrete sine transform.
When inverse transform

$$f(t) = \frac{2}{\pi} \int_0^\infty F(\omega) \sin \omega t d\omega \qquad (3.5)$$

is to be calculated, the subroutine FSINT can be called for *n* pieces of data as follows:

$$\frac{2}{T} F\left(\frac{\pi}{T}k\right), k = 0,1,...,n-1$$

The subroutine FSINT can obtain an approximation of:

$$f\left(\frac{T}{n}j\right), j = 0,1,...,n-1$$

See Example 2.

- Examples

Example 1:
This example expands the following odd function with period 2π having auxiliary variable *p*

$$f(t) = \frac{\sin t}{1 - 2p \cos t + p^2}$$

in a sine series according to the required accuracy of $\varepsilon_a = 5 \cdot 10^{-5}$ and $\varepsilon_r = 5 \cdot 10^{-5}$. Where NMIN = 8 and NMAX = 256 are assumed.

The theoretical sine series expansion of *f(t)* is as follows:

$$f(t) = \sum_{k=1}^{\infty} p^{k-1} \sin kt$$

This example prints sine series coefficients when p = 1/4, 1/2 and 3/4.

```
C      **EXAMPLE**
       DIMENSION B(256),TAB(127)
       EXTERNAL FUN
       COMMON P
       TH=ATAN(1.0)*4.0
       EPSA=0.5E-04
       EPSR=EPSA
       NMIN=8
       NMAX=256
       P=0.25
     1 CALL FSINF(TH,FUN,EPSA,EPSR,
      *NMIN,NMAX,B,N,ERR,TAB,ICON)
       IF(ICON.GT.10000) GO TO 10
       WRITE(6,600) N,ERR,ICON,P
       WRITE(6,601) (B(I),I=1,N)
       P=P+0.25
       IF(P.LT.1.0) GO TO 1
       STOP
    10 WRITE(6,602) ICON
       STOP
   600 FORMAT('0',5X,'EXPANSION OF',
      *' FUNCTION FUN(T)',3X,'N=',I4,
      *5X,'ERR=',E15.5,5X,'ICON=',I6,5X,
      *'P=', E15.5)
   601 FORMAT(/(5E15.5))
   602 FORMAT('0',5X,'CONDITION CODE',I8)
       END
       FUNCTION FUN(T)
       COMMON P
       FUN=SIN(T)/(1.0-2.0*P*COS(T)+P*P)
       RETURN
       END
```

Example 2:
Sine transform and inverse transform
This example transform odd function

$$F(\omega) = \int_0^{\infty} te^{-x^2} \sin \omega t dt$$

in a sine series according to the required accuracy of $\varepsilon_a = 5 \cdot 10^{-5}$ and $\varepsilon_r = 5 \cdot 10^{-5}$ and compares the results with analytical solution

$$F(\omega) = \frac{\sqrt{\pi}}{4} \omega \cdot e^{-\omega^2/4}$$

Then, this example performs inverse transform of the function by using the subroutine FSINT and checks the accuracy of the results.

```
C      **EXAMPLE**
       DIMENSION B(256),TAB(127),
      *          ARG(256),T(256)
       EXTERNAL FUN
       COMMON PI,SQPI
       PI=ATAN(1.0)*4.0
       SQPI=SQRT(PI)
       TH=SQRT(ALOG(4.0/AMACH(TH)))
       EPSA=0.5E-04
       EPSR=0.0
       NMIN=8
       NMAX=256
C      SINE TRANSFORM
       CALL FSINF(TH,FUN,EPSA,EPSR,NMIN,
      *NMAX,B,N,ERR,TAB,ICON)
       IF(ICON.GT.10000) GO TO 10
       TQ=TH*0.5
       H=PI/TH
       DO 1 K=1,N
       ARG(K)=H*FLOAT(K-1)
       B(K)=B(K)*TQ
       T(K)=TRFN(ARG(K))
     1 CONTINUE
       WRITE(6,600) N,ERR,ICON
       WRITE(6,610)
       WRITE(6,601) (ARG(K),B(K),T(K), K=1,N)
C      INVERSE TRANSFORM
       Q=1.0/TQ
       DO 2 K=1,N
       B(K)=B(K)*Q
     2 CONTINUE
       CALL FSINT(B,N,TAB,ICON)
       IF(ICON.NE.0) GO TO 10
       H=TH/FLOAT(N)
       DO 3 K=1, N
       ARG(K)=H*FLOAT(K-1)
       T(K)=FUN(ARG(K))
     3 CONTINUE
       WRITE(6,620)
       WRITE(6,610)
       WRITE(6,601) (ARG(K),B(K),T(K),K=1,N)
       STOP
    10 WRITE(6,602)ICON
       STOP
   600 FORMAT('0',5X,' CHECK THE SINE',
      *' TRANSFORM OF FUNCTION FUN(T)',
      *3X,'N=',I4,5X,'ERR=',E15.5,5X,
      *'ICON=',I5)
   610 FORMAT('0',6X,'ARGUMENT',7X,
      *'COMPUTED',11X,'TRUE')
   620 FORMAT('0',5X,'CHECK THE INVERSE'
      *,' TRANSFORM')
   601 FORMAT(/(3E15.5))
   602 FORMAT('0',5X,'CONDITION CODE',I6)
       END
       FUNCTION FUN(T)
       FUN=T*EXP(-T*T)
       RETURN
       END
       FUNCTION TRFN(W)
       COMMON PI,SQPI
       TRFN=W*EXP(-W*W*0.25)*SQPI*0.25
       RETURN
       END
```

Method

This subroutine applies discrete fast sine transform (based on the trapezoidal rule) to sine transform for entry of functions.

- Sine series expansion

  For simplicity, an odd function $f(t)$ with a period of 2π. The function can be expanded in a sine series as shown below:

$$f(t)=\sum_{k=1}^{\infty} b_k \sin kt \qquad (4.1)$$

$$b_k = \frac{2}{\pi}\int_0^{\pi} f(t)\sin kt\,dt \qquad (4.2)$$

  This subroutine uses the trapezoidal rule to compute (4.2) by dividing the closed interval $[0, \pi]$ equally. By using resultant coefficients $\{b_k\}$ this subroutine approximates (4.1) by finite number of terms.

  If this integrand is smooth, the number of terms is doubled as far as the required accuracy of ε_a and ε_r is satisfied. If sampling is sufficient, (4.3) will be satisfied.

$$\left| f(t)-\sum_{k=0}^{n-1} b_k \sin kt \right| < \max\{\varepsilon_a, \varepsilon_r \| f \|\} \qquad (4.3)$$

  where n indicates the number of samples (power of 2+1) and b_0=0. The resultant trigonometric polynomial is a trigonometric interpolation polynomial in which each breakpoint used by the trapezoidal rule is an interpolation point as shown below:

$$f\left(\frac{\pi}{n} j\right)=\sum_{k=0}^{n-1} b_k \sin\frac{\pi}{n}kj, \qquad (4.4)$$
$$j = 0,1,\ldots,n-1$$

  The sine series expansion is explained in detail below. Assume that coefficients obtained by the trapezoidal rule using n sample points ($n=n_p$, $n_p=2^p$) are

$$b_k^p = \sum_{j=0}^{n_p-1} f\left(\frac{\pi}{n_p} j\right)\sin\frac{\pi}{n_p}kj, \qquad (4.5)$$
$$k = 1,2,\ldots,n_p-1$$

  Where the ordinary scaling factor $2/n_p$ is omitted from (4.5).

  When the number of terms is doubled as $n_{p+1} = 2n_p$ each coefficient can efficiently be determined by making a good use of complementary relation between the trapezoidal rule and the midpoint rule. At each midpoint between sample points used by the trapezoidal rule (4.5), $f(t)$ can be sampled as shown below:

$$f\left(\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right), j = 0,1,\ldots,n_p-1 \qquad (4.6)$$

Discrete sine transform (using the midpoint rule) for (4.6) can be defined as shown below:

$$\tilde{b}_k^p = \sum_{j=0}^{n_p-1} f\left(\frac{\pi}{n_p}\left(j+\frac{1}{2}\right)\right)\sin\frac{\pi}{n_p}k\left(j+\frac{1}{2}\right), \qquad (4.7)$$
$$k = 1,2,\ldots,n_p$$

Since (4.8) is satisfied at this stage $\{b_k^{p+1}\}$ can be determined.

$$\left.\begin{aligned} b_k^{p+1} &= b_k^p + \tilde{b}_k^p \\ b_{n_{p+1}-k}^{p+1} &= \tilde{b}_k^p - b_k^p \\ b_{n_p}^{p+1} &= \tilde{b}_{n_p}^{p+1} \end{aligned}\right\}, k = 1,2,\ldots,n_p-1 \qquad (4.8)$$

  By using this recurrence formula for $\{b_k^p\}$, $f(t)$ can be expanded in a sine series of higher degree while the number of terms is doubled as far as the required accuracy is satisfied.

  Then $\{b_k^p\}$ is normalized by multipling by factor $2/n_p$.

- Error evaluation for sine series

  The following relationship exists between the theoretical sine coefficients $\{b_k\}$ of $f(t)$ and discrete sine coefficients $\{b_k^p\}$, taken at the p-th stage:

$$b_k^p = b_k + \sum_{m=1}^{\infty}\left(b_{2mn_p-k} + b_{2mn_p+k}\right), \qquad (4.9)$$
$$k = 1,2,\ldots,n_p-1$$

  This results from (4.2) and (4.5) as well as orthogonality of trigonometric functions.

  The error evaluation for a sine series at the p-th stage

$$\left| f(t)-\sum_{k=0}^{n_p-1} b_k^p \sin kt \right| \le \left|b_{n_p}\right| + 2\sum_{k=n_p+1}^{\infty}\left|b_k\right| \qquad (4.10)$$

  can be deduced from (4.9).

  If $f(t)$ is an analytical periodic function, its series coefficients $\{b_k\}$ decrease according to exponential function order $O(r^k)$ $(0 < r < 1)$ as k increases. Then r can be estimated from a discrete sine coefficient at the p-th stage. Let $b_k^p = Ar_k$ (A: constant). Since k is at most $n_p - 1, r^{n_p/4}$ can be estimated from the ration of the coefficient of the last term n_p-1 to the coefficient of term $3/4n_p$-1. This subroutine does not allow the two coefficients to be zero by accident. Therefore it uses the (n_p-2)-th and ($3/4n_p$-2)-th coefficients together with those coefficients to estimate a value of r as shown below.

$$r = \min\left\{\left(\frac{\left|b_{n_p-2}^p\right| + \left|b_{n_p-1}^p\right|}{\left|b_{\frac{3}{4}n_p-2}^p\right| + \left|b_{\frac{3}{4}n_p-1}^p\right|}\right)^{\frac{4}{n_p}}, 0.99\right\}$$

If r is greater than 0.99, this subroutine cannot actually expand $f(t)$ in a sine series because the convergence rate of the series becomes weak. By using the resultant r, the p-th stage error.

$$e_p = \frac{r}{1-r}\left(\left|b^p_{n_p-2}\right| + \left|b^p_{n_p-1}\right|\right) \qquad (4.11)$$

can be estimated from (4.10).

- Computational process

Step 1: Initialization

– Initialization of Trigonometric function table
At three points which divides interval [0, $\pi/2$] equally, three values for the cosine function is obtained in reverse binary order. The trigonometric function table is not initialized if this subroutine is called in advance. The trigonometric function table is used for discrete sine transform.

– Initial sine series expansion
This subroutine performs 4(p=2) in (4.5) and calculates $0.0, b_1^2, b_2^2, b_3^2$. At this time, it also obtains $\|f\|$ based on the norm definition shown in (1.3).

Step 2: Convergence criterion
If n_p < NMIN is satisfied this subroutine does not perform a convergence test but immediately executes Step 3.
If n_p > NMIN is satisfied, this subroutine performs a convergence test as described below:
This subroutine estimates computational error limit

$$\rho = \sqrt{n_p}\,\|f\|(2u), \qquad (4.12)$$

where u is the unit round off.
and a tolerance for convergence test as

$$\varepsilon = \max\{\varepsilon_a, \varepsilon_r\|f\|\} \qquad (4.13)$$

If the last two terms at the p-th stage have been lost significant digits, that is, if the coefficients satisfy (4.14).

$$\left|b^p_{n_p-2}\right| + \frac{1}{2}\left|b^p_{n_p-1}\right| < \rho \qquad (4.14)$$

the computational accuracy cannot be increased even if this computation continues.
Therefore, this subroutine replaces the absolute error e_p of the sine series by the computational error ρ, assuming that the sine series is converged.
If $\rho < \varepsilon$ is satisfied, this subroutine sets a condition code of 0 to ICON. If $\rho \geq \varepsilon$ is satisfied, this subroutine sets a condition code of 10000 to ICON assuming that ε_a or ε_r is relatively smaller than unit round off u.
If (4.14) is not satisfied, this subroutine estimates absolute error e_p based on (4.11). If $e_p \geq \varepsilon$ is satisfied, this subroutine sets a condition code of 0 to ICON and terminates normally. If $e_p < \varepsilon$ is not satisfied but $2n_p \leq$ NMAX is satisfied, this subroutine immediately executes Step 3. Otherwise this subroutine sets a condition code of 20000 to ICON and terminates abnormally assuming that the required accuracy is not satisfied even when the number of terms to be expanded its reached the upper limit. Note that each coefficient is normalized whether this subroutine terminates normally or abnormally.

Step 3: Calculation of sample points
Sample points to be used for sampling of $f(t)$ at the p-th stage can be expressed as follows:

$$t_j = \frac{\pi}{n_p}\left(j + \frac{1}{2}\right), j = 0,1,...,n_p - 1$$

They can be obtained in reverse binary order through use of the recurrence formula shown below:

$$\begin{aligned}&t_0 = \pi / 2^{p+1}\\&t_{(2j+1)2^{p-l-1}} = t_{j2^{p-l}} + 2^{-p+l}\pi, \qquad (4.16)\\&j = 0,1,...,2^l - 1, l = 0,1,..., p-1,\end{aligned}$$

where $n_p = 2^p$.

Step 4: Sampling of $f(t)$ and calculation of the norm
This subroutine obtains values of $f(t)$ for n sample points based on (4.16) and overwrites them on the sample points.
It also calculates norm $\|f\|$ based on the norm definition shown in (1.3).

Step 5: Trigonometric function table creation
This subroutine produces the trigonometric function table required by Step 6. The trigonometric function table is not recalculated each time this subroutine is called.

Step 6: Discrete sine transform (using the midpoint rule)
For sample points obtained by Step 4, this subroutine performs discrete sine transform using the Fast Fourier Transform (FFT) method to determine { $\tilde{b}_k^p$ }.

Step 7: Calculation of { b_k^{p+1} }
This subroutine combines { b_k^p } obtained previously with { $\tilde{b}_k^p$ } by using (4.8) to obtain the coefficients { b_k^{p+1} } of the discrete sine series consisting of $2n_p$ + 1 terms.
Then, this subroutine executes Step 2 after it increases a value of p by one.

Step 4 and 6 consume most of the time required to execute this subroutine.
The number of multiplications required to determine the coefficients of a sine series consisting of n

terms is about $n\log_2 n$.
To save storage this subroutine overwrites sample points, samples and expansion coefficients onto a one-dimensional array B.

For further information, see Reference [59].
For detailed information about discrete sine transform, see an explanation of the subroutines FSINT and FSINM.

F11-21-0201 FSINM, DFSINM

Discrete sine transform (midpoint rule, radix 2 FFT)
CALL FSINM (A, N, ISN, TAB, ICON)

Function

Given n sample points $\{x_{j+1/2}\}$,

$$x_{j+1/2} = x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right), j = 0,1,...,n-1 \qquad (1.1)$$

by equally dividing the half period of the odd function $x(t)$ with period 2π, a discrete sine transform or its inverse transform based on the midpoint rule is performed by using the Fast Fourier Transform (FFT).
Here $n = 2^l$ (l = 0 or positive integer).

- Sine transform
  By inputting $\{x_{j+1/2}\}$ and performing the transform defined in (1.2), the Fourier coefficients $\{n/2 \cdot b_k\}$ are obtained.

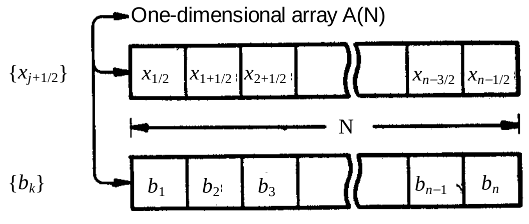
$$\frac{n}{2}b_k = \sum_{j=0}^{n-1} x_{j+1/2} \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right), k = 1,2,...,n \qquad (1.2)$$

- Sine inverse transform
  By inputting $\{b_k\}$ and performing the transform defined in (1.3), the value of the Fourier series $\{x_{j+1/2}\}$ are obtained.

$$x_{j+1/2} = \sum_{k=0}^{n-1} b_k \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right) + \frac{1}{2}b_n \sin\pi\left(j+\frac{1}{2}\right), \qquad (1.3)$$
$$j = 0,1,...,n-1$$

Parameters

A.....	Input. $\{x_{j+1/2}\}$ or $\{b_k\}$
	Output. $\{n/2 \cdot b_k\}$ or $\{x_{j+1/2}\}$
	One-dimensional array of size n
	See Fig. FSINM-1.
N.....	Input. Sample point number n



Note: $\{n/2 \cdot b_k\}$ is handled in the same way as for $\{b_k\}$.

Fig. FSINM-1 Data storing method

ISN...	Input. Transform or inverse transform is indicated.
	For transform: ISN = +1
	For inverse transform: ISN = -1
TAB...	Output. Trigonometric function table used in the transform is stored.
	One-dimensional array of size n-1.
	See "Notes".
ICON..	Output. Condition code
	See Table FSINM-1.

Table FSINM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	ISN ≠ 1, ISN ≠ -1 or N ≠ 2^l (l = 0 or positive integer)	Bypassed

Comments on use

- Subprograms used
  SSL II ... UPNR2, UTABT, USINM and MGSSL
  FORTRAN basic functions ... SQRT and MAX0

- Notes
  General definition of Fourier Transform:
  The discrete sine transform and its inverse transform based on the midpoint rule are generally defined by (3.1) and (3.2)

$$b_k = \frac{2}{n}\sum_{j=0}^{n-1} x_{j+1/2} \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right), \qquad (3.1)$$
$$k = 1,2,...,n$$

$$x_{j+1/2} = \sum_{k=1}^{n-1} b_k \sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right) + \frac{1}{2}b_n \sin\pi\left(j+\frac{1}{2}\right), \qquad (3.2)$$
$$j = 0,1,...,n-1$$

  The subroutine obtains $\{n/2 \cdot b_k\}$ and $\{x_{j+1/2}\}$ which correspond to the left-hand side of (3.1) and (3.2), respectively, and the user has to scale the results, if necessary.
  Calculation of trigonometric polynomial:
  When obtaining the values $x(\pi/n(j+1/2))$ of the n-th order trigonometric polynomial

$$x(t) = b_1 \sin t + b_2 \sin 2t + ... + b_n \sin nt$$

  by using the inverse transform, the highest order coefficient b_n must be doubled in advance. See example (b).
  Use of the trigonometric function table:
  When the subroutine is called successively for transforms of a fixed dimension, the trigonometric function table is calculated and created only once. Therefore, when calling the subroutine subsequently, the contents of the parameter TAB must be kept intact.

Even when the dimension differs, the trigonometric function table need not be changed. A new trigonometric function table entry can be made on an as-required basis.

- Example
  (a) By imputing n sample point $\{x_{j+1/2}\}$ performing the transform in the subroutine and scaling the results, the discrete Fourier coefficients $\{b_k\}$ are obtained. By performing the inverse transform after that, $\{x_{j+1/2}\}$ are obtained.
  Here $n \le 512$.

```
C     **EXAMPLE**
      DIMENSION X(512),TAB(511)
C     SINE TRANSFORM
      ISN=1
      READ(5,500) N,(X(I),I=1,N)
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,N)
      CALL FSINM(X,N,ISN,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
C     NORMALIZE
      CN=2.0/FLOAT(N)
      DO 10 K=1,N
      X(K)=X(K)*CN
   10 CONTINUE
      WRITE(6,602) ISN
      WRITE(6,601) (X(I),I=1,N)
C     SINE INVERSE TRANSFORM
      ISN=-1
      CALL FSINM(X,N,ISN,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602) ISN
      WRITE(6,601) (X(I),I=1,N)
   20 WRITE(6,603) ICON
      STOP
  500 FORMAT(I5/(6F12.0))
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA',5X,
     *       'ISN=',I2)
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

  (b) By inputting sine coefficients $\{b_k\}$ based on the midpoint rule, the values $\{x(\pi(j+1/2)/n)\}$ of the n-th trigonometric polynomial.

$$x(t) = b_1 \sin t + b_2 \sin 2t + ... + b_n \sin nt$$

  are obtained at the sample points $\{\pi(j+1/2)/n\}$. The coefficient of the last term bn must be doubled before it is input.
  Here $n \le 512$.

```
C     **EXAMPLE**
      DIMENSION B(512),TAB(511)
      ISN=-1
      READ(5,500) N,(B(I),I=1,N)
      WRITE(6,600) N
      WRITE(6,601) (B(I),I=1,N)
      B(N)=B(N)*2.0
      CALL FSINM(B,N,ISN,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602) ISN
      WRITE(6,601) (B(I),I=1,N)
   20 WRITE(6,603) ICON
      STOP
  500 FORMAT(I5/(6F12.0))
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA',5X,
     *       'ISN=',I2)
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

Method

The discrete sine transform based on the midpoint rule of dimension $n(= 2^l, l=0, 1, ...)$ is performed by using the Fast Fourier Transform (FFT).

The transform based on the midpoint rule can be accomplished by considering the odd function *x(t)* to be a complex valued function and performing the discrete complex Fourier transform for the odd function based on the midpoint rule of dimension 2n. In this case, however, it is known that the use of the characteristics of the complex transform permits efficient transform.

For the complex valued function *x(t)* with period 2π, the discrete Fourier transform based on the midpoint rule of dimension $2n$ $(=2^{l+1}, l=0, 1, 2, ...)$ is defined as

$$2n\alpha_k = \sum_{j=0}^{2n-1} x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)\exp\left(-\frac{\pi i}{n}k\left(j+\frac{1}{2}\right)\right), \quad (4.1)$$
$$k = 0,1,...,2n-1$$

The basic idea of the FFT is to accomplish the objective transform by repeating the elementary discrete Fourier transform of small dimension (i.e., if the radix is 2, the dimension is 2). In other words, considering sample data which consist of (j+1)th element and subsequent ones with the interval $\tilde{n}_p (= 2^{l+1-p}, p = 1,2,...)$ and defining the transform

$$x^p(j,k) = \sum_{j=0}^{n_p-1} x\left(\frac{\pi}{n}\left(r\tilde{n}_{p+} j + \frac{1}{2}\right)\right)$$
$$\cdot \exp\left(-\frac{2\pi i}{n_p}\left(r + \left(j + \frac{1}{2}\right)\Big/\tilde{n}_p\right)k\right)$$
$$j = 0,1,...,\tilde{n}_p - 1, \quad k = 0,1,...,n_p - 1 \quad (4.2)$$

of dimension $n_p = 2^p$, then the transform can be satisfied with the FFT algorithm (4.3) of radix 2.

Initial value

$$
\begin{aligned}
&x^0(j,0) = x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right), j = 0,1...,2n-1\\
&x^p(j,k) = x^{p-1}(j,k) + x^{p-1}(j+\tilde{n}_p,k)\\
&x^p(j,k+n_{p-1}) = \{x^{p-1}(j,k) - x^{p-1}(j+\tilde{n}_p,k)\}\cdot\exp\left(-\frac{\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right)\\
&j = 0,1,...,\tilde{n}_p - 1, k = 0,1,...,n_{p-1}-1\\
&p = 1,2,...,l-1
\end{aligned}
\tag{4.3}
$$

The values obtained in the final step of these recurrence equations are the discrete Fourier coefficients (4.4).

$$2n\alpha_k = x^{l+1}(0,k), \qquad k = 0,1,...,2n-1 \tag{4.4}$$

If $x(t)$ is an odd function, the fact that $x(t)$ is real, i.e., $x(t) = \bar{x}(t)$ ($\bar{x}(t)$ is a complex conjugate of $x(t)$), and skew-symmetric, i.e.,

$$x(2\pi - t) = -\bar{x}(t)$$

affects the intermediate results $\{x^p(j, k)\}$ of the FFT as follows:

Real property: $x^p(j, 0)$ is real

$$
\begin{aligned}
&x^p(j,n_p-k) = \bar{x}^p(j,k)\exp\left(-\frac{2\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right),\\
&\quad j = 0,1,...,\tilde{n}_p-1, k = 1,2,...,n_p-1
\end{aligned}
\tag{4.5}
$$

Skew-symmetric property:

$$
\begin{aligned}
&x^p(\tilde{n}_p - j - 1,k) = -\bar{x}^p(j,k)\\
&j = 0,1,...,\tilde{n}_p - 1, \quad k = 0,1,...,n_p - 1
\end{aligned}
$$

On the other hand, the relationship

$$b_k = 2i\alpha_k, k = 1,2,...,n$$

can be satisfied between the complex Fourier coefficients $\{\alpha_k\}$ for the odd function $x(t)$ and the Fourier coefficients defined by the following discrete sine transform

$$
b_k = \frac{2}{n}\sum_{j=0}^{n-1} x\left(\frac{\pi}{n}\left(j+\frac{1}{2}\right)\right)\sin\frac{\pi}{n}k\left(j+\frac{1}{2}\right)
\qquad k = 1,2,\ldots,n
\tag{4.6}
$$

Therefore, by using the characteristic (4.5) in the FFT algorithm (4.3) as well as the relationship above, the number of computations and the memory used can be reduced to a quarter of those without using them. That is, the range of j and k used in (4.3) are halved, so the FFT algorithm of radix 2 for the discrete sine transform (4.6) can be represented by

$$
\begin{aligned}
&x^p(j,k) = x^{p-1}(j,k) - \bar{x}^{p-1}(\tilde{n}_p - j - 1,k)\\
&x^p(j,n_{p-1}-k) = \{\bar{x}^{p-1}(j,k) + x^{p-1}(\tilde{n}_p - j - 1,k)\}\\
&\qquad\qquad\cdot\exp\left(-\frac{\pi i}{\tilde{n}_p}\left(j+\frac{1}{2}\right)\right)\\
&j = 0,1,...,\frac{\tilde{n}_p}{2}-1, \quad k = 0,1,...,\frac{n_{p-1}}{2}-1\\
&p = 1,2,...,l
\end{aligned}
\tag{4.7}
$$

Normally, the area to store the intermediate results $\{x^p(j, k)\}$ needs one-dimensional array of size n, but if the input data is permuted in advance by reverse binary transformation, the above computation can be carried out by using the same area in which the data was input. The number of real number multiplications necessary for the sine transform of dimension n is about $n\log_2 n$.

- Transform procedure in the subroutine
  - (a) The necessary trigonometric function table (size n-1) for the transform is created in reverse binary order.
  - (b) The sample points $\{x(\pi/n(j+1/2))\}$ (size n) are permuted in reverse binary order.
  - (c) The FFT algorithm (4.7) which takes into consideration the skew-symmetric property of the input data is performed in the same area to obtain the Fourier coefficients in the order, b_n, b_{n-1}, ..., b_1.
  - (d) The Fourier coefficients $\{b_k\}$ are rearranged in ascending order.

- Inverse transform procedure
  - (a) The necessary trigonometric function table for the inverse transform is created. This table is the same one as used in the transform procedure (a).
  - (b) Rearranging the Fourier coefficients $\{b_k\}$ in the order, b_n, b_{n-1}, ..., b_1, $\{b_{n-k}\}$ is obtained.
  - (c) By inputting $\{b_{n-k}\}$ and tracing back words the recurrence equations (4.7) with respect to p, the function value $\{x(\pi/n(j+1/2)\}$ of the odd function $x(t)$ is obtained in reverse binary order.
  - (d) By permuting the obtained n data by reverse binary transformation, the function values $\{x(\pi/n(j+1/2)\}$ are obtained in normal order.

For further details, refer to the Reference [58].

F11-21-0101, FSINT, DFSINT

Discrete sine transform (Trapezoidal rule, radix 2 FFT)
CALL FSINT (A, N, TAB, ICON)

Function

Given n sample points $\{x_j\}$,

$$x_j = x\left(\frac{\pi}{n} j\right), \quad j = 0,1,...,n-1 \tag{1.1}$$

by dividing equally into n the half period of the odd function *x(t)* with period 2π, a discrete sine transform or its inverse transform based on the trapezoidal rule is performed by using the Fast Fourier Transform (FFT). Here $n = 2^l$ (n: positive integer)

- Sine transform
  By inputting $\{x_j\}$ and performing the transform defined in (1.2), the Fourier coefficients $\{n/2 \cdot b_k\}$ are obtained.

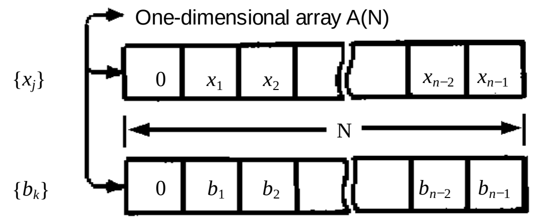
$$\frac{n}{2} b_k = \sum_{j=0}^{n-1} x_j \sin\frac{\pi}{n} kj, \quad k = 0,1,...,n-1 \tag{1.2}$$

- Sine inverse transform
  By inputting $\{b_k\}$ and performing the transform defined in (1.3), the values of the Fourier series $\{x_j\}$ are obtained.

$$x_j = \sum_{j=0}^{n-1} b_k \sin\frac{\pi}{n} kj, \quad j = 0,1,...,n-1 \qquad b_0 = 0. \tag{1.3}$$

Parameters

A.....	Input. $\{x_j\}$ or $\{b_k\}$. Output. $\{n/2 \cdot b_k\}$ or $\{x_j\}$. One-dimensional array of size n. See Fig. FSINT-1.



Note: $\{\frac{n}{2} b_k\}$ is handled the same way as for $\{b_k\}$.

Fig. FSINT-1 Data storing method

N....	Input. Sample point number n
TAB...	Output. Trigonometric function table used in the transform is stored. One dimensional array of size n/2-1. See “Notes”.
ICON..	Output. Condition code. See Table FSINT-1.

Table FSINT-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N ≠ 2^l (l: positive integer)	Bypassed

Comments on use

- Subprograms used
  SSL II ... UPNR2, UTABT, USINM and MGSSL
  FORTRAN basic functions ... SQRT and MAX0

- Notes
  General definition of Fourier transform:
  The discrete sine transform and its inverse transform based on the trapezoidal rule are generally defined by (3.1) and (3.2),

$$b_k = \frac{2}{n}\sum_{j=1}^{n-1} x_j \sin\frac{\pi}{n} kj, \quad k = 1,2,...,n-1 \tag{3.1}$$

$$x_j = \sum_{k=1}^{n-1} b_k \sin\frac{\pi}{n} kj, \quad j = 1,2,...,n-1 \tag{3.2}$$

The subroutine obtains $\{n/2 \cdot b_k\}$ and $\{x_j\}$ which correspond to the left-hand side of (3.1) and (3.2), respectively, and the user has to scale the results, if necessary.
Use of the trigonometric function table:
When the subroutine is called successively for transforms of a fixed dimension, the trigonometric function table is calculated and created only once. Therefore when calling the subroutine subsequently, the contents of parameter TAB must be kept intact.
Even when the dimension differs, the trigonometric function table need not be changed. A new trigonometric function table entry can be made on an as-required basis.

- Example
  By inputting n sample points $\{x_j\}$, performing the transform by the subroutine and scaling the results, the discrete Fourier coefficients $\{b_k\}$ are obtained. Also by performing the inverse transform after that, $\{x_j\}$ are obtained.
  Here $n \le 512$.

```
C      **EXAMPLE**
       DIMENSION X(512),TAB(255)
C      SINE TRANSFORM
```

```
      READ(5,500) N,(X(I),I=1,N)
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,N)
      CALL FSINT(X,N,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
C     NORMALIZE
      CN=2.0/FLOAT(N)
      DO 10 K=1,N
   10 X(K)=X(K)*CN
      WRITE(6,602)
      WRITE(6,601)(X(I),I=1,N)
C     SINE INVERSE TRANSFORM
      CALL FSINT(X,N,TAB,ICON)
      IF(ICON.NE.0) GO TO 20
      WRITE(6,602)
      WRITE(6,601) (X(I),I = 1,N)
   20 WRITE(6,603) ICON
      STOP
  500 FORMAT(I5/(6F12.0))
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA')
  603 FORMAT('0',5X,'CONDITION CODE',
     *       I8)
      END
```

Method

The discrete sine transform based on the trapezoidal rule of dimension $n(=2^l,\ l=1, 2, ...)$ is performed by using the radix 2 Fast Fourier Transform (FFT).

The transform based on the trapezoidal rule can be accomplished efficiently by using the transform based on the midpoint rule. The subroutine uses this method.

Dividing equally into n_p (= 2, p = 1, 2, ...) the half period [0, π] of an odd function $x(t)$ with period 2π, the discrete sine transform based on the trapezoidal rule is defined as

$$b_k^p = \sum_{j=0}^{n_p-1} x\left(\frac{\pi}{n_p} j\right) \sin\frac{\pi}{n_p} kj, k = 1,2,...,n_p - 1 \qquad (4.1)$$

Also the discrete sine transform based on the midpoint rule of dimension n_p is defined as

$$\hat{b}_k^p = \sum_{j=0}^{n_p-1} x\left(\frac{\pi}{n_p} k\left(j+\frac{1}{2}\right)\right) \sin\frac{\pi}{n_p} k\left(j+\frac{1}{2}\right), \qquad (4.2)$$
$$k = 1,2,...,n_p$$

In Eqs. (4.1) and (4.2), however, the ordinary scaling factor $2/n_p$ is omitted.

Doubling the division number to n_{p+1} the $\{b_k^{p+1}\}$ can be expressed as follows by using $\{b_k^p\}$ and $\{\hat{b}_k^p\}$ both of which have half as many dimension as $\{b_k^{p+1}\}$.

$$\left.\begin{aligned} b_k^{p+1} &= \hat{b}_k^p + b_k^p \\ b_{n_{p+1}-k}^{p+1} &= \hat{b}_k^p + b_k^p \end{aligned}\right\} k = 1,2,...,n_p - 1 \qquad (4.3)$$
$$b_{n_p}^{p+1} = \hat{b}_{n_p}^p$$

The equations shown in (4.3) can be considered as recurrence formula with respect to p to obtain $\{b_k^{p+1}\}$ from $\{b_k^p\}$. Therefore, when performing transform with division number 2^l, if the initial condition $b_1^1 = x(\pi/2)$ is given and the discrete sine transform based on the midpoint rule of dimension n_p is performed each at the p-th stage (p=1, 2, ..., l-1) then the discrete sine transform series $\{b_k^p;\ k = 1, 2, ..., n_p\text{-}1\}$ (p = 1, 2, ..., l) based on the trapezoidal rule can be obtained by using equations (4.3).

The number of multiplications of real numbers executed is about $n\log_2 n$ ($n = 2^l$).

Procedural steps taken in the subroutine

Permuting the sample points, $x_0, x_1, ..., x_{n-1}$, in reverse binary order, they are denoted as $x(0), x(1), ..., x(n\text{-}1)$, where $x(0) = x_0 = 0$.

Next, the trigonometric function table (size n/2 -1) 1)necessary for the transform is made in reverse binary order corresponding to the sample point number.

Finishing with the preparatory processing, the sine transform is performed as follows:

(a) Initialization
$b_1^1 = x(1)$ and $p = 1$

(b) Sine transform based on the midpoint rule of dimension n_p
By inputting n_p sample points $x(n_p), x(n_{p+1}), ..., x(n_{p+1}\text{-}1)$ and performing the sine transform based on the midpoint rule, $\{b_k^p\}$ are obtained in the same area in the order $\hat{b}_{n_p}^p, \hat{b}_{n_{p-1}}^p, ..., \hat{b}_1^p$. For the sine transform based on the midpoint rule, see Method for subroutine FSINM.

(c) Calculation of recurrence equations with respect to $\{\hat{b}_k^p\}$
At the p-th stage, the intermediate results, $x(0), b_1^p, b_2^p, ..., b_{n_p-1}^p,$ $\hat{b}_{n_p}^p, ..., \hat{b}_1^p$ are stored in the array elements, $x(0)$, $x(1), ..., x(n_{p+1}\text{-}1)$. The $\{b_k^{p+1}\}$ are obtained by using only this area. That is, the two elements, b_k^{p+1} and $b_{n_{p+1}-k}^{p+1}$ at the (p+1)th stage are calculated from the two elements b_k^p and $\hat{b}_k^p\ (k = 1,2,...,n_p - 1)$ at the p-th stage, according to the recurrence equations (4.3), and they are stored in the corresponding array elements of the p-th stage.
By repeating procedure (b) and (c) above with p = 1, 2, ..., l-1 the $\{b_k\}$ can be obtained.
For further details, refer to the Reference [58].

B52-11-0101 GBSEG, DGBSEG

Eigenvalues and eigenvectors of a real symmetric band generalized eingeproblem (Jennings method)
CALL GBSEG (A, B, N, NH, M, EPSZ, EPST, LM, E, EV, K, IT, VW, ICON)

Function

This subroutine obtains *m* eigenevalues and corresponding eigenvectors of a generalized eigenproblem

$$\boldsymbol{Ax}=\lambda\boldsymbol{Bx} \qquad (1.1)$$

consisting of real symmetric matrices ***A*** and ***B*** of order *n* and bandwidth *h* in the ascending or descending order of absolute values by using *m* given initial vectors. It adopts the Jennings' simultaneous iteration method with the Jennings' acceleration. When starting with the largest or smallest absolute value, matrix ***B*** or ***A*** must be positivedefinite respectively. The eigenvectors is normalized such that:

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{BX}=\boldsymbol{I} \qquad (1.2)$$

or

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{AX}=\boldsymbol{I} \qquad (1.3)$$

$1 \le m << n$ and $0 \le h << n$

Parameters

A..... Input. Real symmetric band matrix ***A***.
When eigenvalues are obtained in the ascending order of absolute values, the contents of ***A*** are altered on output.
Compressed mode for symmetric band matrices.
A is a one-dimensional arrays of size *n* (*h*+1) − *h* (*h*+1)/2.

B..... Input. Real symmetric band matrix ***B***.
The contents of ***B*** are altered on output.
Compressed mode for symmetric band matrices.
B is a one-dimensional array of size *n* (*h*+1) − *h*(*h*+1)/2.
See "Comments on use".

N..... Input. Order *n* of matrices ***A*** and ***B***.

NH... Input. Bandwidth *h* of matrices ***A*** and ***B***.
See "Comments on use".

M.... Input. Number of eigenvalues and eigenvectors to be obtained, *m*.
M = *m* ... *m* eigenvalues are obtained in the descending order of absolute values.
M = −*m* .. *m* eigenvalues are obtained in the ascending order of absolute values.

EPSZ.. Input. Relative zero criterion for pivoting associated with LL$^{\mathrm{T}}$ decomposition of matrix ***A*** or ***B***.
If zero or negative value is given, an appropriate default value is used.
See "Comments on use".

EPST.. Input. Constant ε used for convergence criterion of eigenvectors. If zero or negative value is given, an appropriate default value is used.
See "Comments on use".

LM... Input. Upper limit for the number of iterations. If the number of iterations exceeds the limit, the processing is terminated.
See "Comments on use".

E.... Output. Eigenvalues. Each eigenvalue is stored in the sequence as specified by the M parameter.
E is a one-dimensional array of size *m*.

EV... Input. *m* initial vectors stored in the *m*-th columns (in columnwise direction).
See "Comments on use".
Output. Eigenvectors. Eigenvectors are stored in the *m*-th columns (in columnwise direction).
EV (K, *m*+2) is a two-dimensional array.

K..... Input. Adjustable dimension of EV.

IT... Output. Number of iterations which are made until eigenvectors are obtained.

VW... Work area. VW is a on-dimensional array of size 2*n*+*m*(3*m*+1)/2.

ICON.. Output. Condition code.
See Table GBSEG-1.

Table GBSEG-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The number of iterations exceeded upper limit LM.	Terminated. E and EV contain the approximations of eigenvalues and eigenvectors obtained so far.
25000	Orthogonalization of eigenvectors at each interation cannot be attained.	Discontinued.
28000	Matrix A or matrix B is not positive-definite.	Discontinued.
29000	Matrix A or matrix B is singular.	Discontinued.
30000	NH < 0, NH ≥ N, N > K, M = 0 or $\lvert M \rvert$ > N	Discontinued.

Comments on use

- Subprograms used
  SSL II ... AMACH, MSBV, TRID1, TEIG1, TRBK, UCHLS, UBCHL, UBCLX, UERST, MGSSL
  FORTRAN basic functions ... IABS, ABS, AMAX1, FLOAT, SQRT

- Notes

When eigenvalues are obtained in the ascending order of absolute values, the contents of matrix ***B*** are saved into array A. When this subroutine handles several generalized eigenproblems involving the identical matrix ***B***, it can utilize the contents of matrix ***B*** in array A.

The bandwidth of matrix ***A*** must be equal to that of matrix ***B***. If the bandwidth of matrix ***A*** is not equal to that of matrix ***B***, the greater band-width is assumed; therefore zeros are added to the matrix of smaller band-width as required.

The EPSZ default value is 16 *u* when *u* is the unit round off. When EPSZ contains 10^{-s} this subroutine regards the pivot as zero if the cancellation of over *s* significant digits occurs for the pivot during LL^T decomposition of matrix ***A*** or ***B***.
Then this subroutine sets a condition code ICON to 29000 and terminates abnormally. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value but the result is not always guaranteed. If the pivot becomes negative during LL^T decomposition of matrix ***A*** or ***B***, the matrix is regarded as singular and a condition code ICON to 28000. This subroutine terminates abnormally.

The parameter EPST is used to examine the convergence of eigenvector normalized as $\|\boldsymbol{x}\|_2 = 1$. Whenever an eigenvector converges for the convergence criterion constant ε, the corresponding eigenvalue converges at least with accuracy $\|\boldsymbol{A}\| \cdot \varepsilon$ and in most cases is higher. It is therefore better to choose somewhat a larger EPST value. When defining the unit round off as *u*, the default value is $\varepsilon = 16u$. When the eigenvalues are very close to each other, however, conveygence may not be attained. If so, it is safe to choose ε such that $\varepsilon \geq 100u$.

The upper Limit LM for the number of iteration is used to forcefully terminate the iteration when convergence is not attained. It should be set taking into consideration the required accuracy and how close the eigenvalues are to each other. The standard value is 500 to 1000.

It is desirable for the initial eigenvectors to be a good approximation to the eigenvectors corresponding to the obtained eigenvalues. If approximate vectors are not available, the standard way to choose initial vectors is to use the first *m* column vectors of the unit matrix ***I***. The number of eigenevalues and eigenvectors, *m* had better be smaller than *n* such that $m/n < 1/10$. The numbering of the eigenvalues is from the largest (or smallest) absolute value of eigenvalues such as λ_1, $\lambda_2, ..., \lambda_n$. It is desirable if possible, to choose *m* in such a way that $|\lambda_{m-1} / \lambda_m| << 1$ (or $|\lambda_{m+1} / \lambda_m| >> 1$) to achieve convergence faster.

- Example

This example obtains eigenvalues and corresponding eigenvectors of generalized eigenproblem
$\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$
consisting of real symmetric matrices ***A*** and ***B*** of order *n* and bandwidth *h*.
$n \leq 100$, $h \leq 10$ and $m \leq 10$.

```
C     **EXAMPLE**
      DIMENSION A(1100),B(1100),E(10),
     *          EV(100,12),VW(400)
   10 READ(5,500) N,NH,M,EPSZ,EPST
      IF(N.EQ.0) STOP
      MM=IABS(M)
      NN=(NH+1)*(N+N-NH)/2
      READ(5,510) (A(I),I=1,NN),
     *            (B(I),I=1,NN),
     * ((EV(I,J),I=1,N),J=1,MM)
      WRITE(6,600) N,NH,M
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=MIN0(NH+1,I)+NE
   20 WRITE (6,610) I,(A(J),J=NI,NE)
      WRITE(6,620) N,NH,M
      NE=0
      DO 30 I=1,N
      NI=NE+1
      NE=MIN0(NH+1,I)+NE
   30 WRITE(6,610) I,(B(J),J=NI,NE)
      CALL GBSEG(A,B,N,NH,M,EPSZ,EPST,500,
     *     E,EV,100,IT,VW,ICON)
      WRITE(6,630) ICON,IT
      IF(ICON.GE.20000) GO TO 10
      CALL SEPRT(E,EV,100,N,MM)
      GO TO 10
  500 FORMAT(3I5,2E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1',20X,'ORIGINAL MATRIX A',
     * 5X,'N=',I3,5X,'NH=',I3,5X,'M=',I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',20X,'ORIGINAL MATRIX B',
     * 5X,'N=',I3,5X,'NH=',I3,5X,'M=',I3/)
  630 FORMAT('0',20X,'ICON=',I5,
     * 5X,'IT=',I5)
      END
```

The SEPRT subroutine used in this example prints eigenvalues and eigenvectors of a real symmetric matrix. For further information see an example of using the subroutine SEIG1.

Method

This subroutine obtains *m* eigenvalues and corresponding eigenvectors of generalized eigenproblem.

$$\boldsymbol{Ax} = \lambda \boldsymbol{Bx} \qquad (4.1)$$

consisting of real symmetric band matrices ***A*** and ***B*** of order *n* and bandwidth *h* in the ascending of descending order of absolute values of the eigenvalues by using *m* given initial vectors. It adopts the Jennings' simultaneous iteration method with Jennings' acceleration.

For detailed information about the Jennings' simultaneous iteration method see the "Method" of the subroutine BSEGJ.

- Computational procedures

  If m eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ are to be determined in the ascending order of absolute values, (4.1) must be transformed as shown in (4.2)

$$\boldsymbol{Bx} = \frac{1}{\lambda}\boldsymbol{Ax} = \mu\boldsymbol{Ax} \tag{4.2}$$

  and m eigenvalues $\mu_1, \mu_2, ..., \mu_m$ can be determined by using the following:

$$\lambda_i = 1/\mu_i, \quad i = 1,...,m$$

  Therefore, the following explanation is concerned about only the case when m eigenvalues are to be determined in the descending order of absolute values.

  1) Matrix $\boldsymbol{B}$ is decomposed into LL^{T} by the subroutine UBCHL.

$$\boldsymbol{B} = \tilde{\boldsymbol{B}}\tilde{\boldsymbol{B}}^{\mathrm{T}} \tag{4.3}$$

     This procedure transforms a general eigenproblem shown in (4.1) to a standard eigenproblem such that:

$$\tilde{\boldsymbol{B}}^{-1}\boldsymbol{A}\tilde{\boldsymbol{B}}^{-\mathrm{T}}\boldsymbol{u} = \lambda\boldsymbol{u} \tag{4.4}$$

     where

$$\boldsymbol{u} = \tilde{\boldsymbol{B}}^{\mathrm{T}}\boldsymbol{x} \tag{4.5}$$

  2) Let m approximate eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_m$ be formed into m columns of $\boldsymbol{U}$ which is an $n \times m$ matrix.
     These vectors are assumed to be formed into an orthonormal matrix as shown in (4.6):

$$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{I_m} \tag{4.6}$$

     where $\boldsymbol{I_m}$ indicates an m-order unit matrix.
     By multiplying $\boldsymbol{U}$ by $\tilde{\boldsymbol{B}}^{-\mathrm{T}}$, $\boldsymbol{A}$ and $\tilde{\boldsymbol{B}}^{-1}$ in this order from the left,

$$\boldsymbol{V} = \tilde{\boldsymbol{B}}^{-1}\boldsymbol{A}\tilde{\boldsymbol{B}}^{-\mathrm{T}}\boldsymbol{U} \tag{4.7}$$

     results. Multiplying $\boldsymbol{V}$ by $\boldsymbol{U}^T$ from the left,

$$\boldsymbol{C} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{U}^{\mathrm{T}}\tilde{\boldsymbol{B}}^{-1}\boldsymbol{A}\tilde{\boldsymbol{B}}^{\mathrm{T}}\boldsymbol{U} \tag{4.8}$$

     results.
     This processing is actually performed in parallel to save storage as follows:
     For i=1, 2, ..., m (4.9) results by using auxiliary vector $\boldsymbol{y}$.

$$\begin{aligned}\boldsymbol{y} &= \boldsymbol{u_i} \\ \boldsymbol{v_i} &= \tilde{\boldsymbol{B}}^{-1}\boldsymbol{A}\tilde{\boldsymbol{B}}^{-\mathrm{T}}\boldsymbol{y}\end{aligned} \tag{4.9}$$

     This processing can be performed by using the subroutines UBCLX and MSBV. Since $\boldsymbol{C}$ is an m-dimensional symmetric matrix, its lower triangular portion can be denoted by:

$$c_{ii} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{v_i}, \tag{4.10}$$

$$c_{ji} = \boldsymbol{u_j}^{\mathrm{T}}\boldsymbol{v_i}, \quad j = i+1,...,m \tag{4.11}$$

     Thus, by taking $\boldsymbol{u_i}$ for each column of $\boldsymbol{U}$ and obtaining $\boldsymbol{v}_i$, c_{ii} and c_{ji}, $\boldsymbol{V}$ is produced directly in the area for $\boldsymbol{U}$.

  3) Solving the eigenproblem for $\boldsymbol{C}$, $\boldsymbol{C}$ is decomposed to the form

$$\boldsymbol{C} = \boldsymbol{PMP}^{\mathrm{T}} \tag{4.12}$$

     where $\boldsymbol{M}$ is a diagonal matrix using eigenvalues of $\boldsymbol{C}$ as diagonal elements and $\boldsymbol{P}$ is a m-dimensional orthogonal matrix.
     This processing is performed by using the subroutines TRID1, TEIG1 and TRBK, which are called successively. Then the eigenvectors corresponding to the eigenvalues are sorted such that the largest absolute values comes first using the subroutine UESRT.

  4) Multiply $\boldsymbol{V}$ by $\boldsymbol{P}$ form the right, and produce an $n \times m$ matrix $\boldsymbol{W}$ shown below:

$$\boldsymbol{W} = \boldsymbol{VP} \tag{4.13}$$

  5) To orthogonalize each column of $\boldsymbol{W}$, produce an m - dimensional real symmetric positive-definite matrix $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W}$ and decompose it into LL^{T} shown below:

$$\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{LL}^{\mathrm{T}} \tag{4.14}$$

     This processing is performed by using the subroutine UCHLS.
     If LL^{T} decomposition is not possible this subroutine sets a condition code ICON to 25000 and terminates abnormally.

  6) Solve the equation $\boldsymbol{U}^*\boldsymbol{L}^{\mathrm{T}} = \boldsymbol{W}$ to compute

$$\boldsymbol{U}^* = \boldsymbol{WL}^{-\mathrm{T}} \tag{4.15}$$

     $\boldsymbol{U}^*$ has an orthonormal system as in equation (4.6). The m-th columns $\boldsymbol{u}_m$ and $\boldsymbol{u}_m^*$ of $\boldsymbol{U}$ and $\boldsymbol{U}^*$ are examined to see if

$$d = \left\|\boldsymbol{u}_m^* - \boldsymbol{u}_m\right\|_\infty \le \varepsilon \tag{4.16}$$

is satisfied.
If this convergence condition is not satisfied, see $\boldsymbol{U}^*$ as a new $\boldsymbol{U}$ and go to Step 2).

7) If it is satisfied, the iteration is stopped and the diagonal elements of $\boldsymbol{M}$ obtained in Step 3) become eigenvalues, and the first m row of

$$\boldsymbol{X} = \tilde{\boldsymbol{B}}^{-\mathrm{T}}\boldsymbol{U}^* \qquad (4.17)$$

become the corresponding eigenvectors. This processing is performed by using the subroutine UBCLX.
These steps given above are a general description of the Jennings's method.
For further information see References [18] and [19].

- Jennings' acceleration
To accelerate the simultaneous Jennings' iteration method previously explained, the Jennings' acceleration method for vector series in incorporated in this subroutine. See explanations about subroutine BSEGJ for the principle and application of Jennings' acceleration.

E51-30-0301 GCHEB, DGCHEB

Differentiation of a Chebyshev series
CALL GCHEB (A, B, C, N, ICON)

Function

Given an n-terms Chebyshev series defined on the interval $[a, b]$

$$f(x)=\sum_{k=0}^{n-1}{}' c_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \qquad (1.1)$$

this subroutine computes its derivative in a Chebyshev series

$$f'(x)=\sum_{k=0}^{n-2}{}' c'_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \qquad (1.2)$$

and determines its coefficients $\{c'_k\}$.

Symbol Σ' denotes to make sum but the initial term only is multipled by factor 1/2.

Where $a \neq b$ and $n \geq 1$.

Parameters

A..... Input. Lower limit a of the interval for the Chebyshev series.

B..... Input. Upper limit b of the interval for the Chebyshev series.

C..... Input. Coefficients$\{c_k\}$.
Each coefficient is stored as shown below:
C(1) $=c_0$, C(2) $=c_1$, ..., C(N) $=c_{n-1}$
Output. Coefficients $\{c'_k\}$ for the derivative.
Each coefficient is stored as shown below:
C(1) $=c'_0$, C(2) $=c'_1$, ..., C(N-1) $=c'_{n-2}$
C is a one-dimensional array of size N.

N..... Input. Number of terms n.
Output. Number of terms of the derivative n–1.
See Notes.

ICON.. Output. Condition code.
See Table GCHEB-1.

Table GCHEB-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the two conditions occurred: 1 N < 1 2 A = B	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... FLOAT

- Notes
  When a derivative of an arbitrary function is required, this subroutine can be used together with the subroutine FCHEB for Chebyshev series expansion.
  When a differential coefficient is determined at arbitrary point, this subroutine should be used together with the subroutine ECHEB for evaluation of the Chebyshev series. See Example.
  This subroutine can be called repeatedly to compute a derivative of higher order.
  The error of a derivative can be estimated from the absolute sum of the last two terms. Note that the error of a derivative increases as order increases.
  If only one term is entered this subroutine produces only one term.

- Example
  This example expands exponential function

$$f(x)=e^x\left(=\sum_{n=0}^{\infty}\frac{x^n}{n!}\right)$$

defined on the interval [-2, 2] in a Chebyshev series according to the required accuracy of $\varepsilon_a = 0$ and $\varepsilon_r = 5 \cdot 10^{-5}$ by using the subroutine FCHEB. It then computes the derivative from the resultant Chebyshev series by using this subroutine. It also evaluates differential coefficients by using the ECHEB subroutine while increasing the value of x from –2 to 2 with increment 0.05 and compares them with the true values.

```
C      **EXAMPLE**
       DIMENSION C(257),TAB(127)
       EXTERNAL FUN
       EPSR=5.0E-5
       EPSA=0.0
       NMIN=9
       NMAX=257
       A=-2.0
       B=2.0
       CALL FCHEB(A,B,FUN,EPSA,EPSR,NMIN,
      *           NMAX,C,N,ERR,TAB,ICON)
       IF(ICON.NE.0) GOTO 20
       WRITE(6,600) N,ERR,ICON
       WRITE(6,601) (C(K),K=1,N)
       CALL GCHEB(A,B,C,N,ICON)
       IF(ICON.NE.0) GOTO 20
       WRITE(6,602)
       WRITE(6,601) (C(K),K=1,N)
       WRITE(6,603)
       H=0.05
       X=A
    10 CALL ECHEB(A,B,C,N,X,Y,ICON)
       IF(ICON.NE.0) GOTO 20
       ERROR=FUN(X)-Y
       WRITE(6,604) X,Y,ERROR
       X=X+H
       IF(X.LE.B) GOTO 10
       STOP
    20 WRITE(6,605) ICON
       STOP
```

```
600 FORMAT('0',3X,'EXPANSION OF',
   1' FUNCTION FUN(X)',3X,'N=',I4,3X,
   2'ERROR=',E13.3,3X,'ICON=',I6)
601 FORMAT(/(5E15.5))
602 FORMAT('0',5X,'DERIVATIVE OF',
   1' CHEBYSHEV SERIES')
603 FORMAT('0',10X,'X',7X,
   1'DIFFERENTIAL',6X,'ERROR'/)
604 FORMAT(1X,3E15.5)
605 FORMAT('0',5X,'CONDITION CODE',I8)
    END
    FUNCTION FUN(X)
    REAL*8 SUM,XP,TERM
    EPS=AMACH(EPS)
    SUM=1.0
    XP=X
    XN=1.0
    N=1
 10 TERM=XP/XN
    SUM=SUM+TERM
    IF(DABS(TERM).LE.
   1   DABS(SUM)*EPS) GOTO 20
    N=N+1
    XP=XP*X
    XN=XN*FLOAT(N)
    GOTO 10
 20 FUN=SUM
    RETURN
    END
```

Method

This subroutine performs termwise differentiation of an *n*-terms Chebyshev series defined on the interval [*a*, *b*] and expresses its derivative in a Chebyshev series. Let a derivative to be defined as follow:

$$\frac{d}{dx}\sum_{k=0}^{n-1}{}' c_k T_k\left(\frac{2x-(b+a)}{b-a}\right)=\sum_{k=0}^{n-2}{}' c'_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \tag{4.1}$$

The following relationships exist between coefficients for the derivative:

$$\begin{aligned} c'_{n-1} &= 0 \\ c'_{n-2} &= \left(\frac{4}{b-a}\right)(n-1)c_{n-1} \\ c'_{k-2} &= \left(\frac{4}{b-a}\right)kc_k + c'_{k+1}, \\ &\quad k = n-2, n-3, \ldots, 1 \end{aligned} \tag{4.2}$$

This subroutine determines coefficients { c'_k } by using differential formula (4.2) for Chebyshev polynomials. The number of multiplications required to compute a derivative from an *n*-terms series is about 2*n*.

For further information about recurrence formula (4.2), see explanation of the subroutine ICHEB.

A25-31-0101 GINV, DGINV

Moore-Penrose generalized inverse of a real matrix (the singular value decomposition method)
CALL GINV (A, KA, M, N, SIG, V, KV, EPS, VW, ICON)

Function

This subroutine obtains the generalized inverse $\boldsymbol{A}^+$ of an $m \times n$ matrix $\boldsymbol{A}$ using the singular value decomposition method.

$m \geq 1$ and $n \geq 1$.

Parameters

A.....	Input. Matrix $\boldsymbol{A}$. Output. Transposed matrix of $\boldsymbol{A}^+$. A is a two-dimensional array A (KA, N). See Notes.
KA....	Input. Adjustable dimension of array A. KA ≥ M.
M.....	Input. Number of rows in matrix $\boldsymbol{A}$, m.
N.....	Input. Number of columns in matrix $\boldsymbol{A}$ or number of rows in matrix $\boldsymbol{V}$, n.
SIG...	Output. Singular values of matrix $\boldsymbol{A}$. One-dimensional array of size n. See Notes.
V.....	Output. Orthogonal transformation matrix produced by the singular value decomposition. V is a two-dimensional array V (KV, K). K = min (M + 1, N).
KV.....	Input. Adjustable dimension of array V. KV ≥ N.
EPS...	Input. Tolerance for relative zero test of the singular value. EPS ≥ 0.0 If EPS is 0.0 a standard value is used. See Notes.
VW.....	Work area. VW is a one-dimensional array of size n.
ICON..	Output. Condition code. See Table GINV-1.

Table GINV-1 Condition codes

Code	Meaning	Processing
0	No error	
15000	Any singular value could not be obtained.	Discontinued
30000	KA < M, M < 1, N < 1, KV < N or EPS < 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II ... ASVD1, AMACH, MGSSL
  FORTRAN basic function ... None

- Notes
  Note that the transposed matrix $(\boldsymbol{A}^+)^T$ instead of the generalized inverse $\boldsymbol{A}^+$ is placed on A.
  Singular values are non-negative. They are stored in descending order.
  When ICON is set to 15000, unobtained singular values are -1. In this case, resultant singular values aren't arranged in descending order.
  Since the EPS has direct effects on the determination of the rank of $\boldsymbol{A}$, it must be specified carefully.
  The least squares minimal norm solution of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ can be expressed as $\boldsymbol{X} = \boldsymbol{A}^+ \boldsymbol{b}$ by using the generalized inverse $\boldsymbol{A}^+$. However this subroutine should not be used except when generalized inverse $\boldsymbol{A}^+$ is required. The subroutine LAXLM is provided by SSL II for this purpose.

- Example
  This example obtains a generalized inverse of an $m \times n$ real matrix.
  $1 \leq n \leq m \leq 100$

```
C     **EXAMPLE**
      DIMENSION A(100,100),SIG(100),
     *          V(100,100),VW(100)
   10 READ(5,500) M,N
      IF(M.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,M),J=1,N)
      WRITE(6,600) M,N,
     *  ((I,J,A(I,J),J=1,N),I=1,M)
      CALL GINV(A,100,M,N,SIG,V,100,0.0,
     *          VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) GO TO 10
      WRITE(6,620) N,M,
     *  ((I,J,A(J,I),I=1,M),J=1,N)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX'/
     *  6X,'ROW NUMBER=',I4,5X,
     *  'COLUMN NUMBER=',I4/
     *  (10X,4('(',I3,',',I3,')',E17.7,
     *  3X)))
  610 FORMAT(' ',10X,'CONDITION CODE='
     *  ,I6)
  620 FORMAT('1',5X,
     *  'GENERALIZED INVERSE'/6X,
     *  'ROW NUMBER=',I4,5X,
     *  'COLUMN NUMBER=',I4/(10X,
     *  4('(',I3,',',I3,')',E17.7,3X)))
      END
```

Method

The $n \times m$ matrix satisfying all conditions listed in (4.1) with regard to an $m \times n$ real matrix $\boldsymbol{A}$ is called (Moore-Penrose matrix) the generalized inverse of $\boldsymbol{A}$.

$$\left.\begin{aligned} \boldsymbol{AXA} &= \boldsymbol{A} \\ \boldsymbol{XAX} &= \boldsymbol{X} \\ (\boldsymbol{AX})^{\mathrm{T}} &= \boldsymbol{AX} \\ (\boldsymbol{XA})^{\mathrm{T}} &= \boldsymbol{XA} \end{aligned}\right\} \tag{4.1}$$

A generalized inverse is uniquely defined for a real matrix $\boldsymbol{A}$ and is denoted by $\boldsymbol{A}^+$. If matrix $\boldsymbol{A}$ is a square and nonsingular matrix, $\boldsymbol{A}^+$ is equal to $\boldsymbol{A}^-$. Based on the uniqueness of generalized inverse (4.2) and (4.3) can be established from (4.1).

$$\left(\boldsymbol{A}^+\right)^+ = \boldsymbol{A}\,, \tag{4.2}$$

$$\left(\boldsymbol{A}^{\mathrm{T}}\right)^+ = \left(\boldsymbol{A}^+\right)^{\mathrm{T}} \tag{4.3}$$

Therefore $m \ge n$ can be assumed without loss of generality.

- Singular value decomposition and a generalized inverse
  Let the singular value decomposition of $\boldsymbol{A}$ be defined as follows:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{\mathrm{T}} \tag{4.4}$$

When $\boldsymbol{U}$ is an $m \times n$ matrix satisfying $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{I}$, $\boldsymbol{V}$ is an $n \times n$ orthogonal matrix and $\boldsymbol{\Sigma}$ is an $n \times n$ diagonal matrix.

By adding m–n column vectors to the right of $\boldsymbol{U}$ to produce an orthogonal matrix $\boldsymbol{U}_c$ of order m and adding a zero matrix with the $(m–n)$-th row, the n-th column to the lower portion of $\boldsymbol{\Sigma}$ to obtain a matrix $\boldsymbol{\Sigma}_c$, then the singular value decomposition of $\boldsymbol{A}$ can be expressed as follows:

$$\boldsymbol{A} = \boldsymbol{U}_c\boldsymbol{\Sigma}_c\boldsymbol{V}^{\mathrm{T}} \tag{4.5}$$

If $\boldsymbol{\Sigma}$ is determined as follows:

$$\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, ..., \sigma_n) \tag{4.6}$$

then an $n \times n$ diagonal matrix $\boldsymbol{\Sigma}^+$ is expressed as shown in (4.7)

$$\boldsymbol{\Sigma}^+ = \mathrm{diag}\left(\sigma_1{}^+, \sigma_2{}^+, ..., \sigma_n{}^+\right) \tag{4.7}$$

where,

$$\sigma_i{}^+ = \begin{cases} 1/\sigma_i & ,\sigma_i > 0 \\ 0 & ,\sigma_i = 0 \end{cases} \tag{4.8}$$

The matrix obtained by adding an $n \times (m\text{-}n)$ zero matrix to the right of $\boldsymbol{\Sigma}^+$ is equal to $\boldsymbol{\Sigma}_c^+$.

Let $\boldsymbol{X} = \boldsymbol{V\Sigma}_c^+\boldsymbol{U}_c^{\mathrm{T}}$.

Substituting such a $\boldsymbol{X}$ and (4.5) into (4.1), (4.1) is repressed as shown in (4.9) due to orthogonality of $\boldsymbol{U_c}$ and $\boldsymbol{V}$.

$$\left.\begin{aligned} \boldsymbol{\Sigma}_c\boldsymbol{\Sigma}_c^+\boldsymbol{\Sigma}_c &= \boldsymbol{\Sigma}_c \\ \boldsymbol{\Sigma}_c^+\boldsymbol{\Sigma}_c\boldsymbol{\Sigma}_c^+ &= \boldsymbol{\Sigma}_c^+ \\ \left(\boldsymbol{\Sigma}_c\boldsymbol{\Sigma}_c^+\right)^{\mathrm{T}} &= \boldsymbol{\Sigma}_c\boldsymbol{\Sigma}_c^+ \\ \left(\boldsymbol{\Sigma}_c^+\boldsymbol{\Sigma}_c\right)^{\mathrm{T}} &= \boldsymbol{\Sigma}_c^+\boldsymbol{\Sigma}_c \end{aligned}\right\} \tag{4.9}$$

Thus, due to the uniqueness of the generalized inverse, $\boldsymbol{A}^+$ can be defined as

$$\boldsymbol{A}^+ = \boldsymbol{V\Sigma}_c^+\boldsymbol{U}_c^{\mathrm{T}} \tag{4.10}$$

By using the characteristics of $\boldsymbol{\Sigma}_c^+$ and definitions of $\boldsymbol{U_c}$, (4.10) can be rewritten as follows:

$$\boldsymbol{A}^+ = \boldsymbol{V\Sigma}^+\boldsymbol{U}^{\mathrm{T}} \tag{4.11}$$

- Computational procedures
  This subroutine determines matrices $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$ by performing the singular value decomposition of $\boldsymbol{A}$.
  The resultant matrix $\boldsymbol{U}$ is placed in the area containing matrix $\boldsymbol{A}$ by subroutine ASVD1. For detailed information, see an explanation of subroutine ASVD1.
  A transposed matrix $(\boldsymbol{A}^+)^{\mathrm{T}}$ corresponding to $\boldsymbol{A}^+$ is placed in the area containing matrix $\boldsymbol{A}$ through use of

$$\left(\boldsymbol{A}^+\right)^{\mathrm{T}} = \boldsymbol{U\Sigma}^+\boldsymbol{V}^{\mathrm{T}} \tag{4.12}$$

This subroutine tests the zero criterion for singular value σ_i when producting $\boldsymbol{\Sigma}^+$.

The zero criterion is σ_1 EPS, where σ_1 is the maximum singular value.

If a singular value is less than the zero criterion, it is regarded as zero. If EPS contains 0.0, this subroutine assumes a value of 16u as the standard value, where u is the unit round off.

For further information, see Reference [11] and an explanation of the subroutine ASVD1.

B22-10-0402 GSBK, DGSBK

Back transformation of the eigenvectors of the standard form to the eigenvectors of the real symmetric generalized eigenproblem
CALL GSBK (EV, K, N, M, B, ICON)

Function

m number of eigenvectors $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_m$, of *n* order real symmetric matrix $\boldsymbol{S}$ are back transformed to eigenvectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$ for the generalized eigenproblem $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$, where $\boldsymbol{S}$ is a matrix given by

$$\boldsymbol{S}=\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-\mathrm{T}} \quad (1.1)$$

$\boldsymbol{B} = \boldsymbol{LL}^{\mathrm{T}}$, $\boldsymbol{L}$ is a lower triangular matrix and $n \geq 1$.

Parameters

EV.....	Input. *m* number of eigenvectors of real symmetric matrix $\boldsymbol{S}$. Output. Eigenvectors for the generalized eigenproblem $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$. Two dimensional array, EV (K, M) (See "Comments on use").
K.....	Input. Adjustable dimension of input EV.
N.....	Input. Order *n* of real symmetric matrix $\boldsymbol{S}$, $\boldsymbol{A}$ and $\boldsymbol{B}$.
M.....	Input. The number of eigenvalue, *m* (See "Comments on use").
B.....	Input. Lower triangular matrix $\boldsymbol{L}$. (See Fig. GSBK-1). One dimensional array of size $n(n+1)/2$ (See "Comments on use").
ICON..	Output. Condition code. See Table GSBK-1.

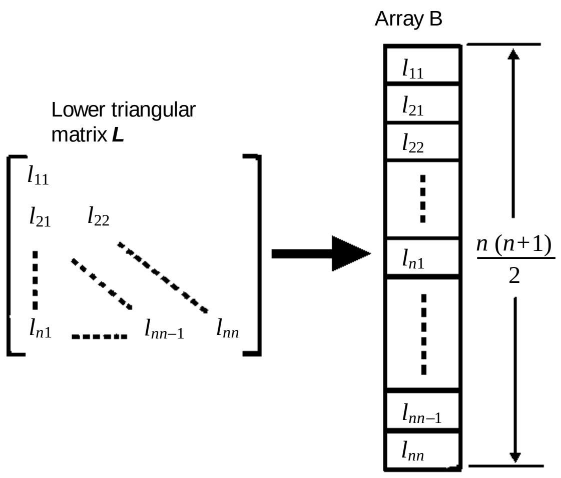
Table GSBK-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	EV (1, 1) = 1.0 / B(1)
30000	N < |M|, K < N or M = 0	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... IABS

- Notes
  Output parameter B of subroutine GSCHL can be used as input parameter B of this subroutine.
  If input vectors $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_m$ are normalized in such a way that $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y} = \boldsymbol{I}$ is satisfied, then eigenvectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$ are output in such a way that $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{BX} = \boldsymbol{I}$ is satisfied, where $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_m]$ and $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m]$.
  When parameter M is negative, its absolute value is used.



Note: Lower triangular matrix $\boldsymbol{L}$ must be stored into array B in the compressed storage mode for a symmetric matrix.

Fig. GSBK-1 Correspondence between matrix $\boldsymbol{L}$ and array B

- Example
  All eigenvalues and corresponding eigenvectors of the generalized eigenproblem with *n* order real symmetric matrix $\boldsymbol{A}$ and *n* order positive definite symmetric matrix $\boldsymbol{B}$ are obtained using subroutines GSCHL, TRID1, TEIG1, TRBK and GSBK. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),
     *          SD(100),E(100),
     *EV(100,100),D(100)
   10 READ(5,500) N,M,EPSZ,EPST
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      READ(5,510) (B(I),I=1,NN)
      WRITE(6,600) N,M,EPSZ,EPST
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      WRITE(6,620)
      NE=0
      DO 30 I=1,N
      NI=NE+1
      NE=NE+I
   30 WRITE(6,610) I,(B(J),J=1,NI,NE)
      CALL GSCHL(A,B,N,EPSZ,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL TRID1(A,N,D,SD,ICON)
      CALL TEIG1(D,SD,N,E,EV,100,M,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL TRBK(EV,100,N,M,A,ICON)
      CALL GSBK(EV,100,N,M,B,ICON)
      MM=IABS(M)
      CALL SEPRT(E,EV,100,N,MM)
      GO TO 10
```

```
500 FORMAT(2I5,2E15.7)
510 FORMAT(5E15.7)
600 FORMAT('1',10X,'**ORIGINAL ',
   * 'MATRIX A**',11X,'** ORDER =',I5,
   * 10X,'** M =',I3/46X,'EPSZ=',E15.7,
   * 'EPST=',E15.7)
610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
620 FORMAT('1',10X,'**ORIGINAL ',
   * 'MATRIX B**')
630 FORMAT(/11X,'** CONDITION CODE =',I5/)
    END
```

Subroutine SEPRT in this example is used to print eigenvalues and eigenvectors of a real symmetric matrix. For details, refer to the example of subroutine SEIG1.

Method

Eigenvector $\boldsymbol{y}$ of n order real symmetric matrix $\boldsymbol{S}$ is back transformed to eigenvector $\boldsymbol{x}$ of the generalized eigenvalue problem.

$$\boldsymbol{Ax} = \lambda\boldsymbol{Bx} \tag{4.1}$$

where $\boldsymbol{A}$ is a symmetric matrix and $\boldsymbol{B}$ is a positive definite symmetric matrix. In this case, reduction of eq. (4.1) to the standard form (4.2) must be done in advance.

$$\boldsymbol{Sy} = \lambda\boldsymbol{y} \tag{4.2}$$

This process is shown in eqs. (4.3) to (4.6). Decomposing positive symmetric matrix $\boldsymbol{B}$ into

$$\boldsymbol{B} = \boldsymbol{LL}^{\mathrm{T}} \tag{4.3}$$

and substituting this into eq. (4.1), we get

$$\begin{aligned}
&\boldsymbol{Ax} = \lambda\boldsymbol{LL}^{\mathrm{T}}\boldsymbol{x}\\
&\boldsymbol{L}^{-1}\boldsymbol{Ax} = \lambda\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\\
&\boldsymbol{L}^{-1}\boldsymbol{A}\left(\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}\right)\boldsymbol{x} = \lambda\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\\
&\boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}}\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right) = \lambda\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right)
\end{aligned} \tag{4.4}$$

therefore,

$$\boldsymbol{S} = \boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}} \tag{4.5}$$
$$\boldsymbol{y} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} \tag{4.6}$$

Since $\boldsymbol{L}$ is known, $\boldsymbol{x}$ can be obtained from eq. (4.6) as follows:

$$\boldsymbol{x} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{y} \tag{4.7}$$

For details see Reference [13] pp. 303-314.

B22-21-0302 GSCHL, DGSCHL

Reduction of a real symmetric matrix system $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$ to a standard form
CALL GSCHL (A, B, N, EPSZ, ICON)

Function

For n order real symmetric matrix $\boldsymbol{A}$ and n order positive definite symmetric matrix $\boldsymbol{B}$, the generalized eigenvalue problem

$$\boldsymbol{Ax} = \lambda\boldsymbol{Bx} \qquad (1.1)$$

is reduced to the standard form.

$$\boldsymbol{Sy} = \lambda\boldsymbol{y} \qquad (1.2)$$

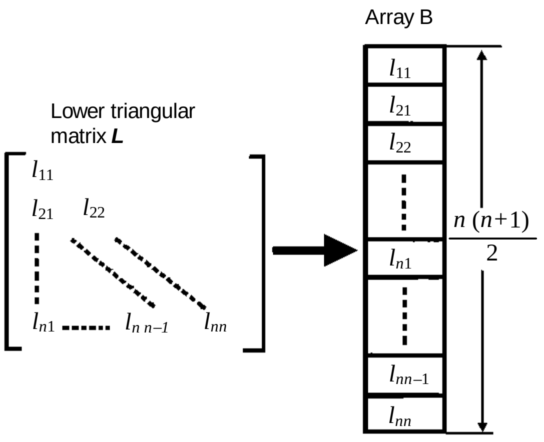
where $\boldsymbol{S}$ is a real symmetric matrix and $n \geq 1$.

Parameter

A.....	Input. Real symmetric matrix $\boldsymbol{A}$. Output. Real symmetric matrix $\boldsymbol{S}$. In the compressed storage mode for a symmetric matrix. One dimensional array of size $n(n+1)/2$.
B.....	Input. Positive definite symmetric matrix $\boldsymbol{B}$. Output. Lower triangular matrix $\boldsymbol{L}$ (See Fig. GSCHL-1). In the compressed storage mode for a symmetric matrix. One dimensional array of size $n(n+1)/2$.
N.....	Input. The order n of the matrices.
EPSZ..	Input. A tolerance for relative accuracy test of pivots in LL^{T} decomposition of $\boldsymbol{B}$. When specified 0.0 or negative value, a default value is taken (See "Comments on use").
ICON..	Output. Condition code. See Table GSCHL-1.

Table GSCHL-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	A(1) = A (1) / B(1) B(1)=SQRT (B(1))
28000	Pivot became negative in LL^{T} decomposition of matrix $\boldsymbol{B}$. Input matrix $\boldsymbol{B}$ is not positive-definite.	Bypassed
29000	Pivot was regarded as relatively zero in LL^{T} decomposition of matrix $\boldsymbol{B}$. The input matrix $\boldsymbol{B}$ is possible singular.	Bypassed
30000	N < 1	Bypassed



Note: The lower triangular portion of matrix $\boldsymbol{L}$ is stored into one-dimensional array B in the compressed storage mode for a symmetric matrix.

Fig. GSCHL-1 Correspondence between matrix $\boldsymbol{L}$ and array B

Comments on use

- Subprograms used
  SSL II ... AMACH, UCHLS, and MGSSL
  FORTRAN basic function ... SQRT

- Note
  The default value for parameter EPSZ is represented as EPSZ = $16 \cdot u$ where u is the unit round-off. (Refer to subroutine LSX)
  If EPSZ is set to 10^{-s}, when a pivot has cancellation of more than s decimal digits in LL^{T} decomposition of positive definite symmetric matrix $\boldsymbol{B}$, the subroutine considers the pivot to be relative zero, sets condition code ICON to 29000 and terminates the processing.
  To continue the processing even when the pivot becomes smaller, set a very small value into EPSZ.
  When the pivot becomes negative in LL^{T} decomposition of $\boldsymbol{B}$, $\boldsymbol{B}$ is considered not to be a positive definite matrix. This subroutine, in this case, sets condition code ICON to 28000 and terminates the processing.

- Example
  The generalized eigenvalue problem $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$ with n order real symmetric matrix $\boldsymbol{A}$ and n order positive definite symmetric matrix $\boldsymbol{B}$ is reduced to a standard form for the eigenvalue problem using the subroutine GSCHL, and the standard form is reduced to a real symmetric tridiagonal matrix using subroutine TRID1. After that, the m number of eigenvalues are obtained by using the subroutine BSCT1. This is for $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),VW(300),
     *     E(100),D(100),SD(100)
   10 CONTINUE
      READ(5,500) N,M,EPSZ,EPST
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      READ(5,510) (B(I),I=1,NN)
      WRITE(6,600) N,M,EPSZ,EPST
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      WRITE(6,620)
      NE=0
      DO 30 I=1,N
      NI=NE+1
      NE=NE+I
   30 WRITE(6,610) I,(B(J),J=1,NI,NE)
      CALL GSCHL(A,B,N,EPSZ,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL TRID1(A,N,D,SD,ICON)
      CALL BSCT1(D,SD,N,M,EPST,E,VW,ICON)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,640)
      MM=IABS(M)
      WRITE(6,650) (I,E(I),I=1,MM)
      GO TO 10
  500 FORMAT(2I5,2E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX A'/
     *11X,'** ORDER =',I5,10X,'** M =',I3,
     *10X,'** EPSZ =',E15.7,10X,'EPST =',
     *E15.7)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',10X,'** ORIGINAL MATRIX B')
  630 FORMAT(/11X,'** CONDITION CODE =',I5/)
  640 FORMAT('0'/11X,'** EIGENVALUES')
  650 FORMAT(5X,'E(',I3,')=',E15.7)
      END
```

Method

When $\boldsymbol{A}$ is an n order symmetric matrix and $\boldsymbol{B}$ is a positive definite symmetric matrix, the generalized engenvalue problem

$$\boldsymbol{Ax} = \lambda \boldsymbol{Bx} \tag{4.1}$$

is reduced to the standard form

$$\boldsymbol{Sy} = \lambda \boldsymbol{y} \tag{4.2}$$

as shown in eqs. (4.3) to (4.6)

Since $\boldsymbol{B}$ is a positive definite symmetric matrix, it can be decomposed to

$$\boldsymbol{B} = \boldsymbol{LL}^{\mathrm{T}} \tag{4.3}$$

where $\boldsymbol{L}$ is a lower triangular matrix. This decomposition can be uniquely determined if the diagonal elements of $\boldsymbol{L}$ are chosen positive. From (4.1) and (4.3)

$$\boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}}\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right) = \lambda\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right) \tag{4.4}$$

where $\boldsymbol{L}^{-\mathrm{T}}$ means $(\boldsymbol{L}^{-1})^{\mathrm{T}}$ or $(\boldsymbol{L}^{\mathrm{T}})^{-1}$

Therefore, putting

$$\boldsymbol{S} = \boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}} \tag{4.5}$$

$$\boldsymbol{y} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} \tag{4.6}$$

then eq. (4.2) can be derived.

The decomposition in (4.3) is done by the Cholesky method, that is , the elements of matrix $\boldsymbol{L}$ are successively obtained for each row as shown in eqs. (4.7) and (4.8)

$$l_{11} = (b_{11})^{\frac{1}{2}} \tag{4.7}$$

$$\left.\begin{aligned} l_{ij} &= \left(b_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}\right)\Big/ l_{jj}\ ,\ j = 1,\ldots,i-1 \\ l_{ii} &= \left(b_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)^{\frac{1}{2}} \end{aligned}\right\} i = 2,\ldots,n \tag{4.8}$$

where $\boldsymbol{L} = (l_{ij})$, $\boldsymbol{B} = (b_{ij})$

For details, see Reference [13] PP.303-314.

B22-21-0201 GSEG2, DGSEG2

Eigenvalues and corresponding eigenvectors of a real symmetric generalized eigenproblem $\boldsymbol{Ax}=\lambda\boldsymbol{Bx}$ (bisection method and inversed iteration method)
CALL GSEG2 (A, B, N, M, EPSZ, EPST, E, EV, K, VW, ICON)

Function

The *m* largest or *m* smallest eigenvalues of the generalized eigenvalue problem

$$\boldsymbol{Ax}=\lambda\boldsymbol{Bx} \tag{1.1}$$

are obtained by bisection method, and the corresponding *m* number of eigenvectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$ are obtained by the inverse iteration method, where $\boldsymbol{A}$ is an *n* order real symmetric matrix and $\boldsymbol{B}$ an *n* order positive definite symmetric matrix. The eigenvectors satisfy.

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{BX}=\boldsymbol{I} \tag{1.2}$$

where $\boldsymbol{X}=[\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m]$ and $n \geq m \geq 1$.

Parameters

A.....	Input. Real symmetric matrix $\boldsymbol{A}$ in the compressed mode storage for a symmetric matrix. One dimensional array of size $n(n+1)/2$. The contents will be altered on output.
B.....	Input. Positive definite matrix $\boldsymbol{B}$ in the compressed storage mode for a symmetric matrix. One dimensional array of size $n(n+1)/2$. The contents will be altered on output.
N.....	Input. Order *n* of real symmetric matrix $\boldsymbol{A}$ and positive definite symmetric matrix $\boldsymbol{B}$.
M.....	Input. The number *m* of eigenvalues obtained. M = + *m* .... The number of largest eigenvalues desired. M = – *m* .... The number of smallest eigenvalues desired.
EPSZ..	Input. A tolerance for relative accuracy test of pivots in LL$^{\mathrm{T}}$ decomposition of $\boldsymbol{B}$. If specifying 0.0 or negative value, a default value is taken. (See "Comments on use").
EPST...	Input. An absolute error tolerance used as a convergence criterion for eigenvalues. If specifying 0.0 or negative value, a default value is taken. (See "Comments on use").
E.....	Output. *m* eigenvalues, Stored in the sequences as specified by parameter M. One dimensional array of size *m*.
EV.....	Output. Eigenvectors. The eigenvectors are stored in columnwise direction. Two dimensional array, EV (K, M)
K....	Input. Adjustable dimension of array EV.
VW....	Work area. One dimensional array of size 7*n*.
ICON..	Output. Condition code. See Table GSEG2-1.

Table GSEG2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	E(1)=A(1)/B(1) X(1, 1)=1.0/SQRT (B(1))
15000	Some eigenvectors could not be determined.	The eigenvectors are set to zero vectors.
20000	Eigenvectors could not be obtained.	All the eigenvectors are set to zero vectors.
28000	Pivot became negative in LL$^{\mathrm{T}}$ decomposition of *B*. Input matrix *B* is not positive-definite.	Discontinued.
29000	Pivot was regarded as relative zero in LL$^{\mathrm{T}}$ decomposition of *B*. Input matrix *B* is possible singular.	Discontinued.
30000	M = 0 or N < IMI or K < N	Bypassed.

Comments on use

- Subprogram used
  SSL II ... GSCHL, TRID1, UTEG2, TRBK, GSBK, AMACH, UCHLS, and MGSSL
  FORTRAN basic functions ... IABS, SQRT, SIGN, ABS, AMAX1, and DSQRT

- Notes
  The default value for parameter EPSZ is represented as EPSZ = $16 \cdot u$ where *u* is the unit round-off.
  If EPSZ is set to 10^{-s}, when a pivot has cancellation of more than *s* decimal digits in LL$^{\mathrm{T}}$ decomposition of positive definite symmetric matrix $\boldsymbol{B}$, the subroutine considers the pivot to be relative zero, sets condition code ICON to 29000 and terminates the processing. If the processing is to proceed even at a low pivot value, EPSZ has to be given the minimum value but the result is not always guaranteed. When the pivot becomes negative in LL$^{\mathrm{T}}$ decomposition of $\boldsymbol{B}$, $\boldsymbol{B}$ is considered not to be positive definite.
  This subroutine, in this case, sets condition code ICON to 28000 and terminates the processing.
  The default value for parameter EPST is

$$\mathrm{EPST} = u \max\left(\left|\lambda_{\max}\right|, \left|\lambda_{\min}\right|\right),$$

where *u* is the unit round-off, and $\lambda_{\max}$ and $\lambda_{\min}$

are upper and lower limits, respectively, of the existing range (given by the Gershgorins theorem) of eigenvalues obtained from $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$. When both very large and small eigenvalues exist, it may be difficult to obtain the smaller eigenvalues with good accuracy. It is possible, though, by setting a small value into EPST, but the computation time may increase.
For details of the method to enter a value into EPST, refer to subroutine BSCT1.

- Example
  This example obtains the *m* largest or *m* smallest eigenvalues and corresponding eivenvectors for the generalized eigenproblem $\boldsymbol{Ax} = \lambda\boldsymbol{Bx}$ which has *n* order real symmetric matrix $\boldsymbol{A}$ and *n* order positive definite symmetric matrix $\boldsymbol{B}$. $n \le 100$, $m \le 100$

```
C     ** EXAMPLE**
      DIMENSION A(5050),B(5050),E(10),
     *EV(100,10),VW(700)
   10 CONTINUE
      READ(5,500) N,M,EPSZ,EPST
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      READ(5,510) (B(I),I=1,NN)
      WRITE(6,600) N,M,EPSZ,EPST
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
      WRITE(6,610) I,(A(J),J=NI,NE)
   20 CONTINUE
      WRITE(6,620)
      NE=0
      DO 30 I=1,N
      NI=NE+1
      NE=NE+I
      WRITE(6,610) I,(B(J),J=1,NI,NE)
   30 CONTINUE
      CALL GSEG2(A,B,N,M,EPSZ,EPST,
     *E,EV,100,VW,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) GO TO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,100,N,MM)
      GO TO 10
  500 FORMAT(2I5,2E15.7)
  510 FORMAT(5E15.7)
  600 FORMAT('1','OROGINAL MATRIX A',5X,
     * 'N=',I3,' M=',I3,' EPSZ=',
     * E15.7,' EPST=',E15.7)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('1','OROGINAL MATRIX B')
  630 FORMAT('0',20X 'ICON=',I5)
      END
```

Subroutine SEPRT in this example is used to print eigenvalues and eigenvectors of a real symmetric matrix. For details, refer to the Example of subroutine SEIG1.

Method
Eigenvalues and eigenvectors of the generalized eigenvalue problem

$$\boldsymbol{Ax} = \lambda\boldsymbol{Bx} \tag{4.1}$$

which has *n* order real symmetric matrix $\boldsymbol{A}$ and *n* order positive definite symmetric matrix $\boldsymbol{B}$ are obtained as follows:

- Reduction of the generalized eigenvalue problem into the standard form.
  Since $\boldsymbol{B}$ in eq. (4.1) is a positive definite symmetric matrix, it can be decomposed as follows:

$$\boldsymbol{B} = \boldsymbol{LL}^{\mathrm{T}} \tag{4.2}$$

where $\boldsymbol{L}$ is an *n* order lower triangular matrix. This decomposition can be uniquely determined when the diagonal elements of $\boldsymbol{L}$ are chosen positive values. From eqs. (e.1) and (4.2), if eq. (4.1) is multiplied by $\boldsymbol{L}^{-1}$ from the left of A and by $\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}$ from the right, the following eq.

$$\boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}}\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right) = \lambda\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}\right) \tag{4.3}$$

can be obtained. Where $\boldsymbol{L}^{-\mathrm{T}}$ is used instead of $(\boldsymbol{L}^{\mathrm{T}})^{-1}$ or $(\boldsymbol{L}^{-1})^{\mathrm{T}}$. Putting

$$\boldsymbol{S} = \boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}} \tag{4.4}$$

$$\boldsymbol{y} = \boldsymbol{L}^{-1}\boldsymbol{x} \tag{4.5}$$

then $\boldsymbol{S}$ becomes a real symmetric matrix and eq. (4.3) becomes

$$\boldsymbol{Sy} = \lambda\boldsymbol{y} \tag{4.6}$$

which is the standard form for the eigenvalue problem.

- Eigenvalues and eigenvectors of a real symmetric matrix.
  Real symmetric matrix $\boldsymbol{S}$ is reduced to real symmetric tridiagonal matrix, and eigenvlalue λ and corresponding eigenvector $\boldsymbol{y}'$ of $\boldsymbol{T}$ are obtained by the bisection method and inverse iteration method. $\boldsymbol{y}'$ is further back transformed to eigenvector $\boldsymbol{y}$ of $\boldsymbol{S}$.
- Eigenvectors of a generalized eigenvalue problem.
  Eigenvector $\boldsymbol{x}$ in eq. (4.1) can be obtained from the eigenvector $\boldsymbol{y}$ determined above, such as

$$\boldsymbol{x} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{y} \tag{4.7}$$

The above processing are accomplished using subroutine GSCHL for the 1-st step., TRID1 and UTEG2 for the 2nd step and GSBK for the last step.
For details see Reference [13] pp. 303-314.

H11-20-0121 HAMNG, DHAMNG

A system of first order differential equations (Hamming method)
CALL HAMNG (Y, N1, H, XEND, EPS, SUB, OUT, VW, ICON)

Function

This subroutine solves a system of first order differential equations:

$$\left.\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_n), & y_{10} &= y_1(x_0) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n), & y_{20} &= y_2(x_0) \\ &\vdots & &\vdots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n), & y_{n0} &= y_n(x_0) \end{aligned}\right\} \quad (1.1)$$

with initial values $y_1(x_0)$, $y_2(x_0)$, ..., $y_n(x_0)$ throughout the interval $[\boldsymbol{x}_0, \boldsymbol{x}_e]$, by Hamming's method.

Initially a specified stepsize is used, however it may be made smaller to achieve the specified accuracy, or larger so that the computation will proceed more rapidly, while still meeting the desired accuracy.

Parameters

Y..... Input. Initial values x_0, y_{10}, y_{20}, ..., y_{n0}. One-dimensional array of size n+1. The contents of Y are altered on exit.

N1... Input. n+1, where n is the number of equations in the system.

H..... Input. Initial step size H ≠ 0.0. The value of H is changed during the computations.

XEND.. Input. Final value x_e, of independent variable x. Calculation is finished when the approximation at x_e is obtained.

ESP... The relative error tolerance. If 0.0 is specified, the standard value is used. (See the Comments and Method sections.)

SUB... Input. The name of subroutine subprogram which evaluates f_i (i = 1, 2, ..., n) in (1.1). The subprogram is provided by the user as follows:
SUBROUTINE SUB (YY, FF)
Parameters
YY: Input. One-dimensional array of size n + 1, where
YY(1) = x, YY(2) = y_1, YY(3) = y_2, ..., YY(n+1) = y_n
FF: Output. One-dimensional array of size n + 1, where
FF(2) = f_1, FF(3) = f_2,
FF(4) = f_3, ..., FF(n+1) = f_n
(See the example.)
Nothing may be substituted in FF(1).

OUT... Input. The name of subroutine subprogram which receives the approximations. In other words, at each integration step with stepsize HH (not necessarily the same as the initial stepsize), subroutine HAMNG transfers the results to this subprogram. This subprogram is provided by the user as follows:
SUBROUTINE OUT (YY, FF, N1, HH, IS)
Parameters
YY: Input. Calculated results of x, y_1, ..., y_n
One-dimensional array of size n+1
where
YY(1) = x,
YY(2) = y_1, ...,
YY(n+1) = y_n
FF: Input. Calculated results of y'_1, ..., y_n
..., y'_2, ...,
FF(n+1) = y'_nFF(1) is usually set to 1.
N1: Input. n+1.
HH: Input. The stepsize with which y_i and y'_i were determined.
IS: Input. Indicator that gives relative magnitude of HH to the initial stepsize H. i.e. HH is related to H as HH = 2^{IS}*H (See the Comments section.)
If IS is set equal to -11 in the subprogram, calculation can terminate at that moment. In this subprogram, except for parameter IS, the contents of the parameters must not be changed.

VW..... Work area. One-dimensional array of size 18(n+1).

ICON.. Output. Condition code.
See Table HAMNG-1.

Table HAMNG-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	0.0 < EPS < 64 u, where u is the unit round off.	The standard value (64u) is used for EPS, and processing is continued.
20000	Calculation was performed with a stepsize 2^{-10} times the specified stepsize H, however, a relative error less than or equal to EPS could not be achieved.	Terminated
30000	N1 < 2, EPS < 0.0, 1.0 < EPS, or (XEND-Y(1))*H≤0.0.	Bypassed

Comments

- Subprogram used
  SSL II...RKG,AMACH
  FORTRAN basic functions...ABS,AMAX1

- Note
  SUB and OUT must be declared as EXTERNAL in the program from which this subroutine is called.
  If IS becomes less than -10, ICON is set to 20000 and calculation is terminated. ICON is set to 0 when the user sets IS = -11 in subroutine OUT.EPS must satisfy the condition $64u \le \mathrm{EPS} < 1$

- Example
  Initial value problem (3.1) is obtained for H=0.1, XEND=5.0 and EPS=10^{-4}

$$\left.\begin{aligned} y_1' &= y_2 && , \; y_{10} = y_1(1) = 5 \\ y_2' &= \frac{4}{x^2} y_1 + \frac{2}{x} y_2 && , \; y_{20} = y_2(1) = 3 \end{aligned}\right\} \tag{3.1}$$

```
C     **EXAMPLE**
      DIMENSION Y(3),VW(54)
      EXTERNAL SUB,OUT
      Y(1)=1.0
      Y(2)=5.0
      Y(3)=3.0
      H=0.1
      EPS=1.0E-4
      WRITE(6,600)
      CALL HAMNG(Y,3,H,5.0,EPS,SUB,OUT,VW,
     *           ICON)
      WRITE(6,610) ICON
      STOP
  600 FORMAT('1'/' ',17X,'X',15X,'Y1',14X,
     *'Y2',14X,'F1',14X,'F2',14X,'HH',12X,
     *'IS'//)
  610 FORMAT(' ',20X,'ICON=',I5)
      END
      SUBROUTINE SUB(YY,FF)
      DIMENSION YY(3),FF(3)
      FF(2)=YY(3)
      FF(3)=4.0*YY(2)/(YY(1)*YY(1))+2.0
     *      *YY(3)/YY(1)
      RETURN
      END
      SUBROUTINE OUT(YY,FF,N1,HH,IS)
      DIMENSION YY(N1),FF(N1)
      WRITE(6,600) (YY(I),I=1,N1),(FF(I),
     *             I=2,N1),HH,IS
      RETURN
  600 FORMAT(' ',10X,6E16.8,I10)
      END
```

Method

Considering the independent variables as a function;

$$y_0(x) = x, y_{00} = y_0(x_0) = x_0 \tag{4.1}$$

initial value problem of a system of first order ordinary differential equations (1.1) can be rewritten as:

$$\begin{aligned} y_0' &= f_0(y_0, y_1, \dots, y_n), & y_{00} &= x_0 \\ y_1' &= f_1(y_0, y_1, \dots, y_n), & y_{10} &= y_1(x_0) \\ y_2' &= f_2(y_0, y_1, \dots, y_n), & y_{20} &= y_2(x_0) \\ &: & &: \\ y_n' &= f_n(y_0, y_1, \dots, y_n), & y_{n0} &= y_n(y_0) \end{aligned} \tag{4.2}$$
$$\text{where } f_0(y_0, y_1, \dots, y_n) = 1$$

Then, the following vectors are introduced to simplify notations

$$\begin{aligned} \boldsymbol{y} &= (y_0, y_1, y_2, \dots, y_n)^{\mathrm{T}} \\ \boldsymbol{y}_0 &= (y_{00}, y_{10}, y_{20}, \dots, y_{n0})^{\mathrm{T}} \\ \boldsymbol{f}(\boldsymbol{y}) &= (f_0(\boldsymbol{y}), f_1(\boldsymbol{y}), \dots, f_n(\boldsymbol{y}))^{\mathrm{T}} \\ &\text{with} \quad f_i(\boldsymbol{y}) = f_i(\mathrm{y}_0, \mathrm{y}_1, \dots, y_n) \\ &\qquad i = 0, 1, \dots, n \end{aligned} \tag{4.3}$$

Let the vector whose elements are y_i (i=0,1,......,n) be represented as $\boldsymbol{y}'$. Then (4.2) can be simplified to

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}), \boldsymbol{y}_0 = \boldsymbol{f}(x_0) \tag{4.4}$$

- Hamming's formula
  Let the approximation and true solution at $x=x_j$, be $\boldsymbol{y}_j$ and $\boldsymbol{y}(\boldsymbol{x})$, respectively, and their corresponding derivatives be $\boldsymbol{y}'_j = \boldsymbol{f}(\boldsymbol{y}_j)$ and $\boldsymbol{y}'(x_j) = \boldsymbol{f}(\boldsymbol{y}(x_j))$.
  $2h,x_k$, the calculated solutions and derivatives

$$\begin{aligned} &\boldsymbol{y}_{k-3}, \boldsymbol{y}_{k-2}, \boldsymbol{y}_{k-1}, \boldsymbol{y}_k \\ &\boldsymbol{y}_{k-3}', \boldsymbol{y}_{k-2}', \boldsymbol{y}_{k-1}', \boldsymbol{y}_k' \end{aligned}$$

are known. $\boldsymbol{y}_{k+1}$ is determined from quantities $\boldsymbol{p}_{k+1}, \boldsymbol{m}_{k+1}$ and $\boldsymbol{c}_{k+1}$ through the following procedures.

- Calculating $\boldsymbol{p}_{k+1}$

$$\boldsymbol{p}_{k+1} = \boldsymbol{y}_{k-3} + \frac{4h}{3}(2\boldsymbol{y}_k' - \boldsymbol{y}_{k-1}' + 2\boldsymbol{y}_{k-2}') \tag{4.5}$$

In(4.5), value $\boldsymbol{P}_{k+1}$ roughly predicts $\boldsymbol{y}_{k+1}$,to a degree and is called the predictor. By the Taylor theorem, it can be expanded as:

$$\begin{aligned} \boldsymbol{y}(x_{k+1}) &= \boldsymbol{y}(x_{k-3}) + \frac{4h}{3}[2\boldsymbol{y}'(x_k) - \boldsymbol{y}'(x_{k-1}) \\ &+ 2\boldsymbol{y}'(x_{k-2})] + \frac{14}{45} h^5 \boldsymbol{y}^{(5)}(\eta_1) \end{aligned} \tag{4.6}$$

where, n_1 depends on the elements of $\boldsymbol{y}$ and it lies between $x_{k-3}$and x_{k+1}. In (4.6), the term $(14/45)h^5y^{(5)}(\boldsymbol{\eta}_1)$ is called the truncation error of $\boldsymbol{p}_{k+1}$. This means that even if y_i' and y_i were true solution and derivative and there were no round-off errors, $\boldsymbol{p}_{k+1}$ has an error of$(14/45)h^5y^{(5)}(\eta_1)$.

- Calculation $\boldsymbol{m}_{k+1}$

$$\boldsymbol{m}_{k+1} = \boldsymbol{p}_{k+1} + \frac{112}{121}(\boldsymbol{c}_k - \boldsymbol{p}_k) \tag{4.7}$$

($\boldsymbol{c}_\mathrm{k}$ will be described later.)
In(4.7), $\boldsymbol{m}_{k+1}$ is a "modified" value of $\boldsymbol{p}_{k+1}$, called the modifier. The second term on the right hand side of (4.7) is the estimated truncation error of $\boldsymbol{p}_{k+1}$(its derivation is not shown here.).

- Calculating $\boldsymbol{c}_{k+1}$
With

$$\begin{aligned} \boldsymbol{m}'_{k+1} &= \boldsymbol{f}(\boldsymbol{m}_{k+1}) \\ \boldsymbol{c}_{k+1} &= \frac{1}{8}(9\boldsymbol{y}_k - \boldsymbol{y}_{k-2}) + \frac{3h}{8}(\boldsymbol{m}'_{k+1} + 2\boldsymbol{y}'_k - \boldsymbol{y}'_{k-1}) \end{aligned} \tag{4.8}$$

$\boldsymbol{c}_{\mathrm{k}+1}$ is the finally corrected value of $\boldsymbol{p}_{\mathrm{k}+1}$ called the corrector. Using the same rationale as in (4.6), truncation error in $\boldsymbol{c}_{\mathrm{k}+1}$ can be derived as $(-1/40)h^5y^{(5)}(\eta_2)$.
If $\boldsymbol{c}_{k+1}$ has the specified accuracy, it is used as the approximation $\boldsymbol{y}_{k+1}$

- Calculating the starting values
In the above method, the previously-described four points are required. Therefore, to start the calculation, the values

$$\begin{gathered} y'_0, y'_1, y'_2, y'_3 \\ y'_0, y'_1, y'_2, y'_3 \end{gathered}$$

must be calculated using another method. This is performed in this subroutine using the Runge-Kutta-Gill method (subroutine RKG). In calculating $\boldsymbol{m}_4$, the quantity $\boldsymbol{c}_3$-$\boldsymbol{p}_3$ is required, but only in this case $\boldsymbol{c}3$-$\boldsymbol{p}3=0$ is assumed.

- Estimating the error in $\boldsymbol{c}_{k+1}$
Since the truncation errors in predictor $\boldsymbol{p}_{k+1}$ and modifier $\boldsymbol{c}_{k+1}$ are respectively $(-1/40)h^5y^5(\eta_{22})$, if the round-off error can be disregarded in comparison with these errors, it can be seen that

$$\begin{aligned} \boldsymbol{y}(x_{k+1}) &= \boldsymbol{p}_{\mathrm{k}+1} + \frac{14}{45}h^5\boldsymbol{y}^{(5)}(\eta_1) \\ \boldsymbol{y}(x_{k+1}) &= \boldsymbol{c}_{\mathrm{k}+1} - \frac{1}{40}h^5\boldsymbol{y}^{(5)}(\eta_2) \end{aligned} \tag{4.9}$$

Furthermore, if it can be assumed that $\boldsymbol{y}^{(5)}(x)$ does not change greatly been η_1 and η_2, from (4.9) we obtain

$$h^5\boldsymbol{y}^{(5)}(\eta_1) \approx h^5\boldsymbol{y}^{(5)}(\eta_2) \approx \frac{360}{121}(\boldsymbol{c}_{k+1} - \boldsymbol{p}_{k+1})$$

and finally the truncation error of $\boldsymbol{c}_{\mathrm{k}+1}$ can be expressed as

$$-\frac{1}{40}h^5\boldsymbol{y}^{(5)}(\eta_2) \approx -\frac{9}{121}(\boldsymbol{c}_{k+1} - \boldsymbol{p}_{k+1}) \tag{4.10}$$

- Step size control
If calculation is made with a constant stepsize, the desired accuracy may not be achieved at certain points. On the other hand, the accuracy of the results can be so much higher than the required accuracy that the effect of round-off errors is greater than that of truncation errors. As a result, the lower digits of $\boldsymbol{y}_i$ will oscillate, unless the stepsize is properly controlled. Since truncation error of modifier $\boldsymbol{c}_{\mathrm{k}+1}$ use as the approximation, is given in (4.10), the stepsize is controlled as follows:

(a) If (4.11) holds with respect to all elements, $\boldsymbol{c}_{\mathrm{k}+1}$ is used as the approximation.

$$\frac{9}{121}|\boldsymbol{c}_{k+1} - \boldsymbol{p}_{k+1}| \le \mathrm{EPS} \cdot \max(|\boldsymbol{y}_k|, |\boldsymbol{c}_{k+1}|) \tag{4.11}$$

where, EPS is the specified tolerance for the relative error. Its standard value is $64u$.

(b) If (4.12) holds with respect to all elements, c_{k+1} is used as the approximation.

$$\frac{9}{121}|\boldsymbol{c}_{k+1} - \boldsymbol{p}_{k+1}| \le \mathrm{TOL} \cdot \max(|\boldsymbol{y}_k|, |\boldsymbol{c}_{k+1}|) \tag{4.12}$$

where TOL=EPS/32
And also, the stepsize is doubled for the next step. At this time, the value of the previous $\boldsymbol{y}_{k-5}$ is needed again. Thus, the informations prior to $\boldsymbol{y}_{k-5}$ are kept as long as possible.

(c) If (4.13) holds for a particular elements,$\boldsymbol{c}_{k+1}$ is not used as the approximation.

$$\frac{9}{121}|\boldsymbol{c}_{k+1} - \boldsymbol{p}_{k+1}| > \mathrm{EPS} \cdot \max(|\boldsymbol{y}_k|, |\boldsymbol{c}_{k+1}|) \tag{4.13}$$

In this case, the current stepsize is halved, and calculations for the step are made again.

At this time, $\boldsymbol{y}_{k-1/2}$ and $\boldsymbol{y}_{k-3/2}$ are needed. Using $\boldsymbol{y}_{k-1}$ and $\boldsymbol{y}_{k-2}$ as the starting values, the Runge-Kutta-Gill method is used to calculated them.

If the stepsize is changed due to (b) or (c), the quantity $\boldsymbol{c}_k$-$\boldsymbol{p}_k$ is recalculated as

$$\boldsymbol{c}_k - \boldsymbol{p}_k = \frac{242}{27}[\boldsymbol{y}_k - \boldsymbol{y}_{k-3} - \frac{3h}{8}(\boldsymbol{y}'_{k-3} + 3\boldsymbol{y}'_{k-2} + 3\boldsymbol{y}'_{k-1} + \boldsymbol{y}'_k)] \qquad (4.14)$$

where h, $\boldsymbol{y}_i$, and $\boldsymbol{h}'_i$ on the right side are all quantities after the stepsize has been changed.

- Calculating the solution at XEND
  If the previously described method is used, the approximation at XEND may not necessarily be obtained. To make sure that it is obtained at XEND, this subroutine uses the following algorithm.

Assume the state just before the independent variable x exceeds XEND and let the value of x at that time be x_i and the stepsize be h (Fig. HAMG-1)

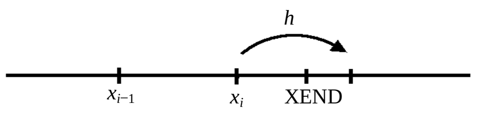


Fig.HAMNG-1 Last stage(h>0)

The following relation exists among XEND, x_i and h.

[XEND-(x_i+h)]h<0

When this state is reached, the Hamming method is no longer used. Instead, the Runge-Kutta-Gill method is applied using x_i as the starting point.
Calculating two approximate solutions with stepsize(XEND-x_i)/2 and (XEND)-x_i), solution $\boldsymbol{y}_e$ at XEND is obtained as

$$\boldsymbol{y}_e = \boldsymbol{y}_e^{(2)} + \left(\boldsymbol{y}_e^{(2)} - \boldsymbol{y}_e^{(1)}\right)/15 \qquad (4.15)$$

where

$\boldsymbol{y}_e^{(2)}$ is approximate solution at XEND with stepsize h=(XEND-x_i)/2
$\boldsymbol{y}_e^{(1)}$ is approximate solution at XEND with stepsize h=(XEND-x_i).

For further information, see Reference[70].

B21-11-0602 HBK1, DHBK1

Back transformation and normaliazation of the eigenvectors of a real Hessenberg matrix.
CALL HBK1(EV,K,N,IND,M,P,PV,DV,ICON)

Function

Back transformation is performed on m eigenvectors of an n-order real Hessenberg matrix $\boldsymbol{H}$ to obtain the eigenvectors of a real matrix $\boldsymbol{A}$. Then the resulting eigenvectors are normalized such that $\|x\|_2 = 1$. $\boldsymbol{H}$ is obtained from $\boldsymbol{A}$ using the Householder method.
$1 \le m \le n$.

Parameters

EV......	Input. m eigenvectors of real Hessenberg matrix $\boldsymbol{H}$(see "Comments on use"). EV is a two-dimensional array, EV(K,M) Output. Eigenvectors of real matrix $\boldsymbol{A}$.
K......	Input. Adjustable dimension of array EV and P.
N......	input. Order n of real Hessenberg matrix $\boldsymbol{H}$.
IND......	Input. Specifies, for each eigenvector, whether the eigenvector is real or complex. If the J-th column of EV is a real eigenvector, then IND(J)=1,; if it is the real part of a complex eigenvector, then IND(J)=-1, if it is the imaginary part of a complex eigenvector, then IND(J)=0. IND is a one-dimensional array of size M.
M......	Input. Size of array IND.
P......	Input. Transformation matrix provided by Householder method (see "Comments on use"). P(K,N) is a two-dimensional array.
PV......	Input. Transformation matrix provided by Householder method (see "Comments on use").
DV......	Input. Scaling factor used for balancing of matrix $\boldsymbol{A}$. DV is a one-dimensional array of size n. If matrix $\boldsymbol{A}$ was not balanced, DV=0.0 can be specified since it need not be a one-dimensional array.
ICON......	Output. Condition code. See Table HBK1-1.

Table HBK1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	EV(1,1)=1.0
30000	N<M,M<1orK<N	Bypassed

Comments on use

- Subroutines used
  SSL II......NRML, and MGSSL
  FORTRAN basic functions......ABS and SQRT

- Notes
  Eigenvectors are stored in EV such that each real eigenvector occupies one column and each complex eigenvector occupies two columns (one for the real part and one for the imaginary part).
  Refer to Fig. HBK1-1. After subroutine HVEC is executed, parameters EV, IND, and M can be used as input parameters to this routine.

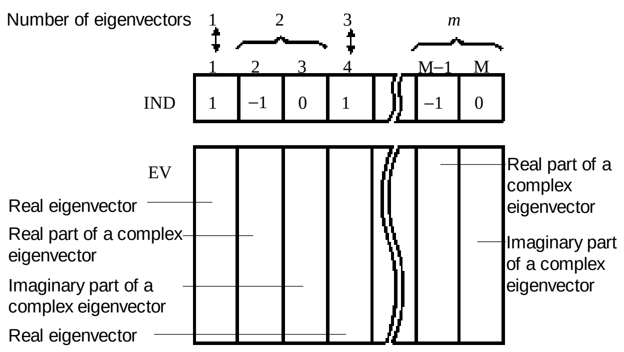

Number of eigenvectors
1
2
3
m
1
2
3
4
M−1
M
IND
1
−1
0
1
−1
0
EV
Real eigenvector
Real part of a complex eigenvector
Imaginary part of a complex eigenvector
Real eigenvector
Real part of a complex eigenvector
Imaginary part of a complex eigenvector

Fig. HBK1-1 Relationship between IND and EV

$$
\mathrm{P}\begin{bmatrix}
\times & \cdots & \cdots & \cdots & \cdots & \times \\
\sigma_1^{\frac{1}{2}} & \times & \cdots & \cdots & \cdots & \times \\
a_{31}^{(1)} & \ddots & \ddots & & & \vdots \\
a_{41}^{(1)} & \ddots & \sigma_k^{\frac{1}{2}} & \ddots & & \vdots \\
\vdots & & a_{k+2k}^{(k)} & \ddots & \ddots & \vdots \\
\vdots & & & \sigma_{n-2}^{\frac{1}{2}} & \ddots & \vdots \\
a_{n1}^{(1)} & \cdots & a_{nk}^{(k)} & \cdots\ a_{nn-2}^{(n-2)} & \times & \times
\end{bmatrix}
\qquad
\mathrm{PV}\begin{bmatrix}
\times \\
a_{21}^{(1)} \pm \sigma_1^{\frac{1}{2}} \\
\vdots \\
a_{k+1k}^{(k)} \pm \sigma_k^{\frac{1}{2}} \\
\vdots \\
a_{n-1n-2}^{(n-2)} \pm \sigma_{n-2}^{\frac{1}{2}} \\
\times
\end{bmatrix}
$$

Note: $\times$ is not used.

Fig. HBK1-2 Input format of P and PV

Parameters A and PV of subroutine HES1 correspond to P and PV of this routine and can be used as input parameters to this routine.
The information of the transformation matrix in the Householder method should be entered in P and PV as shown in Fig. HBK1-2.
$a_{ij}^{(k)}$ are elements of A_k used in

$$\boldsymbol{A}_{k+1} = \boldsymbol{P}_k^{\mathrm{T}} \boldsymbol{A}_k \boldsymbol{P}_k,\ k = 1,2,\ldots,n-2 \tag{3.1}$$

$\boldsymbol{\sigma}_k$ are determined by

$$\sigma_k = \left(a_{k+1k}^{(k)}\right)^2 + \left(a_{k+2k}^{(k)}\right)^2 + \ldots + \left(a_{nk}^{(k)}\right)^2 \tag{3.2}$$

For further information, refer to the section on HES1.
Refer to the section on BLNC for the contents of scaling factor DV.

- Example
  Eigenvalues and eigenvectors of an *n*-order real matrix are calculated using the following subroutines. Eigenvectors are calculated in the order that the eigenvalues are determined.

BLNC...	balances an *n*-order real matrix
HES1...	reduces a balanced real matrix to a real Hessenberg matrix
HSQR...	determines eigenvalues of a real Hessenberg matrix.
HVEC...	determines eigenvectors of a real Hessenberg matrix
HBK1...	back transforms eigenvectors of a real Hessenberg matrix into eigenvectors of a real matrix, then normalizes the resultant eigenvectors.

$n \le 100$, $m \le 10$

```
C     **EXAMPLE**
      DIMENSION A(100,100),DV(100),
     *PV(100),IND(100),ER(100),EI(100),
     *AW(100,104),EV(100,100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL BLNC(A,100,N,DV,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL HES1(A,100,N,PV,ICON)
      MP=N
      DO 35 II=1,N
      I=1+N-II
      DO 30 J=1,MP
   30 AW(J,I)=A(J,I)
   35 MP=I
      CALL HSQR(AW,100,N,ER,EI,M,ICON)
      WRITE(6,620)ICON
      IF(ICON.EQ.20000) GO TO 10
      DO 40 I=1,M
   40 IND(I)=1
      CALL HVEC(A,100,N,ER,EI,IND,M,EV,100,
     *AW,ICON)
      WRITE(6,620)ICON
      IF(ICON.GE.20000) GO TO 10
      CALL HBK1(EV,100,N,IND,M,A,PV,DV,ICON)
      CALL EPRT(ER,EI,EV,IND,100,N,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'
     * //11X,'** ORDER =',I5)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     *E14.7))
  620 FORMAT(/11X,'** CONDITION CODE =',I5/)
      END
```

In this example, subroutine EPRT is used to print the eigenvalues and corresponding eigenvectors of the real matrix. For further information see the example in the EIG1 section.

Method

Back transformation of the eigenvectors of a real Hessenberg matrix $\boldsymbol{H}$ to the eigenvectors of the balanced real matrix $\tilde{\boldsymbol{A}}$ and back transformation of the eigenvectors of $\tilde{\boldsymbol{A}}$ to the eigenvectors of the real matrix $\boldsymbol{A}$ are performed. The resulting eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$.

The real matrix $\boldsymbol{A}$ is balanced using the diagonal similarity transformation shown in (4.1). The balanced real matrix $\tilde{\boldsymbol{A}}$ is reduced to a real Hessenberg matrix using the Householder method which performs the (*n*-2) orthogonal similarity transformations shown in (4.2).

$$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D} \qquad (4.1)$$

(where $\boldsymbol{D}$ is a diagonal matrix.)

$$\boldsymbol{H} = \boldsymbol{P}_{n-2}^{\mathrm{T}} \ldots \boldsymbol{P}_2^{\mathrm{T}} \boldsymbol{P}_1^{\mathrm{T}} \tilde{\boldsymbol{A}} \boldsymbol{P}_1 \boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2} \qquad (4.2)$$

(where $\boldsymbol{P}_i = \boldsymbol{I} - \boldsymbol{u}_i \boldsymbol{u}_i^{\mathrm{T}} / h_i$)

Let eigenvalues and eigenvectors of H be λ and $\boldsymbol{y}$, then obtain

$$\boldsymbol{H}\boldsymbol{y} = \lambda \boldsymbol{y} \qquad (4.3)$$

From (4.1) and (4.2),(4.3) becomes:

$$\boldsymbol{P}_{n-2}^{\mathrm{T}} \ldots \boldsymbol{P}_2^{\mathrm{T}} \boldsymbol{P}_1^{\mathrm{T}} \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{D} \boldsymbol{P}_1 \boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2} \boldsymbol{y} = \lambda \boldsymbol{y} \qquad (4.4)$$

If both sides of (4.4) are premultiplied by $\boldsymbol{D}\boldsymbol{P}_1\boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2}$,

$$\boldsymbol{A}\boldsymbol{D}\boldsymbol{P}_1\boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2}\boldsymbol{y} = \lambda \boldsymbol{D}\boldsymbol{L}\boldsymbol{P}_1\boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2}\boldsymbol{y} \qquad (4.5)$$

results, and eigenvectors $\boldsymbol{x}$ of $\boldsymbol{A}$ becomes

$$\boldsymbol{x} = \boldsymbol{D}\boldsymbol{P}_1\boldsymbol{P}_2 \ldots \boldsymbol{P}_{n-2}\boldsymbol{y} \qquad (4.6)$$

which is calculated as shown in (4.7) and (4.8). However, $y=x_{n-1}$.

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{P}_i \boldsymbol{x}_{i+1} \\ &= \boldsymbol{x}_{i+1} - \left(\boldsymbol{u}_i \boldsymbol{u}_i^{\mathrm{T}} / h_i\right) \boldsymbol{x}_{i+1}, \; i = n-2, \ldots, 2, 1 \end{aligned} \qquad (4.7)$$

$$\boldsymbol{x} = \boldsymbol{D}\boldsymbol{x}_1 \qquad (4.8)$$

For further information on the Householder method and balancing, refer to the sections on HES1 and BLNC. NRML is used to normalize the eigenvectors. For details, see Reference [13] pp339-358

B21-25-0201 HEIG2, DHEIG2

Eigenvalues and corresponding eigenvectors of a Hermitian matrix (Householder method, bisection method and inverse iteration method)
CALL HEIG2(A,K,N,M,E,EVR,EVI,VW,ICON)

Function

The *m* largest or *m* smallest eigenvalues of an *n*-order Hermitian matrix ***A*** are obtained using the bisection method, where 1≤*m*≤*n*. Then the corresponding eigenvectors are obtained using the inverse iteration method. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$.

Parameters

A......	Input. Hermitian matrix *A* in the compressed storage mode. Refer to "2.8 Data Storage". A is a two-dimensional array, A(K,N). The contents of A are altered on output.
K......	Input. Adjustable dimension of array A,EVER, and EVI(≥*n*).
N......	Input. Order *n* of the Hermitian matrix.
M......	Input. M=+*m*... The number of largest eigenvalues desired. M=-*n*... The number of smallest eigenvalues desired.
E......	Output. Eigenvalues. One dimensional array of size *m*.
EVR,EVI	Output. The real part of the eigenvectors are stored into EVR and the imaginary part into EVI both in columnwise direction. The *l*-th element of the eigenvector corresponding to the *j*-th eigenvalue E(J) is represented by EVR(L,J)+*i*·EVI(L,J), where *i*=$\sqrt{-1}$. EVR and EVI are two dimensional arrays EVR(K,M), and EVI(K,M).
VW......	Work area. One dimensional array of size 9*n*.
ICON......	Output. Condition code. See Table HEIG2-1.

Table HEIG2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	E(1)=A(1,1) EVR(1,1)=1.0 EVI(1.1)=0.0
15000	Some eigenvectors could not be determined.	The eigenvectors are set to zero vectors.
20000	None of eigenvectors could be determined.	All the eigenvectors are set to zero vectors.
30000	M=0,N<|M| or K<N	Bypassed

Comments on use

- Subprograms used
  SSL II...TRIDH, TEIG2, TRBKH, AMACH, MGSSL, and UTEG2
  FORTRAN basic functions ... IABS,SQRT,ABS, and AMAX1.

- Note
  This subroutine is provided for a Hermitian matrix and not for a general complex matrix.
  This subroutine should be used to obtain both eigen values and eigenvectors. For determining only eigenvalues, subroutines TRIDH and BSCT1 should be used.

- Example
  The *m* largest (or smallest) eigen values of *n*-order Hermitian matrix ***A*** as well as the corresponding eigenvectors are obtained in this example for *n*≤100 and *m*≤ 10.

```
C     **EXAMPLE**
      DIMENSION A(100,100),E(10),
     * EVR(100,10),EVI(100,10),VW(900)
   10 CONTINUE
      READ(5,500) N,M
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,M
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL HEIG2(A,100,N,M,E,EVR,EVI,VW,
     *           ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      MM=IABS(M)
      CALL HEPRT(E,EVR,EVI,100,N,MM)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'///40X,'**ORIGINAL MATRIX**'
     * 5X,'N=',I3,5X,'M=',I3//)
  610 FORMAT(/4(4X,'A(',I3,',',I3,')=',
     * E15.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

Subroutine HEPRT used in above example is for printing eigenvalues and eigenvectors of a Hermitian matrix, shown as follows:

```
      SUBROUTINE HEPRT(E,EVR,EVI,K,N,M)
      DIMENSION E(M),EVR(K,M),EVI(K,M)
      WRITE(6,600)
      DO 10 I=1,M
   10 WRITE(6,610) I,E(I)
      KAI=(M-1)/3+1
      LST=0
      WRITE(6,620)
      DO 20 I=1,KAI
      INT=LST+1
      LST=LST+3
      IF(LST.GT.M) LST=M
      WRITE(6,630) (J,J=INT,LST)
      WRITE(6,640)
      DO 20 J=1,N
   20 WRITE(6,650) J,(EVR(J,L),EVI(J,L),
     *             L=INT,LST)
      RETURN
  600 FORMAT('1'///5X,'**HERMITIAN ',
     * 'MATRIX**',///5X,'**EIGENVALUES**'/)
  610 FORMAT(5X,'E(',I3,')=',E20.8)
  620 FORMAT('0',///5X,'**EIGENVECTORS**'//)
  630 FORMAT('0',3X,3(20X,'X(',I3,')', 14X))
  640 FORMAT(//13X,3('REAL PART',10X,
     * 'IMAG. PART',11X))
  650 FORMAT(1X,I3,6E20.8)
      END
```

Method

Eigenvalues and eigenvectors of *n* order Hermitian matrix $\boldsymbol{A}$ are obtained through the following procedures:

- Reduction of the Hermitian matrix to a real symmetric tridiagonal matrix.
  This is done first by reducting the Hermitian matrix $\boldsymbol{A}$ to Hermitian tridiagonal matrix $\boldsymbol{H}$ using the Householder method,

  $$\boldsymbol{H}=\boldsymbol{P}^{*}\boldsymbol{AP} \tag{4.1}$$

  then by reducing it further to a real symmetric tridiagonal matrix $\boldsymbol{T}$ by diagonal unitary transformation,

  $$\boldsymbol{T}=\boldsymbol{V}^{*}\boldsymbol{HV} \tag{4.2}$$

  where $\boldsymbol{P}$ is a unitary matrix and $\boldsymbol{V}$ is a diagonal unitary matrix.

- Eigenvalues and eigenvectors of the real symmetric tridiagonal matrix
  m eigenvalues of $\boldsymbol{T}$ are obtained by the bisection method, then the corresponding eigenvector $\boldsymbol{y}$ is obtained using the inverse iteration method. The inverse iteration method solves

  $$(\boldsymbol{T}-\mu\boldsymbol{I})\boldsymbol{y}_r=\boldsymbol{y}_{r-1}, \quad \boldsymbol{r}=1,2,... \tag{4.3}$$

  repeatedly to get eigenvector $\boldsymbol{y}$, where μ is the eigenvalue obtained by the bisection method and $\boldsymbol{y}_0$ is an appropriate initial vector.

- Eigenvectors of the Hermitian matrix Eigenvector $\boldsymbol{x}$ of $\boldsymbol{A}$ can be obtained using eigenvector $\boldsymbol{y}$ obtained in (4.3) as

  $$\boldsymbol{x}=\boldsymbol{PVy} \tag{4.4}$$

Eq. (4.4) is the back transformation of eqs. (4.1) and (4.2). The above three are performed with subroutines TRIDH,TEIG2 and TRBKIH respectively. For details, see references[12], [13] pp. 259-269, and [17].

B21-11-0302 HES1,DHES1

Reduction of a real matrix to a real Hessenberg matrix (Householder method)
CALL HES1(A,K,N,PV,ICON)

Function

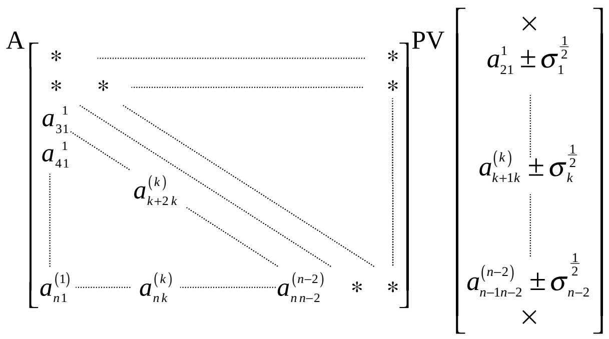
An *n*-order real matrix ***A*** is reduced to a real Hessenverg matrix ***H*** using the Householder method (orthogonal similarity method).

$$\boldsymbol{H}=\boldsymbol{P}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{P}$$

P is the transformation matrix $n \geq 1$.

Parameters

A......	Input. Real matrix ***A***. Output. Real Hessenberg matrix ***H*** and transformation matrix ***P***. (See Fig. HES1-1) A is a two-dimensional array A (K, N)
K......	Input. Adjustable dimension of array A($\geq n$)
N......	Input. Order *n* of real matrix ***A***.
PV......	Output. Transformation matrix ***P*** (See Fig. HES1-2). PV is a one-dimensional array of size *n*.
ICON...	Output. Condition code. See Table HES1-1.


A
PV
$*$
a_{31}^{1}
a_{41}^{1}
$a_{k+2\,k}^{(k)}$
$a_{n1}^{(1)}$
$a_{nk}^{(k)}$
$a_{n\,n-2}^{(n-2)}$
$\times$
$a_{21}^{1} \pm \sigma_{1}^{\frac{1}{2}}$
$a_{k+1k}^{(k)} \pm \sigma_{k}^{\frac{1}{2}}$
$a_{n-1n-2}^{(n-2)} \pm \sigma_{n-2}^{\frac{1}{2}}$

Note: The section indicated with * is the Hessenberg matrix; the rest contains some information for the transformation matrix. X indicates work areas.

Fig HEW1-1 A and PV after Householder transformation

Table HES1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1 or N=2	Transformation is not performed.
30000	K<N or N<1	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, and MGSSL
  FORTRAN basic functions ... ABS, DSQRT, and SIGN

- Notes
  Output arrays A and PV are necessary for determining the eigenvalues and corresponding eigenvectors of real matrix ***A***.
  The precision of eigenvalues is determined in the real Hessenberg matrix transformation process. For that reason, this subroutine has been implemented so that real Hessenberg matrices can be determined at as high a precision as possible; however, in case of matrix containing very large and very small eigenvalues, the precision of the smaller values, some of which are difficult to precisely determine tends to be more affected by the reduction process.

- Example
  After an *n*-order real matrix is reduced to a real Hessenberg matrix, subroutine HSQR is used to determine the eigenvalues. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),PV(100),
     *          ER(100),EI(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL HES1(A,100,N,PV,ICON)
      WRITE(6,620)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,610) ((I,J,A(I,J),J=1,N),
     *             I=1,N)
      CALL HSQR(A,100,N,ER,EI,M,ICON)
      WRITE(6,640)
      WRITE(6,630) ICON
      IF(ICON.EQ.20000) GO TO 10
      WRITE(6,650) (I,ER(I),I,EI(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'
     * //11X,'** ORDER =',I5/)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     * E14.7))
  620 FORMAT('0'/11X,'** HESSENBERG MATRIX')
  630 FORMAT(/11X,'** CONDITION CODE =',I5/)
  640 FORMAT('0'/11X,'** EIGENVALUES')
  650 FORMAT(5X,'ER(',I3,')=',E14.7,
     * 5X,'EI(',I3,')=',E14.7)
      END
```

Method

An n-order real matrix $\boldsymbol{A}$ is reduced to a real Hessenberg Matrix through n-2 iterations of the orthogonal similarity transformation.

$$\boldsymbol{A}_{k+1} = \boldsymbol{P}_k{}^{\mathrm{T}} \boldsymbol{A}_k \boldsymbol{P}_k, \quad k = 1,2,...,n-2 \tag{4.1}$$

where, $\boldsymbol{A}_1$=$\boldsymbol{A}$.$\boldsymbol{P}_k$ is the transformation (and orthogonal) matrix.

When transformation is completed, $\boldsymbol{A}_{n-1}$ is in the real Hessenberg matrix.

The k-th transformation is performed according to the following procedure: Let $\boldsymbol{A}_k = \left(a_{ij}^{(k)}\right)$

$$\sigma_k = \left(a_{k+1k}^{(k)}\right)^2 + \left(a_{k+2k}^{(k)}\right)^2 + ... + \left(a_{nk}^{(k)}\right)^2 \tag{4.2}$$

$$\boldsymbol{u}_k{}^{\mathrm{T}} = \left(0,...,0,a_{k+1k}^{(k)} \pm \sigma_k^{1/2}, a_{k+2k}^{(k)},...,a_{nk}^{(k)}\right) \tag{4.3}$$

$$h_k = \sigma_k \pm a_{k+1k}^{(k)} \sigma_k^{1/2} \tag{4.4}$$

$$\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^{\mathrm{T}} / h_k \tag{4.5}$$

By applying $\boldsymbol{P}_k$ in (4.5) to (4.1), $a_{k+2\,k}^{(k)}$ to $a_{n\,k}^{(k)}$ of $\boldsymbol{A}_k$ can be eliminated. The following precatuions are taken during the transformation. To avoid possible underflow and overflow in the computations of (4.2) to (4.4), the elements on the right side of (4.2) are scaled by

$$\sum_{i=k+1}^{n} \left|a_{ik}^{(k)}\right|$$

- When determining $\boldsymbol{u}_k{}^{\mathrm{T}}$ of (4.3), to insure that cancellation cannot take place in computation of $a_{k+1k}^{(k)} \pm \sigma_k{}^{1/2}$, the sign of $\pm \sigma_k{}^{1/2}$ is taken to be that of $a_{k+1k}^{(k)}$

- Instead of determining $\boldsymbol{P}_k$ for the transformation of (4.1), $\boldsymbol{A}_{k+1}$ if determined by (4.7) and (4.8).

$$\boldsymbol{B}_{k+1} = \boldsymbol{P}_k^{\mathrm{T}} \boldsymbol{A}_k = \boldsymbol{A}_k - \boldsymbol{u}_k \left(\boldsymbol{u}_k^{\mathrm{T}} \boldsymbol{A}_k\right) / h_k \tag{4.7}$$

$$\boldsymbol{A}_{k+1} = \boldsymbol{B}_{k+1} \boldsymbol{P}_k = \boldsymbol{B}_{k+1} - \left(\boldsymbol{B}_{k+1} \boldsymbol{u}_k / h_k\right) \boldsymbol{u}_k^{\mathrm{T}} \tag{4.8}$$

The elements of $\boldsymbol{u}_k$ obtained from (4.3) are stored in array A and one-dimensional array PV in the form shown in Fig.HES1-2, because transformation matrix is needed for determining the eigenvectors of real matrix $\boldsymbol{P}_k$. When n=2 or n=1, transformation is not performed. For further information see Reference[13]pp.339-358.

F20-02-0201 HRWIZ,DHRWIZ

Judgment on Hurwitz polynomials
CALL HRWIZ(A,NA,ISW,IFLG,SA,VW,ICON)

Function

This subroutine judges whether the following polynomial of degree n with real coefficients is a Hurwitz polynomial (all zeros lies in the lefthalf plane Re(s)<0:

$$P(s) = a_1 s^n + a_2 s^{n-1} + \cdots + a_n s + a_{n+1}$$

If $P(s)$ is not a Hurwitz polynomial, the subroutine searches for α_0 such that $P(s+\alpha)$ becomes a Hurwitz polynomial for $\alpha > \alpha_0$ (≥ 0)

Parameters

A......	Input. Coefficents of $P(s)$. One-dimensional array of size $n+1$, assigned in the order of A(1) = a_1, A(2) = a_2, ... ,A(n+1) = a_{n+1}.
NA......	Input. Degree n of $P(s)$.
ISW......	Input. Control information. 0: Only judges whether $P(s)$ is a Hurwitz polynomial. 1: Judges whether $P(s)$ is a Hurwitz polynomial, and if it is not a Hurwitz polynomial, searches for α_0. Others: 1 is assumed.
IFLG...	Output Result of judgment. 0: $P(s)$ is Hurwitz polynomial. 1: $P(s)$ is not a Hurwitz polynomial.
SA......	Output. The value of α_0:0.0 if $P(s)$ is a Hurwitz polynomial.
VW......	Work area: One-dimensional array of size n+1.
ICON......	Output. Condition code. (See Table HRWIZ-1.)

Table HRWIZ-1 Condition codes

Code	Meaning	Processing
0	No error.	
20000	Value α_0 has not been found.	Bypassed.
30000	NA<1 or A(1)=0.	Bypassed.

Comments on use

- Subprograms used
  SSL II...MGSSL,AMACH
  FORTRAN basic functions...ABS,ALOG10

- Notes
  Since the function of this subroutine relates to obtaining the inverse Laplace transform $f(t)$ of a rational function $F(s) = Q(s)/P(s)$, it can also be used to roughly check the characteristic of $f(t)$. This means that $F(s)$ has singularities in the domain of Re(s)≥0 if $P(s)$ is not Hurwitz polynomial, and the value of the inverse transform function $f(t)$increases exponentially as the value of t approaches infinity.
  To obtain the inverse Laplace transform $f(t)$ of a rational function $F(s)$, use subroutine LAPS1 or LAPS2 depending on whether $F(s)$ is known to be regular in the domain of Re(s)>0. (See Chapter 8 for details about numerical calculation of the inverse Laplace transforms.)

- Example
  Given a polynomial $P(s) = s^4 - 12s^3 + 54s^2 - 108s + 81$, the following judges whether it is a Hurwitz polynomial.

```
C     **EXAMPLE**
      DIMENSION A(5),VW(5)
      NA=4
      NA1=NA+1
      A(1)=1.0
      A(2)=-12.0
      A(3)=54.0
      A(4)=-108.0
      A(5)=81.0
      ISW=1
      WRITE(6,600) NA,(I,A(I),I=1,NA1)
      CALL HRWIZ(A,NA,ISW,IFLG,SA,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0)STOP
      WRITE(6,620)ISW,IFLG,SA
      STOP
  600 FORMAT(//24X,'NA=',I3//
     *(20X,'A(',I3,')=',E15.8/))
  610 FORMAT(/24X,'ICON=',I6)
  620 FORMAT(/24X,'ISW=',I2,5X,
     *'IFLG=',I2,5X,'SA=',E15.8)
      END
```

Method

See 8.6 for details about judgment on Hurwitz polynomials.

B21-11-0402 HSQR,DHSOR

Eigenvalues of a real Hessenberg matrix (two-stage QR method)
CALL HSQR(A,K,N,ER,EI,M,ICON)

Function

All eigenvalues of an *n*-order real Hessenberg matrix are determined using the double QR method. $n \geq 1$.

Parameters

A......	Input. Real Hessenberg matrix. The contents of A are altered on output. A is a two-dimensional array, A(K,N).
K......	Input. Adjustable dimensions of array A.
N......	Input. Order *n* of the real Hessenberg matrix.
ER,EI......	Output. Eigenvalues. The *J*th eigenvalue is ER(*J*)+*i*·EI(*J*) (*J*=1,2, ...,M); $i=\sqrt{-1}$ ER and EI are one-dimensional arrays of size *n*.
M......	Output. The number of determined eigenvalues.
ICON......	Output. Condition code. See Table HSQR-1.

Table HSQR-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	ER(1)=A(1,1) and EI(1)=0.0
15000	Some of the eigenvalues could not be determined.	M is set to the number of eigenvalues that were obtained. 1≤M<N.
20000	No eigenvalues could be determined.	M is set to 0.
30000	K≤N or N <1	Bypassed

Comments on use

- Subprograms used
  SSLII...AMACH and MGSSL
  FORTRAN basic functions...ABS,SQRT and SIGN

- Notes
  Normally, after executing subroutine HES1, this routine is used to determine all eigenvalues.
  If eigenvectors are also needed, array A should be copied onto another area before calling this routine.

- Example
  After reducing an *n*-order real matrix to a real Hessenberg matrix using HES1, the eigenvalues are determined. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),PV(100),
     *          ER(100),EI(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL HES1(A,100,N,PV,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      CALL HSQR(A,100,N,ER,EI,M,ICON)
      WRITE(6,630)
      WRITE(6,620) ICON
      IF(ICON.EQ.20000) GO TO 10
      WRITE(6,640) (I,ER(I),I,EI(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'
     * //11X,'** ORDER =',I5/)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     * E14.7))
  620 FORMAT(/11X,'** CONDITION CODE =',I5/)
  630 FORMAT('0'/11X,'** EIGENVALUES')
  640 FORMAT(5X,'ER(',I3,')=',E14.7,
     * 5X,'EI(',I3,')=',E14.7)
      END
```

Method

In the QR method, the diagonal elements become eigenvalues by making the subdiagonal elements of Hessenberg matrix $\boldsymbol{A}$ converge to zero. To accomplish this the orthogonal similarity transformation shown in (4.1) is applied repeatedly.

$$\boldsymbol{A}_{s+1} = \boldsymbol{Q}_s^{\mathrm{T}} \boldsymbol{A}_s \boldsymbol{Q}_s \tag{4.1}$$

$\boldsymbol{Q}_s$ is the orthogonal matrix which is uniquely determined in the QR decomposition shown in (4.2). $\boldsymbol{R}_s$ is an upper triangular matrix whose diagonal elements are positive real numbers.

$$\boldsymbol{A}_s = \boldsymbol{Q}_s \boldsymbol{R}_s \tag{4.2}$$

By applying $\boldsymbol{Q}_s$ determined in (4.2) to (4.1), the lower subdiagonal elements of $\boldsymbol{A}$ gradually converge to zero.

Normally, to improve the rate of convergence, QR decomposition is applied to origin-shifted matrix $(\boldsymbol{A}_s - \boldsymbol{k}_s \boldsymbol{I})$ instead of $\boldsymbol{A}_s$. Then orthogonal similarity transformation is performed. k_s is the origin shift. Since both real and/or complex numbers are present in the eigenvalues of a real matrix, it is necessary to select a complex number as k_s. As a result, complex arithmetic throughout the process is required. To avoid this problem, the double QR method, in which arithmetic is kept real, is adopted in the subroutine. In this case, (4.1) and (4.2) become (4.3) and (4.4) respectively.

$$\boldsymbol{A}_{s+2} = \left(\boldsymbol{Q}_s \boldsymbol{Q}_{s+1}\right)^{\mathrm{T}} \boldsymbol{A}_s \left(\boldsymbol{Q}_s \boldsymbol{Q}_{s+1}\right) \tag{4.3}$$

$$\left(\boldsymbol{A}_s - k_s \boldsymbol{I}\right)\left(\boldsymbol{A}_s - k_{s+1} \boldsymbol{I}\right) = \left(\boldsymbol{Q}_s \boldsymbol{Q}_{s+1}\right)\left(\boldsymbol{R}_{s+1} \boldsymbol{R}_s\right) \tag{4.4}$$

The left side of (4.4) is always treated as a real matrix, if k_s and k_{s+1} are a pair of complex conjugate. Since product of orthogonal matrices is also an orthogonal matrix, assuming that

$$\boldsymbol{Q} = \boldsymbol{Q}_s \boldsymbol{Q}_{s+1} = \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-1} \tag{4.5}$$

(4.3) is converted to

$$\boldsymbol{A}_{s+2} = \boldsymbol{P}_{n-1}^{\mathrm{T}} \cdots \boldsymbol{P}_2^{\mathrm{T}} \boldsymbol{P}_1^{\mathrm{T}} \boldsymbol{A}_s \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-1} \tag{4.6}$$

P_1 is the transformation matrix which determines the element of the first column of upper triangular matrix $R_{s+1}R_s$ during the QR decomposition in (4.4). $P_i(i$=2,3,...,n-1) is a series of transformation matrices determined when using the Householder method to transform $P_1A_sP_1$to$\boldsymbol{A}_{s+2}$ in (4.6).

The process for the double QR method is described below.

1) (4.7) determines whether there any elements which can be regarded as relative zero among lower subdiagonal elements $a_{n\,n-1}$,......,$a_{21}^{(s)}$ of A_s

$$\left|a_{l\,l-1}^{(s)}\right| \le u\left(\left|a_{l-1\,l-1}^{(s)}\right| + \left|a_{ll}^{(s)}\right|\right), \quad l = n, n-1, \ldots, 2 \tag{4.7}$$

u is the unit round-off.

$a_{l\,l-1}^{(s)}$ is relative zero if it satisfies (4.7). If not, step 2 is performed.

(a) If $l = n$, $a_{n\,n}^{(s)}$ is adopted as an eigenvalue, order n of the matrix is reduced to n-1, and the process returns to step 1). If n=0, the process is terminated.

(b) If l=n-1, two eigenvalues are obtained from the lowest 2×2 principal submatrix, order n of the matrix is reduced to n-2, and the process returns to step 1.

(c) When 2≤ l <n-1, the matrix is split as shown in Fig.HSOR-1, the process proceeds to step 2, in this case, submatrix $\boldsymbol{D}$ is used for $\boldsymbol{A}_s$.

$$\begin{array}{c} \\ \\ l \\ \\ \\ \\ \\ \end{array}\left[\begin{array}{cc|ccccc} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ \hline & \varepsilon & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \\ & & & & & * & * \end{array}\right] \rightarrow \left[\begin{array}{c|c} \boldsymbol{B} & \boldsymbol{C} \\ \hline \boldsymbol{0} & \boldsymbol{D} \end{array}\right]$$

Note: Element ε is regarded as zero.

Fig. HSQR-1 A direct sum of submatrices for a Hassenberg matrix

2) The two eigenvalues of the lowest 2×2 principal submatrix are used for origin shifts. $\boldsymbol{K}_s$ and k_{s+1} are set to these values.

3) The first column of $(\boldsymbol{A}_s - k_s\boldsymbol{I})(\boldsymbol{A}_s - k_{s+1}\boldsymbol{I})$ on the left side of (4.4) is obtained. Since $\boldsymbol{A}$, is Hessenberg matrix, its first column $\boldsymbol{m}_1$ is

$$\boldsymbol{m}_1 = (x_1, y_1, x_1, 0, \ldots\ldots, 0)^{\mathrm{T}} \tag{4.8}$$

where

$$\left.\begin{aligned} x_1 &= a_{11}^{(s)2} - a_{11}^{(s)}(k_s + k_{s+1}) + k_s k_{s+1} + a_{12}^{(s)} a_{21}^{(s)} \\ y_1 &= a_{21}^{(s)}\left(a_{11}^{(s)} + a_{22}^{(s)} - k_s - k_{s+1}\right) \\ z_1 &= a_{32}^{(s)} a_{21}^{(s)} \end{aligned}\right\} \tag{4.9}$$

4) $\boldsymbol{P}_1$ is determined. If the first column of upper triangular matrix $\boldsymbol{R}_{s+1}\boldsymbol{R}_s$ is given by (4.10) and (4.11), $\boldsymbol{P}_1$ can be determined as shown in (4.12).

$$\boldsymbol{r}_1 = (\sigma_1, 0, \ldots, 0)^{\mathrm{T}} \tag{4.10}$$

$$\sigma_1 = \pm\sqrt{{x_1}^2 + {y_1}^2 + {z_1}^2} \tag{4.11}$$

$$\boldsymbol{P}_1 = \boldsymbol{I} - 2\boldsymbol{w}_1 {\boldsymbol{w}_1}^{\mathrm{T}} \tag{4.12}$$

$$\text{where,} \quad \boldsymbol{w}_1 = (\boldsymbol{m}_1 - \boldsymbol{r}_1) / \|\boldsymbol{m}_1 - \boldsymbol{r}_1\|_2 \tag{4.13}$$

To avoid cancellation the sign of σ_1 is taken to be the opposite of the sign of x_1.

5) $\boldsymbol{P}_1^{\mathrm{T}}\boldsymbol{A}_s\boldsymbol{P}_1$ is computed. The form of $\boldsymbol{P}_1^{\mathrm{T}}\boldsymbol{A}_s\boldsymbol{P}_1$ is shown in Fig. HSQR-2.

$$\left[\begin{array}{ccccccc} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \\ & & & & & * & * \end{array}\right]$$

Fig. HSQR-2 Form of $P_1^{\mathrm{T}} A_s P_1$

6) $\boldsymbol{P}_1^{\mathrm{T}}\boldsymbol{A}_s\boldsymbol{P}_1$ is reduced to a Hessenberg matrix using the Householder method. The process then returns to step 1. (Refer to the method section HSE1 for more about the Householder method.) By repeating this process, the lowest subdiagonal elements will converge to zero, and eigenvalues should be obtained using (a) and (b) of 1. If eigenvalues can not be obtained. In (a), (b) of 1 after 30 iterations, ICON is set to 15000 or 20000. For further information see References [12],[13] PP.359-371 and [16] PP.177-206.

B21-11-0502 HVEC,DHVEC

Eigenvectors of a real Hessenberg matrix (inverse iteration method)
CALL HVEC(A,K,N,ER,EI,IND,M,EV,MK,AW,ICON)

Function

Eigenvectors $\boldsymbol{x}$ which correspond to selected eigenvalues μ of an n-order real Hessenberg matrix $\boldsymbol{A}$ are obtained using the inverse iteration method. Eigenvectors are not normalized. $n \geq 1$.

Parameters

A...... Input. Real Hessenberg matrix.
A is a two-dimensional array, A(K,N)

K...... Input. Adjustable dimension of arrays A, EV, and AW.

N...... Input. Order n of the real Hessenberg matrix.

ER,EI.. Input. Eigenvalues μ.
The real parts of μ are stored in ER and the imaginary parts are stored in EI. The jth eigenvalue μ_j is represented as:

μ_j=ER(j)+i·EI(j)

If the j-th eigenvalue μ_j is complex, μ_j and μ_{j+1} should be a pair of complex conjugate eigenvalues. See Fig. HVEC-1. ER and EI are one-dimensional array of size M.

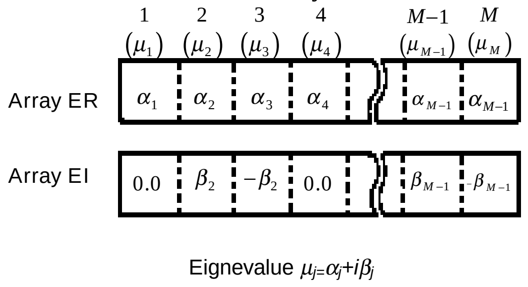

1 2 3 4 $M-1$ M
(μ_1) (μ_2) (μ_3) (μ_4) (μ_{M-1}) (μ_M)
Array ER
α_1 α_2 α_3 α_4 α_{M-1} α_{M-1}
Array EI
0.0 β_2 $-\beta_2$ 0.0 β_{M-1} $-\beta_{M-1}$
Eignevalue $\mu_j=\alpha_j+i\beta_j$

Fig. HVEC-1 Storage of eigenvalues

IND... Input. Indicate whether or not eigenvectors are to be determined. When the eigenvector which corresponds to the jth eigenvalue μ_j is to be determined, IND(J)=1. If it is not to be determined, IND(J)=0. See "Comments on use". IND is a one-dimensional array of size M.

M... Input. The number of eigenvalues stored in arrays ER and EI.

EV... Output. Eigenvectors $\boldsymbol{x}$.
Eigenvectors are stored in columns of EV. Real eigenvectors of real eigenvalues are stored in one column in EV. Complex eigenvectors of complex eigenvalues are split into the real and imaginary parts and stored in two consecutive column. See "Comments on use". EV is a two-dimensional array,EV(K,MK).

MK... Input. The number of columns of array EV.
See "Comments on use".

AW... Work area.
AW(K,N+4) is a two-dimensional array.

ICON... Output. Condition code.
See Table HVEC-1.

Table HVEC-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	EV(1,1)=1.0
15000	The engenvector of some eigenvalue could not be determined.	The IND information of eigenvectors that could not be determined is cleared to zero.
16000	There are not enough columns in array EV to store all eigenvectors that were requested.	Only as many eigenvectors as can be contained in array EV are computed. The IND information of those eigenvectors which could not be computed is set to zero.
20000	No eigenvectors could be determined.	All of the IND information is set to zero.
30000	M<1,N<M or K<N	Bypassed

Comments on use

- Subprograms used
  SSL II...AMACH and MGSSL.
  FORTRAN basic functions...ABS,SQRT,and SIGN

- Notes
  Parameter IND and storage of eigenvectors. If arrays ER, EI, and IND are specified as shown in Fig. HVEC-2, the eigenvectors which correspond to μ_1, μ_2, μ_3, μ_4, μ_5, μ_6, and μ_8 are computed and then stored in array EV as shown in the same figure.
  Since the eigenvectors which correspond to μ_3 and μ_7 are not computed, IND(3) and IND (7) are set to 0 by this routine. Also, since μ_2 and μ_6 are complex eigenvalues, IND(2) and IND(6) are set to -1.
  When IND is specified as shown in Fig. HVEC-3, the eigenvectors are successively stored in array EV from the first column. In this case, note that the eigenvectors are closely stored in order from the first column in EV.
  Based on the eigenvector storage described above, parameter MK should be set to the number of column required to contain the eigenvectors. If the actual number of columns required for the

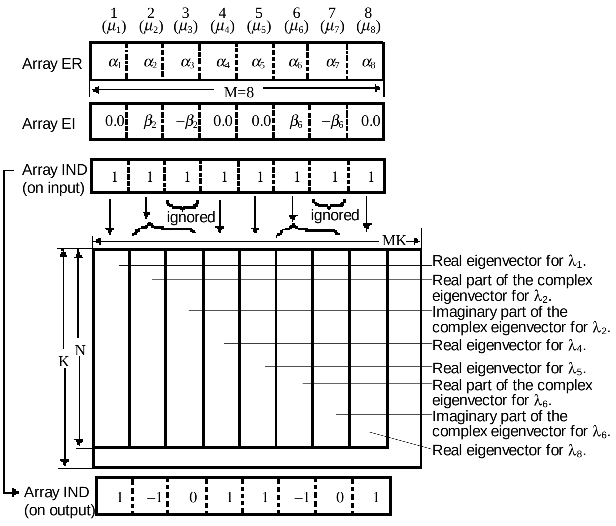
eigenvectors is larger than the number specified in MK, as many eigenvectors as can be stored in the number of columns specified in MK are computed, the rest are ignored and ICON is set to 16000.

The eigenvalues used by this routine can be determined by subroutine HSQR. The parameters ER, EI and M provided by subroutine HSQR can be input as the parameters for this subroutine.

When selected eigenvectors of a real matrix are to be determined:

– A real matrix is first transformed into a real Hessenberg matrix using subroutine HES1;
– Eigenvalues are determined using subroutine HSQR;
– Selected eigenvectors are determined using this routine; and
– Back transformation is applied to the above eigenvectors using subroutine HBK1 to obtain the eigenvectors of a real matrix.

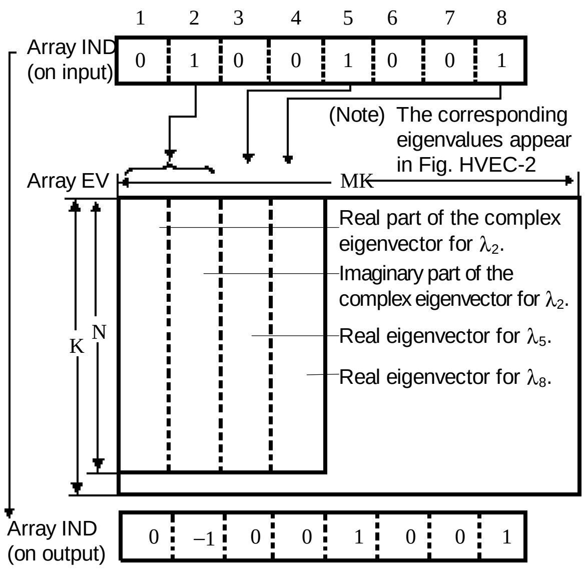
Note that subroutine EIG1 can be applied in order to obtain all the eigenvectors of a real matrix for convenience.



1 (μ_1) 2 (μ_2) 3 (μ_3) 4 (μ_4) 5 (μ_5) 6 (μ_6) 7 (μ_7) 8 (μ_8)
Array ER
α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8
M=8
Array EI
0.0 β_2 $-\beta_2$ 0.0 0.0 β_6 $-\beta_6$ 0.0
Array IND (on input)
1 1 1 1 1 1 1 1
ignored
ignored
MK
K
N
Real eigenvector for λ_1.
Real part of the complex eigenvector for λ_2.
Imaginary part of the complex eigenvector for λ_2.
Real eigenvector for λ_4.
Real eigenvector for λ_5.
Real part of the complex eigenvector for λ_6.
Imaginary part of the complex eigenvector for λ_6.
Real eigenvector for λ_8.
Array IND (on output)
1 −1 0 1 1 −1 0 1

Fig. HVEC-2 IND information and eigenvectors storage (Example 1)

– The resulting eigenvectors by this routine have not been normalized yet. Subroutine NRML should be used, if necessary, to normalize the eigenvectors.
– When using subroutines HBK1 or NRML, parameters IND, M, and EV of this subroutine can be used as their input parameters.



1 2 3 4 5 6 7 8
Array IND (on input)
0 1 0 0 1 0 0 1
(Note) The corresponding eigenvalues appear in Fig. HVEC-2
Array EV
MK
Real part of the complex eigenvector for λ_2.
Imaginary part of the complex eigenvector for λ_2.
Real eigenvector for λ_5.
Real eigenvector for λ_8.
K
N
Array IND (on output)
0 −1 0 0 1 0 0 1

Fig. HVEC-3 IND information and eigenvectors storage (Example 2)

- Example

The eigenvalues of an *n*-order real matrix are first obtained by subroutines HES1 and HSQR, then eigenvectors are obtained using this subroutine and HBK1. $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),AW(100,104),
     *ER(100),EI(100),IND(100),EV(100,100),
     *PV(100)
   10 CONTINUE
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510)((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL HES1(A,100,N,PV,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      MP=N
      DO 35 II=1,N
      I=1+N-II
      DO 30 J=1,MP
   30 AW(J,I)=A(J,I)
   35 MP=I
      CALL HSQR(AW,100,N,ER,EI,M,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      DO 40 I=1,M
   40 IND(I)=1
```

```
    CALL HVEC(A,100,N,ER,EI,IND,M,EV,100,
   *     AW,ICON)
    WRITE(6,620) ICON
    IF(ICON.GE.20000) GO TO 10
    CALL HBK1(EV,100,N,IND,M,A,PV,0.0,
   *     ICON)
    CALL EPRT(ER,EI,EV,IND,100,N,M)
    GO TO 10
500 FORMAT(I5)
510 FORMAT(5E15.7)
600 FORMAT('1',5X,'ORIGINAL MATRIX',5X,
   * 'N=',I3)
610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
   * E14.7))
620 FORMAT('0',20X,'ICON=',I5)
    END
```

Subroutine ERPT used in this example is for printing the eigenvalues and eigenvectors of the real matrix. For details, see the example in section EIG1.

Method

The inverse iteration method is used to determine eigenvectors $\boldsymbol{x}$ which correspond to eigenvalues of an n-order real Hessenberg matrix $\boldsymbol{A}$. In the inverse iteration method, if matrix $\boldsymbol{A}$ and approximation μ_j for eigenvalue of $\boldsymbol{A}$ are given, an appropriate initial vector $\boldsymbol{x}_0$ is used to solve equation (4.1) iteratively. When convergence conditions have been satisfied, $\boldsymbol{x}_r$ is determined as the resultant eigenvector.

$$(\boldsymbol{A}-\mu_j\boldsymbol{I})x_r=\boldsymbol{x}_{r-1},\qquad r=1,2,3,... \tag{4.1}$$

Where, r is the number of iterations, $\boldsymbol{x}_r$ is the vector determined at the r-th iteration.

Now let the eigenvalues of an n-order matrix $\boldsymbol{A}$ be $i(i=1,2,...,n)$, and the eigenvectors which correspond to λ_i be $\boldsymbol{u}_i$.

Since the appropriate initial vector $\boldsymbol{x}_0$ can be expressed, as shown in (4.2) by linear combination of $\boldsymbol{u}_r$, $\boldsymbol{x}_r$ can be written as shown in (4.3) if all eigenvalues λ_i are different.

$$\boldsymbol{x}_0=\sum_{i=1}^{n}\alpha_i\boldsymbol{u}_i \tag{4.2}$$

$$\boldsymbol{x}_r=1/(\lambda_j-\mu_j)^r\Big[\alpha_i\boldsymbol{u}_i+\sum_{\substack{i=1\\ i\neq j}}^{n}\alpha_i\boldsymbol{u}_i(\lambda_j-\mu_j)^r/(\lambda_j-\mu_j)^r\Big] \tag{4.3}$$

Since $1/(\lambda_j-\mu_j)^r$ is a constant, it can be omitted and (4.3) is rewritten as:

$$\boldsymbol{x}_r=\alpha_i\boldsymbol{u}_i+\sum_{\substack{i=1\\ i\neq j}}^{n}\alpha_i\boldsymbol{u}_i(\lambda_j-\mu_j)^r/(\lambda_j-\mu_j)^r \tag{4.4}$$

Since in general $\left|(\lambda_j-u_j)/(\lambda_j-u_j)\right|<<1.0$, (4.4) indicates that, if $\alpha_j\neq 0$, as r grows greater, x_r tends rapidly to $\alpha_j\mu_j$.

The system of linear equations shown in (4.1) are solved using (4.5) after decomposition of $(\boldsymbol{A}-\mu_j\boldsymbol{I})$ to a lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{LUx}_r=\boldsymbol{Px}_{r-1} \tag{4.5}$$

Where, $\boldsymbol{P}$ is the permutation matrix used for pivoting. (4.5) can be solved as follows.

$$\boldsymbol{Ly}_{r-1}=\boldsymbol{Px}_{r-1}\text{(forwardsubstitution)} \tag{4.6}$$

$$\boldsymbol{Ux}_r=\boldsymbol{y}_{r-1}\text{(backwardsubstitution)} \tag{4.7}$$

Since any vector may be used for initial vector $\boldsymbol{x}_0$, $\boldsymbol{x}_0$ may be given such that $\mathbf{y}_0$ of (4.8) has a form such as $\boldsymbol{y}_0=(1,1,1,...,1)^{\mathrm{T}}$.

$$\boldsymbol{y}_0=\boldsymbol{L}^{-1}\boldsymbol{Px}_0 \tag{4.8}$$

Therefore, for the first iteration, the forward substitution in (4.6) can be omitted. In general, the eigenvectors can be obtained by repeating forward substitution and backward substitution for the second and following iterations. The following apply to this routine:

1) Selection of the initial vector
   $\boldsymbol{y}_0$ in (4.10) is used for the initial vector.

$$\boldsymbol{y}_0=(\mathrm{EPS1},\mathrm{EPS1},...,\mathrm{EPS1})^{\mathrm{T}} \tag{4.10}$$

where

$$\mathrm{EPS1}=u\|\boldsymbol{A}\|_\infty \tag{4.11}$$

And u is the unit round-off

2) Method for convergence criterion
   After backward substitution, the condition in (4.12) is tested to determine whether the eigenvectors have been accepted.

$$\|\boldsymbol{x}_r\|_1\geq 0.1/\sqrt{n} \tag{4.12}$$

   If (4.12) is satisfied, $\mathbf{x}_\mathrm{r}$ is accepted as the eigenvectors. If (4.12) is not satisfied, the initial vector is regarded inappropriate. A new initial vector is substituted and backward substitution is continued.
3) When eigenvalues have multiple roots or close roots
   If eigenvalue μ_j whose eigenvector is to be determined and eigenvalue u_j whose eigenvector has already been computed satisfy the condition in (4.13), correct eigenvectors will not be determined no matter how many iterations are performed.

$$|\mu_j-\mu_i|\leq\mathrm{EPS1}(i=1,2,...,j-1) \tag{4.13}$$

if $\lambda_j = \lambda_i$ then from (4.4) we have:

$$\boldsymbol{x}_1 = \alpha_j \boldsymbol{u}_j + \alpha_i \boldsymbol{u}_i + \sum_{k=1}^{n} \alpha_k \boldsymbol{u}_k \left(\lambda_j - \mu_j\right) / \left(\lambda_k - \mu_j\right)$$
$$\left(k \neq i, j\right) \tag{4.14}$$

Therefore, as long as the same initial vector is used, eigenvectors $\boldsymbol{x}_1^{(j)}$ and $\boldsymbol{x}_1^{(i)}$ which are computed corresponding to μ_j and μ_i will be approximately equal.
in such situations, this routine adjusts μ_j in EPS1 units, and when (4.15) is satisfied, $\tilde{\mu}_j$ is used to compute the eigenvectors.

$$\left|\tilde{\mu}_j - \mu_i\right| > \mathrm{EPS1}\left(i = 1, 2, \ldots, j-1\right) \tag{4.15}$$

For further information, see References [12] and [13]. PP.418-439.

E51-30-0401 ICHEB, DICHEB

Indefinite integral of a Chebyshev series
CALL ICHEB(A,B,C,N,ICON)

Function

Given an n-terms Chebyshev series defined on interval $[a,b]$

$$f(x)=\sum_{k=1}^{n-1}{}' c_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \tag{1.1}$$

this subroutine computes its indefinite integral in a Chebyshev series

$$\int f(x)dx=\sum_{k=0}^{n}{}' \bar{c}_k T_k\left(\frac{2x-(b+a)}{b-a}\right) \tag{1.2}$$

and obtains its coefficient $\{\bar{c}_k\}$. Where arbitrary constant $\{\bar{c}_0\}$ is assumed to be zero. Σ' denotes to make sum but the first term only is multiplied by factor 1/2.
Where $a\neq b$ and $n\geq 1$.

Parameters

A......	Inputs. Lower limit a of the interval for the Chebyshev series.
B......	Input. Upper limit b of the interval for the Chebyshev series.
C......	Input. Coefficients$\{c_k\}$. Stored as C(1)=c_0,C(2)=c_1,..., C(N)=c_{n-1}. Output. Coefficients $\{\bar{c}_k\}$ for the indefinite integral. Stored as C(1)=0.0, C(2)= $\bar{c}_1$,..., C(N+1)= $\bar{c}_n$. One-dimensional array of size N+1.
N......	Input. Number of terms n. Output. Number of terms n+1 of the indefinite integral.
ICON...	Output. Condition code. See Table ICHEB-1.

Table ICHEB-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	Either of the following occurred. 1 N<1 2 A=B	Bypassed

Comments on use

- Subprograms used
  SSL II.....MGSSL
  FORTRAN basic function...FLOAT

- Notes
  When a indefinite integral of an arbitrary function is required, this subroutine can be used together with the subroutine FCHEB for Chebyshev series expansion. The ECHEB subroutine used for evaluation of Chebyshev series can be subsequently called to obtain the integral value at point $v\in[a,b]$ in an interval.

  Determination of arbitrary constant $\bar{c}_0$ of an integral this subroutine outputs zero as arbitrary constant $\bar{c}_0$. If a constant is defined so that an indefinite integral at point $v\in[a, b]$ in an interval is y_v, it should be computed by using the subroutine ECHEB use for evaluation of Chebyshev series as follows:

```
       :
  CALL ICHEB(A,B,C,N,ICON)
  CALL ECHEB(A,B,C,N,V,Y,ICON)
  C(1)=(YV-Y)*2.0
       :
```

  where C(1) corresponds to c_0, V corresponds to v and YV corresponds to yv respectively. To obtain the definite integral

$$\int_a^{x_i} f(t)dt,\quad x_i\in[a,b],\quad i=1,2,...,m \tag{3.1}$$

  by changing the upper limit of the integral interval for function $f(x)$, it should determines at first the value of arbitrary constant $\bar{c}_0$. So that the value of the indefinite integral at end point a is zero and then should calls the subroutine ECHEB m times repeatedly. (See Example).

  The error in an indefinite integral can be estimated by the absolute sum of the last two term coefficients.

- Example
  The value of function

$$\tilde{f}(x)=\frac{1}{4}+\int_0^x\frac{dt}{1+100t^2},\quad x\in[0,1] \tag{3.2}$$

  is obtained with increment 0.05 starting from x=0
  The subroutine FCHEB expands the integrand in Chebyshev series (required accuracy: absolute error $5\cdot 10^{-5}$). Then this subroutine obtains an indefinite integral. Integral constant is determined so that the value of an indefinite integral where x=0 is 1/4
  The subroutine ECHEB obtains the function value at each step and computes the error for true value

$$f(x)=\frac{1}{4}+\frac{1}{10}\tan^{-1}10x \qquad (3.3)$$

```
C      **EXAMPLE**
       DIMENSION C(258),TAB(255)
       EXTERNAL FUN
       EPSA=5.0E-5
       EPSR=5.0E-5
       NMIN=9
       NMAX=257
       A=0.0
       B=1.0
       CALL FCHEB(A,B,FUN,EPSA,EPSR,NMIN,
      *           NMAX,C,N,ERR,TAB,ICON)
       IF(ICON.NE.0) GO TO 20
       WRITE(6,600) N,ERR,ICON
       WRITE(6,601) (C(K),K=1,N)
       CALL ICHEB(A,B,C,N,ICON)
       IF(ICON.NE.0) GO TO 20
       WRITE(6,602)
       WRITE(6,601) (C(K),K=1,N)
       CALL ECHEB(A,B,C,N,A,F,ICON)
       IF(ICON.NE.0) GO TO 20
       C(1)=(0.25-F)*2.0
       WRITE(6,603)
       H=0.05
       X=A
    10 CALL ECHEB(A,B,C,N,X,Y,ICON)
       IF(ICON.NE.0) GO TO 20
       ERROR=G(X)-Y
       WRITE(6,604) X,Y,ERROR
       X=X+H
       IF(X.LE.B) GO TO 10
       STOP
    20 WRITE(6,604) ICON
       STOP
   600 FORMAT('0',3X,'EXPANSION OF',
      1' FUNCTION FUN(X)',3X,'N=',I4,3X,
      2'ERROR=',E13.3,3X,'ICON=',I6)
   601 FORMAT(/(5E15.5))
   602 FORMAT('0',5X,'INTEGRATION OF',
      1' CHEBYSHEV SERIES')
   603 FORMAT('0',10X,'X',7X,
      1'INTEGRATION ',6X,'ERROR'/)
   604 FORMAT(1X,3E15.5)
   605 FORMAT('0',5X,'CONDITION CODE',I8)
       END
       FUNCTION FUN(X)
       FUN=1.0/(1.0+100.0*X*X)
       RETURN
       END
       FUNCTION G(X)
       G=0.25+ATAN(10.0*X)/10.0
       RETURN
       END
```

Method

This subroutine performs termwise indefinite integral of an n-terms Chebyshev series defined on interval $[a,b]$ and expresses it in a Chebyshev series.

Given

$$\int \sum_{k=0}^{n-1}{}' c_k T_k\left(\frac{2x-(b-a)}{b-a}\right)dx = \sum_{k=0}^{n} \bar{c}_k T_k\left(\frac{2x-(b-a)}{b-a}\right) \qquad (4.1)$$

Substituting the following to the left side of (4.1),

$$\begin{aligned}
&\int T_0(y)dx = \frac{b-a}{2}T_1(y)\\
&\int T_1(y)dx = \frac{b-a}{2}T_2(y)\\
&\int T_k(y)dx = \frac{b-a}{2}\left\{\frac{T_{k+1}(y)}{k+1}-\frac{T_{k-1}(y)}{k-1}\right\},\quad k\ge 2
\end{aligned} \qquad (4.2)$$

where the integral constant is omitted and $y=\dfrac{2x-(b+a)}{b-a}$, the following relation is established for coefficients$\{c_k\}$and $\{\ \bar{c}_k\ \}$.

$$\begin{aligned}
&\bar{c}_{n+1} = 0\\
&\bar{c}_n = \frac{b-a}{4n}c_{n-1}\\
&\bar{c}_k = \frac{b-a}{4k}(c_{k-1}+c_{k+1}),\\
&\qquad k = n-1, n-2, \ldots, 1
\end{aligned} \qquad (4.3)$$

This subroutine obtains $\{\ \bar{c}_k\ \}$ by using an integral formula for Chebyshev polynomial in (4.3).

Zero is used for arbitrary constant $\bar{c}_0$.

n times multiplications and divisions are required for obtaining an indefinite integral of n-terms series.

I11-71-0301 IERF, DIERF

Inverse error function $\mathrm{erf}^{-1}(x)$
CALL IERF (X,F,ICON)

Function

This subroutine evaluates the inverse function $\mathrm{erf}^{-1}(x)$ of the error function $\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$, using the minimax approximation formulas of the form of the polynomial and rational functions.
Limited $|x|<1$.

Parameters

X......	Input. Independent variable x.
F......	Output. Function value $\mathrm{erf}^{-1}(x)$.
ICON...	Output. Condition code. See Table IERF-1.

Table IERF-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	|X|≥1	F is set to 0.0.

Comments on use

- Subprograms used
  SSL II...IERFC,MGSSL
  FORTRAN basic functions...ABS,SQRT,ALOG

- Notes
  The range of argument X is limited as |X|<1.
  This range is the definition area of this function.
  When considering the relationship $\mathrm{erf}^{-1}(x)=\mathrm{erfc}^{-1}(1-x)$ the inverse error function may be evaluated by subroutine IERFC. But, if values of x are within the range $|x|\le 0.8$, higher accuracy and less computation time are accomplished by the IERF subroutine.

- Example
  A table of function values computed at x from 0 to 0.99 with step-size 0.01 is made for $\mathrm{erf}^{-1}(x)$.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,100
       X=FLOAT(K-1)/100.0
       CALL IERF(X,F,ICON)
       IF(ICON.EQ.0)WRITE(6,610)X,F
       IF(ICON.NE.0)WRITE(6,620)X,F,ICON
   10 CONTINUE
       STOP
  600 FORMAT('1','EXAMPLE OF INVERSE ERROR',
     * ' FUNCTION'///6X,'X',7X,'IERF(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'IERF=',E17.7,5X,
     * 'CONDITION=',I10)
       END
```

Method

Some approximation formulas are used for evaluation of the inverse error function $\mathrm{erf}^{-1}(x)$, corresponding if $|x|\le 0.8$ or not.

- $|x|\le 0.8$
  The minimax approximation in the sense of relative error based on the Taylor series expansion (See Reference[87]) of $\mathrm{erf}^{-1}(x)$ at x=0 is used.
  Single precision:

$$\mathrm{erf}^{-1}(x)=x\cdot\sum_{k=0}^{3}a_kt^k \Big/ \sum_{k=0}^{4}b_kt^k \qquad (4.1)$$
$$t=\left(1-(x/0.8)^2\right)$$

  Theoretical precision = 9.2 digits
  Double precision:

$$\mathrm{erf}^{-1}(x)=x\cdot\sum_{k=0}^{7}a_kt^k \Big/ \sum_{k=0}^{8}b_kt^k \qquad (4.2)$$
$$t=\left(1-(x/0.8)^2\right)$$

  Theoretical precision=18.8 digits

- $|x|>0.8$
  Considering the relationship $\mathrm{erf}^{-1}(1-x)=\mathrm{erfc}^{-1}(1-x)$, it is computed by subroutine IERFC. For details refer to the descriptions of IERFC.

I11-71-0401 IERFC,DIERFC

Inverse complementary error function $\mathrm{erfc}^{-1}(x)$
CALL IERFC(X,F,ICON)

Function

This subroutine evaluates the inverse function $\mathrm{erfc}^{-1}(x)$ of the complementary error function $\mathrm{erfc}\,(x)=\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt$ using the minimax approximations of the form of the polynomial and rational functions.
Limited $0<x<2$.

Parameters

X......	Input. Independent variable x.
F......	Input. Function value $\mathrm{erfc}^{-1}(x)$.
ICON...	Output. Condition code. See Table IERFC-1.

Table IERFC-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	X≤0 or X≥2	F is set to 0.0.

Comments on use

- Subprograms used
  SSL II...IERF,MGSSL
  FORTRAN basic functions...ABS, SQRT, ALOG

- Notes
  The range of argument X is limited as 0<X<2. This range is the definition area of this function. When considering the relationship $\mathrm{erfc}^{-1}(x)=\mathrm{erf}^{-1}(1-x)$ the inverse complementary error function may be evaluated by subroutine IERF. But, if values of x are within the range $0<x<0.2$, higher accuracy and less computation time are accomplished by this IERFC.

- Example
  A table of function values computed at x from 0.01 to 1.00 with stepsize 0.01 is made for $\mathrm{erfc}^{-1}(x)$

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,100
      X=FLOAT(K)/100.0
      CALL IERFC(X,F,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,F
      IF(ICON.NE.0) WRITE(6,620) X,F,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF INVERSE',
     * ' COMPLEMENTARY ERROR FUNCTION',
     * ///6X,'X',6X,'IERFC(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'IERFC=',E17.7,5X,
     * 'CONDITION=',I10)
      END
```

Method

Some approximation formulas are used for evaluation of inverse complementary error function erfc $^{-1}$(x), corresponding if $0<x<0.2$ or if $1.8<x<2$, or other.

- $0<x<0.2$ or $1.8<x<2$

Placing $\beta=\sqrt{-\log(x(2-x))}$ and $\mathrm{erfc}^{-1}(x)=\beta\cdot R(\beta)$, the auxiliary function R(β) is evaluated using the minimax approximation in the sense of relative error base on the Strecok theory (See Reference [87]).

(a) $0<x<5.10^{-16}$ or $(2-5\cdot10^{-16})<x<2$
Letting $c\Big/\sqrt{\beta+d}$
Single precision:

$$\mathrm{erfc}^{-1}(x)=\beta\cdot\sum_{k=0}^{7}a_kt^k \tag{4.1}$$

Theoretical precision=9.7 digits
where $c=-4.5999961$
$d=1.8974954$

Double precision:

$$\mathrm{erfc}^{-1}(x)=\beta\cdot\sum_{k=0}^{16}a_kt^k \tag{4.2}$$

Theoretical precision -18.9 digits
where $c=-4.59999617941507552$
$d=1.89749541006229975$

(b) $5.10^{-16}\le x<2.5\cdot10^{-3}$ or $2\text{-}2.510^{-3}<x\le2-5\cdot10^{-16}$
$t=c\cdot\beta+d$
Single precision:

$$\mathrm{erfc}^{-1}(x)=\beta\cdot\sum_{k=0}^{4}a_kt^k\Big/\sum_{k=0}^{4}b_kt^k \tag{4.3}$$

Theoretical precision = 8.4 digits
where $c=0.27972881$
$d=-0.64395786$

Double precision:

$$\mathrm{erfc}^{-1}(x)=\beta\cdot\sum_{k=0}^{11}a_kt^k\Big/\sum_{k=0}^{11}b_kt^k \tag{4.4}$$

Theoretical precision=18.7 digits
where $c=0.279728815664916161$
$d=-0.643957858131678820$

(c) $2.5\ 10^{-3} \le x < 0.2$ or $1.8 < x \le 1.9975$
Letting $t = c \cdot b + d$
Single precision:

$$\mathrm{erfc}^{-1}(x) = \beta \cdot \sum_{k=0}^{7} a_k t^k \qquad (4.5)$$

Theoretical precision = 8.9 digits
where $c = -0.77440652$
$d = 1.7827451$

Double precision:

$$\mathrm{erfc}^{-1}(x) = \beta \cdot \sum_{k=0}^{20} a_k t^k \qquad (4.6)$$

Theoretical precision = 18.5 digits
where $c = -0.774406521186630830$
$d = 1.78274506157390808$

- $0.2 \le x \le 1.8$
  Considering the relationship $\mathrm{erfc}^{-1}(x) = \mathrm{erf}^{-1}(1-x)$, it is computed by subroutine IERF. For details refer to the descriptions of IERF.

I11-61-0101 IGAM1, DIGAM1

Incomplete Gamma Function of the first kind $\gamma(v,x)$
CALL IGAM1(V,X,F,ICON)

Function

This subroutine computes incomplete Gamma function of the first kind,

$$\gamma(v,x)=\int_0^x e^{-t}t^{v-1}dt$$

by series expansion, asymptotic expansion, and numerical integration, where $v>0$, $x\geq 0$.

Parameters

V......	Input. Independent variable v.
X......	Input. Independent variable x.
F......	Output. Value of $\gamma(v, x)$.
ICON...	Output. Condition code. See Table IGAM1-1

Table IGAM1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	V≤0 or X<0	F=0.0

Comments on use

- Subprograms used
  SSL II...AFMAX,AMACH,EXPI,IGAM2,MGSSL,and ULMAX
  FORTRAN basic functions...FLOAT,EXP, and GAMMA

- Notes
  When X≥23.0(for single precision) or X≥46.0(for double precision), the value of $\gamma(v, x)$ may be obtained simply by the complete GAMMA(v) function in the FORTRAN basic function, because $\gamma(v,x) \approx \Gamma(v)$ in the above ranges.

- Example
  The following example generates a table of $\gamma(v,x)$ for $v,x=a+b$, where a=0,1,2 and b=0.1,0.2,0.3.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 20 IV=1,9
      V=FLOAT(IV+7*((IV-1)/3))/10.0
      DO 10 IX=1,9
      X=FLOAT(IX+7*((IX-1)/3))/10.0
      CALL IGAM1(V,X,F,ICON)
      IF(ICON.EQ.0) WRITE(6,610) V,X,F
      IF(ICON.NE.0) WRITE(6,620) V,X,F,ICON
   10 CONTINUE
   20 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF',
     *' INCOMPLETE GAMMA FUNCTION'
     *//5X,'V',9X,'X',10X,'FUNC'/)
  610 FORMAT(' ',2F10.5,E17.7)
  620 FORMAT(/' ','** ERROR **',5X,'V=',
     *F10.5,5X,'X=',F10.5,5X,'IGAM1=',
     *E17.7,5X,'CONDITION=',I10/)
      END
```

Methods

Two different approximation formulas are used depending upon the ranges of x divided at x_1=3.5(or 5.5 for double precision).

- For $x\leq 2(v+1)$ or $x<x_1$
  The computation is based on

$$\gamma(v,x)=x^v e^{-x}\left\{1+\frac{x}{v+1}+\frac{x^2}{(v+1)(v+2)}+\ldots\right\}/v \quad (4.1)$$

The value of $\gamma(v,x)$ is computed as a partial sum of the first N terms with N being taken large enough so that the last term no longer affects the value of the partial sum significantly.

- For $x>2(v+1)$and $x\geq x_1$
  The computation is based on

$$\gamma(v,x)=\Gamma(v)-\Gamma(v,x) \quad (4.2)$$

where $\Gamma(v)$ is computed by calling GAMMA, which is one of the FORTRAN basic functions, and F(v,x) is computed by subroutine IGAM2.

For further information, see Reference [85] pp.14-16

I11-61-0201 IGAM2, DIGAM2

Incomplete Gamma Function of the second kind $\Gamma(v,x)$
CALL IGAM2(V,X,F,ICON)

Function

This subroutine computes incomplete Gamma function of the second kind

$$\Gamma(v,x)=\int_x^\infty e^{-t}t^{v-1}dt=e^{-x}\int_0^\infty e^{-t}(x+t)^{v-1}dt$$

by series expansion, asymptotic expansion, and numerical integration, where $v\geq 0, x\geq 0 (x\neq 0$ when $v=0)$.

Parameters

V......	Input. Independent variable v.
X......	Input. Independent variable x.
F......	Output. Value of $\Gamma(v.x)$.
ICON...	Output. Condition code. See Table IGAM2-1

Table IGAM2-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$x^{v-1}e^{-x}>fl_{max}$	F=fl_{max}
30000	V<0,X<0 or V=0 and X=0	F=0.0

Comments on use

- Subprograms used
  SSL II...AFMAX,AMACH,MGSSL,EXPI, and ULMAX
  FORTRAN basic functions...FLOAT,EXP,GAMMA, and ATAN

- Notes
  For X≥log(fl_{max}), the value or $\Gamma(v,x)$ becomes smaller enough to underflow.
  The condition, $x^{v-1}e^{-x}>\log(fl_{max})$ will hold when $x>1$ and v is very large. Then, the value of $\Gamma(v,x)$ overflows.

- Example
  The following example generates a table of $\Gamma(v,x)$ for $v,x=a+b$, where a=0,1,2 and b=0.1,0.2,0.3.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 20 IV=1,9
       V=FLOAT(IV+7*((IV-1)/3))/10.0
       DO 10 IX=1,9
       X=FLOAT(IX+7*((IX-1)/3))/10.0
       CALL IGAM2(V,X,F,ICON)
       IF(ICON.EQ.0) WRITE(6,610) V,X,F
       IF(ICON.NE.0) WRITE(6,620) V,X,F,ICON
    10 CONTINUE
    20 CONTINUE
       STOP
   600 FORMAT('1','EXAMPLE OF',
      *' INCOMPLETE GAMMA FUNCTION'
      *//5X,'V',9X,'X',10X,'FUNC'/)
   610 FORMAT(' ',2F10.5,E17.7)
   620 FORMAT(/' ','** ERROR **',5X,'V=',
      *F10.5,5X,'X=',F10.5,5X,'IGAM2=',
      *E17.7,5X,'CONDITION=',I10/)
       END
```

Methods

Since $\Gamma(v,x)$ is reduced to an exponential integral when v=0 and x>0, subroutine EXPI is used; where as it is the complete Gamma function when v>0 and x=0, so the FORTRAN basic function GAMMA is used. The calculation procedures for v>0 and x>0 are described below.
Two different approximation formulas are used depending upon the ranges of x divided at x_1=20.0(or 40.0 for double precision).

- When v=integer or $x>x_1$
  The computation is based on the following asymptotic expansion:

$$\Gamma(v,x)=x^{v-1}e^{-x}\left\{1+\frac{v-1}{x}+\frac{(v-1)(v-2)}{x^2}+\cdots+\frac{(v-1)\cdots(v-n)}{x^n}+\cdots\right\} \quad (4.1)$$

The value of $\Gamma(v,x)$ is computed as a partial sum of the first n terms, either with n being taken large enough so that the last term no longer affects the value of the partial sum significantly, or with n being take as the largest integer to satisfy $n\leq v+x$.

- When $v\neq$integer and $x\leq x_1$
  The computation is based on the following representation:

$$\Gamma(v,x)=x^{v-1}e^{-x}\left\{1+\frac{v-1}{x}+\cdots+\frac{(v-1)(v-2)\cdots(v-m+1)}{x^{m-1}}\right\}+(v-1)(v-2)\cdots(v-m)\Gamma(v-m,x) \quad (4.2)$$

Where m is the largest integer to satisfy $m\leq v$. When v<1, the firt term is regarded as 0, and (v-1)(v-2)......(v-m)=1. And, because 0<v-m<1, $\Gamma(v-m,x)$, in the second term can be expressed as:

$$\Gamma(v-m,x)=e^{-x}\int_0^\infty e^{-t}(x+t)^{v-m-1}dt \qquad (4.3)$$

This integral, since function(x+1)$^{v-m-1}$ has a certain singularity, can be calculated efficiently by the double exponential formula for numerical integration. (See Reference [68].) For further information, see Reference [85].

I11-91-0301 INDF, DINDF

Inverse normal distribution function $\phi^{-1}(x)$
CALL INDF(X,F,ICON)

Function

This subroutine computes the value of inverse function $\phi^{-1}(x)$ of normal distribution

function $\phi(x)=\frac{1}{\sqrt{2\pi}}\int_0^x e^{-t^2/2}dt$ by the relation,

$$\phi^{-1}(x)=\sqrt{2}\operatorname{erf}^{-1}(2x) \tag{1.1}$$

where $|x|<1/2$.

Parameters

X......	Input. Independent variable x
F......	Output. Function value $\phi^{-1}(x)$
ICON...	Output. Condition code See Table INDF-1

Table INDF-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	|x| ≥ ½	F=0.0

Comments on use

- Subprograms used
  SSL II...IERF,IERFC,MGSSL
  FORTRAN basic functions...ABS,SQRT,ALOG

- Notes
  $|x|<1/2$
  The value of $\phi^{-1}(x)$ can be obtained by the subroutine INDFC if the following relation is used.

$$\phi^{-1}(x)=\psi^{-1}(1/2-x) \tag{3.1}$$

Note that in the range of $|x|\le 0.4$, however, this leads to less accurate and less efficient computation than calling INDF.

- Example
  The following example generates a table of $\phi^{-1}(x)$ in which x varies from 0.0 to 0.49 with increment 0.01.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,50
      X=FLOAT(K-1)/100.0
      CALL INDF(X,F,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,F
      IF(ICON.NE.0) WRITE(6,620) X,F,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF INVERSE NORMAL'
     *,' DISTRIBUTION FUNCTION'
     *//6X,'X',7X,'INDF(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',E17.7
     *,5X,'INDF=',E17.7,5X,'CONDITION=',I10)
      END
```

Method

There exists the relation (4.1) between $\phi(t)$ and erf(t), the error function.

$$\phi(t)=\operatorname{erf}\left(t/\sqrt{2}\right)/2, -\infty<t<\infty \tag{4.1}$$

Letting both sides of (4.1) be x, we see t=$\phi^{-1}(x)$ from $\phi(t)$=x, or $t=\sqrt{2}\operatorname{erf}^{-1}(x)$ from $\operatorname{erf}\left(t/\sqrt{2}\right)/2=x$.

Therefore

$$\phi^{-1}(x)=\sqrt{2}\operatorname{erf}^{-1}(2x) \tag{4.2}$$

This subroutine computes $\phi^{-1}(x)$ based on (4.2) by using the subroutine IERF.

I11-91-0401 INDFC, DINDFC

Inverse complementary normal distribution function $\psi^{-1}(x)$
CALL INDFC(X,F,ICON)

Function

This subroutine computes the value of inverse function $\psi^{-1}(x)$ of complementary normal distribution function $\psi(x)=\frac{1}{\sqrt{2\pi}}\int_x^\infty e^{t^2/2}dt$ by the relation,

$$\psi^{-1}(x)=\sqrt{2}\mathrm{erfc}^{-1}(2x) \tag{1.1}$$

$0<x<1$

Parameters

X......	Input. Independent variable x
F......	Output. Function value $\psi^{-1}(x)$
ICON...	Output. Condition code See Table INDFC-1

Table INDFC-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	X≤0 or X≥1	F=0.0

Comments on use

- Subprograms used
  SSL II...IERFC,IERFC,MGSSL
  FORTRAN basic functions...ABS,SQRT,ALOG

- Notes

$0<x<1$.

The value of $\psi^{-1}(x)$ can be obtained by the subroutine INDF if the following relation is used.

$$\psi^{-1}(x)=\phi^{-1}(1/2-x) \tag{3.1}$$

Note that in the range of $0<x<0.1$, however, this leads to less accurate and less efficient computation than calling INDFC.

- Example
  The following example generates a table of $\psi^{-1}(x)$ in which x varies from 0.01 to 0.50 with increment 0.01.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,50
      X=FLOAT(K)/100.0
      CALL INDFC(X,F,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,F
      IF(ICON.NE.0) WRITE(6,620) X,F,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF INVERSE COMPL',
     * 'EMENTARY NORMAL DISTRIBUTION ',
     * 'FUNCTION'//6X,'X',7X,'INDFC(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'INDFC=',E17.7,5X,
     * 'CONDITION=',I10)
      END
```

Method

There exists the relation (4.1) between $\psi(t)$ and erf(t), the error function.

$$\psi(t)=\mathrm{erfc}\left(t/\sqrt{2}\right)/2, -\infty<t<\infty \tag{4.1}$$

Letting both sides of (4.1) be x, we see $t=\psi^{-1}(x)$ from $\psi(t)=x$ or $t=\sqrt{2}\mathrm{erfc}^{-1}(2x)$ from $\mathrm{erfc}\left(t/\sqrt{2}\right)/2=x$. Therefore

$$\psi^{-1}(x)=\sqrt{2}\mathrm{erfc}^{-1}(2x) \tag{4.2}$$

This subroutine computes $\psi^{-1}(x)$ based on (4.2) by using the subroutine IERFC subroutine.

E12-21-0101 INSPL, DINSPL

Cubic spline interpolation coefficient calculation
CALL INSPL(X,Y,DY,N,C,D,E,ICON)

Function

Given discrete points x_1, x_2, ..., x_n ($x_1<x_2<...<x_n$), their corresponding function values $y_i=f(x_i)$, i=1,...,n, and the second derivatives at both ends y_1'' and y_m'', this subroutine obtains the interpolating cubic spline represented as (1.1) below to $f(x)$.

$$S(x)=\begin{cases} \text{When} \quad x<x_1 \\ y_1+c_1(x-x_1)+d_1(x-x_1)^2 \\ x_i \le x \le x_{i+1}, i=1,...,n-1 \ \text{When} \\ y_i+c_i(x-x_i)+d_i(x-x_i)^2 \\ +e_i(x-x_i)^3 \\ \text{When} \quad x>x_n \\ y_n+c_n(x-x_n)+d_n(x-x_n)^2 \end{cases} \tag{1.1}$$

Parameters

X	Input. Discrete points x_i. One-dimensional array of size n.
Y	Input. Function values y_i. One-dimensional array of size n.
DY	Input. 2nd order derivatives y_1'' and y_n'' at both ends. One-dimensional array of size 2. DY(1) is set to y_1'' and DY(2) is set to y_n''. (See Comments)
N	Input. Number (n) of discrete points. n≥2.
C	Output. Coefficients c_i of (1.1) One-dimensional array of size n.
D	Output, Coefficients d_i of (1.1) One-dimensional array of size n.
E	Output. Coefficients e_i of (1.1) Usually E(N)=0.0 One-dimensional array of size n.
ICON ..	Output. Condition code. See Table INSPL-1.

Table INSPL-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	$n<2$ or $X_i \ge X_{i+1}$	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... none

- Notes
  There are no restrictions on specifying 2nd order derivatives y''_i and y''_n at both ends. However, when those values are unknown it is possible to specify y_i''=0.0 and y_n''=0.0. In that case the integral $\int_{x_1}^{x_n}[S''(x)]^2 dx$ will be minimized. Where $S''(x)$s 2nd order derivative of the cubic spline that is to be determined.
  If the function $f(x)$ to be interpolated can be assumed to have periodic intervals of x_n-x_1, it is best to specify 2nd derivatives such that $y''_i=y''_n$.

- Example
  The number of discrete points n, discrete points x_i, and function values y_i, i=1,...,n are input. The cubic spline function is determined in order to interpolate a value at $x=v$ in the interval $[x_i,x_{i+1}]$.
  y''_i=0.0, y''_n=0.0, n≤10.

```
C     **EXAMPLE**
      DIMENSION X(10),Y(10),DY(2),C(10),
     *          D(10),E(10)
      READ(5,500) N
      READ(5,510) (X(I),Y(I),I=1,N)
      DY(1)=0.0
      DY(2)=0.0
      CALL INSPL(X,Y,DY,N,C,D,E,ICON)
      WRITE(6,600) ICON
      IF(ICON.GT.10000) STOP
      READ(5,520) I,V
      XX=V-X(I)
      YY=Y(I)+(C(I)+(D(I)+E(I)*XX)*XX)*XX
      WRITE(6,610) YY
      STOP
  500 FORMAT(I2)
  510 FORMAT(2E16.8)
  520 FORMAT(I2,E16.8)
  600 FORMAT('1',10X,'ICON=',I5)
  610 FORMAT('0',10X,'INTERPOLATED VALUE=',
     * E16.8)
      END
```

Method

Consider obtaining the interpolating function shown in (1.1) when given discrete points $x_1,x_2...,x_n$ ($x_1<x_2<...<x_n$) and their corresponding function values $y_i=f(x_i)$, i=1,...,n. In (1.1) for each interval $[x_n,x_{i+1}]$, different polynomials of degree three or less (for $(-\infty,x_1)$ and (x_n, ∞) polynomials of degree two or less) are shown.
 Let the interpolating function to be obtained be represented function to be obtained be represented by S(x), which is expressed separately as

$$S(x)=\begin{cases} S_0(x), x<x_1 \\ S_i(x), x_i \le x \le x_{i+1}, i=1,...,n-1 \\ S_n(x), x>x_n \end{cases} \tag{4.1}$$

Where $S_0(x)$ and $S_n(x)$ are polynomials of degree two or less and $S_1(x),...,S_{n-1}(x)$ are polynomials of degree three or less.

In this routine $S(x)$ is determined so that the derivatives of $S(x)$ up to 2nd order should be continuous over $(-\infty,\infty)$. For this to be true $S_i(x)$ must satisfy

$$\left.\begin{array}{l} S_{i-1}(x_i)=S_i(x_i)=y_i \\ S'_{i-1}(x_i)=S''_i(x_i) \\ S''_{i-1}(x_i)=S''_i(x_i) \\ i=1,...,n \end{array}\right\} \quad (4.2)$$

The coefficients c_i, d_i and e_i of (1.1) are determined by the conditions in (4.2). This is shown in (4.3).
When i=1,...,n-1

$$\left.\begin{array}{l} \text{When } i=1,...n-1 \\ c_i=\dfrac{y_{i+1}-y_i}{h_i}-\dfrac{h_i}{6}\left(y''_{i+1}+2y''_i\right) \\ d_i=\dfrac{y''_i}{2} \\ e_i=\dfrac{y''_{i+1}-y''_i}{6h_i} \\ \text{When } i=n \\ c_n=\dfrac{y_n-y_{n-1}}{h_{n-1}}+\dfrac{h_{n-1}}{6}\left(2y''_n+y''_{n-1}\right) \\ d_n=\dfrac{y''_n}{2} \end{array}\right\} \quad (4.3)$$

where $h_i=x_{i+1}-x_i$, $y''_i=S''(x_i)$. From the second condition of (4.2), y''_i satisfies the three term relation.

$$\begin{array}{l} h_{i-1}y''_{i-1}+2\left(h_{i-1}+h_i\right)y''_i+h_iy''_{i+1} \\ \quad =6\left(\dfrac{y_{i+1}-y_i}{h_i}-\dfrac{y_i-y_{i-1}}{h_{i-1}}\right) \\ \quad ,i=2,...,n-1 \end{array} \quad (4.4)$$

If y''_i, y''_n are given, (4.4) becomes a system of linear equations with respect to y''_2, y''_3, ..., y''_{n-1}.
Since the coefficient matrix is a positive definite symmetric matrix, this routine uses the modified Cholesky method.

- Characteristics of $S(x)$ when $y''_i = y''_n = 0$
  If $y''_i = y''_n = 0$ is given, $S(x)$ becomes linear over $(-\infty,x_1)$, (x_m, ∞). At this time it becomes a cubic natural spline with the following characteristics.
  Let $g(x)$ be any interpolating function which satisfies

$$g(x)=y_i, \quad i=1,...,n \quad (4.5)$$

$g(x)$, $g'(x)$, $g''(x)$ are continuous in $[x_1, x_n]$.
If $S(x)$ is the interpolating cubic spline corresponding to $y''_1=y''_n=0$, then

$$\int_{x_1}^{x_n}\{g''(x)\}^2dx \geq \int_{x_1}^{x}\{S''(x)\}^2dx \quad (4.6)$$

(The equality occurs only when $g(x)=S(x)$.)
In the sense that $S(x)$ minimizes

$$\int_{x_1}^{x_n}\{g''(x)\}^2dx$$

it can be considered the "smoothest" interpolating function. For further information, see References [48], [49] and [50] pp.349 - 356.

F20-01-0101 LAPS1, DLAPS1

Inversion of Laplace transform of a rational function (regular in the right-half plane)
CALL LAPS1(A,NA,B,NB,T,DELT,L,EPSR,FT, T1,NEPS,ERRV,ICON)

Function

Given a rational function $F(s)$ expressed by (1.1), this subroutine calculates values of the inverse Laplace transform $f(t_0)$, $f(t_0+\Delta t)$, ..., $f(t_0+\Delta t(L\text{-}1))$:

$$\left.\begin{aligned}
&F(s)=\frac{Q(s)}{P(s)}\\
&\quad Q(s)=b_1s^m+b_2s^{m-1}+\ldots+b_ms+b_{m+1}\\
&\quad P(s)=a_1s^n+a_2s^{n-1}+\ldots+a_ns+a_{n+1}\\
&n\geq m,\quad a_1,a_2,\ldots,a_{n+1},b_1,b_2,\ldots,b_{m+1}:\text{real value}
\end{aligned}\right\}\quad(1.1)$$

where $F(s)$ must be regular in the domain of $\mathrm{Re}(s)>0$.

Parameters

A	Input. Coefficients of $P(s)$. One-dimensional array of size $n+1$, assigned in the order of $A(1)=a_1$, $A(2)=a_2$, ..., $A(n+1)=a_{n+1}$.
NA	Input. Degree n of $P(s)$.
B	Input. Coefficients of $Q(s)$. One-dimensional array of size $m+1$, assigned in the order of $B(1)=b_1$, $B(2)=b_2$, ..., $B(m+1)=b_{m+1}$.
NB	Input. Degree m of $Q(s)$.
T	Input. Initial value $t_0(\geq 0)$ from which the values of $f(t)$ are required.
DELT ..	Input. Increment $\Delta t(\geq 0)$ of variable t. If DELT=0.0, only $f(t_0)$ is calculated.
L	Input. The number (L≥1) of points at which values of $f(t)$ are required.
EPSR ...	Input. The relative error tolerance for the values of $f(t)$: 10^{-2} to 10^{-4} for single precision and 10^{-2} to 10^{-7} for double precision will be typical. If EPSR=0.0, the default value 10^{-4} is used.
FT	Output. A set of values $f(t_0)$, $f(t_0+\Delta t)$,..., $f(t_0+\Delta t(\text{L-1}))$. One-dimensional array of size L.
T1	Output. A set of values t_0, $t_0+\Delta t$, ..., $t_0+\Delta t(\text{L-1})$. One-dimensional array of size L.
NEPS ...	Output. The number of terms: N in the truncated expansion. (See "Method"). One-dimensional array of size L. The number of terms N used to calculate FT(I) is stored in NEPS(I).
ERRV ..	Output. Estimates of the relative error of resultant value FT. One-dimensional array of size L. The relative error of FT(I) is stored in ERRV(I).
ICON ..	Output. Condition code. (See Table LAPS1-1.)

Table LAPS1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Some of the results did not meet the required accuracy.	Continued. Values representing accuracy for are output to array ERRV.
30000	One of the following conditions: (1) NB<0 or NB>NA (2) T<0 or DELT<0 (3) L<1 (4) EPSR<0 (5) A(1) = 0	Bypassed.

Comments on use

- Subprograms used
  SSL II ... MGSSL, AFMAX
  FORTRAN basic functions ... FLOAT, ALOG, ALOG10, CMPLX, AIMAG, EXP, ABS, INT, ATAN

- Notes
  Rational function F(s) must be regular in the domain for Re(s)>0.
  If F(s) is singular or if its regularity is not known, use subroutine LAPS2.
  If t_0=0, the value of $f(0)$ is calculated according to the theorem on initial values as

$$f(0)=\begin{cases}b_1/a_1 & (n=m+1)\\ 0 & (n>m+1)\end{cases}$$

For NA=NB, (1.1) is written as

$$F(s)=\frac{Q(s)}{P(s)}=F_1(s)+F_2(s)\qquad(3.1)$$

where, $F_1(s)\equiv b_1/a_1$

$$F_2(s)\equiv\frac{c_2s^{n-1}+c_3s^{n-2}+\cdots+c_{n+1}}{a_1s^n+a_2s^{n-1}+\cdots+a_{n+1}}$$

Inverse transform $f_1(t)$ of $F_1(s)$ is given as

$$f_1(t)=\frac{b_1}{a_1}\delta(t)\ \text{ where }\delta(t)\text{ is the delta function.}$$

Since $F_2(s)$ satisfies the condition of (8.20) in Chapter 8, inverse transform $f_2(t)$ of $F_2(s)$ can be calculated from (8.22). Therefore, when NA=NB the subroutine calculates the inverse of $F_2(s)$ for t>0. When t=0, the maximum value of the floating point numbers is returned.

- Example

  Given a rational function $F(s)$ regular (non-singular) in the area for Re(s)>0:

$$F(s) = \frac{s^2+4}{s^4+12s^3+54s^2+108s+81}$$

the inverse Laplace transform $f(t)$ is obtained at points, i=1,2,...,9, with EPSR=10^{-4}.

```
C       **EXAMPLE**
        DIMENSION A(5),B(3),T1(9),FT(9),
       *ERRV(9),NEPS(9)
        NB=2
        NB1=NB+1
        B(1)=1.0
        B(2)=0.0
        B(3)=4.0
        NA=4
        NA1=NA+1
        A(1)=1.0
        A(2)=12.0
        A(3)=54.0
        A(4)=108.0
        A(5)=81.0
        T=0.2
        DELT=0.2
        L=9
        EPSR=1.0E-4
        WRITE(6,600) NB,(I,B(I),I=1,NB1)
        WRITE(6,610) NA,(I,A(I),I=1,NA1)
        WRITE(6,620) EPSR
        CALL LAPS1(A,NA,B,NB,T,DELT,L,EPSR,
       *FT,T1,NEPS,ERRV,ICON)
        WRITE(6,630) ICON
        IF(ICON.EQ.30000) STOP
        WRITE(6,640) (T1(I),FT(I),ERRV(I),
       *NEPS(I),I=1,L)
        STOP
  600 FORMAT(//24X,'NB=',I3//
     *(20X,'B(',I3,')=',E15.8/))
  610 FORMAT(/24X,'NA=',I3//
     *(20X,'A(',I3,')=',E15.8/))
  620 FORMAT(22X,'EPSR=',E15.8/)
  630 FORMAT(22X,'ICON=',I6//)
  640 FORMAT(15X,'F(',F8.5,')=',E15.8,2X,
     *'ERROR=',E15.8,2X,'N=',I3/)
        END
```

Method

The method for obtaining the inverse Laplace transform is explained in Chapter 8, where $f(t)$ is approximated as:

$$\begin{aligned} f_N(t,\sigma_0) &= \frac{e^{\sigma_0}}{t}\sum_{n=1}^{N} F_n \\ &= \frac{e^{\sigma_0}}{t}\left\{\sum_{n=1}^{k-1} F_n + \frac{1}{2^{p+1}}\sum_{r=0}^{p} A_{p,r}F_{k+r}\right\} \end{aligned} \qquad (4.1)$$

$$\text{where}\quad F_n = (-1)^n \operatorname{Im}\left[F\left(\frac{\sigma_0 + i(n-0.5)\pi}{t}\right)\right]$$

$$N = k + p$$

$$A_{p,p} = 1, A_{p,r-1} = A_{p,r} + \binom{p+1}{r}$$

In this subroutine, σ_0, N, and p are determined as follows:

Since $f(t,\sigma_0)$ satisfies

$$f(t,\sigma_0) = f(t) - e^{-2\sigma_0} f(3t) + e^{-4\sigma_0} f(5t) - \cdots$$

σ_0 is determined as

$$\sigma_0 = -\left[\frac{\log(\text{EPSR})}{2}\right] + 2$$

where [•] is the Gaussian notation, so that the user specified relative error tolerance (EPSR) may be expressed as:

$$\text{EPSR} \approx \left|\frac{f_N(t,\sigma_0) - f_{N-1}(t,\sigma_0)}{f_N(t,\sigma_0)}\right| = e^{-2\sigma_0} \qquad (4.2)$$

Since value N that satisfies the condition of (4.2) depends on the value of t, it is adaptively determined from the following empirical formula

$$N = [5\sigma_0] + [\sigma_0 \cdot t/2.5] \qquad (4.3)$$

and value p is chosen as

5 for $\sigma_0 \le 4$

$[\sigma_0+1]$ for 4 $<\sigma_0 \le 9$,

9 for 9 $<\sigma_0$

Then relative errors of results $f_N(t,\sigma_0)$ are output to array ERRV, which are estimated as follows:

$$\text{ERRV} = \left|\frac{f_N(t,\sigma_0) - f_{N-1}(t,\sigma_0)}{f_N(t,\sigma_0)}\right|$$

F20-02-0101 LAPS2, DLAPS2

Inversion of Laplace transform of a general rational function
CALL LAPS2(A,NA,B,NB,T,DELT,L,EPSR,FT, T1,NEPS,ERRV,IFLG,VW,ICON)

Function

Given a rational function $F(s)$ expressed by (1.1), this subroutine calculates values of the inverse Laplace transform $f(t_0)$, $f(t_0+\Delta t)$, ..., $f(t_0+\Delta t(L\text{-}1))$:

$$\left.\begin{aligned} &F(s)=\frac{Q(s)}{P(s)} \\ &\quad Q(s)=b_1s^m+b_2s^{m-1}+\cdots+b_ms+b_{m+1} \\ &\quad P(s)=a_1s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1} \\ &n\geq m, a_1,a_2,\cdots,a_{n+1},b_1,b_2,\cdots,b_{m+1}:\text{real value} \end{aligned}\right\} \quad (1.1)$$

In this case, $F(s)$ need not be regular in the domain for Re(s)>0.

Parameters

A ..... Input. Coefficients of $P(s)$.
One-dimensional array of size n+1, assigned in the order of A(1)=a_1, A(2)=a_2, ..., A(n+1)=a_{n+1}.

NA .... Input. Degree n of P(s).

B ..... Input. Coefficients of Q(s).
One-dimensional array of size m+1, assigned in the order of B(1)=b_1, B(2)=b_2, ..., B(m+1)=b_{m+1}.

NB .... Input. Degree m of Q(s).

T ..... Input. Initial value t_0(≥0.0) from which the values of $f(t)$ are required.

DELT .. Input. Increment Δt(>0.0) of variable t. If DELT=0.0, only $f(t_0)$ is calculated.

L ..... Input. The number (≥1) of points at which values of $f(t)$ are required.

EPSR ... Input. The relative error tolerance for the values of $f(t)$: 10^{-2} to 10^{-4} for single precision and 10^{-2} to 10^{-7} for double precision will be typical. If EPSR=0.0, the default value 10^{-4} is used.

FT .... Output. A set of values $f(t_0)$, $f(t_0+\Delta t)$,...,$f(t_0+\Delta t$(L-1)).
One-dimensional array of size L.

T1 .... Output. A set of values t_0, $t_0+\Delta t$, ..., $t_0+\Delta t$(L-1).
One-dimensional array of size L.

NEPS ... Output. The number of terms: N in the truncated expansion.
(See "Method").
One-dimensional array of size L. The number of terms N used to calculate is stored in NEPS(I).

ERRV .. Output. Estimates of the relative error of result FT. One-dimensional array of size L. The relative error of FT(I) is stored in ERRV(I).

IFLG .. Output. Regularity judgment result.
IFLAG=0 if $F(s)$ is regular in the domain of Re(s)>0, IFLAG=1 otherwise. (See "Notes on Use".)

VW .... Work area. One-dimensional array of size n+m+2

ICON .. Output. Condition code. (See Table LAPS2-1.)

Table LAPS2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Some of the results did not meet the required accuracy.	Continued. Values representing accuracy for $f(t_0+\Delta t\cdot i)$: i=0,1,...,L-1 are output to array ERRV.
20000	The subroutine failed to obtain the non-negative real value γ_0 such that the F(s) is regular in the domain of Re(s)>γ_0.	Bypassed
30000	One of the following conditions: (1) NB<0 (2) NB>NA (3) T<0 (4) DELT<0 (5) L<1 EPSR<0 A(1) = 0	Bypassed.

Comments on use

- Subprograms used
  SSL II ... LAPS1, HRWIZ, MGSSL, AMACH, AFMAX
  FORTRAN basic functions ... FLOAT, ALOG, ALOG10, CMPLX, AIMAG, EXP, ABS, INT, ATAN

- Notes
  Rational function $F(s)$ need not be regular in the domain for Re(s)>0. However, if it is known that $F(s)$ is regular, use subroutine LAPS1 for efficiency.
  If IFLG=1 is output, $F(s)$ is not regular in the domain of Re(s). This means that $f(t)$ increases exponentially as t approaches infinity.
  If t_0=0, the value of $f(0)$ is calculated according to the theorem on initial values as

$$f(0)=\begin{cases} b_1\ /a_1 & (n=m+1) \\ 0 & (n>m+1) \end{cases}$$

For NA=NB, (1.1) is written as

$$F(s) = \frac{Q(s)}{P(s)} = F_1(s) + F_2(s) \qquad (3.1)$$

where $F_1(s) \equiv b_1 / a_1$

$$F_2(s) \equiv \frac{c_2 s^{n-1} + c_3 s^{n-2} + \cdots + c_{n+1}}{a_1 s^n + a_2 s^{n-1} + \cdots + a_{n+1}}$$

Inverse transform $f_1(t)$ of $F_1(s)$ becomes

$$f_1(t) = \frac{b_1}{a_1} \boldsymbol{\delta}(t)$$

where $\delta(t)$ is the delta function.

Since $F_2(s)$ satisfies the condition of (8.20) in Chapter 8, inverse transform $f_2(t)$ of $F_2(s)$ can be calculated from (8.22). Therefore, when NA=NB the subroutine calculates the inverse of $F_2(s)$ for t>0. When t=0, the maximum value of the floating point numbers is returned.

- Example

Given a rational function $F(s)$:

$$F(s) = \frac{s^2 + 4}{s^4 + 12s^3 + 54s^2 + 108s + 81}$$

the inverse Laplace transform $f(t)$ is obtained at points, t_i=0.2+0.2(i-1),where i=1,2,...,9 with EPSR=10^{-4}.

```
C      **EXAMPLE**
       DIMENSION A(5),B(3),T1(9),FT(9),
      *ERRV(9),NEPS(9),VW(8)
       NB=2
       NB1=NB+1
       B(1)=1.0
       B(2)=0.0
       B(3)=4.0
       NA=4
       NA1=NA+1
       A(1)=1.0
       A(2)=-12.0
       A(3)=54.0
       A(4)=-108.0
       A(5)=81.0
       T=0.2
       DELT=0.2
       L=9
       EPSR=1.0E-4
       WRITE(6,600) NB,(I,B(I),I=1,NB1)
       WRITE(6,610) NA,(I,A(I),I=1,NA1)
       WRITE(6,620) EPSR
       CALL LAPS2(A,NA,B,NB,T,DELT,L,EPSR,
      *FT,T1,NEPS,ERRV,IFLG,VW,ICON)
       WRITE(6,630) ICON,IFLG
       IF(ICON.GE.20000) STOP
       WRITE(6,640) (T1(I),FT(I),ERRV(I),
      *NEPS(I),I=1,L)
       STOP
   600 FORMAT(//24X,'NB=',I3//
      *(20X,'B(',I3,')=',E15.8/))
   610 FORMAT(/24X,'NA=',I3//
      *(20X,'A(',I3,')=',E15.8/))
   620 FORMAT(22X,'EPSR=',E15.8/)
   630 FORMAT(22X,'ICON=',I6,20X,'IFLG=',I2)
   640 FORMAT(15X,'F(',F8.5,')=',E15.8,2X,
      *'ERROR=',E15.8,2X,'N=',I3/)
       END
```

Method

The method for obtaining the inverse Laplace transform is explained in Chapter 8. This subroutine proceeds as follows:

Obtain real value γ_0 for which $F(s+\gamma_0)$ becomes regular in the domain of Re(s)>0 using subroutine HRWIZ.

Calculate inverse Laplace transform $g(t)$ of $G(s) \equiv F(s+\gamma_0)$ by using subroutine LAPS1.

Calculate $f(t)$ from $f(t)= e^{r_0 t} g(t)$

F20-03-0101 LAPS3, DLAPS3

Inversion of Laplace transform of a general function
CALL LAPS3(FUN,T,DELT,L,EPSR,R0,FT,T1, NEPS,ERRV,ICON)

Function

Given a function $F(s)$ (including non-rational function), this subroutine calculates values of the inverse Laplace transform $f(t_0)$, $f(t_0+\Delta t)$, ..., $f(t_0+\Delta t(\text{L-1}))$

In this case, $F(s)$ must be regular in the domain of $\text{Re}(s)>\gamma_0$: Convergence coordinate).

Parameters

FUN ...	Input. The name of the function subprogram which calculates the imaginary part of function $F(s)$ for complex variable s. The form of the subprogram is as follows: FUNCTION FUN(S) where S stands for a complex variable. (See Example.)
T	Input. Initial value t_0(>0.0) from which the values of $f(t)$ are required.
DELT ..	Input. Increment Δt(≥0.0) of variable t. If DELT=0.0, only $f(t_0)$ is calculated.
L	Input. The number (≥1) of points at which values of $f(t)$ are required.
EPSR ...	Input. The relative error tolerance for the values of $f(t)$ (≥0.0): 10^{-2} to 10^{-4} for single precision and 10^{-2} to 10^{-7} for double precision are typical. If EPSR=0.0 or EPSR≥1.0 is input, the default value 10^{-4} is used.
R0	Input. The value of γ which satisfies the condition of $\gamma\geq\gamma_0$ when function $F(s)$ is regular in the domain of $\text{Re}(s)>\gamma_0$. If a negative is input as the value of R0, this subroutine assumes R0=0.0.
FT	Output. A set of values $f(t_0)$, $f(t_0+\Delta t)$,...,$f(t_0+\Delta t(\text{L-1}))$. One-dimensional array of size L.
T1	Output. A set of values t_0, $t_0+\Delta t$,..., $t_0+\Delta t$ (L - 1). One-dimensional array of size L.
NEPS ...	Output. The number of truncation items. One-dimensional array of size L. The number of terms N used to calculate FT(I) is stored in NEPS(I).
ERRV ..	Output. Estimates of the relative error of the result. One-dimensional array of size L. The relative error of FT(I) is stored in ERRV(I).
ICON ..	Output. Condition code. (See Table LAPS3-1.)

Table LAPS3-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Some of the results did not meet the required accuracy.	Continued. Values representing accuracy for $f(t_0+\Delta t\cdot i)$, i=0,1,..., L-1 are output to array ERRV.
20000	The value of EXP(R0*T1(I)+σ_0)/T1(I) may overflow for a certain value of I.	Bypassed. The result may not be guaranteed.
30000	One of the following conditions: (1) T≤0 or DELT<0 (2) L<1	Bypassed.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic functions ... FLOAT, ALOG, CMPLX, EXP, INT, ATAN, ABS

- Notes
  When $F(s)$ is a rational function, use subroutine LAPS2 for efficiency.
  When $F(s)$ is regular in the domain for $\text{Re}(s)>\gamma_0$, input $\gamma\geq\gamma_0$ as parameter R0.
  When $\gamma_0\leq 0.0$, simply set R0=0.0. If a negative value is input as parameter R0, R0=0.0 is assumed in the subroutine.
  If function $f(t)$ for R0=0.0 and function $f(t)$ for R0>0.0 are significantly different, it is possible because of $\gamma_0>0.0$. To estimate value γ_0 using this subroutine, perform the following procedure:
  Calculate $f(t)$ using appropriate values (e.g. R0=0.0, R0=0.5) as R0. Estimate the value of t at which the values of $f(t)$ are not the same with R0=0.0, 0.5, and 1.0. Let this value be t_a. If the singular point of $F(s)$ is $s_0=\nu_0+i\mu_0(\nu_0>0)$ (if more than one singular point exists, use the largest value of the real part as the value of s_0), $f(t)$ varies with the values of R0 in the domain of $t>t_a\approx\sigma_0/(\nu_0-R0)$, where value σ_0 is as follows:

$$\sigma_0=\left[-\frac{\log(\text{EPSR})}{2}\right]+2$$

  Therefore, $\gamma_0(=\nu_0)$ can be estimated from:

$$\gamma_0\approx R0+\frac{\sigma_0}{t_a} \tag{3.1}$$

An example of the above procedure is given below.
Example:

$$F(s)=1\Big/\sqrt{s^2-4s+8} \tag{3.2}$$

The results of *f*(*t*) with R0=0.0, R0=0.5, and R0=1.0 on the assumption that EPSR=10^{-2} are shown by the solid lines in (a), (b), and (c), respectively of Fig. LAPS3-1. The dotted lines show the correct results. These figures show that each *f*(*t*) largely varies with the values of R0.

f(*t*) sharply varies at the point of $t_a \approx 2$, $t_a \approx 3.5$, and $t_a \approx 5$ for R0=0.0, R0=0.5, and R0=1.0. Since σ_0=5 for EPSR=10^{-2}, the values of γ_0 are estimated by using (3.1). The estimated values are as follows:
For R0=0.0, $\gamma_0 \approx 2.5$
For R0=0.5, $\gamma_0 \approx 1.93$
For R0=1.0, $\gamma_0 \approx 2.0$
Since *F*(*s*) in (3.2) has the singular point at $s_0 = 2 \pm i2\sqrt{3}$, the values of γ_0 above are proper estimates. The dotted lines in the figure show the results for input of 2.0 as the value of R0.

Since the factor $e^{\sigma_0}\big/t$ is included in the expression for calculating *f*(*t*) ((4.4) in method), *f*(0.0) cannot be calculated. Use the value of *f*(0.01), *f*(0.0001) or so instead of the value of *f*(0.0). Note that an overflow may occur for an excessively small value of *t*.

In this subroutine, the value of the imaginary part of function *F*(*s*) is evaluated at the following points:

$$s_n = \frac{\sigma_0 + i(n-0.5)\pi}{t}, \quad n=1,2,3,...$$
$$\sigma_0 > 0, t > 0$$

When, for such a point, function *F*(*s*) is a multi-valued function, a proper branch for *F*(*s*) must be calculated in the function subprogram. For example, suppose the following:

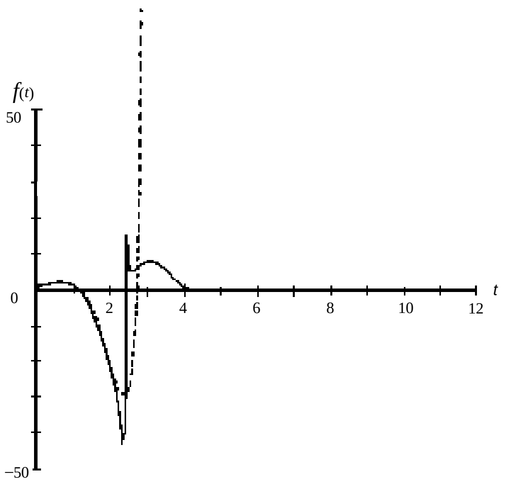
$$F(s)=1\Big/\sqrt{s^2+1}$$

$\sqrt{s^2+1}$ generates the branches for the above s_n in Quadrants 1 and 3. In this case, the branch in Quadrant 1 should be employed.

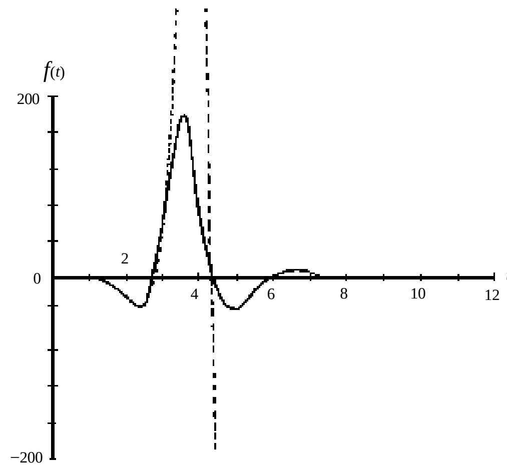
Suppose the following functions:

$$F(s)=\exp\left(-\sqrt{4s^2+1}\right)\Big/s \tag{3.4}$$

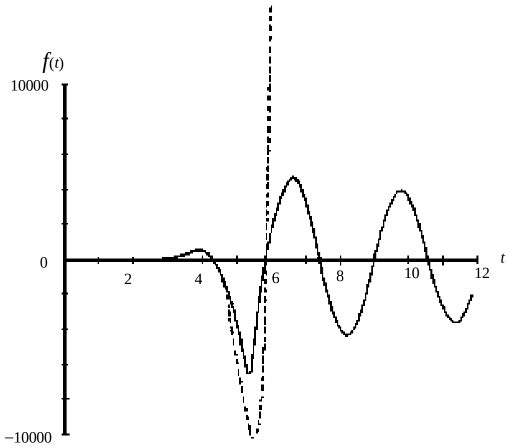
$$F(s)=1\Big/\left(s\cosh\sqrt{s^2+1}\right) \tag{3.5}$$



(a) EPSR=10^{-2}, R0=0.0



(b) EPSR=10^{-2}, R0=0.5



(c) EPSR=10^{-2}, R0=1.0

Fig. LAPS3-1 Estimotion of γ_0 for *F*(s)= $1\Big/\sqrt{s^2-4s+8}$

In each of the above, the delay factor exp(-*as*) (*a*>0) is included in $F(s)$ for $s \to \infty$ .In such a case, the function subprogram should be carefully defined. For example, if (3.4) is employed without any modification, the result of *f*(t) is as shown in 1) in Fig. LAPS3-2. In this case, for *t*<2, *f*(*t*)≠0.0, and vibration occurs between 2 and 4. This is because the valid condition for the Euler transformation ((8.25) in Section 8.6) may not be satisfied. This is called the Gibbs phenomenon.

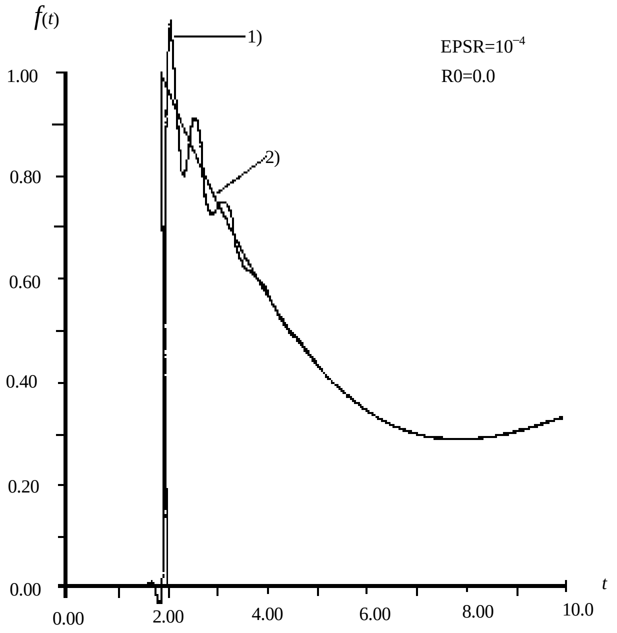


Fig. LAPS3-2 Conversion of $F(s)=\exp(-\sqrt{4s^2+1}/s)$

For (3.4), if *t*<2, *f*(*t*)=0 is theoretically derived. Thus, the following expression can be sufficiently manipulated:

$$\begin{aligned} G(s) &\equiv \exp(2s)\cdot F(s) \\ &= \exp\left(2s-\sqrt{4s^2+1}\right)\Big/ s \end{aligned} \tag{3.6}$$

Note that the above function is the image function of:

$$g(t') \equiv f(t'+2) \qquad t'>0$$

G(*s*) gives a more accurate result than *F*(*s*). However, since $\left(2s-\sqrt{4s^2+1}\right)$ may cause numerical cancellation, it should be transformed to the following:

$$2s-\sqrt{4s^2+1} = -\frac{1}{2s+\sqrt{4s^2+1}}$$

2) in Fig. LAPS3-2 is the result of *G*(*s*) above. *F*(*s*) in (3.5) should be expanded into the following, then transformed by terms as in (3.4):

$$\begin{aligned} F(s) = \frac{2}{s}\Big\{&\exp\left(-\sqrt{s^2+1}\right)-\exp\left(-3\sqrt{s^2+1}\right) \\ &+\exp\left(-5\sqrt{s^2+1}\right)-\cdots\Big\} \end{aligned}$$

• Example

$$F(s)=\frac{\exp\left(-X\sqrt{s^2+1}\right)}{s^2+2} \quad ,X=40$$

Suppose that *f*(*t*), inversion of Laplace transform is to be calculated at each point of $(t_i-X)=0.2+0.2(i-1)$, $i=1,2,...,800$. Use EPSR=10^{-4}. Instead of *F*(*s*), the following expression is employed in the function subprogram:

$$\begin{aligned} G(s) &= \exp(Xs)\cdot F(s) \\ &= \frac{\exp\left\{\left(s-\sqrt{s^2+1}\right)X\right\}}{s^2+2} \\ &= \frac{\exp\left\{\dfrac{-X}{s+\sqrt{s^2+1}}\right\}}{s^2+2} \end{aligned}$$

f(*t*) is derived by using inverse transform of *G*(*s*), *g*(*t*) as follows:

$$f(t)=\begin{cases} 0 & ,t<X \\ g(t-X) & ,t\ge X \end{cases}$$

Figure LAPS3-3 shows the result of *g*(*t*-X) in which *t*-X is used as the axis of abscissas. (Refer to [101] in References.)

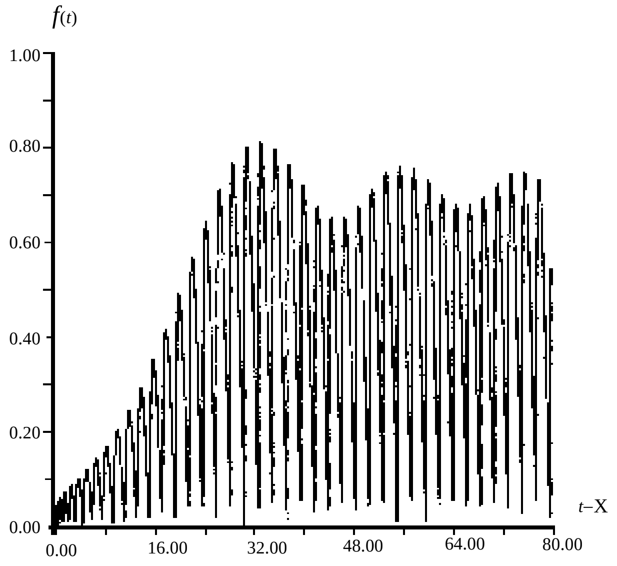


Fig. LAPS3-3 Result of the example

```
C      **EXAMPLE**
       DIMENSION FT(800),T1(800),NEPS(800),
      *          ERRV(800)
       EXTERNAL FUN
       T=0.2
       DELT=0.2
       L=800
       EPSR=1.0E-4
       R0=0.0
       CALL LAPS3(FUN,T,DELT,L,EPSR,R0,FT,
      *T1,NEPS,ERRV,ICON)
       WRITE(6,600) ICON
       WRITE(6,610) (T1(I),FT(I),NEPS(I),
      *             ERRV(I),I=1,L)
   600 FORMAT('1',20X,'ICON=',I5//)
   610 FORMAT(6X,'F(',F8.3,')=',E16.7,3X,
      *'N=',I3,3X,'ERR=',E16.7)
       STOP
       END
       FUNCTION FUN(S)
       COMPLEX*8 S,SR
       SR=CSQRT(S*S+1.0)
       IF(REAL(SR).LT.0.0) SR=-SR
       SR=CEXP(40.0/(S+SR))*(S*S+2.0)
       FUN=AIMAG(1.0/SR)
       RETURN
       END
```

Method

Calculation of inverse Laplace transform is described in Section 8.6. Let the value of R0 be γ, then for $\gamma>0$, $G(s)=F(s+\gamma)$ is regular in the domain of Re(s)>0. By calculating inverse transform $g(t)$, function $f(t)$ can be derived as:

$$f(t)=e^{\gamma t}g(t) \tag{4.1}$$

In the following description, a function which is regular in the domain of Re(s)>0, will be set to $F(s)$ again. Then determination of parameters σ_0, p, and N described in Section 8.6, will be described.

- Value of σ_0 (truncation error)
  From (8.23) in Section 8.6, σ_0 approximation $f(t,\sigma_0)$ of $f(t)$ can be expressed as

$$f(t,\sigma_0)=f(t)-e^{-2\sigma_0}f(3t)+e^{-4\sigma_0}f(5t)-\cdots \tag{4.2}$$

  The value of σ_0 is determined so that the required relative error tolerance EPSR may be as follows:

$$\text{EPSR}=\left|\frac{f(t,\sigma_0)-f(t)}{f(t)}\right|\approx\left|\frac{f(t,\sigma_0)-f(t)}{f(3t)}\right|$$
$$\approx e^{-2\sigma_0}$$

  This leads to

$$\sigma_0=\left[-\frac{\log(\text{EPSR})}{2}\right]+2.0 \tag{4.3}$$

  where [•] is the Gaussian notation.
  2.0 is added to the second term of the right part in (4.3) to produce σ_0=3 even for EPSR=10^{-1}. The value of σ_0 is almost the same as the number of significant digits of $f_N(t,\sigma_0)$.

- Value of p
  Approximation of $f(t)$, $f_N(t,\sigma_0)$ is calculated by using the following expression, derived from equation (8.28) in Section 8.6.

$$f_N(t,\sigma_0)=\frac{e^{\sigma_0}}{t}\left\{\sum_{n=1}^{k-1}F_n+\sum_{q=0}^{p}\frac{1}{2^{p+1}}D^qF_k\right\} \tag{4.4}$$

  where (p+1) stands for the number of terms of the Euler transformation. In this subroutine, p is determined from experience as

$$p=\sigma_0+2 \tag{4.5}$$

  When $F(s)$=O(s^α) as $s\rightarrow\infty$, and if σ_0, that is, EPSR is given so that the value of p satisfies the condition:

$$p\ge(\alpha+5) \tag{4.6}$$

  then, (4.4) can also be applied to $F(s)$ for which f(t) may be a distribution. Below is an example in which $f(t)$ may be distribution. Inverse transform of $F(s)=\sqrt{s}$, can be written as

$$f(t)=\frac{\delta(t)}{\sqrt{\pi t}}-\frac{U(t)}{\left(2\sqrt{\pi}t^{3/2}\right)}$$

  where $\delta(t)$ stands for the Dirac delta function, and $U(t)$ stands for the following:

$$U(t)=\begin{cases}1, & t\ge 0\\ 0, & t<0\end{cases}$$

  Because $\lim\limits_{|s|\to\infty}F(s)\neq 0$, the above method cannot be applied to $F(s)=\sqrt{s}$ theoretically. However, when p is set to 6(>1/2+5) or more in the range of t>0, this subroutine can be used for this function as usual.

- Value of N (number of truncation terms)
  With k and p used in (4.4):

$$N=k+p$$

  Stands for the number of truncation terms and is equal to the evaluation count for function $F(s)$. The value of k must be determined so that the valid condition for the Euler transformation ((8.25) in Section 8.6) may be satisfied. Such a value of k depends upon t, and can be expressed by:

$$k=k_1+k_2t \tag{4.7}$$

  (where k_1 and k_2 are constants.)

This subroutine calculates $f(t)$ for the range of $t_0 \le t \le t_0 + \Delta t(\mathrm{L}-1)$ with use of the following method, which determines the values of k_1 and k_2. First, by setting $t=t_0$, define k which allows truncation error ERRV(1) to in the range of ERRV(1)<EPSR. Let this be k'.

Next, by setting $t=t_0+\Delta t(\mathrm{L}-1)$, define k which allows ERRV(L)<EPSR. Let this be k''. From these, the following simultaneous equations are obtained:

$$\begin{aligned} k' &= k_1 + k_2 t_0 \\ k'' &= k_1 + k_2\{t_0 + \Delta t(\mathrm{L}-1)\} \end{aligned} \qquad (4.8)$$

Determine the value of k_1 and k_2 by solving these equations. Then, calculate k by applying these values to equation (4.7) (For the fraction part, round up to the integer part.) The value of N is output to array NEPS.

A22-11-0101 LAX, DLAX

A system of linear equations with a real general matrix (Crout's method)
CALL LAX(A,K,N,B,EPSZ,ISW,IS,VW,IP,ICON)

Function

This subroutine solves a real coefficient linear equations (1.1) using the Crout's method.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

Where $\boldsymbol{A}$ is an $n \times n$ regular real matrix, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is the n-dimensional solution vector. $n \geq 1$.

Parameters

A	Input. Coefficient matrix $\boldsymbol{A}$. The contents of A are altered on output. A is a two-dimensional array, A(K,N).
K	Input. Adjustable dimension of array A (≥N).
N	Input. Order n of the coefficient matrix $\boldsymbol{A}$.
B	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. B is a one-dimensional array of size n.
EPSZ ..	Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ (≥0.0). If EPSZ is 0.0, a standard value is used.
ISW ...	Input. Control information. When l (≥1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows: ISW=1, the first system is solved. ISW=2, the 2nd to lth systems are solved. However, only parameter B is specified for each constant vector $\boldsymbol{b}$ of the systems of equations, with the rest unchanged. (See Notes.)
IS	Output. Information for obtaining the determinant of matrix $\boldsymbol{A}$. If the n elements of the calculated diagonal of array A are multiplied by IS, the determinant is obtained.
VW	Work area. VW is a one-dimensional array of size n.
IP	Work area. IP is a one-dimensional array of size n.
ICON ..	Output. Condition code. Refer to Table LAX-1.

Table LAX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Either all of the elements of some row were zero or the pivot became relatively zero. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	K<N, N<1, EPSZ<0.0 or ISW≠1,2	Bypassed

Comments on use

- Subprogram used

  SSL II ..... ALU, LUX, AMACH, MGSSL

  FORTRAN basic functions ..... ABS

- Notes

  The solution $\boldsymbol{x}$ obtained by this subroutine may be refined in accuracy by calling subroutine LAXR successively.

  If EPSZ is set to 10^{-s}, this value has the following meaning; while performing the LU-decomposition by Crount's method, if the loss of over s significant digits occurred for the pivot, the LU-decomposition should be discontinued with ICON=20000 regarding the pivot to be relatively zero. The standard value of EPSZ is $16u$, u being the unit round off. If the processing is to proceed at a lower pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.

  When solving successive systems of linear equations with the identical coefficient matrix, computation can be performed by setting ISW=2 after the first system of equations are processed. By setting ISW=2, LU-decomposition of coefficient matrix $\boldsymbol{A}$ is bypassed so the computation time is reduced. In this case, the value of IS is the same as when ISW=1.

- Example

  In this example, l systems of linear equations in n unknown with the identical coefficient matrix are solved. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(100),
     *          VW(100),IP(100)
      READ(5,500) N
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),
     *             I=1,N)
      READ(5,500) L
      M=1
      ISW=1
      EPSZ=1.0E-6
```

```
 10 READ(5,510) (B(I),I=1,N)
    WRITE(6,610) (I,B(I),I=1,N)
    CALL LAX(A,100,N,B,EPSZ,ISW,IS,VW,IP,
   *ICON)
    WRITE(6,620) ICON
    IF(ICON.GE.20000) STOP
    WRITE(6,630) (I,B(I),I=1,N)
    IF(L.EQ.M) GOTO 20
    M=M+1
    ISW=2
    GOTO 10
 20 DET=IS
    DO 30 I=1,N
    DET=DET*A(I,I)
 30 CONTINUE
    WRITE(6,640) DET
    STOP
500 FORMAT(I5)
510 FORMAT(4E15.7)
600 FORMAT('1',10X,'** COEFFICIENT MATRIX'
   */12X,'ORDER=',I5/(10X,4('(',I3,',',I3,
   *')',E16.8)))
610 FORMAT(///10X,'CONSTANT VECTOR'
   */(10X,5('(',I3,')',E16.8)))
620 FORMAT('0',10X,'CONDITION CODE=',I5)
630 FORMAT('0',10X,'SOLUTION VECTOR'
   */(10X,5('(',I3,')',E16.8)))
640 FORMAT(///10X,
   *'DETERMINANT OF COEFFICIENT MATRIX=',
   *E16.8)
    END
```

Method

A system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (4.1)$$

is solved using the following procedure.

- LU-decomposition of coefficient matrix ***A*** (Crout's method)
  The coefficient matrix ***A*** is decomposed into the product of a lower triangular matrix ***L*** and a unit upper triangular matrix ***U***. To reduce rounding off errors, the partial pivoting is performed in the decomposition process.

  $$\boldsymbol{PA} = \boldsymbol{LU} \qquad (4.2)$$

  P is the permutation matrix which performs the row exchanges required in partial pivoting.
  Subroutine ALU is used for this operation.
- Solving ***LU***=***Pb*** (forward and backward substitutions)
  Solving equation (4.1) is equivalent to solving the linear equation (4.3).

  $$\boldsymbol{LUx} = \boldsymbol{Pb} \qquad (4.3)$$

  Equation (4.3) is decomposed into two equations

  $$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad (4.4)$$
  $$\boldsymbol{Ux} = \boldsymbol{y} \qquad (4.5)$$

  then the solution is obtained using forward substitution and backward substitution.

  Subroutine LUX is used for these operations. For more information, see References [1], [3], and [4].

A25-11-0101 LAXL, DLAXL

Least squares solution with a real matrix (Householder transformation)
CALL LAXL(A,K,M,N,B,ISW,VW,IVW,ICON)

Function

This subroutine solves the overdetermined system of linear equations (1.1) for the least squares solution $\tilde{\boldsymbol{x}}$ using Householder transformation,

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

Given $m \times n$ real matrix $\boldsymbol{A}$ of rank n and m-dimensional constant vector $\boldsymbol{b}$ where m is not less than n. That is, this subroutine determines the solution vector $\boldsymbol{x}$ such that

$$\|\boldsymbol{b} - \boldsymbol{Ax}\|_2$$

is minimized. where $n \geq 1$.

Parameters

A ..... Input. Coefficient matrix $\boldsymbol{A}$.
The contents of $\boldsymbol{A}$ are altered on output.
The matrix $\boldsymbol{A}$ is a two-dimensional array, A(K,N).

K ..... Input. Adjustable dimension of array **A** (≥M).

M ..... Input. Number of rows, m, in matrix $\boldsymbol{A}$.

N ..... Input. Number of columns, n, in matrix $\boldsymbol{A}$.

B ..... Input. Constant vector $\boldsymbol{b}$.
Output. The least squares solution $\tilde{\boldsymbol{x}}$. One-dimensional array of size m. (Refer to Notes.)

ISW ... Input. Control information.
When solving l (≥ 1) systems of linear equations with the identical coefficient matrix, specify as follows:
ISW = 1 ... The first system is solved.
ISW = 2 ... The 2nd to l th systems are solved. however, the values of B are to be replaced by the new constant vector b with the rest unchanged. (Refer to Notes)

VW ... Work area.
One-dimensional array of size 2n.

IVW ... Work area.
One-dimensional array of size n.

ICON .. Output. Condition code.
Refer to Table LAXL-1.

Table LAXL-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Rank (***A***)<n	Discontinued
30000	K<M, M<N, N<1 or ISW≠1,2	Bypassed

Comments on use

- Subprogram used
  SSL II ..... ULALH, ULALB, AMACH, MGSSL
  FORTRAN basic functions ..... SQRT

- Notes
  The least squares solution $\tilde{\boldsymbol{x}}$ is stored in the first n elements of array B.
  When solving successive systems of linear equations with the identical coefficient matrices, ISW=2 should be specified after the first system. Then reducing the coefficient matrix to an upper triangular matrix is bypassed by setting ISW=2, hence computation time is reduced. Refer to "Method" for reducing to an upper triangular matrix.

- Example
  This example shows the method to solve l overdetermined systems of linear equations with the identical coefficient matrix where m, the number of equations, is not less than n, the number of unknowns. $m \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(100),
     *          VW(200),IVW(100)
      READ(5,500) M,N,L
      READ(5,510) ((A(I,J),I=1,M),J=1,N)
      WRITE(6,600) M,N,((I,J,A(I,J),J=1,N),
     *             I=1,M)
      LL=1
      ISW=1
   10 READ(5,510) (B(I),I=1,M)
      WRITE(6,610) (I,B(I),I=1,M)
      CALL LAXL(A,100,M,N,B,ISW,VW,IVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,630) (I,B(I),I=1,N)
      IF(L.EQ.LL) GO TO 20
      ISW=2
      LL=LL+1
      IF(LL.LE.L) GO TO 10
   20 WRITE(6,640)
      STOP
  500 FORMAT(3I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,
     *'LINEAR LEAST SQUARES SOLUTION'/6X,
     *'ROW NUMBER=',I4,5X,'COLUMN NUMBER='
     *,I4/6X,'COEFFICIENT MATRIX='/
     *(10X,4('(',I3,',',I3,')',E17.8,3X)))
```

```
610 FORMAT(///10X,'CONSTANT VECTOR='
   */(10X,4('(',I3,')',E17.8,3X)))
620 FORMAT(' ',10X,'CONDITION CODE=',I6)
630 FORMAT(' ',10X,'SOLUTION VECTOR='
   */(10X,4('(',I3,')',E17.8,3X)))
640 FORMAT('0',5X,
   *'LINEAR LEAST SQUARES SOLUTION END')
    END
```

Method

Let $\boldsymbol{A}$ be an $m \times n$ real matrix with $m \geq n$ and of rank n and $\boldsymbol{b}$ be an m-dimensional constant vector. This subroutine determines the vector $\tilde{\boldsymbol{x}}$ (the least squares solution) so that

$$\left\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\right\|_2 \tag{4.1}$$

is minimized.

Since the Euclidean norm is invariant with the orthogonal transformation.

$$\left\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\right\|_2 = \left\|\boldsymbol{Q}\boldsymbol{b}-\boldsymbol{Q}\boldsymbol{A}\boldsymbol{x}\right\|_2 = \left\|\boldsymbol{C}-\boldsymbol{R}\boldsymbol{x}\right\|_2 \tag{4.2}$$

where $\boldsymbol{Q}$ is an orthogonal matrix, $\boldsymbol{C} = \boldsymbol{Qb}$ and $\boldsymbol{R} = \boldsymbol{QA}$. Choosing $\boldsymbol{Q}$ such that

$$\boldsymbol{QA} = \boldsymbol{R} = \begin{bmatrix} \tilde{\boldsymbol{R}} \\ \cdots \\ 0 \end{bmatrix} \}(m-n)\times n \tag{4.3}$$

where $\tilde{\boldsymbol{R}}$ is an $n \times n$ upper triangular matrix, clearly the equation (4.2) is represented by the equation (4.4) as follows:

$$\left\|\boldsymbol{C}-\boldsymbol{R}\boldsymbol{x}\right\|_2 = \left\|\begin{matrix} \tilde{\boldsymbol{C}}-\tilde{\boldsymbol{R}}\boldsymbol{x} \\ \cdots\cdots \\ \boldsymbol{C_1} \end{matrix}\right\|_2 \tag{4.4}$$

where $\tilde{\boldsymbol{C}}$ denotes the first n-dimensional vector of $\boldsymbol{C}$, and $\boldsymbol{C}_1$ is the $(m - n)$-dimensional vector of $\boldsymbol{C}$ except for $\tilde{\boldsymbol{C}}$. Therefore, the equation (4.4) is minimum when $\tilde{\boldsymbol{C}} - \tilde{\boldsymbol{R}}\boldsymbol{x} = \boldsymbol{0}$. In other words, if matrix $\boldsymbol{A}$ can be reduced to an upper triangular matrix by an orthogonal transformation as shown in the equation (4.3), the least squares solution $\tilde{\boldsymbol{x}}$ of the equation (4.1) can be obtained by the following the equation (4.5).

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{R}}^{-1}\tilde{\boldsymbol{C}} \tag{4.5}$$

In this subroutine, the least squares solution is determined by the following procedures:

- Reducing a matrix $\boldsymbol{A}$ to an upper triangular matrix ... Transforming a matrix $\boldsymbol{A}$ to an upper triangular matrix with the Householder transformation, matrix $\boldsymbol{R}$ is obtained using the equation (4.3).

- Obtaining a solution ... The least squares solution $\tilde{\boldsymbol{x}}$ is obtained using the equation (4.5).
  The actual procedure is performed as follows.

– Transforming a matrix $\boldsymbol{A}$ into a triangular matrix-

In transforming a matrix $\boldsymbol{A}$ into an upper triangular matrix, let $a^{(k)}_{ij}$ be the elements of a matrix $\boldsymbol{A}^{(k)}$.

Then the transformation is as follows:

$$\begin{aligned} &\boldsymbol{A}^{(1)} = \boldsymbol{A} \\ &\boldsymbol{A}^{(k+1)} = \boldsymbol{P}^{(k)}\boldsymbol{A}^{(k)}, \quad k = 1,...,n \end{aligned} \tag{4.7}$$

is accomplished by orthogonal matrix $P^{(k)}$ as shown in equation (4.8) such that $a_{ik}^{(k+1)}=0$, $i=k+1,...,m$.
(Refer to Fig. LAXL-1)

$$\boldsymbol{P}^{(k)} - \boldsymbol{I} = \beta_k \boldsymbol{u}^{(k)}\boldsymbol{u}^{(k)\mathrm{T}} \tag{4.8}$$

$$\text{Where}, \ \sigma_k = \left(\sum_{i=k}^{m}\left(a_{ik}^{(k)}\right)^2\right)^{\frac{1}{2}}$$

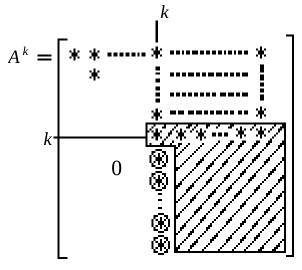
$$\boldsymbol{u}^{(k)\mathrm{T}} = \left(u_1^{(k)}, u_2^{(k)}, ..., u_m^{(k)}\right)$$

$$\beta_k = \left[\sigma_k\left(\sigma_k + \left|a_{kk}^{(k)}\right|\right)\right]^{-1}$$

$$u_i^{(k)} = 0, \quad i < k$$

$$u_k^{(k)} = \operatorname{sign}\left(a_{kk}^{(k)}\right)\left(\sigma_k + \left|a_{kk}^{(k)}\right|\right)$$

$$u_i^{(k)} = a_{ik}^{(k)}, \quad i > k$$



The orthogonal matrix $\boldsymbol{P}$ is chosen so that the elements marked by ○ become zeros in the transformation $\boldsymbol{A}^{k+1}=\boldsymbol{P}^k\boldsymbol{A}^k$.
The elements surrounded by ▭ are all changed by the transformation.

Fig. LAXL-1 Process of transforming a matrix into an upper triangular matrix

Consequently, $\boldsymbol{R}$ is given by the following equation:

$$\boldsymbol{R} = \boldsymbol{A}^{(n+1)} = \boldsymbol{P}^{(n)}\boldsymbol{P}^{(n-1)}\cdots\boldsymbol{P}^{(1)}\boldsymbol{A} = \boldsymbol{QA}$$

This is called the Householder transformation. In this subroutine the following items are taken into consideration:

- Previous to the k -th transformation, the k -th column is chosen in such a way that σ_k value becomes maximum in equation (4.8) in order to minimize calculation errors. In other words, let the l -th column be chosen out of the equation (4.9) so that $S_1^{(k)} = \max S_j^{(k)}$ and then the l-th column is exchanged with k -th column.

$$s_j^{(k)} = \sum_{i=k}^{m} \left(a_{ij}^{(k)}\right)^2, \quad i = k,..., \tag{4.9}$$

If $s_l^{(k)}$ value satisfies the following condition,

$$s_l^{(k)} \le u \cdot \max_{1 \le i \le n} s_i^{(1)}$$

where u is a unit round off.
Then rank ($\boldsymbol{A}$) < n is assumed and the processing is discontinued with ICON = 20000.

- In order to reduce the calculation for the transformation, $\boldsymbol{P}^{(k)}$ is not computed explicitly, but rather the transformation (4.7) is done by way of the following transformation:

$$\begin{aligned} \boldsymbol{A}^{(k+1)} &= \left(\boldsymbol{I} - \beta_k \boldsymbol{u}^{(k)} \boldsymbol{u}^{(k)\mathrm{T}}\right) \cdot \boldsymbol{A}^{(k)} \\ &= \boldsymbol{A}^{(k)} - \boldsymbol{u}^{(k)} \boldsymbol{y}_k^{\mathrm{T}} \\ \text{where}, \boldsymbol{y}_k^{\mathrm{T}} &= \beta_k \boldsymbol{u}^{(k)\mathrm{T}} \boldsymbol{A}^{(k)} \end{aligned}$$

taking into consideration the fact that the first (k-1) elements of $\boldsymbol{u}^{(k)}$ are all zeros when the vector $\boldsymbol{y}^{(k)}$ and the matrix $\boldsymbol{A}^{(k+1)}$ are computed.

Obtaining a solution
Since

$$\boldsymbol{R} = \boldsymbol{P}^{(n)} \boldsymbol{P}^{(n-1)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{A} \tag{4.10}$$

the constant vector is also transformed in the same way.

$$\boldsymbol{C} = \boldsymbol{Qb} = \boldsymbol{P}^{(n)} \boldsymbol{P}^{(n-1)} \cdots \boldsymbol{P}^{(1)} \boldsymbol{b} \tag{4.11}$$

Solving the system of linear equations (4.12) by using this $\tilde{\boldsymbol{C}}$, the least squares solution for the equations (4.1) can be obtained as follows:

$$\boldsymbol{R}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{C}} \tag{4.12}$$

Taking into consideration that the matrix $\tilde{\boldsymbol{R}}$ is an upper trangular matrix, the equation (4.12) can be solved using backward substitution.

A25-21-0101 LAXLM, DLAXLM

Least squares minimal norm solution of a real matrix (singular value decomposition method)
CALL LAXLM(A,KA,M,N,B,ISW,EPS,SIG,V,KV, VW,ICON)

Function

This subroutine obtains least squares minimal norm solution $\boldsymbol{x}^{+}$ for a system of linear equations with an $m \times n$ real matrix $\boldsymbol{A}$.

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

where $\boldsymbol{b}$ is an m-dimensional real constant vector, this subroutine determines the n order solution vector x so that

$$\|\boldsymbol{x}\|_2$$

is minimized while

$$\|\boldsymbol{b} - \boldsymbol{Ax}\|_2$$

is minimized.

$m \geq 1$, $n \geq 1$

Parameters

A	Input. Coefficient matrix $\boldsymbol{A}$. The contents of A are altered on output. Two-dimensional array, A(KA,N).
KA	Input. Adjustable dimension of array A (≥M).
M	Input. Number of rows in coefficient matrix $\boldsymbol{A}$, m.
N	Input. Number of columns in coefficient matrix $\boldsymbol{A}$ and number of rows in matrix $\boldsymbol{V}$, n.
B	Input. Constant vector $\boldsymbol{b}$. Output. Least squares minimal norm solution $\boldsymbol{x}^{+}$. One-dimensional array of size max(m,n). (See Notes.)
ISW ...	Input. Control information. Specify depending upon the conditions that one system of linear equations is solved or some systems of linear equations with the identical coefficient matrix is solved as follows: ISW=0: One system is solved. ISW=1: The first system is solved and the information to solve the subsequent systems are left. ISW=2: The second and subsequent systems are solved. However the values of B are to be replaced by the new constant vector b with the rest specifying ISW=1. (See Notes.)
EPS...	Input. Tolerance for relative zero test of singular values (≥0.0). When EPS=0.0 is specified, the default value is used. (See Notes.)
SIG ...	Output. Singular values. One-dimensional array of size n. (See Notes.)
V	Work area. Two-dimensional array, V(KV,K) where K=min(M+1,N). (See Notes.)
KV ...	Input. Adjustable dimension of array V (≥N).
VW ...	Work area. One-dimensional array of size n.
ICON ..	Output. Condition code. See Table LAXLM-1.

Table LAXLM-1 Condition codes

Code	Meaning	Processing
0	No error	
15000	Any singular values could not be obtained.	Discontinued
30000	KA<M, M<1, N<1, KV<N, EPS<0.0 or ISW≠1,2	Bypassed

Comments on use

- Subprograms used
  SSL II ..... ASVD1, AMACH, MGSSL
  FORTRAN basic functions ..... MIN0, SIGN, SQRT, AMAX1, ABS

- Notes
  The least squares minimal norm solution $\boldsymbol{x}^{+}$ is stored in the first n elements of array B.
  When only one least squares minimal norm solution is required, if ISW=0 is specified, this subroutine does not compute transformation matrix $\boldsymbol{U}$ by singular value decomposition. Therefore computational time is reduced. Matrix $\boldsymbol{V}$ is returned in the first l columns of array $\boldsymbol{V}$ with l=min(m,n). This matrix $\boldsymbol{V}$ can be obtained on array A.
  To obtain the least squares minimal norm solutions of a number of systems of linear equations with identical coefficient matrices, specify ISW=1 for the first system. Next specify ISW=2 for the second and subsequent system. Since the singular value decomposition for coefficient matrices is omitted in the second and following systems, the computational time is reduced. Matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are stored in the first l columns of A and first l columns of $\boldsymbol{V}$ respectively.
  See Method.
  All singular values are non-negative and are stored in descending order. When ICON=15000, unobtained singular values are defined -1 and are not arranged in descending order.
  This subroutine should be used when rank deficient of $\boldsymbol{A}$ is or may be found (rank ($\boldsymbol{A}$) in (m,n)). When rank ($\boldsymbol{A}$) = min(m,n), the subroutine LAXL should be used. Input parameter EPS is used for determining the rank of $\boldsymbol{A}$. It must be carefully specified. See Method.

- Example
  This example shows the method to solve the least squares minimal norm solutions for systems of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ associated with $m \times n$ coefficient matrix $\boldsymbol{A}$. Matrix $\boldsymbol{V}$ is obtained on $\boldsymbol{A}$.
  $1 \le m \le 100$, $1 \le n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(100),
     *SIG(100),VW(100)
   10 READ(5,500) M,N
      IF(M.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,M),J=1,N)
      WRITE(6,600) M,N,
     *((I,J,A(I,J),J=1,N),I=1,M)
      READ(5,510) (B(I),I=1,M)
      WRITE(6,610) (I,B(I),I=1,M)
      CALL LAXLM(A,100,M,N,B,0,0.0,SIG,
     *           A,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL SVPRT(SIG,A,100,N,N)
      WRITE(6,630)(I,B(I),I=1,N)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',5X,
     *'LEAST SQUARES AND MINIMAL',
     *' NORM SOLUTION'/6X,
     *'ROW NUMBER=',I4,5X,
     *'COLUMN NUMBER=',I4/6X,
     *'COEFFICIENT MATRIX='/
     *(10X,4('(',I3,',',I3,')',
     *E17.7,3X)))
  610 FORMAT(///10X,
     *'CONSTANT VECTOR='
     */(10X,4('(',I3,')',E17.7,3X)))
  620 FORMAT(' ',10X,'CONDITION CODE=',
     *I6)
  630 FORMAT('1',10X,
     *'SOLUTION VECTOR='
     */(10X,4('(',I3,')',E17.7,3X)))
      END
```

The subroutine SVPRT in this example prints the singular values and eigenvectors. The subroutine SVPRT is described in the example of the subroutine ASVD1.

Method

Given $m \times n$ matrix $\boldsymbol{A}$ and m-dimensional constant vector $\boldsymbol{b}$, this subroutine solves least squares minimal norm solution $\boldsymbol{x}^{+}$ for a system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

This subroutine obtains the solution $\boldsymbol{x}$ which minimizes the norm of $\boldsymbol{x}$

$$\|\boldsymbol{x}\|_2 \tag{4.3}$$

in solution $\boldsymbol{x}$ to minimize the residual norm

$$\|\boldsymbol{b} - \boldsymbol{Ax}\|_2 \tag{4.2}$$

This subroutine can handle matrix $\boldsymbol{A}$ independent of the size of m and n. $m \ge n$ is assumed to make the explanation easy in the following example.

- Singular value decomposition and least squares minimal norm solution
  Given a singular value decomposition of $\boldsymbol{A}$

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^{\mathrm{T}} \tag{4.4}$$

where $\boldsymbol{U}$ is an $m \times n$ matrix as shown in (4.5)

$$\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{I} \tag{4.5}$$

$\boldsymbol{V}$ is an n $\times$ n orthogonal matrix

$$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = \boldsymbol{VV}^{\mathrm{T}} = \boldsymbol{I} \tag{4.6}$$

$\boldsymbol{\Sigma}$ is an $n \times n$ diagonal matrix

$$\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$$

where

$$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_n \ge 0$$

If $m \times m$ orthogonal matrix $\boldsymbol{U}_c$ is produced by adding m - n column vectors in the right of $\boldsymbol{U}$ and $\boldsymbol{\Sigma}_c$ is produced by adding m - $n \times n$ zero matrix below $\boldsymbol{\Sigma}$, the singular value decomposition can be represented like following:

$$\boldsymbol{A} = \boldsymbol{U_c \Sigma_c V}^{\mathrm{T}} \tag{4.8}$$

Suppose

$$\left.\begin{aligned} \tilde{\boldsymbol{b}}_c &= \boldsymbol{U}_c^{\mathrm{T}}\boldsymbol{b}, \\ \tilde{\boldsymbol{x}} &= \boldsymbol{V}^{\mathrm{T}}\boldsymbol{x} \end{aligned}\right\} \tag{4.9}$$

Since transformation by orthogonal matrix $\boldsymbol{U}_c$ does not change the value of $\| \;\|_2$ norm, (4.10) can be obtained based upon (4.8) and (4.9)

$$\|\boldsymbol{b} - \boldsymbol{Ax}\|_2 = \left\|\boldsymbol{U}_c^{\mathrm{T}}(\boldsymbol{b} - \boldsymbol{Ax})\right\|_2 = \left\|\tilde{\boldsymbol{b}}_c - \boldsymbol{\Sigma}_c\tilde{\boldsymbol{x}}\right\|_2 \tag{4.10}$$

Therefore to minimize $\|\boldsymbol{b} - \boldsymbol{Ax}\|_2$ is reduced to minimize $\left\|\tilde{\boldsymbol{b}}_c - \boldsymbol{\Sigma}_c\tilde{\boldsymbol{x}}\right\|_2$. The last m - n rows of $\boldsymbol{\Sigma}_c$ is a zero matrix, then (4.10) is represented by

$$\left\|\tilde{\boldsymbol{b}}_c - \boldsymbol{\Sigma}_c\tilde{\boldsymbol{x}}\right\|_2 = \left\|\begin{array}{c}\tilde{\boldsymbol{b}} - \boldsymbol{\Sigma}\tilde{\boldsymbol{x}} \\ \hdashline \tilde{\boldsymbol{b}}_1\end{array}\right\|_2 \tag{4.11}$$

Where $\tilde{\boldsymbol{b}}$ is an n-dimensional vector consisting of the first n elements of $\tilde{\boldsymbol{b}}$. It can be given by

$$\tilde{\boldsymbol{b}} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{b} \tag{4.12}$$

$\tilde{\boldsymbol{b}}_1$ is m-n dimensional vector reducing $\tilde{\boldsymbol{b}}$ from $\tilde{\boldsymbol{b}}_c$. Thus, it is conclusion to minimize

$$\left\|\tilde{\boldsymbol{b}} - \boldsymbol{\Sigma}\,\tilde{\boldsymbol{x}}\right\|_2 \tag{4.13}$$

Suppose the rank of matrix $\boldsymbol{A}$ to be r, that is $\sigma_1 \geq \cdots \geq \sigma_r > 0, \sigma_{r+1} = \cdots = \sigma_n = 0$

Let $\tilde{\boldsymbol{x}} = (\tilde{x}_1,...,\tilde{x}_n)^{\mathrm{T}}$,

$\tilde{\boldsymbol{b}} = (\tilde{\boldsymbol{b}}_1,...,\tilde{\boldsymbol{b}}_n)^{\mathrm{T}}$, the first r components of the least squares minimal solution can be given by

$$\tilde{x}_i = \tilde{\boldsymbol{b}}_i / \sigma_i, \quad i = 1,2,...,r \tag{4.14}$$

and the other are arbitrary.

That is, the least squares solution is not unique. If the condition that the norm of the least squares solution is minimized is added to the condition, the solution will be obtained uniquely. For this purpose, components excepting those given in (4.14) must be

$$\tilde{x}_i = 0, \quad i = r+1,...,n \tag{4.15}$$

Taking (4.9) into consideration, since $\boldsymbol{V}$ is orthogonal transformation matrix which makes $\|\ \|_2$ norm in variable, then $\boldsymbol{x}^+$ obtained by

$$\boldsymbol{x}^+ = \boldsymbol{V}\,\tilde{\boldsymbol{x}} \tag{4.16}$$

is the least squares minimal norm solution.

Next, let $\boldsymbol{x}^+$ represented by using a matrix.

$\boldsymbol{\Sigma}^+$ which is the generalized inverse of $\boldsymbol{\Sigma}$ can be represented by

$$\boldsymbol{\Sigma}^+ = \mathrm{diag}(\sigma^+, \sigma^+, ..., \sigma_n{}^+) \tag{4.17}$$

where

$$\sigma_i{}^+ = \begin{cases} 1/\sigma_i & ,\sigma_i > 0 \\ 0 & ,\sigma_i = 0 \end{cases} \tag{4.18}$$

When this $\boldsymbol{\Sigma}^+$ is used,

$$\tilde{\boldsymbol{x}} = \boldsymbol{\Sigma}^+\tilde{\boldsymbol{b}} \tag{4.19}$$

can be obtained from (4.14) and (4.15).

From (4.16), (4.19) and (4.12), least squares minimal norm solution $\boldsymbol{x}^+$ is given as

$$\boldsymbol{x}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^{\mathrm{T}}\boldsymbol{b} \tag{4.20}$$

(4.20) can be represented by $\boldsymbol{x}^+ = \boldsymbol{A}^+\boldsymbol{b}$ by using the generalized inverse $\boldsymbol{A}^+$.

See Method of the subroutine GINV.

- Computational procedures

1) $\boldsymbol{A}$ is reduced to upper bidiagonal matrix $\boldsymbol{J}_0$ by performing the Householder transformation alternatively from left and right.

$$\boldsymbol{J_0} = \boldsymbol{P_n}\boldsymbol{P_{n-1}} \cdots \boldsymbol{P}_1\boldsymbol{A}\boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q_{n-2}} \tag{4.21}$$

For details, see Method of the subroutine ASVD1.

2) $\boldsymbol{J}_0$ is reduced to diagonal matrix $\boldsymbol{\Sigma}$ by performing orthogonal transformation alternatively from left and right.

$$\boldsymbol{\Sigma} = \boldsymbol{S}_q^{\mathrm{T}} \cdots \boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{J}_0\boldsymbol{T}_1 \cdots \boldsymbol{T}_q \tag{4.22}$$

Each $\boldsymbol{S}_i$ and $\boldsymbol{T}_i$ are given as products of two-dimensional rotational transformation represented by

$$\boldsymbol{S}_i = \boldsymbol{L}_2\boldsymbol{L}_3 \cdots \boldsymbol{L}_n \tag{4.23}$$

$$\boldsymbol{T}_i = \boldsymbol{R}_2\boldsymbol{R}_3 \cdots \boldsymbol{R}_n \tag{4.24}$$

For details, see Method of the subroutine ASVD1.

3) Matrices $\boldsymbol{U}^{\mathrm{T}}$ and $\boldsymbol{V}$ are given from (4.4), (4.21) and (4.22) as follows:

$$\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{S}_q^{\mathrm{T}} \cdots \boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{P}_n \cdots \boldsymbol{P}_1 \tag{4.25}$$

$$\boldsymbol{V} = \boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}\boldsymbol{T}_1 \cdots \boldsymbol{T}_q \tag{4.26}$$

$\boldsymbol{U}$ and $\boldsymbol{V}$ are obtained on array $\boldsymbol{A}$ and $\boldsymbol{V}$ respectively by multiplying a transformation matrix sequentially from the right.

The above discussion is adapted when ISW=1 is specified. When ISW=0 is specified, this subroutine directly computes $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{b}$ without producing $\boldsymbol{U}$. For this purpose, the transformation matrix constructing $\boldsymbol{U}^{\mathrm{T}}$ should be sequentially multiplied from the left of $\boldsymbol{b}$.

4) $\boldsymbol{x}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^{\mathrm{T}}\boldsymbol{b}$

is produced by sequentially multiplying $\boldsymbol{U}^{\mathrm{T}}$, $\boldsymbol{\Sigma}^+$, and $\boldsymbol{V}$ from the left of $\boldsymbol{b}$. When ISW=0 is specified, $\boldsymbol{U}^{\mathrm{T}}$ is not multiplied.

At multiplying $\boldsymbol{\Sigma}^+$, relative zero test is carried out for σ_i using $\|\boldsymbol{J}_0\|_\infty$ EPS as tolerance. If the singular value is less than the tolerance, it is assumed to be zero. When EPS=0.0 is specified, EPS=16u is assumed, where u is the unit round off.

When ISW=0, this subroutine directly processes these procedures. When ISW=1 is specified, this subroutine performs the singular value decomposition by using subroutine ASVD1.

For details, see Method of the subroutine ASVD1 and Reference [11].

A25-11-0401 LAXLR, DLAXLR

Iterative refinement of the least squares solution with a real matrix
CALL LAXLR(X,A,K,M,N,FA,FD,B,IP,VW,ICON)

Function

Given an approximate least squares solution $\tilde{\boldsymbol{x}}$ to the linear equations with an $m \times n$ real matrix $\boldsymbol{A}$ (rank ($\boldsymbol{A}$)=n) such as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

This subroutine refines the solution by the method of iterative modification, where $\boldsymbol{b}$ is an m-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector.

The coefficient matrix $\boldsymbol{A}$ must have been decomposed using the Householder transformation with partial pivoting which exchanges columns of the matrix as shown in Eq. (1.2),

$$\boldsymbol{R} = \boldsymbol{QA} \tag{1.2}$$

where $\boldsymbol{R}$ is an upper triangular matrix, and $\boldsymbol{Q}$ is an orthogonal matrix, also $m \geq n \geq 1$.

Parameters

X	Input. Approximate least squares solution $\boldsymbol{x}$. Output. Iteratively refined least squares solution $\tilde{\boldsymbol{x}}$. One-dimensional array of size n.
A	Input. Coefficient matrix $\boldsymbol{A}$. Two-dimensional array such as A(K,N)
K	Input. Adjustable dimension of array A (≥M).
M	Input. Number of rows m in matrix $\boldsymbol{A}$.
N	Input. Number of columns n in matrix $\boldsymbol{A}$. (See Notes.)
FA	Input. Upper triangular portion of matrix $\boldsymbol{R}$, and matrix $\boldsymbol{Q}$. Two-dimensional array such as FA(K,N). (See Notes.)
FD	Input. Diagonal portion of matrix $\boldsymbol{R}$. One-dimensional array of size n. (See Notes.)
B	Input. Constant vector $\boldsymbol{b}$. One-dimensional vector of size m.
IP	Input. Transposition vector which indicates the history of exchanging rows of the matrix $\boldsymbol{A}$ required in partial pivoting. One-dimensional array of size n. (See Notes.)
VW	Work area. One-dimensional array of size m.
ICON ..	Output. Condition code. See Table LAXLR-1.

Table LAXLR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Rank($\boldsymbol{A}$)<n was found.	Discontinued
25000	The convergence condition was not met because of very ill-conditioned coefficient matrix.	Discontinued (Refer to "Method" for the convergence condition.)
30000	K<M, M<N or N<1	Bypassed

Comments on use

- Subprograms used
  SSL II ... ULALB, MAV, AMACH, and MGSSL
  FORTRAN basic functions ... IABS, ABS and SQRT.

- Notes
  This subroutine repeatedly corrects the approximate least squares solution $\tilde{\boldsymbol{x}}$ obtained by the subroutine LAXL, and improves its accuracy.
  Therefore, prior to calling this subroutine to obtain the refined least squares solution approximate least squares solution $\tilde{\boldsymbol{x}}$ must have been obtained by calling subroutine LAXL and then the results, B, A, VW and IVW, of the subroutine LAXL must be input as the parameters X, FA, FD and IP to be used for this subroutine. In addition, this subroutine needs both the coefficient matrix $\boldsymbol{A}$ and the constant vector $\boldsymbol{b}$, therefore they must be saved before calling the subroutine LAXL in order not to lose them.
  Refer to the example shown below for a more practical use. By specifying N=-n, and Euclidean norm of the residual vector $\left(\|\boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{x}}\|_2\right)$ for the approximate least squares solution obtained by the subroutine LAXL.
  When specified, this subroutine does not perform iterative refinement for the solution, but only computes the Euclidean norm of the residual vector and outputs it to the parameter VW(1).

- Example
  A least squares solution $\tilde{\boldsymbol{x}}$ to linear equations with m unknowns and n equations (with $m \geq n$) is obtained by calling the subroutine LAXL. The least squares solution $\tilde{\boldsymbol{x}}$ is iteratively refined by this subroutine. Here $m \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(50,50),FA(50,50),X(50),
     *   B(50),VW(100),IVW(50),FD(100)
      CHARACTER*4 NT1(6),NT2(4),NT3(4)
      DATA NT1/'CO  ','EF  ','FI  ','CI  ',
     *         'EN  ','T   '/,
     *     NT2/'CO  ','NS  ','TA  ','NT  '/,
     *     NT3/'SO  ','LU  ','TI  ','ON  '/
      READ(5,500) M,N
      WRITE(6,600) M,N
      READ(5,510) ((A(I,J),I=1,M),J=1,N)
      CALL PGM(NT1,6,A,50,M,N)
      READ(5,510) (B(I),I=1,M)
      CALL PGM(NT2,4,B,M,M,1)
      DO 20 I=1,M
      X(I)=B(I)
      DO 10 J=1,N
      FA(I,J)=A(I,J)
   10 CONTINUE
   20 CONTINUE
      ISW=1
      CALL LAXL(FA,50,M,N,X,ISW,FD,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) STOP
      CALL LAXLR(X,A,50,M,N,FA,FD,B,IVW,
     * VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) STOP
      CALL PGM(NT3,4,X,N,N,1)
      STOP
  500 FORMAT(2I5)
  510 FORMAT(10F8.3)
  600 FORMAT('1',
     *  /6X,'LINEAR LEAST SQUARES SOLUTION'
     *  /6X,'ROW NUMBER=',I4
     *  /6X,'COLUMN NUMBER=',I4)
  610 FORMAT(' ',5X,'ICON OF LAXL=',I6)
  620 FORMAT(' ',5X,'ICON OF LAXLR=',I6)
      END
```

The subroutine PGM is used in this example only to print out a real general matrix, it is described in the example for the subroutine MGSM.

Method

Given the approximate least squares solution (approximate solution, hereafter), $\tilde{\boldsymbol{x}}$ to the linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the approximate solution is iteratively refined as follows:

- Principle of iterative refinement
  The iterative refinement is a method to obtain a successive improved approximate solution $\boldsymbol{x}^{(s+1)}$ (s=1,2,...) to the linear equations (4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)}=\boldsymbol{x}$

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$

$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$

s=1,2,...

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to equation (4.1).

If Eq. (4.2) is accurately computed, a refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained.

If, however, the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable, an improved solution is not obtained. (Refer to "Iterative refinement of a solution" in Section 3.4.)

- Procedure performed in this subroutine
  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by the subroutine LAXL.
  - The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq. (4.2). This is performed by calling the subroutine MAV.
  - The correction $\boldsymbol{d}^{(s)}$ is obtained by using Eq. (4.3). This is performed by calling the subroutine ULALB.
  - Finally, the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq. (4.4).

The convergence of iteration is tested as follows:
Considering u as a unit round off, the iteration refinement is assumed to converge if, at the s-th iteration step, the following relationship is satisfied.

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2 \cdot u \tag{4.5}$$

The obtained $\boldsymbol{x}$(s+1) is then taken as the final solution. However, if the relationship,

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable. The iteration refinement is assumed not to converge, and consequently the processing is terminated with ICON=25000.

A22-11-0401 LAXR, DLAXR

Iterative refinement of the solution to a system of linear equations with a real general matrix
CALL LAXR(X,A,K,N,FA,B,IP,VW,ICON)

Function

When an approximate solution $\tilde{x}$ is given to linear equations with an $n \times n$ real matrix $\boldsymbol{A}$ such as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

This subroutine refines the approximate solution by the method of iterative modification, where $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector.

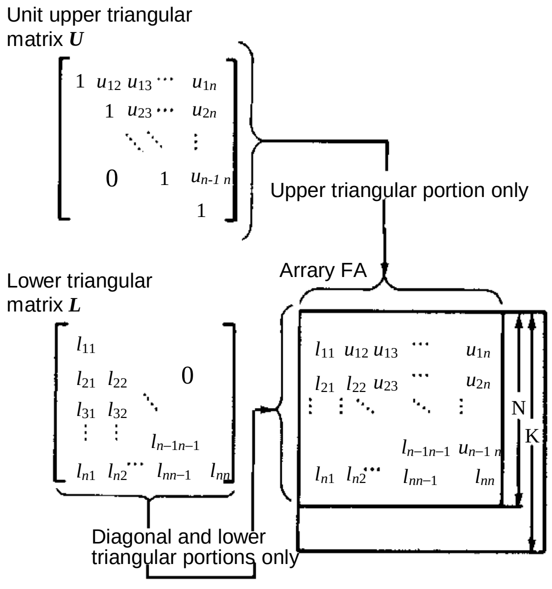
Prior to calling this subroutine, the coefficient matrix $\boldsymbol{A}$ must be LU-decomposed as shown in Eq. (1.2),

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{1.2}$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are an $n \times n$ lower triangular matrix and unit upper triangular matrix, respectively, and $\boldsymbol{P}$ is a permutation matrix which exchanges rows of the matrix $\boldsymbol{A}$ required in partial pivoting. $n \geq 1$.

Parameters

X	Input. Approximate solution vector $\boldsymbol{x}$. Output. Refined solution vector. One-dimensional array of size n.
A	Input. Coefficient matrix. Two-dimensional array, A(K,N)
K	Input. The adjustable dimension of array A (≥N).
N	Input. The order n of matrix $\boldsymbol{A}$ (See Notes.)
FA	Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$. See Fig. LAXR-1. Two-dimensional array, FA(K,N). See Notes.
B	Input. Constant vector $\boldsymbol{b}$. One-dimensional vector of size n.
IP	Input. The transposition vector which indicates the history of the rows exchange in partial pivoting. One-dimensional array of size n. Refer to Notes.
VW	Work area. One-dimensional array of size n.
ICON ..	Output. Condition code. See Table LAXR-1.



Unit upper triangular matrix $\boldsymbol{U}$
$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & \ddots & \vdots \\ 0 & & & 1 & u_{n-1\,n} \\ & & & & 1 \end{bmatrix}$
Upper triangular portion only
Arrary FA
Lower triangular matrix $\boldsymbol{L}$
$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & 0 & \\ l_{31} & l_{32} & \ddots & & \\ \vdots & \vdots & & l_{n-1\,n-1} & \\ l_{n1} & l_{n2} & \cdots & l_{n\,n-1} & l_{nn} \end{bmatrix}$
$\begin{matrix} l_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ & & & l_{n-1\,n-1} & u_{n-1\,n} \\ l_{n1} & l_{n2} & \cdots & l_{n\,n-1} & l_{nn} \end{matrix}$
N
K
Diagonal and lower triangular portions only

Fig. LAXR-1 Storage of elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ in array FA

Table LAXR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The coefficient matrix was singular.	Discontinued
25000	The convergence condition was not met because of very ill-conditioned coefficient matrix.	Discontinued (Refer to "Method" for the convergence condition.)
30000	K<N or N<1	Bypassed

Comments on use

- Subprograms used
  SSL II ... LUX, MAV, AMACH, and MGSSL
  FORTRAN basic functions ... ABS

- Notes
  This subroutine iteratively corrects the approximate solution $\tilde{x}$ obtained by subroutine LAX to get solution $\boldsymbol{x}$ with refined precision. Therefore prior to calling this subroutine, $\tilde{x}$ must be obtained by LAX and the results must be input as the parameters X, FA and IP to be used for this subroutine. In addition, because this subroutine also requires the coefficient matrix $\boldsymbol{A}$ and constant vector $\boldsymbol{b}$, they must also be prepared separately before calling LAX. Refer to the example for details. If N=$-n$ is specified, an estimated accuracy (relative error) for the approximate solution $\tilde{x}$ that is given by the subroutine LAX can be obtained. When specified, this subroutine calculates the relative error and out-

puts it to work area VW(1) without performing the iterative refinements of accuracy. Refer to "method" for estimation of accuracy.

- Example

An approximate solution vector $\tilde{\boldsymbol{x}}$ for a system of linear equations in *n* unknowns is obtained by subroutine LAX, then $\tilde{\boldsymbol{x}}$ is refined to $\boldsymbol{x}$.
$n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),FA(100,100),
     *  X(100),B(100),VW(100),IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),
     *             I=1,N)
      READ(5,510) (B(I),I=1,N)
      WRITE(6,610) (I,B(I),I=1,N)
      DO 20 I=1,N
      X(I)=B(I)
      DO 20 J=1,N
      FA(J,I)=A(J,I)
   20 CONTINUE
      EPSZ=0.0E0
      ISW=1
      K=100
      CALL LAX(FA,K,N,X,EPSZ,ISW,IS,VW,IP,
     * ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL LAXR(X,A,K,N,FA,B,IP,VW,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,640) (I,X(I),I=1,N)
      DET=IS
      DO 30 I=1,N
      DET=DET*FA(I,I)
   30 CONTINUE
      WRITE(6,650) DET
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',10X,'**COEFFICIENT MATRIX'
     * /12X,'ORDER=',I5/(10X,4('(',I3,',',
     * I3,')',E17.8)))
  610 FORMAT(///10X,'CONSTANT VECTOR'
     * /(10X,4('(',I3,')',E17.8)))
  620 FORMAT('0',10X,'LAX  ICON=',I5)
  630 FORMAT('0',10X,'LAXR ICON=',I5)
  640 FORMAT('0',10X,'SOLUTION VECTOR'
     * /(10X,4('(',I3,')',E17.8)))
  650 FORMAT(///10X,
     * 'DETERMINANT OF COEFFICIENT MATRIX=',
     * E17.8)
      END
```

Method

Given the approximate solution $\tilde{\boldsymbol{x}}$, to the linear equations,

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principle of iterative refinement

The iterative refinement is a method to obtain a successively improved approximate solution $\boldsymbol{x}^{(s+1)}$ (s=1,2,...) to the linear equations (4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)} = \tilde{\boldsymbol{x}}$

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$
$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$
$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$
$$s=1,2,...$$

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to equation (4.1). If Eq. (4.2) is accurately computed, a refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained.
If, however, the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable, an improved solution is not obtained.
(Refer to "Iterative refinement of a solution" in Section 3.4.)

- Procedure performed in this subroutine

Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by the subroutine LAX.
Then this subroutine repeats the following steps:
The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq. (4.2).
This is performed by calling the subroutine MAV.
The correction $\boldsymbol{d}^{(s)}$ is obtained next by using Eq. (4.3).
This is performed by calling the subroutine LUX.
Finally the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq. (4.4).

The convergence of iteration is tested as follows:
Considering u as a unit round off, the iterative refinement is assumed to converge if, as the s-th iteration step, the following relationship is satisfied.

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2u \tag{4.5}$$

The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as the final solution.
However, if the relation,

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of coefficient matrix $\boldsymbol{A}$ is not suitable. The iterative refinement is assumed not to converge, and consequently the processing is terminated with ICON = 25000.

Accuracy estimation for approximate solution

Suppose the error for the approximate solution $\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}$ ($= \boldsymbol{x}^{(1)} - \boldsymbol{x}$), its relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(1)}\right\|_\infty$ If this iteration method converges, $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$. The relative error for the approximate solution is therefore estimated by $\left\|\boldsymbol{d}^{(1)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(1)}\right\|_\infty$ (Refer to "Accuracy estimation for approximate solution" in Section 3.4.)

For further details, see to References [1], [3], and [5].

A52-11-0101 LBX1,DLBX1

A system of linear equations with a real general band matrix (Gaussian elimination method)
CALL LBX1(A,N,NH1,NH2,B,EPSZ,ISW,IS,FL, VW,IP,ICON)

Function

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

This subroutine solves a system of linear equations(1.1) by using the Gaussian elimination method.

Where $\boldsymbol{A}$ is an $n\times n$ real general band matrix with lower band width h_1, and upper band width h_2, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. $n>h_1\geq 0$, $n>h_2\geq 0$.

Parameters

A..... Input. Coefficient matrix $\boldsymbol{A}$.
The contents of A are altered on output.
Matrix $\boldsymbol{A}$ is stored in one-dimensional array of size $n\cdot\min(h_1+h_2+1,n)$ in the compressed mode for real general band matrices.

N..... Input. Order n of coeffcient matrix $\boldsymbol{A}$.

NH1..... Input. Lower band width h_1.

NH2..... Input. Upper band width h_2.

B..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

ESPZ.. Input. Tolerance for relative zero test of pivots in decomposition process of matrix $\boldsymbol{A}(\geq 0.0)$.When this is 0.0, the standard value is used.
(See Notes.)

ISW... Input. Control informaltion
When solving l (≥ 1) systems of linear equations with the identical coefficient matrix, ISW can be specifled as follows:
ISW=1... The first system is solved.
ISW=2... The 2nd to l th systems are solved. However, only parameter B is specified for each constant vector $\boldsymbol{b}$ of the systems with the rest unchanged.
(See Motes.)

IS.... Output. Information for obtaining the determinant of the matrix $\boldsymbol{A}$. (Refer to Notes.)

FL.... Work area.
One-dimensional array of size $(n-1)\cdot h_1$.

VW.... Work area. One-dimensional array of size n.

IP.... Work area. One-dimensional array of size n.

ICON.. Output. Condition code. Refer to Table LBX1-1.

Table LBX1-1. Condition codes

Code	Meaning	Processing
0	No error	
20000	The relatively zero pivot occured. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	N≤NH1,N≤NH2,NH1<0,NH2<0, EPSZ<0.0 or ISW≠1,2.	Bypassed

Comments on use

- Subprograms used
SSL II..... BLU1, BLUX1, AMACH, MGSSL
FORTRAN basic functions..... ABS, MIN0

- Notes
This subroutine assumes that the relatively zero pivot occurs when the absolute value of the pivot is smaller than the largest absolute value of the elements, in the coefficient matrix, multiplied by EPSZ in the LU-decomposition using the Gaussian elimination method. In such a case, the processing is discontinued with ICON = 20000. The standrd value of EPSZ is 16 u, where u is the unit round off.

If the processing is to proceed at a lower pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.

When solving successive systems of linear equations with the identical coefficient matrix, ISW=2 should be given for the second time and subsequently. By setting ISW = 2,LU-decomposed coefficient matrix $\boldsymbol{A}$ is bypassed so that the execution time is reduced. In this case, the IS value is the same as when ISW=1.

The determinant of the coefficient matrix $\boldsymbol{A}$ can be obtained by multiplying the product of the n array elements A ($i\cdot h+1$) , i= 0,1,...,$n-1$ by the IS value, where $h = \min(h_1+h_1+1,n)$.

This subroutine, by making use of band matrix characteristics, saves data storage area. In some cases, however, depending on the size of the band width, a larger data storage area may be required (including work area) than used by subroutine LAX provided for real general matrices. If that is the case, subroutine LAX may be used to save more data storage area. This subroutine is especially useful for the case where the upper and lower band widths of the coefficient matrix of order n are approximately less than n / 3, provided both the band widths are equal.

- Example
  In this example, l systems of linear equations in n unknown with the identical matrix are solved. $n \leq 100$, $h_1 \leq 20$ and $h_2 \leq 20$.

```
C     ** EXAMPLE **
      DIMENSION A(4100),B(100),IP(100),
     *   FL(1980),VW(100)
      CHARACTER*4 NT1(6),NT2(4),NT3(4)
      DATA NT1/'CO  ','EF  ','FI  ','CI  ',
     *         'EN  ','T   '/,
     *     NT2/'CO  ','NS  ','TA  ','NT  '/,
     *     NT3/'SO  ','LU  ','TI  ','ON  '/
      READ(5,500) N,NH1,NH2,L
      WRITE(6,600) N,NH1,NH2
      NT=N*MIN0(N,NH1+NH2+1)
      READ(5,510) (A(I),I=1,NT)
      M=1
      ISW=1
      EPSZ=1.0E-6
      CALL PBM(NT1,6,A,N,NH1,NH2)
   10 READ(5,510) (B(I),I=1,N)
      CALL PGM(NT2,4,B,N,N,1)
      CALL LBX1(A,N,NH1,NH2,B,EPSZ,ISW,IS,
     *          FL,VW,IP,ICON)
      WRITE(6,610) ICON
      IF(ICON.EQ.0) GOTO 20
      WRITE(6,620)
      STOP
   20 CALL PGM(NT3,4,B,N,N,1)
      M=M+1
      ISW=2
      IF(L.GT.M) GO TO 10
      WRITE(6,630)
      STOP
  500 FORMAT(4I4)
  510 FORMAT(4E15.7)
  600 FORMAT('1'
     * ///5X,'LINEAR EQUATIONS  AX=B'
     * /5X,'ORDER=',I4
     * /5X,'SUB-DIAGONAL LINES=',I4
     * /5X,'SUPER-DIAGONAL LINES=',I4)
  610 FORMAT(' ',4X,'ICON=',I5)
  620 FORMAT(' '/5X,
     * '** ABNORMAL END **')
  630 FORMAT(' '/5X,'** NORMAL END **')
      END

C     ** MATRIX PRINT (REAL BAND) **
      SUBROUTINE PBM(ICOM,L,A,N,NH1,NH2)
      DIMENSION A(1)
      CHARACTER*4 ICOM(1)
      WRITE(6,600) (ICOM(I),I=1,L)
      M=MIN0(NH1+NH2+1,N)
      IE=0
      IB=1
      DO 10 I=1,N
      J=MAX0(1,I-NH1)
      KIB=IB
      KIE=IE+MIN0(NH1+1,I)+MIN0(NH2,N-I)
      WRITE(6,610) I,J,(A(K),K=KIB,KIE)
      IE=IE+M
      IB=IB+M
   10 CONTINUE
      RETURN
  600 FORMAT(/10X,35A2)
  610 FORMAT(/3X,'(',I3,',',I3,')',
     *3(2X,E16.7)/(12X,3(2X,E16.7)))
      END
```

Subroutines PGM and PBM are used to print out a real matrix and a real band matrix, respectively. The description on subroutine PGM is shown in the example for subroutine MGSM.

Method

A system of linear equations (4.1) with a real general band matrix $\boldsymbol{A}$ as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

are solved using the following procedure.

- LU-decomposition of the coefficient matrix $\boldsymbol{A}$ (Gaussian elimination method)
  The coefficient matrix $\boldsymbol{A}$ is decomposed into the unit lower band matrix $\boldsymbol{L}$ and the upper band matrix $\boldsymbol{U}$.

$$\boldsymbol{A} = \boldsymbol{LU} \tag{4.2}$$

Subroutine BLU1 is used for this operation.

- Solving $\boldsymbol{LUx} = \boldsymbol{b}$ (Forward and backward substitutions)
  Solving the linear equations (4.1) is equivalent to solving the next linear equations (4.3) .

$$\boldsymbol{LUx} = \boldsymbol{b} \tag{4.3}$$

  This equation (4.3) is resolved into two equations (4.4) and (4.5)

$$\boldsymbol{Ly} = \boldsymbol{b} \tag{4.4}$$
$$\boldsymbol{Ux} = \boldsymbol{y} \tag{4.5}$$

  and then solution is obtained using forward and backward substitutions. Subroutine BLUX1 is used for this procedure. For more information, see Reference [1], [3],[4] and [8].

A52-11-0401 LBX1R, DLBX1R

Interrative refinement of the solution to a system of linear equations with a real band general band matrix.
CALL LBX1R (X, A, N, NH1, NH2, FL, FU, B, IP, VW, ICON)

Function

Give an approximate solution $\tilde{x}$ to linear equations with $n \times n$ real band matrix $\boldsymbol{A}$ of lower band width h_1 and upper band width h_2 such as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

this subroutine refines the approximate solution by the method of iterative modification, where $\boldsymbol{b}$ is an n-dimensional real constant vector and $\boldsymbol{x}$ is an n-dimensional solution vetor.

Befor this subroutine is called, the coeffcient matrix $\boldsymbol{A}$ must be LU-decomposed as shown is Eq. (1.2)

$$\boldsymbol{A} = \boldsymbol{LU} \tag{1.2}$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are an $n \times n$ unit lower band matrix and upper band matrix, repectively. Also $n > h_1 \geq 0$ and $n > h_2 \geq 0$.

Parameters

X..... Input. Approximate solution vector $\tilde{x}$.
Output. Iteratively refined solution vector $\boldsymbol{x}$
One-dimensional array of size n.

A..... Input. Coefficient matrix $\boldsymbol{A}$.
Compressed mode for a band matirx.
One-dimensional array of size $n \cdot \min(h_1+h_2+1,n)$

N..... Input. Order n of the coefficient matrix $\boldsymbol{A}$.
(see "Notes")

NH1... Input. Lower band width h_1 of the coefficient matrix $\boldsymbol{A}$.

NH2... Input. Upper band width h_2 of the coefficient matirx $\boldsymbol{A}$.

FL.... Input. Matrix $\boldsymbol{L}$
Refer to Fig. LBX1R-1
One-dimensional arrary of size $(n-1)\cdot h_1$. (See "Notes".)

FU.... Input. Matrix $\boldsymbol{U}$
Refer to Fig. LBX1R-2
One-dimensional array of size $n \cdot \min(h_1+h_2+1,n)$. (see"Notes".)

B.... Input. Constant vector $\boldsymbol{b}$.
One-dimensional array of size n.

IP.... Input. Transposition vector indicating the history of exchangeing rows in partial pivoting.
One-dimensional array of size n.
(See "Notes".)

VW.... Work area. One-dimensional arrary of size n.

ICON.. Output. Condition code. See Table LBX1R-1.

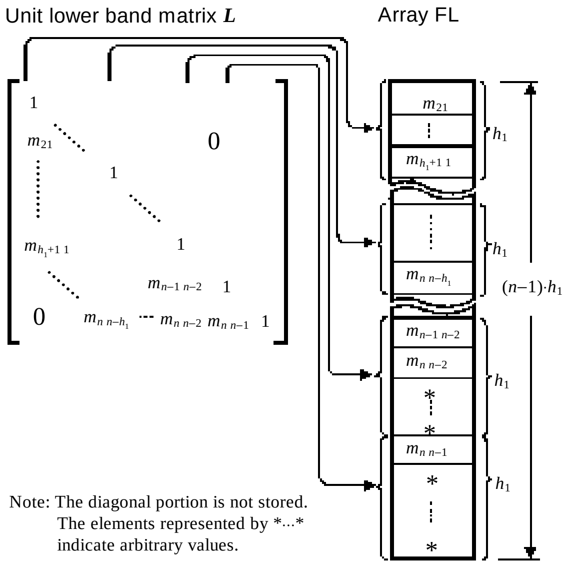


Fig. LBX1R-1 Storing method for each element of $\boldsymbol{L}$ into array FL.

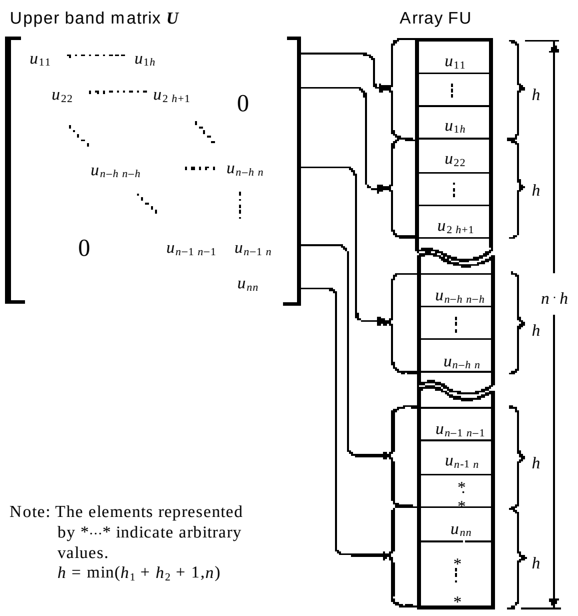


Fig. LBX1R-2 Storing method for each element of $\boldsymbol{U}$ into array FU

Comments on use

- Subprograms used
  SSL II ...BLUX1, MBV, AMACH, and MGSSL
  FORTRAN basic functions ..ABS and MIN0

Table LBX1R-1 Condition codes

code	Meaning	Processing
0	No error	
20000	Coefficient matrix was singular.	Discontinued
25000	The convergence condition was not met because of very illconditioned coefficient matrix.	Discontinued (Refer to "Method" for convergence condition.)
30000	N=0,N ≤ NH1,N ≤ NH2,NH1<0 or NH2<0	Bypassed

- Notes

  This subroutine repeatedly corrects the approximate solution $\tilde{x}$ obtained by subroutine LBX1 and improves its accuracy.

  Therefore, subroutine LBX1 must have been called to obtain the approximate solution $\tilde{x}$ before calling this subroutine to obtain the iteratively refined solution. In this case, the parameters X, FL, FU and IP must be each assigned values of the parameters B, FL, A and IP ,of subroutine LBX1. (Refer to descriptions of subroutine LBX1.) In addition, this subroutine needs both the coefficient matrix $\boldsymbol{A}$ and the constant vector $\boldsymbol{b}$. Therefore they must be saved before calling subroutine LBX1 so as not to lose them. For a practical use, refer to the example shown below.

  By specifying N = $-n$, an estimated accuracy (relative error) for the approximate solution $\tilde{x}$ obtained by subroutine LBX1 can be obtained.

  This subroutine does not carry out iterative refinement of the solution, but only computes the relative error and outputs it to the parameter VW(1). For the accuracy estimation, refer to **Method**.

- Example

  An approximate solution $\tilde{x}$ to n-dimensional linear equations is obtained by calling subroutine LBX1, and after that the $\tilde{x}$ is iteratively refined by using this subroutine. Here $n \le 50$, $h_1 \le 10$ and $h_2 \le 10$.

```
C      **EXAMPLE**
       DIMENSION A(1050),FL(490),FU(1050),
      *  X(50),B(50),VW(50),IP(50)
       CHARACTER*4 NT1(6),NT2(4),NT3(4)
       DATA NT1/'CO  ','EF  ','FI  ','CI  ',
      *         'EN  ','T   '/,
      *     NT2/'CO  ','NS  ','TA  ','NT  '/,
      *     NT3/'SO  ','LU  ','TI  ','ON  '/
       READ(5,500) N,NH1,NH2
       WRITE(6,600) N,NH1,NH2
       NT0=MIN0(NH1+NH2+1,N)
       NT=N*NT0
       READ(5,510) (A(I),I=1,NT)
       CALL PBM(NT1,6,A,N,NH1,NH2)
       READ(5,510) (B(I),I=1,N)
       CALL PGM(NT2,4,B,N,N,1)
       DO 10 I=1,N
    10 X(I)=B(I)
       DO 20 I=1,NT
    20 FU(I)=A(I)
       CALL LBX1(FU,N,NH1,NH2,X,0.0,1,IS,FL,
      *  VW,IP,ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) STOP
       CALL LBX1R(X,A,N,NH1,NH2,FL,FU,B,IP,
      *  VW,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) STOP
       CALL PGM(NT3,4,X,N,N,1)
       DET=IS
       IC=1
       DO 30 I=1,N
       DET=DET*FU(IC)
    30 IC=IC+NT0
       WRITE(6,630) DET
       STOP
   500 FORMAT(3I4)
   510 FORMAT(4E15.7)
   600 FORMAT('1'
      * /6X,'LINEAR EQUATIONS AX=B'
      * /6X,'ORDER=',I4
      * /6X,'SUB-DIAGONAL LINES=',I4
      * /6X,'SUPER-DIAGONAL LINES=',I4)
   610 FORMAT(' ',5X,'ICON OF LBX1=',I4)
   620 FORMAT(' ',5X,'ICON OF LBX1R=',I6)
   630 FORMAT(' ',5X,'DETERMINANT=',E15.7)
       END
```

Subroutines PBM and PGM are used only to print out a band matrix and a real general matrix, respectively.

The descriptions on those programs are shown in the examples of the subroutines LBX1 and MGSM, respectively.

Method

Given an approximate solution $\tilde{x}$ to the linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principale of iterative refinement

  The iterative refinement is a method to obtain a successively improved approximate solution $\boldsymbol{x}$ $(s+1)$ to the linear equations (4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)} = \tilde{\boldsymbol{x}}$:

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$

$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$

$$s = 1,2,...$$

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to equation (4.1). If Eq. (4.2) is accurately computed, a refined solution of $\boldsymbol{x}^{(1)}$ is obtained. If, however, the condition of coefficient matrix $\boldsymbol{A}$ is not suitable, no improved solution is obtained. (See "Iterative refinement of a solution" in Section 3.4.)

- Procedure performed in this subroutine
  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by subroutine LBX1.
  This subroutine repeats the following steps:
  The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq. (4.2). Subroutine MBV is used for this operation. The correction $\boldsymbol{d}^{(s)}$ is obtained next by using Eq. (4.3). Subroutine BLUX1 is used for this operation.
  Finally, the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq. (4.4).

  The convergence of iteration is tested as follows:
  Considering u as a unit round off, the iteration refinement is assumed to converge if, at the s-th iteration step, the following relationship is satisfied.

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2 \cdot u \tag{4.5}$$

The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as the final solution. However, if the relationship.

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s-1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable. The iteration refinement is assumed not to converge, and consequently the processing is terminated with ICON = 25000.

- Accuracy estimation for approximate solution
  Suppose that the error for the approximate solution $\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}$ $(=\boldsymbol{x}^{(1)}-\boldsymbol{x})$, its relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(s)}\right\|_\infty$. If this iteration method converges $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$ .Therefore the relative error for the approximate solution is estimated by $\left\|\boldsymbol{d}^{(1)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(1)}\right\|_\infty$ (See "Accuracy estimation for approximate solution" in Section 3.4.) For further details, refer to References [1], [3] and [5].

A22-15-0101 LCX,DLCX

A system of linear equations with a complex general matrix (Crout's method).
CALL LCX (ZA, K, N, ZB, EPSZ, ISW, IS, ZVW, IP,ICON)

Function

This subroutine solves a system of linear equations, as shown in (1.1) using the Crout's method.

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

$\boldsymbol{A}$ is an $n \times n$ non-singular complex general matrix, $\boldsymbol{b}$ is an n-dimensional complex constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. $n \geq 1$.

Parameter

ZA	Input. Coefficient matrix $\boldsymbol{A}$. The contents of ZA are overridden after operation. ZA is a complex two-dimensional array, ZA (K, N).
K	Input. Adjustable dimension of the array ZA ($\geq$ N)
N	Input. Order n of the coefficient matrix $\boldsymbol{A}$.
ZB	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. ZB is a complex one-dimensional array of size n.
EPSZ	Input. Tolerance for relative zero test of pivots in decomposition of $\boldsymbol{A}$ ($\geq$ 0.0). If EPSZ is 0.0, a standard value is used. (Refer to Notes.)
ISW	Input. Control information When l ($\geq$ 1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows: ISW = 1 .... The first system is solved. ISW = 2 .... The 2nd to l-th systems are solved. However, only parameter ZB is specified for each constant vector $\boldsymbol{b}$ of the systems of equations, with the rest unchanged. (Refer to Notes.).
IS	Output. Information for obtaining the determinant of the matrix $\boldsymbol{A}$. If the n elements of the calculated diagonal of array ZA are multiplied by IS, the determinant is obtained.
ZVW	Work area. ZVW is a complex one-dimensional array of size n.
IP	Work area. IP is a one-dimensional array of size n.
ICON ..	Output. Condition code. Refer to Table LCX-1.

Table LCX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Either all of the elements of some row were zero or the pivot became relatively zero. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	K<N, N<1, EPSZ<0.0 or ISW ≠ 1,2	Bypassed

Comments on use

- Subprograms used
  SSL II ..... CLU, CLUX, AMACH, CSUM, MGSSL
  FORTRAN basic functions ..... REAL, AIMAG, ABS

- Notes
  If EPSZ is set to 10^{-s}, this value has following meaning: while performing the LU-decomposition by Crout's method, if the loss of over s significant digits occured for both real and imaginary parts of the pivot, the LU-decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero.
  Let u be the unit round off, then the standard value of EPSZ is $16u$. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.
  When solving successive systems of linear equations with the idenficial coefficient matrix, computation can be performed by setting ISW = 2 after the first system of equations. By setting ISW = 2, the LU decomposition of the coefficient matrix $\boldsymbol{A}$ is bypassed so the computation time is reduced. In this case, the value of IS is the same as when ISW = 1.

- Example
  l systems of linear equations in n unknown with the identical complex coefficient matrix are solved. $n \leq$ 100.

```
C     **EXAMPLE**
      DIMENSION ZA(100,100),ZB(100),
     *          ZVW(100),IP(100)
      COMPLEX ZA,ZB,ZVW,ZDET
      READ(5,500) N,L
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,ZA(I,J),J=1,N),
     *             I=1,N)
      M=1
      ISW=1
      EPSZ=1.0E-6
   10 READ(5,510) (ZB(I),I=1,N)
      WRITE(6,610) (I,ZB(I),I=1,N)
      CALL LCX(ZA,100,N,ZB,EPSZ,ISW,IS,
     *         ZVW,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,630) (I,ZB(I),I=1,N)
      IF(L.EQ.M) GOTO 20
      M=M+1
      ISW=2
      GOTO 10
   20 ZDET=CMPLX(FLOAT(IS),0.0)
      DO 30 I=1,N
      ZDET=ZDET*ZA(I,I)
   30 CONTINUE
      WRITE(6,640) ZDET
      STOP
  500 FORMAT(2I5)
  510 FORMAT(5(2F8.3))
  600 FORMAT('1',10X,'** COMPLEX MATRIX **'
     * /12X,'ORDER=',I5/(3X,2('(',I3,',',I3,
     * ')',2E15.8,2X)))
  610 FORMAT(///10X,'COMPLEX CONSTANT ',
     * 'VECTOR',/(5X,3('(',I3,')',2E15.8,
     * 2X)))
  620 FORMAT('0',10X,'CONDITION CODE=', I5)
  630 FORMAT('0',10X,'SOLUTION VECTOR',
     * /(5X,3('(',I3,')',2E15.8,2X)))
  640 FORMAT(///10X,'DETERMINANT OF MATRIX',
     * 2E15.8)
      END
```

Method

A system of linear equations (4.1).

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (4.1)$$

is solved using the following procedure.

- LU decomposition of the coefficient matrix ***A*** (Crout's method)
  The coefficient matrix ***A*** is decomposed into a lower triangular matrix ***L*** and a unit upper trangular matrix ***U***. To reduce the rounding error, partial pivoting is performed in the decomposition process.

$$\boldsymbol{PA} = \boldsymbol{LU} \qquad (4.2)$$

P is the permutation matrix which performs the row exchanges required in partial pivoting.
Subroutine CLU is used for this operation.

- Solving ***LUx*** = ***Pb*** (forward and backward substitution)
  To solve equation (4.1) is equivalent to solving the next system of linear equations

$$\boldsymbol{LUx} = \boldsymbol{Pb} \qquad (4.3)$$

Equation (4.3) is resolved into two equations

$$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad (4.4)$$
$$\boldsymbol{Ux} = \boldsymbol{y} \qquad (4.5)$$

Then the solution is obtained using forward substitution and backward substitution. Subroutine CLUX is used for these operations. For more information, see References [1], [3], and [4].

A22-15-0401 LCXR,DLCXR

Iterative refinement of the solution to a system of linear equation with a complex general matrix
CALL LCXR(ZX, ZA, K, N, ZFA, ZB, IP, ZVW, ICON)

Function

When an approximate solution $\tilde{x}$ is given to linear equations with an $n \times n$ complex matrix,

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

this subroutine refines the approximate solution by the method of iterative modificton, where $\boldsymbol{b}$ is an n-dimensional complex constant vector and $\boldsymbol{x}$ is an n-dimensional solution vector.

Prior to calling this subroutine, the coefficient matrix $\boldsymbol{A}$ must be LU-decomposed as shown in Eq.(1.2),

$$\boldsymbol{PA} = \boldsymbol{LU} \qquad (1.2)$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are $n \times n$ lower triangular matrix and unit upper triangular matrix respectively, and $\boldsymbol{P}$ is a permutation matrix which exchanges rows of the matrix $\boldsymbol{A}$ required in partial pivoting.
Also, n≥1.

Parameters

ZX .... Input. Approximate solution vector $\tilde{x}$.
Output. Iteratively refined solution vector $\boldsymbol{x}$.
Complex one-dimensional array of size n.

ZA .... Input. Coefficient matrix $\boldsymbol{A}$.
Complex two-dimensional array, ZA (K,N).

K .... Input. Adjustable dimension of the array ZA, ZFA (≥ N)

N .... Input. Order n of the coefficient matrix $\boldsymbol{A}$.
(See "Comments on use".)

ZFA .... Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.
Refer to Fig. LCXR-1.
Complex two-dimensional array, ZFA (K,N).
(See "Comments on use".)

ZB .... Input. Constant vector $\boldsymbol{b}$.
Complex one-dimensional array of size $\boldsymbol{n}$.

IP .... Input. Transposition vector which indicates the history of exchanging rows required in partial pivoting. One dimensional array of size n.
(See "Comments on use".)

ZVW .... Work area. Complex one-dimensional array of size n.

ICON .. Output. Condition code. See Table LCXR-1.

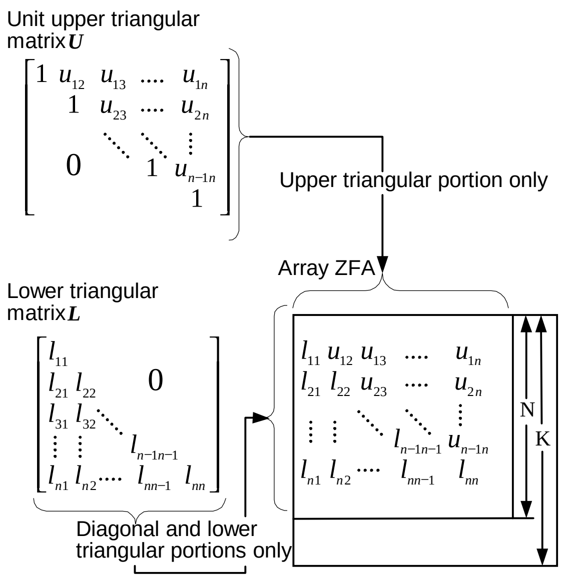

Unit upper triangular matrix $\boldsymbol{U}$
Upper triangular portion only
Array ZFA
Lower triangular matrix $\boldsymbol{L}$
Diagonal and lower triangular portions only
N
K

Fig. LCXR-1 Storing method for each elements of $\boldsymbol{L}$ and $\boldsymbol{U}$ in array ZFA

Table LCXR-1 Condition codes

Code	Meaning	Processing
0	No errors	
20000	Coefficient matrix was singular.	Discontinued
25000	Convergence condition was not met because of very illconditioned coefficient matrix.	Discontinued (Refer to the paragraph "Method" for the convergence condition.)
30000	Either K<N or N<1.	Bypassed

Comments on use

- Subprograms used
SSL II ... CLUX,MCV,AMACH,CSUM,MGSSL
FORTRAN basic functions ... REAL,AIMAG,ABS,IABS

- Notes
This subroutine iteratively corrects the approximate solution $\tilde{x}$ obtained by the subroutine LCX, and refines its precision. Therefore , the subroutine LCX must be called in advance to obtain the approximate solution $\tilde{x}$ prior to calling this subroutine to obtain the refined solution. In this case, the parameters ZX, ZFA and IP must be each assigned the outputs of subroutine LCX that was called prior to this subroutine. In addition, this subroutine requires both the coefficient matrix $\boldsymbol{A}$ and the constant vector $\boldsymbol{b}$, so that they must be saved before calling subroutine LCX in order not to lose them. Refer to the example below for detail. By specifying N = – n, an estimated accuracy (relative error) for the approximate solution $\tilde{x}$ obtained in the

subroutine LCX can be obtained. When specified, this subroutine does not perform iterative refinement for the solution, but only computes a relative error and outputs it to the parameter ZVW(1). For details about the accuracy estimation, refer to the following paragraph "Method".

- Example
  An approximate solution $\tilde{x}$ to n-dimensional linear equations is obtained first by calling the subroutine LCX and after that it is iteratively refined by using this subroutine. Here n≤100.

```
C     **EXAMPLE**
      DIMENSION ZA(100,100),ZFA(100,100),
     *  ZX(100),ZB(100),ZVW(100),IP(100)
      COMPLEX ZA,ZFA,ZX,ZB,ZVW,ZDET
      READ(5,500) N
      IF(N.LE.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,ZA(I,J),J=1,N),
     *             I=1,N)
      READ(5,510) (ZB(I),I=1,N)
      WRITE(6,610) (I,ZB(I),I=1,N)
      DO 10 I=1,N
      ZX(I)=ZB(I)
      DO 10 J=1,N
      ZFA(I,J)=ZA(I,J)
   10 CONTINUE
      K=100
      CALL LCX(ZFA,K,N,ZX,0.0,1,IS,ZVW,IP,
     * ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      CALL LCXR(ZX,ZA,K,N,ZFA,ZB,IP,ZVW,
     * ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,640) (I,ZX(I),I=1,N)
      ZDET=IS
      DO 20 I=1,N
      ZDET=ZDET*ZFA(I,I)
   20 CONTINUE
      WRITE(6,650) ZDET
      STOP
  500 FORMAT(I3)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,'COEFFICIENT MATRIX'
     * /6X,'ORDER=',I5/(6X,2('(',I3,',',I3,
     * ')',2E15.8,1X)))
  610 FORMAT(' ',5X,'CONSTANT VECTOR'
     * /(6X,3('(',I3,')',2E15.8)))
  620 FORMAT(' ',5X,'ICON OF LCX=',I5)
  630 FORMAT(' ',5X,'ICON OF LCXR=',I5)
  640 FORMAT(' ',5X,'IMPROVED SOLUTION'
     * /(6X,3('(',I3,')',2E15.8,1X)))
  650 FORMAT(' ',5X,'DETERMINANT=',2E15.8)
      END
```

Method

Given an approximate solution, $\tilde{x}$, to the linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principle of iterative refinement
  The iteratile refinement is a method to obtain a successively improved approximate solution $\boldsymbol{x}(s+1)$ to the linear equations(4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)}=\tilde{\boldsymbol{x}}$

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$
$$\boldsymbol{Ad}^{(s)}=\boldsymbol{r}^{(s)} \tag{4.3}$$
$$\boldsymbol{x}^{(s+1)}=\boldsymbol{x}^{(s)}+\boldsymbol{d}^{(s)} \tag{4.4}$$
$$s=1,2,....$$

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to equation (4.1). If Eq. (4.2) is accurately computed, an refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained. If the conditions of coeffcient matrix $\boldsymbol{A}$ are not suitable, however, no improved solution is obtained. (See “Iterative refinement of a solution” in Section 3.4.)

- Procedure performed in this subroutine
  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by the subroutine LCX. Then this subroutine repeats the following steps: The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq.(4.2).
  The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq.(4.2) this is done by calling subroutine MCV.
  The correction $\boldsymbol{d}^{(s)}$ is obtained next by using Eq.(4.3) and calling subroutine CLUX.
  Finally the modified approximste solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq.(4.4).

The iteration convergence is tested as follows. Considering u as a unit round off, the itereative refinement is assumed to converge if, in the s-th iteration step, the following relationship is satisfied

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty \Big/ \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2u \tag{4.5}$$

The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as a final solution. However, if the relation,

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2}\cdot\frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ are not suitable. The iterative refinement is assumed not to converge, and consequently the processing is terminated with ICON=25000.

- Accuracy estimation for approximate solution Suppose the error for the approximate solution

$\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}$ ($=\boldsymbol{x}^{(1)} - \boldsymbol{x}$), its relative error is represented by $\|\boldsymbol{e}^{(1)}\|_\infty / \|\boldsymbol{x}^{(1)}\|_\infty$. If this iteration method converges, $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$. The relative error for the approximate solution is therefore estimated by $\|\boldsymbol{d}^{(1)}\|_\infty / \|\boldsymbol{x}^{(1)}\|_\infty$.

(See "Accuracy estimation for approximate solution" in Section 3.4.) For further details, see References [1], [3] and [5]

A22-51-0702 LDIV, DLDIV

The inverse of a positive-definite symmetric matrix decomposed into the factors $\boldsymbol{L}$,$\boldsymbol{D}$ and $\boldsymbol{L}^{\boldsymbol{T}}$
CALL LDIV (FA, N, ICON)

Function

The inverse matrix $\boldsymbol{A}^{-1}$ of an $n \times n$ positive-definite symmetric matrix $\boldsymbol{A}$ given in decomposed form $\boldsymbol{A}=\boldsymbol{LDL}^{\mathrm{T}}$ is computed.

$$\boldsymbol{A}^{-1} = \left(\boldsymbol{L}^{\mathrm{T}}\right)^{-1} \boldsymbol{D}^{-1} \boldsymbol{L}^{-1} \tag{1.1}$$

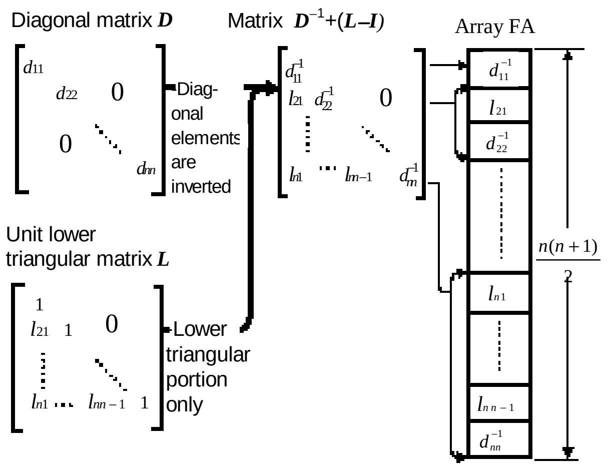
$\boldsymbol{L}$ and $\boldsymbol{D}$ are, respectively, an $n \times n$ unit lower triangular and a diagonal matrices. n≥1.

Parameters

FA........ Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$.
See Fig. LDIV-1.
Output. Inverse $\boldsymbol{A}^{-1}$.
FA is a one-dimensional array of size $n(n+1)/2$ that contains $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$ in the compressed mode for symmetric matrices.

N......... Input. Order n of the matrices $\boldsymbol{L}$ and $\boldsymbol{D}$.

ICON. Output. Condition code.
See Table LDIV-1.



Diagonal matrix D
Matrix D⁻¹+(L–I)
Array FA
Diag-onal elements are inverted
Unit lower triangular matrix L
Lower triangular portion only
n(n+1)/2

Note: The diagonal and lower triangular portions of the matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$ are stored in the one-dimensional array FA in the compressed mode for symmetric matrices.

Fig. LDIV-1 Storage of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$

Table LDIV-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Matrix was not a positive-definite.	Continued
30000	N<1	Bypassed

Comments on use

- Subprograms used
  SSL II........MGSSL
  FORTRAN basic function........none

- Notes
  Prior to calling this subroutine, LDL$^{\mathrm{T}}$-decomposed matrix must be obtained by subroutine SLDL and must be input as the parameter FA to be used. (Refer to the example). In this routine, the diagonal elements of the array D must be given as $\boldsymbol{D}^{-1}$. $\boldsymbol{D}^{-1}$ is output by the subroutine SLDL. The subroutine LSX should be used for solving a system of linear equations. Solving a system of linear equations by first obtaining the inverse matrix should be avoided since more steps of calculation are required. This subroutine should be used only when the inverse matrix is inevitable.

- Example
  The inverse of an $n \times n$ positive symmetric matrix is obtained. $n \le 100$

```
C     **EXAMPLE**
      DIMENSION A(5050)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,620)
      L=0
      LS=1
      DO 20 I=1,N
      L=L+1
      WRITE(6,600) I,(A(J),J=LS,L)
   20 LS=L+1
      CALL SLDL(A,N,0.0,ICON)
      IF(ICON.GE.20000) STOP
      CALL LDIV(A,N,ICON)
      WRITE(6,630)
      L=0
      LS=1
      DO 30 I=1,N
      L=L+1
      WRITE(6,600) I,(A(J),J=LS,L)
   30 LS=L+1
      WRITE(6,610) ICON
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(5E10.2)
  600 FORMAT(' ',I5/(10X,5E16.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(/10X,'INPUT MATRIX')
  630 FORMAT(/10X,'INVERSE MATRIX')
      END
```

Method

Given the LDL$^{\mathrm{T}}$ decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{D}$ of an $n \times n$ positive-definite symmetric matrix $\boldsymbol{A}$, the inverse $\boldsymbol{A}^{-1}$ is computed.

Since

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \tag{4.1}$$

Then, the inverse $\boldsymbol{A}^{-1}$ can be represented using (4.1) as follows: The inverse of $\boldsymbol{L}$ and $\boldsymbol{D}$ are computed and then the inverse $\boldsymbol{A}^{-1}$ is computed as (4.2).

$$\boldsymbol{A}^{-1} = \left(\boldsymbol{L}^{\mathrm{T}}\right)^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1} = \left(\boldsymbol{L}^{-1}\right)^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1} \tag{4.2}$$

Let $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$ be represented as shown in Eq. (4.3) for the following explanation.

$$\boldsymbol{L} = \left(l_{ij}\right), \quad \boldsymbol{D}^{-1} = \mathrm{diag}\left(d_i^{-1}\right) \tag{4.3}$$

- Calculating $\boldsymbol{L}^{-1}$

  Since the inverse $\boldsymbol{L}^{-1}$ of a unit lower triangular matrix $\boldsymbol{L}$ is also a unit lower triangular matrix, if we represent $\boldsymbol{L}^{-1}$ by

$$\boldsymbol{L}^{-1} = (\tilde{l}_{ij}) \tag{4.4}$$

then the Eq. (4.5) is obtained based on the relation $\boldsymbol{LL}^{-1}$=$\boldsymbol{I}$.

$$\sum_{k=1}^{n} l_{ik}\tilde{l}_{kj} = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1 & ,i = j \\ 0 & ,i \neq j \end{cases} \tag{4.5}$$

Since l_{ii}=1,(4.5) can be rewritten with respect to $\tilde{l}_{ij}$ as shown in (4.6).

$$\tilde{l}_{ij} = \delta_{ij} - \sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} \tag{4.6}$$

Then considering that $\tilde{l}_{ii} = 1$ and $\tilde{l}_{jj} = 1$, the elements $\tilde{l}_{ij}$ of the ith row (i=2,...,n) of $\boldsymbol{L}^{-1}$ can be obtained successively using

$$\tilde{l}_{ij} = -l_{ij} - \sum_{k=j+1}^{i-1} l_{ik}\tilde{l}_{kj}, \quad j = 1,...,i-1 \tag{4.7}$$

- Calculation of $(\boldsymbol{L}^{-1})^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1}$

  If we represent the inverse $\boldsymbol{A}^{-1}$ by

$$\boldsymbol{A}^{-1} = \left(\tilde{a}_{ij}\right) \tag{4.8}$$

Then equation (4.9) is derived based on $\boldsymbol{A}^{-1}$=$(\boldsymbol{L}^{-1})^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1}$

$$\tilde{a}_{ij} = \sum_{k=1}^{n} \tilde{l}_{ki} d_k^{-1} \tilde{l}_{kj} \tag{4.9}$$

Considering that $\tilde{l}_{ii} = 1$, the elements $\tilde{a}_{ij}$ of the i-th row (i = 1,...,n) of $\boldsymbol{A}^{-1}$ are successively obtained using

$$\tilde{a}_{ij} = d_i^{-1}\tilde{l}_{ij} + \sum_{k=i+1}^{n} \tilde{l}_{ki} d_k^{-1} \tilde{l}_{kj}, \quad j = 1,...,i \tag{4.10}$$

The precision of the inner product calculation in (4.7) and (4.10) has been raised to minimize the effect of rounding errors.

For further information, see Reference [2].

A22-51-0302 LDLX, DLDLX

A system of linear equations with a positive-definite symmetric matrix decomposed into the factors $\boldsymbol{L}$, $\boldsymbol{D}$ and $\boldsymbol{L}^T$
CALL LDLX (B, FA, N, ICON)

Function

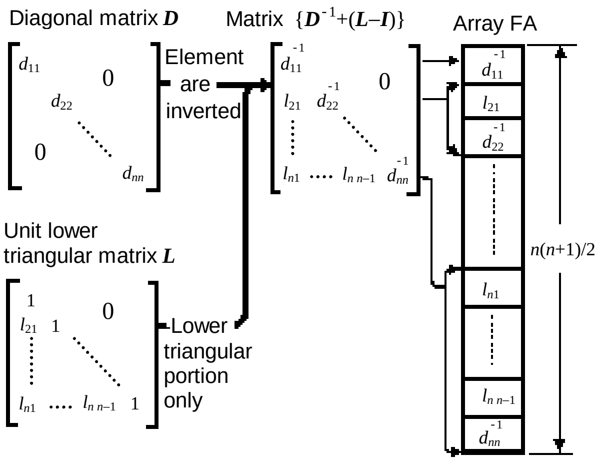
This subroutine solves a system of linear equations

$$\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{b} \qquad (1.1)$$

Where $\boldsymbol{L}$ and $\boldsymbol{D}$ are, respectively, $n \times n$ unit lower triangular and diagonal matrices, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. $n \geq 1$.

Parameters

B..........	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. One-dimensional array of size n.
FA........	Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$. See Fig. LDLX-1. One-dimensional array of size $n(n+1)/2$.
N...........	Input. Order n of the matrices $\boldsymbol{L}$ and $\boldsymbol{D}$.
ICON....	Output. Condition code. See Table LDLX-1.


Diagonal matrix $\boldsymbol{D}$
Matrix $\{\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})\}$
Array FA
Element are inverted
Unit lower triangular matrix $\boldsymbol{L}$
Lower triangular portion only
$n(n+1)/2$

Note: The diagonal and lower triangular portions of the matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$ are contained in the one-dimensional array A in the compressed mode for symmetric matrices.

Fig. LDLX-1 Storage method of matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$

Table LDLX-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The coefficient matrix was not positive-definite.	Discontinued
30000	N<1	Bypassed

Comments on use

- Subprograms used
  SSL II.....MGSSL
  FORTRAN basic function........None

- Notes
  Notes that the diagonal elements of $\boldsymbol{D}^{-1}$ instead of $\boldsymbol{D}$ are required in this subroutine. A system of linear equations can be solved by calling this subroutine after the subroutine SLDL. $\boldsymbol{D}^{-1}$ is output by subroutine SLDL. However, subroutine LSX can be usually called to solve such equations in one step.
  For a positive-definite symmetric band matrix, the subroutine BDLX processes faster than this subroutine because the operation for elements out of the band is omitted.

- Example
  A system of linear equations is solved after first LDL$^{\mathrm{T}}$ decomposition of the $n \times n$ coefficient matrix using subroutine SLDL. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NTOT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NTOT)
      WRITE(6,640)
      L=0
      LS=1
      DO 20 I=1,N
      L=L+1
      WRITE(6,600) I,(A(J),J=LS,L)
   20 LS=L+1
      CALL SLDL(A,N,1.0E-6,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      READ(5,510) (B(I),I=1,N)
      CALL LDLX(B,A,N,ICON)
      WRITE(6,610) ICON
      DET=A(1)
      L=1
      DO 30 I=2,N
      L=L+I
   30 DET=DET*A(L)
      DET=1.0/DET
      WRITE(6,620) (B(I),I=1,N)
      WRITE(6,630) DET
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(5E10.2)
  600 FORMAT(' ',I5/(10X,5E16.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(/10X,'SOLUTION VECTOR'
     *//(10X,5E16.8))
  630 FORMAT(/10X,
     *'DETERMINANT OF COEFFICIENT MATRIX=',
     *E16.8)
  640 FORMAT(/10X,'INPUT MATRIX')
      END
```

Method

To solve a system of linear equations (4.1) is equivalent to solve equations (4.2) and (4.3).

$$\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{b} \tag{4.1}$$

$$\boldsymbol{Ly} = \boldsymbol{b} \tag{4.2}$$

$$\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{y} \tag{4.3}$$

- solving $\boldsymbol{Ly}=\boldsymbol{b}$ (forward substitution)
  $\boldsymbol{Ly}=\boldsymbol{b}$ can be consecutively solved using equation (4.4).

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k , \quad i = 1,...,n \tag{4.4}$$

where, $\boldsymbol{L}=(l_{ij})$, $\boldsymbol{y}^{\mathrm{T}}=(y_1,...,y_n)$, $\boldsymbol{b}^{\mathrm{T}}=(b_1,...,b_n)$

- Solving $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{y}$ (backward substitution)
  $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{y}$ can be consecutively solved using equation (4.5).

$$x_i = y_i d_i^{-1} - \sum_{k=i+1}^{n} l_{ki} x_k , \quad i = n,...,1 \tag{4.5}$$

where, $\boldsymbol{D}^{-1} = \mathrm{diag}(d_i^{-1})$, $\boldsymbol{x}^{\mathrm{T}}=(x_1,...,x_n)$.
For more information, see Reference [2].

E21-20-0101 LESQ1, DLESQ1

Polynomial least squares approximation
CALL LESQ1 (X, Y, N, M, W, C, VW, ICON)

Function

Given n observed data $(x_1,y_1), (x_2,y_2),..., (x_n,y_n)$ and a weight function $w(x_i)$ i=1, 2, .., n, this subroutine obtains polynomial least squares approximations.

Let a polynomial of degree m be $\bar{y}_m(x)$ as

$$\bar{y}_m(x) = c_0 + c_1 x + \cdots + c_m x^m \qquad (1.1)$$

this subroutine determines coefficients $c_0, c_1,..., c_m$ such that (1.2) is minimized.

$$\delta_m^2 = \sum_{i=1}^{n} w(x_i)\{y_i - \bar{y}_m(x_i)\}^2 \qquad (1.2)$$

where m is selected so as to minimize (1.3) in the range $0 \le m \le k$.

$$\mathrm{AIC} = n \log \delta_m^2 + 2m \qquad (1.3)$$

When (1.3) is minimized, m is considered the optimum degree for the least squares approximation.

Where, $0 \le k < n-1$. Also, the weight function $w(x_i)$ must satisfy

$$\begin{aligned} & w(x_i) \ge 0, \quad i = 1,2,...,n \\ & n \ge 2 \end{aligned} \qquad (1.4)$$

Parameters

X .... Input. Discrete points x_i,
One-dimensional array of size n.

Y .... Input. Observed data y_i.
One-dimensional array of size n.

N .... Input. Number (n) of discrete points.

M .... Input. Upper limit k of the degree of the approximation polynomial to be determined.
If M = $-k$ (k > 0), the approximation polynomial of degree k is unconditionally obtained.
Output. The degree k of the approximation polynomial that was determined.
Hence, when M = $-k$, output M is always equal to k.

W .... Input. Weight function values $w(x_i)$.
One-dimensional array of size n.

C .... Output. Coefficient c_i of the determined approximation polynomial. C is a one-dimensional array of size k+1. Letting the output value of M be m ($0 \le m \le k$), the coefficients are stored in the following order:
$c_0,c_1,..., c_m$
Then, $m < k$,elements C(I+1), I=m+1,..., k, are set 0.0.

VW .... Work area. One dimensional array of size 7n.

ICON .. Output. Condition code.
See Table LESQ1-1.

Table LESQ1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	n < 2,k > n-1 or there was a negative value in w(x$_i$).	Bypassed

Comments on use

- Subprograms used
  SSL II ..... MGSSL, AMACH
  FORTRAN basic functions ..... IABS, DABS, DLOG, FLOAT, and AMAX1

- Notes
  Use of single or double precision subroutines
  The degree m of the approximation polynomial to be output may be different between single and double precision subroutines for some data.
  Therefore, it is preferable to use a double precision subroutine when handling a large number of observed data.

  Specifying weight function values $w(x_i)$
  When observed data have nearly the same order, $w(x_i)$=1.0,i= 1,2,...,n may be used. But, when they are ordered irregularly, the weight function should be specified as $w(x_i)=1/y_i^2$ (when y_i=0 specify, $w(x_i)$=1.0).
  The number of discrete points, n should be as high as possible compared to the upper limit k.
  Theoretically, n, is recommended to be equal to or greater than 10k.

- Example
  The number (n) of discrete points, the discrete points x_i, and the observed values y_i,i=1,2,...,n are input, then the coefficients of the least squares approximation polynomial are determined.
  Where $10 \le n \le 50$,and $w(x_i)$ = 1 (i = 1,2,...,n).

```
C     **EXAMPLE**
      DIMENSION X(50),Y(50),W(50),
     *          C(6),VW(350)
      READ(5,500) N
      READ(5,510) (X(I),Y(I),I=1,N)
      WRITE(6,600) (I,X(I),Y(I),I=1,N)
      DO 10 I=1,N
   10 W(I)=1.0
      M=5
      MI=M
      CALL LESQ1(X,Y,N,M,W,C,VW,ICON)
      WRITE(6,610) MI,ICON
      IF(ICON.EQ.30000) STOP
      M1=M+1
      WRITE(6,620) M,(I,C(I),I=1,M1)
      STOP
```

```
500 FORMAT(I5)
510 FORMAT(2F10.0)
600 FORMAT('1'//10X,'INPUT DATA'//
   *20X,'NO.',10X,'X',17X,'Y'//
   *(20X,I3,3X,E15.7,3X,E15.7))
610 FORMAT(10X,
   *'THE UPPER LIMIT OF THE DEGREE',
   *5X,I5/10X,'ICON=',I5)
620 FORMAT(//10X,
   *'THE RESULTANT DEGREE',14X,I5/
   *10X,'THE RESULTANT COEFFICIENTS'/
   *(20X,'C(',I2,')=',E15.7))
    END
```

Method

- Least squares approximation polynomial
  Assume that observed data y_i, i=1,2,...,n, are given for discrete points x_i, i=1,2,...,n, and they include some errors.
  Then, the least squares approximation polynomial for the observed data is the polynomial of degree m, $\bar{y}_m(x)$ which minimizes

$$\delta_m^2 = \sum_{i=1}^{n} w(x_i)\{y_i - \bar{y}_m(x_i)\}^2 \tag{4.1}$$

where $w(x_i)$,i=1,2,...,n is the weights.
Let us express $\bar{y}_m(x)$ as

$$\bar{y}_m(x) = \sum_{j=0}^{m} b_j^{(m)} P_j(x) \tag{4.2}$$

Where $P_j(x)$ is a polynomial of degree j(explained later). To determine $b_j^{(m)}$, $\bar{y}_m(x)$ in (4.2) is substituted into (4.1), then we take the partial derivative of δ_m^2 in (4.1) with respect to $b_j^{(m)}$ and set it equal to zero, thereby obtaining.

$$\left.\begin{aligned} &\sum_{j=0}^{m} d_{jk} b_j^{(m)} = \omega_k, \quad k = 0,1,...,m \\ &\text{where}\quad d_{jk} = \sum_{i=1}^{n} w(x_i) P_j(x_i) P_k(x_i) \\ &\qquad\quad \omega_k = \sum_{i=1}^{n} w(x_i) y_i P_k(x_i) \end{aligned}\right\} \tag{4.3}$$

Equation (4.3) is a system of m+1 linear equations for the m+1 unknown $b_j^{(m)}$. This system is called the normal equations and $b_j^{(m)}$ can be obtained by solving the system.
Now, if polynomials $\{P_j(x)\}$ in (4.3) are chosen as that

$$d_{jk} \begin{cases} = 0, & j \neq k \\ \neq 0, & j = k \end{cases} \tag{4.4}$$

then, elements of the coefficient matrix of the normal equations become zero except the diagonal elements, so

$$\begin{aligned} &b_j^{(m)} = b_j = \omega_j / \gamma_j \\ &\text{where}\quad \gamma_j = d_{jj} = \sum_{i=1}^{n} w(x_i)\{P_j(x_i)\}^2 \end{aligned} \tag{4.5}$$

Thus, $b_j^{(m)}$ can be obtained from (4.5).
The $\{P_j(x)\}$ which satisfies (4.4) can be obtained from the following recurrence formula

$$\begin{aligned} &P_{j+1}(x) = (x - \alpha_{j+1}) P_j(x) - \beta_j P_{j-1}(x) \\ &\qquad j = 0,1,... \end{aligned}$$

$$\begin{aligned} &P_0(x) = 1, P_{-1}(x) = 0 \\ &\alpha_{j+1} = \sum_{i=1}^{n} w(x_i) x_i \{P_j(x_i)\}^2 / \gamma_j \\ &j = 0,1,... \end{aligned} \tag{4.6}$$

$$\beta_j = \begin{cases} 0 & , j = 0 \\ \gamma_j / \gamma_{j-1} & , j = 1,2,... \end{cases}$$

- Selecting the optimum degree
  In obtaining the least squares approximation polynomial, selecting an optimum degree is of importance. This subroutine selects the optimum degree using a quantity AIC as the means to evaluate the optimum condition of degree m. Letting δ_m^2 be

$$\delta_m^2 = \sum_{i=1}^{n} w(x_i) \left\{ y_i - \sum_{j=0}^{m} b_j P_j(x_i) \right\}^2 \tag{4.7}$$

then AIC is defined as

$$\text{AIC} = n \log \delta_m^2 + 2m \tag{4.8}$$

In general, the smaller value of AIC shows the more optimum degree. Thus, this subroutine determines the optimum value which will minimize AIC within the range, 0≤m≤k, then output a least squares approximation polynomial of that degree. The output polynomial is, using $\{b_j\}$ and $\{P_j(x)\}$(j=0,1,...,m), expressed as

$$\bar{y}_m(x) = \sum_{j=0}^{m} b_j P_j(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_m x^m \tag{4.9}$$

in the standard form of polynomials with coefficients c_0, c_1, c_2,..., c_m.
For more details, see Reference [46] pp. 228 to 250.

D11-30-0101 LMINF, DLMINF

Minimization of function with a variable (Quadratic interpolation using function values only)
CALL LMINF (A, B, FUN, EPSR, MAX, F, ICON)

Function

Given a real function $f(x)$ with a variable, the local minimum point x^* and the function value $f(x^*)$ are obtained in interval $[a,b]$, where $f(x)$ is assumed to have up to the second continuous derivative.

Parameters

A..........	Input. End point a of interval $[a,b]$. Output. Minimum point x^*.
B..........	Input. End point b of interval $[a,b]$.
FUN.....	Input. Name of function subprogram which calculates $f(x)$. The form of subprogram is as follows: FUNCTION FUN (X) Parameters X.... Input. Variable x. Substitute values of $f(x)$ in function FUN. (See "Example".)
EPSR..	Input. Convergence criterion (≥0.0). The default value is assumed if 0.0 is specified. (See "Comments on Use".)
MAX..	Input. The upper limit (≠0) of number of evaluations for the function. (See "Comments on use".) Output. The number (>0) of actual evaluations.
F..........	Output. The value of function $f(x^*)$.
ICON..	Condition code. (See Table LMINF-1.)

Table LMINF-1 Condition codes

Code	Meaning	Processing
0	No error.	
10000	The convergence condition has not been satisfied within the specified function estimation count.	The last values obtained are stored in A and F.
30000	EPSR < 0.0 or MAX=0.	Bypassed.

Comments on use

- Subprograms used
  SSL II .... AMACH, MGSSL
  FORTRAN basic functions .... ABS, SQRT, AMAX1

- Notes
  An EXTERNAL statement is necessary to declare the subprogram name correspond to parameters FUN in the calling program.

EPSR
In this subroutine, the convergence condition is checked as follows: During iteration, if

$$|x_1 - x_2| \le \max(1.0, \tilde{x}) \cdot \text{EPSR}$$

is satisfied at two points x_1, and x_2, between which x^* exists, point $\tilde{x}$ is assumed to be minimum point x^* and iteration is stopped, where $\tilde{x} = x_1$ for $f(x_1) \le f(x_2)$ and $\tilde{x} = x_2$ for $f(x_1) > f(x_2)$.

Since $f(x)$ is assumed to be approximately a quadratic function in the vicinity of point x^*, it is appropriate to specify EPSR as EPSR $\approx \sqrt{u}$, where u is the unit round off to obtain value x^* as accurate as the rounding error.

The default value of EPSR is $2\sqrt{u}$.

MAX
The number of function evaluation is incremented by one every time $f(x)$ is evaluated.
This is the same as the call count of subprogram FUN.

The number depends on characteristics of function $f(x)$, interval $[a,b]$, and convergence criterion.

Generally, when an appropriate interval $[a,b]$ is specified and the default value is used for the convergence criterion, it is adequate to specify MAX = 400.

Even if the convergence condition is not satisfied within the specified evaluation count and the subroutine is returned with ICON = 10000, iteration can be resumed by calling this subroutine again. In this case, the user must specify a negative value as the additional evaluation count in the parameter MAX and retain other parameters unchanged.

A and B
If there is only one minimum point of $f(x)$ in interval $[a,b]$, this subroutine obtains the value of the point within the specified error tolerance. If there are several minimum points, it is not guaranteed to which point the result converges. This means that it is desirable to set a and b, end points of the interval containing minimum point x^*, in the vicinity of minimum point x^*.

- Example
  Given the following function,

$$f(x) = x^4 - 4x^3 - 6x^2 - 16x + 4$$

a minimum value in interval [–5, 5] is obtained, assuming that the default value is used for the convergence criterion.

```
C      **EXAMPLE**
       EXTERNAL FUN
       A=-5.0
       B=5.0
       EPSR=0.0
       MAX=400
       CALL LMINF(A,B,FUN,EPSR,
      *MAX,F,ICON)
       WRITE(6,600) ICON,MAX,A,F
   600 FORMAT('1'//1X,'ICON=',I6,2X,
      *'MAX=',I5,2X,'A=',E15.7,2X,
      *'F=',E15.7)
       STOP
       END
C      OBJECTIVE FUNCTION
       FUNCTION FUN (X)
       FUN=(((X-4.0)*X-6.0)*X-16.0)*X+4.0
       RETURN
       END
```

Method

Given a single variable real function $f(x)$ and interval $[a,b]$ in which the minimum value is to be obtained, minimum point x^* and the value of function $f(x^*)$ are obtained using the quadratic interpolation.

It is assumed that $-f(x)$ is unimodal in interval $[a,b]$. If $-f(x)$ is not unimodal, this subroutine obtains one of the minimum points.

The processing comprises two steps:

- Obtains two points x_1 and x_2 between which point x^* exists.
- Based on function values $f(x_1)$, $f(x_2)$ and $f(x_h)$ at points x_1 and x_2, and middle point x_h, determines the minimum point using a quadratic interpolation formula which passes through the points $(x_1, f(x_1))$, $(x_h, f(x_h))$, and $(x_2, f(x_2))$.

There steps are repeated until the convegence condition is satisfied in the interval containing the minimum point.

Calculation procedures

For simplicity, let $f(x_1)$, $f(x_2)$... be denoted as f_1, f_2,...

Procedure step 1 (determination of points x_1 and x_2)

1) Assumes the smaller of a and b to be x_1, the larger to be x_2, and ε=max (EPSR, $2\sqrt{u}$.)
2) Determines middle point x_h in interval $[x_1,x_2]$ from $x_h=(x_1+x_2)/2$
3) If $f_1 > f_h < f_2$, computes $h=x_2-x_h$ and proceeds to step2.
4) If $f_1 \le f_h \le f_2$, assumes $x_2=x_h$, and returns to 2).
   However, if the convergence condition
   $|x_1 - x_h| \le \max(1.0,|x_1|)\cdot\varepsilon$
   is satisfied, assumes x_1 to be x^*, f_1 to be $f(x^*)$, and sets ICON=0, then stops processing.
5) If $f_1 \ge f_h \ge f_2$,, assumes $x_1= x_h$, and returns to 2).
   However, if the convergence condition
   $|x_h - x_2| \le \max(1.0,|x_2|)\cdot\varepsilon$
   is satisfied, assumes x_2 to be x^* and f_2 to be $f(x^*)$, and sets ICON=0, then stops processing.
6) If $f_1 < f_h > f_2$, assumes $x_2= x_h$ (for $f_1 \le f_2$)or $x_1= x_h$ (for $f_1 > f_2$), and returns to 2). However, if the convergence condition
   $|x_1 - x_h| \le \max(1.0, |x_1|)\cdot\varepsilon \qquad (\text{for } f_1 \le f_2)$
   or
   $|x_2 - x_h| \le \max(1.0, |x_2|)\cdot\varepsilon \qquad (\text{for } f_1 > f_2)$
   is satisfied, assumes x_1 or x_2 to be x^*, f_1 or f_2 to be $f(x^*)$, and sets ICON=0, then stops processing.

Procedure step2

7) Obtains minimum point x_m using the quadratic interpolation by means of following calculation:
   $$\Delta h = \frac{h}{2}(f_2 - f_1)/(f_1 - 2f_h + f_2)$$
   $$x_m = x_h - \Delta h$$
8) If the convergence condition
   $|\Delta h| \le \max(1.0,|x_h|)\cdot\varepsilon/2$
   is satisfied, proceeds to 9); otherwise, assumes
   $x_1 = x_m(\text{for } f_m \le f_h)$ or $x_1=x_h$ and
   $x_h = x_m(\text{for } f_m \ge f_h)$, and obtains two points between which point x^* exists, then returns to 7).
9) For the following two points in the vicinity of point $\tilde{x} = x_m(\text{for } f_m < f_h)$ or $\tilde{x} = x_h(\text{for } f_m \ge f_h)$ which minimizes the function value,
   $$\tilde{x}_1 = \tilde{x} + \max(1.0, |\tilde{x}|)\cdot\varepsilon/2$$
   $$\tilde{x}_2 = \tilde{x} - \max(1.0, |\tilde{x}|)\cdot\varepsilon/2$$
   if $f(\tilde{x}_1) < \min(f_m, f_h)$ and
   $f(\tilde{x}_2) < \min(f_m, f_h)$
   are satisfied, assumes $\tilde{x}$ to be x^* and $f(\tilde{x})$ to be $f(x^*)$, and sets ICON=0, then stops processing.
   If these conditions are not satisfied, obtains two points between which point x^* exists, then return to 7).

D11-40-0101 LMING, DLMING

Minimization of function with a variable (Cubic interpolation using function values are its derivatives)
CALL LMING (A, B, FUN, GRAD, EPSR, MAX, F, ICON)

Function

Given a single variable real function $f(x)$ and its derivative $g(x)$, the local minimum point x^* and the function value $f(x^*)$ are obtained in interval $[a, b]$, where $f(x)$ is assumed to have up to the third continuous derivative.

Parameters

A	Input. End point a of interval $[a, b]$. Output. Minimum point x^*.
B	Input. End point b of interval $[a, b]$.
FUN ...	Input. Name of function subprogram which calculates $f(x)$. The form of subprogram is as follows: FUNCTION FUN (X) Parameters X ... Input. Variable x. Substitute values of $f(x)$ in function FUN. (See “Example”.)
GRAD ..	Input. Name of function subprogram which calculates $g(x)$. The form of subprogram is as follows: FUNCTION GRAD (X) Parameters X ... Input. Variable x. Substitute the value of $g(x)$ in function GRAD. (See “Example”.)
EPSR ..	Input. Convergence criterion (≥ 0.0). The default value is assumed if 0.0 is specified. (See “Comments on use”.)
MAX ..	Input. The upper limit ($\neq 0$) of number of evaluations for $f(x)$ and $g(x)$. (See “Comments on use”.) Output. The number (>0) of actual evaluations.
F	Output. The value of function $f(x^*)$.
ICON ..	Condition code. (See Table LMING-1.)

Table LMING-1 Condition codes

Code	Meaning	Processing
0	No error.	
10000	The convergence condition has not been satisfied within the specified function evaluation count.	The last values obtained are stored in A and F.
20000	The value of EPSR is too small.	Bypassed. (The last values obtained are stored in A and F.)
30000	EPSR<0.0 or MAX=0.	Bypassed.

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL
  FORTRAN basic functions ... ABS, SQRT, AMAX1

- Notes
  An EXTERNAL statement is necessary to declare the subprogram names correspond to parameters FUN and GRAD in the calling program.

EPSR

In this subroutine, the convergence condition is checked as follows: During iteration, if

$$|x_1 - x_2| \leq \max(1.0, \tilde{x}) \cdot \text{EPSR}$$

is satisfied for two points x_1 and x_2 between which x^* exists, point $\tilde{x}$ is assumed to be minimum point x^* and iteration is stopped, where $\tilde{x} = x_1$ for $f(x_1) \leq f(x_2)$ and $\tilde{x} = x_2$ for $f(x_1) > f(x_2)$.

Since $f(x)$ is assumed to be approximately a cubic function in the vicinity of point x^*, it is appropriate to specify EPSR as $\text{EPSR} \approx \sqrt{u}$, where u is the unit round off to obtain value x^* as accurate as the rounding error. The default value of EPSR is $2\sqrt{u}$.

MAX

The number of function evaluation count is incremented by one every time $f(x)$ or $g(x)$ is evaluated. This is the same as the call count of subprogram FUN and GRAD.

The number depends on characteristics of function $f(x)$ and $g(x)$, interval $[a, b]$, and convergence criterion.

Generally, when an appropriate interval $[a,b]$ is specified and the default value is used for the convergence criterion, it is adquate to specify MAX=400.

Even if the convergence condition is not satisfied within the specified evaluation count and the subroutine is returned with ICON=10000, iteration can be resumed by calling this subroutine again. In this case, the user must specify a negative value as the additional evaluation count in the parameter MAX and retain other parameters unchanged.

A and B

If there is only one minimum point for $f(x)$ in interval $[a,b]$, this subroutine obtains the value of the point within the specified error tolerance. If there are several minimum points, it is not guaranteed to which point the result converges. This means that it is descrable to set a and b, end points of the interval containing minimum point x^*, in the vicinity of minimum point x^*.

- Example

  Given the following function,

$$f(x) = x^4 - 4x^3 - 6x^2 - 16x + 4$$

a minimum value is obtained in interval $[-5,5]$. Where derivative $g(x)$ of function $f(x)$ is

$$g(x) = 4x^3 - 12x^2 - 12x - 16$$

and the default value is used for the convergence criterion.

```
C      **EXAMPLE**
       EXTERNAL FUN,GRAD
       A=-5.0
       B=5.0
       EPSR=0.0
       MAX=400
       CALL LMING(A,B,FUN,GRAD,EPSR,
      *           MAX,F,ICON)
       WRITE(6,600) ICON,MAX,A,F
   600 FORMAT('1'//1X,'ICON=',I6,2X,
      *'MAX=',I5,2X,'A=',E15.7,2X,
      *'F=',E15.7)
       STOP
       END
C      OBJECTIVE FUNCTION
       FUNCTION FUN(X)
       FUN=(((X-4.0)*X-6.0)*X-16.0)*X+4.0
       RETURN
       END
C      DERIVATIVE
       FUNCTION GRAD(X)
       GRAD=((4.0*X-12.0)*X-12.0)*X-16.0
       RETURN
       END
```

Method

Given a single variable real function $f(x)$, its derivative $g(x)$, and interval $[a,b]$ in which the minimum value is to be obtained, minimum point x^* and the value of function $f(x^*)$ are obtained using cubic interpolation.

It is assumed that $-f(x)$ is unimodal in interval $[a,b]$. If $-f(x)$ is not unimodal, this subroutine obtains one of minimum points.

The processing comprises two steps:

Obtains two points x_1 and x_2 between which point x^* exists.

Based on function values $f(x_1)$ and $f(x_2)$ at points x_1 and x_2 and their derivatives $g(x_1)$ and $g(x_2)$, determines the minimum point using a cubic interpolation formula which passes through points $(x_1, f(x_1))$ and $(x_2, f(x_2))$.

These steps are repeated until the convergence condition is satisfied in the interval containing the minimum point.

Calculation procedures

For simplicity, let $f(x_1)$, $f(x_2)$,...be expressed as f_1, f_2, ...

Procedure step 1 (determination of two points x_1 and x_2)

1) Assumes the smaller of a and b to be x_1, the larger to be x_2, and ε =max(EPSR, $2\sqrt{u}$)

   If g_1<0 and g_2>0, proceeds to step 2.
2) Determines middle point x_h in interval $[x_1, x_2]$ from $x_h = (x_1 + x_2)/2$.
3) If $f_1 \le f_h \le f_2$, assumes $x_2 = x_h$ and returns to 2).

   However, if the convergence condition

   $|x_1 - x_h| \le \max(1.0, |x_1|) \cdot \varepsilon$

   is satisfied, assumes x_1 to be x^* and f_1 to be $f(x^*)$,and sets ICON = 0, then stops processing.
4) If $f_1 \ge f_h \ge f_2$, assumes $x_1 = x_h$, and returns to 2).

   However, if the convergence condition

   $|x_h - x_2| \le \max(1.0, |x_2|) \cdot \varepsilon$

   is satisfied, assumes x_2 to be x^* and f_2 to be $f(x^*)$, and sets ICON =0, then stops processing.
5) If $f_1 < f_h > f_2$, assumes $x_2 = x_h$ (for $f_1 \le f_2$) or $x_1 = x_h$ (for $f_1 > f_2$), and returns to 2). However, if the convergence condition

   $|x_1 - x_h| \le \max(1.0, |x_1|) \cdot \varepsilon$ (for $f_1 \le f_2$)

   or

   $|x_2 - x_h| \le \max(1.0, |x_2|) \cdot \varepsilon$ (for $f_1 > f_2$)

   is satisfied, assumes x_1 or x_2 to be x^* and f_1 or f_2 to be $f(x^*)$, and sets ICON=0, then stops processing.
6) If $f_1 > f_h < f_2$ and

   If $g_h < 0$ and $g_2 > 0$, assumes $x_1 = x_h$, then proceeds to step 2);

   If $g_1 < 0$ and $g_h > 0$, assumes $x_2 = x_h$, then proceeds to step 2);

   If $g_1 \ge 0$ and $g_h \ge 0$, assumes $x_2 = x_h$, then returns to 2);

   If $g_h \le 0$ and $g_2 \le 0$, assumes $x_1 = x_h$, then returns to 2);

Step 2 (cubic interpolation)

7) Obtains minimum point x_m using cubic interpolation by means of following calculation:
$h = x_2 - x_1$, $z = 3(f_1 - f_2)/h + g_1 + g_2$,
$w = (z^2 - g_1 g_2)^{1/2}$,
$\Delta h = h(w + z - g_1)/(g_2 - g_1 + 2w)$,
$x_m = x_1 + |\Delta h|$

8) If $x_m \geq x_2$, assumes x_1 to be x^* and sets ICON=20000, then stops processing.

9) If $x_m < x_2$ and if the convergence condition
$|h| \leq \max(1.0, |x_m|) \cdot \varepsilon$
is satisfied, or if $g_m = 0$, assumes x_m to be x^* and f_m to be $f(x^*)$, and sets ICON=0, then stops processing.
If the convergence condition is not satisfied, assumes $x_1 = x_m (g_m < 0)$ or $x_2 = x_m (g_m > 0)$, then returns to 2).

C21-41-0101 LOWP, DLOWP

Zeros of a low degree polynomial with real coefficients (fifth degree or lower)
CALL LOWP (A, N, Z, ICON)

Function

This subroutine finds zeros of a fifth or less degree polynomial with real coefficients;

$$a_0x^n + a_1x^{n-1} + ... + a_n = 0 \quad (n \le 5, a_0 \ne 0)$$

by the successive substitution method, Newton method, Ferrari method, Bairstom method, and the root formula for quadratic equations.

Parameters

A......	Input. Coefficients of the equations. A is a one-dimensional array of size $n+1$. Where A(1)=a_0, A(2)=a_1, ..., A(N+1)=a_n.
N......	Input. Degree of the equation.
Z.......	Output. n roots Z is a complex one-dimensional array of size n.
ICON..	Output. Condition code. See Table LOWP-1.

Table LOWP-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	When determining a real root of a fifth degree equation, after 50 successive substitutions f_kf_{k+1}<0 was not satisfied.	x_{k+1} is used as the initial value in Newton's method and processing continues. Refer to equation (4.16) in the method.
30000	a0=0.0, n≤0 or n>5	Bypassed

Comments on use

- Subprograms used
  SSL II ... RQER, U3DEG, AMACH, UREDR, and MGSSL
  FORTRAN basic functions ... SQRT, ABS, and CMPLX
- Example
  Order n and coefficient a_i(i = 0 , 1, ..., n) are entered and n roots are determined.

```
C       **EXAMPLE**
        DIMENSION A(6),Z(5)
        COMPLEX Z
        READ(5,500) N
        N1=N+1
        READ(5,510)(A(I),I=1,N1)
        CALL LOWP(A,N,Z,ICON)
        WRITE(6,600) N,ICON
        WRITE(6,610)(A(I),I=1,N1)
        IF(ICON.EQ.30000) STOP
        WRITE(6,620)(Z(I),I=1,N)
        STOP
    500 FORMAT(I1)
    510 FORMAT(6F10.0)
    600 FORMAT(10X,'N=',I1,5X,'ICON=',I7)
    610 FORMAT(10X,'A=',E20.8/(12X,E20.8))
    620 FORMAT(10X,'Z=',2E20.8/(12X,2E20.8))
        END
```

Method

In the explanations below the coefficient of the highest power of the polynomial is assumed to be 1 (there is no loss of generality)

- When n = 2 (quadratic equations)
  The root formula is used. This is done by calling subroutine RQDR.
- For n = 3 (third degree equations)
  If one real root x_1 of $f(x) = x^3 + c_1x^2 + c_2x + c_3 = 0$ is obtained, the equation becomes.

$$f'(x) = (x - x_1)(x^2 + p_ix + p_2) \tag{4.1}$$

$$\text{where,} \begin{cases} p_1 = c_1 + x_1 \\ p_2 = c_2 + x_1p_1 \end{cases}$$

and the problem is reduced to solving a quadratic equation. The single real root x_1 is obtained using Newton's method. The initial value x_a is determined after examining the characteristics of $f(x)$. Namely, let

$$f'(x) = 3x^2 + 2c_1x + c_2 = 0 \tag{4.2}$$

the following three cases are considered when determining x_a.

(a) When $f'(x)$ = 0 has two different real roots x_{m1} and x_{m2}(x_{m1}) and a minimum value $f(x_{m2})$.
If $f(x_{m1}) f(x_{m2})$< 0, the point of inflection of $f(x)$ is used for x_a. If $f(x_{m1}) f(x_{m2})$> 0, the following cases are considered for determining x_a.
When $f(x_{m1}) > 0$
A quadratic equation in h, which is obtained by setting the Taylor series expansion of $f(x_{m1}+h)$ up to h^2 equal to zero, is considered. Let its positive real root be h again, then

$$x_a = x_{m1} - h \tag{4.3}$$

When $f(x_{m1}) < 0$
A quadratic equation in h, which is obtained by setting the Taylor series expansion of $f(x_{m2}+h)$ up to h^2 equal to zero, is considered. Let its positive real root be h again, and then

$$x_a = x_{m2} + h \tag{4.4}$$

(b) When $f'(x) = 0$ has multiple roots

This occurs when the maximum value and then minimum value match the inflection point. Let the inflection point be x_i and perform the successive substitution method once, then

$$x_a = x_i - f(x_i) \tag{4.5}$$

(c) When $f'(x) = 0$ does not have any real roots The inflection point of $f(x)$ is used for x_a.

- When $n = 4$ (fourth degree equation)
  Generally, a fourth degree equation

$$f(x) = x^4 + c_1x^3 + c_2x^2 + c_3x + c_4 = 0 \tag{4.6}$$

can be factored into two quadratic factors using Ferrari's method.

$$f(x) = (x^2 + p_1x + p_2)(x^2 + q_1x + q_2)$$

If one real root μ of the following third degree equation is determined

$$\begin{aligned}\mu^3 - c_2\mu^2 + (c_1c_3 - 4c_4)\mu \\ + (4c_2c_4 - c_3^2 - c_1^2c_4) = 0\end{aligned} \tag{4.8}$$

p_2 and q_2 can be determined as the two roots of

$$v^2 - \mu v + c_4 = 0 \tag{4.9}$$

and p_1 and q_1 can be determined as the two roots of

$$v^2 - c_1v + (c_2 - \mu) = 0 \tag{4.10}$$

It p_2 and q_2 are complex roots, the remaining real roots of (4.8) are examined and real roots within a certain allowable quantity range are used for p_2 and q_2. Later, p_1, q_1, p_2 and q_2 are modified using Bairstow's method. Of the following quantities

$$\left.\begin{aligned} q_1 &= c_1 - p_1 \\ q_2 &= c_2 - p_1q_1 - p_2 \\ q_3 &= c_1 - p_1q_2 - p_2q_1 \\ q_4 &= c_0 - p_1q_3 - p_2q_2 \end{aligned}\right\} \tag{4.11}$$

$$\left.\begin{aligned} d_1 &= q_1 - p_1 \\ d_2 &= q_2 - p_1d_1 - p_2 \\ d_3 &= -p_1d_2 - p_2d_1 \end{aligned}\right\} \tag{4.12}$$

q_1 to q_4 are calculated first, then d_1 to d_3 are calculated. Then the systems of linear equation in (4.13) are solved for $\Delta p_1 p_1$ and Δp_2.

$$\begin{aligned} d_2\Delta p_1 + d_1\Delta p_2 &= q_3 \\ d_3\Delta p_1 + d_2\Delta p_2 &= q_4 \end{aligned} \tag{4.13}$$

$p_1 + \Delta p_1$, and $p_2 + \Delta p_2$ are used as the new values for p_1 and p_2, and the corresponding q_1 and q_2 are obtained using (4.11).

- When $n = 5$ (fifth degree equation) Let one of the real roots of the following fifth degree equation.

$$f(x) = x^5 + c_1x^4 + c_2x^3 + c_3x^2 + c_4x + c_5 = 0 \tag{4.14}$$

be x_1, then $f(x)$ can be rewritten as

$$f(x) = (x - x_1)(x^4 + c_1'x^3 + c_2' + c_3' + c_4') = 0 \tag{4.15}$$

where

$$\begin{aligned} c_1' &= c_1 + x_1 \\ c_2' &= c_2 + x_1c_1' \\ c_3' &= c_3 + x_1c_2' \\ c_4' &= c_1 + x_1c_3' \end{aligned}$$

Therefore once x_1 has been obtained, the problem is reduced to solving a fourth degree equation. x_1 is determined as follows: First, using the successive substitution method

$$\begin{aligned} &x_0 = 0, \quad f_0 = c_5 \\ &x_{k+1} = x_k - f_k \\ &f_k = f(x_k), \quad k = 0, 1, \ldots, 50 \end{aligned} \tag{4.16}$$

x_k and x_{k+1} are determined such that $f_k f_{k+1} < 0$
Next, using the method of regular falsi

$$x_a = x_{k+1} - (x_{k+1} - x_k)f_{k+1}/(f_{k+1} - f_k)$$

x_a is determined. Newton's method is then applied using x_a as the initial value. The upper limit of iteration by (4.16) is set to 50. If $f_k f_{k+1} < 0$ is not satisfied when that limit is reached the last x_{k+1} is used as the initial value in Newton's method.

Convergence criterion
For third and fifth degree equations, Newton's method is used to obtaine one real root. The method used to examine convergence is discussed below. Let third and fifth degree equations be represented in the following general form.

$$c_0x^n + c_1x^{n-1} + \ldots\ldots + c_n = 0 \tag{4.17}$$

(where c_0 =1, n =3 or 5)
and let the k-th approximate value obtained when Newton's method is applied to (4.17) be x_k, x_k is accepted as a root of (4.17) if it satisfies

$$\left|\sum_{j=0}^{n} c_j x_k^{n-j}\right| \le u \sum_{j=0}^{n} \left|c_j x_k^{n-j}\right| \tag{4.18}$$

Where u is the round-off unit.
For futher details, see Reference [25].

D21-10-0101 LPRS1, DLPRS1

Linear programming (Revised simplex method)
CALL LPRS1 (A, K, M, N, EPSZ, IMAX, ISW, NBV, B, VW, IVW, ICON)

Function

This subroutine solves linear programming problem shown below by using the revised simplex method;
Minimize (or maximize) the objective function of linear

$$z = \sum_{j=1}^{n} c_j x_j + c_0$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \le d_i ,\ i = 1,2, \dots ,\ m_l \ ,$$

$$\sum_{j=1}^{n} a_{ij} x_j \ge d_i \ ,\ i = m_l + 1,\ m_l + 2 , \dots , m_l + m_g \ ,$$

$$\sum_{j=1}^{n} a_{ij} x_j = d_i \ ,\ i = m_l + m_g + 1,\ m_l + m_g + 2 , \dots ,$$

$$m_l + m_g + m_e ,$$

$$x_j \ge 0 \ ,\ j = 1, 2 , \dots , n$$

This subroutine solves this problem in the following two phase:

- Phase 1 .... obtaining basic feasible solution.
- Phase 2 .... obtaining the optimal solution.

This subroutine allows entry of the initial feasible basis. There is no constraint on sign of d_i .

Assume $m = m_1 + m_g + m_e$.

- $m \times h$ matrix consisting of elements $\{a_{ij}\}$: called coefficient matrix $\boldsymbol{A}$.
- $\boldsymbol{d} = (d_1 \ ,\ d_2 \ , \dots ,\ d_m)^{\mathrm{T}}$: called a constant vector.
- $\boldsymbol{c} = (c_1 \ ,\ c_2 \ , \dots ,\ c_m)^{\mathrm{T}}$: called a coefficient vector.
- c_0 : called a constant term.

$n \ge 1 ,\ m_l \ge 0 ,\ m_g \ge 0 ,\ m_e \ge 0 \quad m \ge 1$

Parameters

A.... Input. Coefficient matrix $\boldsymbol{A}$, Constant vector $\boldsymbol{d}$, coefficient vector $\boldsymbol{c}$ and constant term c_0 .
See Table LPRS1-1.
Two-dimensional array of size A(K, N+1).

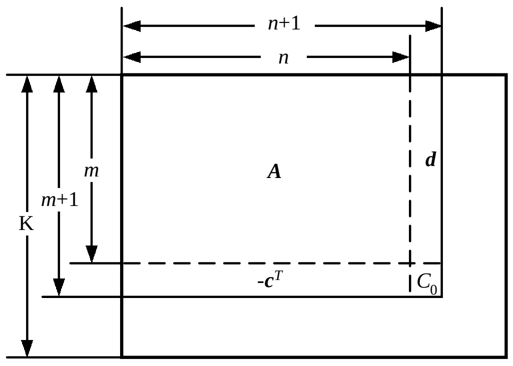


Fig. LPRS1-1 Contents of array A

K	Input. Adjustable dimension of arrays A and B ($\ge m+1$).
M	Input. Number of constraints. One-dimensional array of size 3. M(1), M(2), and M(3) contain m_l, m_s and m_e, respectively.
N	Input. Number of variables n
EPSZ ..	The relative zero criterion for elements (coefficient and constant term) to be used during iteration and the pivot to be used when basic inverse matrix $\boldsymbol{B}^{-1}$ is obtained. EPSZ $\ge$ 0.0. A default value is used when EPSZ=0.0. See Notes.
IMAX ..	Input. Maximum number of iterations ($\ne 0$). See Notes. Output. Number of iterations actually executed (>0).
ISW ..	Input. Control information. ISW specifies whether the objective function is to be minimized or maximized and whether the initial feasible basis is given. ISW is expressed as $10\,d_1 + d_0$. Where $0 \le d_0 \le 1$ and $0 \le d_1 \le 1$. When $d_0 = 0$, the objective function is to be minimized. When $d_0 = 1$, the objective function is to be maximized. When $d_1 = 0$, a basis is not given. When $d_1 = 1$, a basis is given. Output. When an optimal solution or basic feasible solution is obtained, ISW contains a value of $d_1 = 1$. Otherwise the input value remains.

NBV .. Input. (When ISW = 10 or 11 is specified.)
Initial feasible basis.
Output. Optimal or feasible basis.
One-dimensional array of size m.
See Notes.

B ..... Output. Basic inverse matrix $\boldsymbol{B}^{-1}$ for an optimal solution or basic feasible solution, optimal solution or basic feasible solution $\boldsymbol{g}$, simplex multiplier π and objective function value q.
Two-dimensional array B(K, m+1).
See Fig. LPRS1-2.

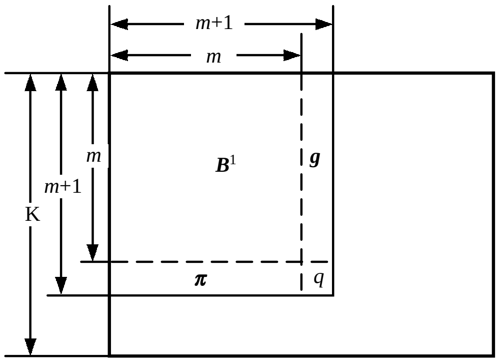


Fig. LPRS1-2 Contents of array B

VW .. Work area. One-dimensional array of size $2n+m+m_l+m_g+1$.

IVW .. Work area. One-dimensional array of size $n+m_l+m_g$.

ICON .. Output. Condition code.
See Table LPRS1-1.

Table LPRS1-1 Condition codes

Code	Meaning	Processing
0	An optimal solution was obtained.	
10000	A basic feasible solution was obtained but the problem has no optimal solution.	The basic feasible solution and corresponding basic inverse matrix, simplex multiplier and objective function value are stored in array B.
11000	The number of iterations reached to the specified upper limit during phase 2. The basic feasible solution was obtained.	
20000	The problem is infeasible. The value EPSZ might not be appropriate.	Bypassed
21000	When ISW=10 or ISW=11, the set of the given variables is not a basis.	Bypassed
22000	When ISW=10 or ISW=11, the set of the given variables is infeasible.	Bypassed
23000	The basic variable could not be interchanged during phase 1. The value EPSZ might not be appropriate.	Bypassed
24000	The number of iterations reached to the upper limit during phase 1.	Bypassed
30000	One of the following conditions occurred: 1 Negative value was contained in M(1), M(2) or M(3). 2 N<1 3 IMAX=0 4 EPSZ<0.0 5 M(1)+M(2)+M(3)<1 6 M(1)+M(2)+M(3)≥K 7 Zero or negative number was contained in the elements of NBV. 8 The number exceeding N+M(1)+M(2) was contained in the elements of NBV. 9 The same basic variable number appeared in the elements of NBV. 10 ISW was incorrectly given.	Bypassed

Comments on use

- Subprograms used
  SSL II ... ALU, LUIV, AMACH, MGSSL
  FORTRAN basic functions ... ABS, IABS

- Notes
  – Contents of NBV
  When the number of basic variable is specified by NBV (when ISW=10 or ISW=11), the number of the slack variable corresponding the i-th $(i \le m_l + m_g)$ unequality constraint must be $n+i$. When the computed basic solution contains the i-th slack variable, the slack variable number is assumed to be $n+i$.
  An artificial variable cannot be specified as a basic variable at entry time. When an artificial variable is combined with the computed basic solution,NBV contains value of zero.
  If the i-th basic variable is an artificial variable, NBV(i) contains a value of zero. When the basic feasible solution has been obtained (when ICON=0, 10000 or 11000), NBV(i)=0 indicates that the i-th constraint is redundant

(which is derived form other constraints with NBV (i) ≠ 0).

Moreover, the content of NBV is ginven corresponding in storage order of the optimal solution or basic feasible solution g (See Fig LPRS 1-2). It is x_3=0 against number j which is not stored in NBV.

– Giving IMAX

The number of iterations is that of criterions of the optimality standards (See Method 4.7) assciated with a basic fesible solution (at phase 1, the basic feasible solution contains artificial variables). The standard value for IMAX is 10m

If the optinal solution could not be obtained within the specified number of iterations and ICON contains a condition code of 11000, this subroutine can be called again to continue iteration. In this case, a negative value for the number of additional iterations must be specified in the parameter IMAX, While other parameters remaim unchanged.

– Giving EPSZ

Suppose the maximum abolute value of the input date (contents of array A) elements to be $a_{\max}$. Zero is assumed for the element in which the absolute value for the elements during iteration is less than $a_{\max}$ ·EPSZ.

When a basic inverse matrix $\boldsymbol{B}^{-1}$ is obtained by using ALU or LUIV sobroutinc, EPSZ is used as the relative zero criterrion value (Refer to Comments on use of the subroutine ALU) .

The standard value for EPSZ is 16 u (u:unit round off). It is desirable that the absolute value must be adjusted as far as possible by multiplying each column or row by constants. When ICON=20000 or 23000, the value of EPSZ may not be appropriate. In this case, use the standard value for EPSZ or change the value for retry.

– When varibale x_j is negative

This subroutine principally used the condition

x_j ≥0, j=1,2,...,n

Therefore if x_j is negative, the subroutine can be used after transformation of the problem as listed below:

- Replce x_j by $x_j^+ - x_j^-$
- Add the constraints of $x_j^+ \geq 0, x_j^- \geq 0$

- Example

This example solves a linear programming problem when the maximum number of variables is 10 and the maximum number of constraints is 20:

```
C     **EXAMPLE**
      DIMENSION A(21,11),B(21,21),M(3),
     *          NBV(20),VW(61),IVW(30)
      READ(5,500) N,M,ISW
      N1=N+1
      MM=M(1)+M(2)+M(3)
      M1=MM+1
      DO 10 I=1,M1
      READ(5,510) (A(I,J),J=1,N1)
   10 CONTINUE
      WRITE(6,600) (J,J=1,N)
      IF(M(1).EQ.0) GOTO 30
      WRITE(6,610)
      DO 20 I=1,M(1)
      WRITE(6,620) I,(A(I,J),J=1,N1)
   20 CONTINUE
   30 IF(M(2).EQ.0) GOTO 50
      WRITE(6,630)
      IS=M(1)+1
      IE=M(1)+M(2)
      DO 40 I=IS,IE
      WRITE(6,620) I,(A(I,J),J=1,N1)
   40 CONTINUE
   50 IF(M(3).EQ.0) GOTO 70
      WRITE(6,640)
      IS=M(1)+M(2)+1
      DO 60 I=IS,MM
      WRITE(6,620) I,(A(I,J),J=1,N1)
   60 CONTINUE
   70 IF(MOD(ISW,10).EQ.0) GOTO 80
      WRITE(6,650) (A(M1,J),J=1,N1)
      GOTO 90
   80 WRITE(6,660) (A(M1,J),J=1, N1)
   90 READ(5,520) EPSZ,IMAX
      WRITE(6,670) EPSZ,IMAX
      IF(ISW.LT.10) GOTO 100
      READ(5,500) (NBV(I),I=1,MM)
      WRITE(6,680) (NBV(I),I=1,MM)
  100 CALL LPRS1(A,21,M,N,EPSZ,IMAX,
     *            ISW,NBV,B,VW,IVW,ICON)
      WRITE(6,720) B(M1,M1)
      WRITE(6,690) ICON,IMAX
      IF(ICON.GE.20000) STOP
      IF(ICON.EQ.0) WRITE(6,700)
      IF(ICON.GE.10000) WRITE(6,710)
      WRITE(6,720) B(M1,M1)
      WRITE(6,730) (NBV(I),B(I,M1),I=1,MM)
      STOP
  500 FORMAT(10I4)
  510 FORMAT(11F6.0)
  520 FORMAT(E10.2,I5)
  600 FORMAT('1','INITIAL TABLEAU'
     *      /'0',10X,11(I6,4X))
  610 FORMAT(' ','LHS.LE.RHS')
  620 FORMAT(' ',I6,4X,11F10.3)
  630 FORMAT(' ','LHS.GE.RHS')
  640 FORMAT(' ','LHS.EQ.RHS')
  650 FORMAT(' ','OBJ.FUNC.(MAX)'
     *      /' ',10X,11F10.3)
  660 FORMAT(' ','OBJ.FUNC.(MIN)'
     *      /' ',10X,11F10.3)
  670 FORMAT('0','EPSZ=',E12.4,5X,
     *       'IMAX=',I4)
  680 FORMAT('0','INITIAL BASIS'
     *      /' ',20I4)
  690 FORMAT('0','ICON=',I5,12X,'IMAX=',I4)
  700 FORMAT('0','OPTIMAL SOLUTION')
  710 FORMAT('0','FEASIBLE SOLUTION')
  720 FORMAT(' ','OBJ.FUNC.',F15.4)
  730 FORMAT(' ',I6,3X,F15.4)
      END
```

Method

First, the following standard linear programming problem is explained.

Minimize the objective function

$$z = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + c_0 \tag{4.3}$$

subject to

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{d} \tag{4.1}$$
$$\boldsymbol{x} \geq \boldsymbol{o} \tag{4.2}$$

Assume an $m \times n$ matrix $\boldsymbol{A}$ where

rank $\boldsymbol{A} = m \leq n$

and assume

$\boldsymbol{x} = (x_1, x_2, ..., x_n)^{\mathrm{T}}$,

$\boldsymbol{d} = (d_1, d_2, ..., d_m)$ and

$\boldsymbol{c} = (c_1, c_2, ..., c_n)^{\mathrm{T}}$.

The *j*-th column of $\boldsymbol{A}$ is expressed by a_j when *m* column of $\boldsymbol{A}$:

$$\boldsymbol{a}_{k_1}, \boldsymbol{a}_{k_2}, ..., \boldsymbol{a}_{k_m}$$

are linearly independent, the corresponding x_{k_1}, x_{k_2},..., x_{k_m} are called bases and x_{k_i} is colled the *i*-th basic variable.

Suppose a set of non-basic variable numbers to be L, a system of linear equations which is equivalent to (4.1) and (4.3) can be expressed below:

$$x_{k_i} + \sum_{j \in L} f_{ij} x_j = g_i, \quad i = 1, 2, ..., m$$
$$z + \sum_{j \in L} p_j x_j = q \tag{4.4}$$

This is called a basic form and one of its solutions

$$x_{k_i} = g_i, \quad i = 1, 2, ..., m$$
$$x_j = 0, \quad j \in L \tag{4.5}$$
$$z = q$$

is colled a basic solution.

When

$$g_i \geq 0, \quad i = 1, 2, ..., m \tag{4.6}$$

holds, the basic solution shown in (4.5) satisfies (4.2). The solution shown in (4.5) is a basic feasible solution In the basic feasible form, if a certain s which satisfies (4.8) exisits,

$$p_j \leq 0, \ j \in L \tag{4.7}$$

holds, (4.5) is the optimal solution. (4.7) is the optimality criterion for the basic feasible solution. In the basic feasible form, if a certain *s* whic satisfies (4.8) exists,

$$p_s > 0, \ s \in L \tag{4.8}$$

it may be possible to get a better basic feasible solution which minimizes the value of z further by replacing any basis, by x_s. Let a null set be ϕ, and

$$I^+ = \{i \mid f_{is} > 0\}$$

holds, if $I^+ \neq \phi$ and

$$\frac{a_r}{f_{rs}} = \min_{i \in I^+} \left(\frac{g_i}{f_{is}} \right) \tag{4.9}$$

holds, the basis in which x_{k_r} is replaced by x_s is a feasible basis. And the basic solution for a new basis is as follows:

$$x_{k'_r} = \frac{g_r}{f_{rs}},$$
$$x_{k'_i} = g_i - f_{is} \frac{g_r}{f_{rs}}, \ i \neq r \quad (4.10)$$
$$x_j = 0 \qquad , \ j \in L',$$
$$z = q - p_s \frac{g_r}{f_{rs}}$$

When k'_i is of a new basic variable number and L' is a set of new non-basic variable numbers. Therefore the following relationships can be established:

$k'_r = s$,

$k'_i = k_i, \quad i \neq r,$

$L' = L - \{s\} + \{k_r\}$

The value z in this new basic feasible solution is smaller than the old value if $g_r > 0$.

When $I^+ = \phi$, the problem has no optimal solution. If

$$\boldsymbol{B} = [\boldsymbol{a}_{k_1}, \boldsymbol{a}_{k_2}, ..., \boldsymbol{a}_{k_m}]$$
$$\boldsymbol{C}_B = [c_{k_1}, c_{k_2}, ..., c_{k_m}] \tag{4.11}$$
$$\boldsymbol{\pi} = \boldsymbol{C}_B \boldsymbol{B}^{-1}$$

are give, the values of coefficients of a basic form shown in (4.4) and the right side are expressed as follows:

$$\boldsymbol{f}_j = \boldsymbol{B}^{-1}\boldsymbol{a}_j, \ j \in L$$
$$\boldsymbol{g} = \boldsymbol{B}^{-1}\boldsymbol{d} \tag{4.12}$$
$$p_j = \boldsymbol{\pi}\boldsymbol{a}_j - c_j, \ j \in L$$
$$q = \boldsymbol{\pi}\boldsymbol{d} - c_0$$

Where $\boldsymbol{f_j} = (f_{1j}, f_{2j}, ..., f_{mj})^{\mathrm{T}}$ and $\boldsymbol{g} = (g_1, g_2, ..., g_m)^{\mathrm{T}}$ hold. $\boldsymbol{B}$, $\boldsymbol{B}^{-1}$ and π are called a basic matrix, basic inverse matrix and simplex multiplier (vector) respectively.

The following summarizes the computational procedures for a standard linear progromming problem:

- Basic inveerse matrix $\boldsymbol{B}^{-1}$ for a fesible basis and simplex multiplier $\boldsymbol{\pi}$ are computed.
- p_j is computed by using the formula shown in (4.12).
- The optimality criterion shown in (4.7) is tested. If the standards are satisfied, the $\boldsymbol{g}$ is assumed as an optimal solution. If they are not satisfied, $\boldsymbol{f}_s$ is computed based on (4.12).
- The variable to be replaced by x_s is selected based on (4.9) and a new fesible base is produced.

The revised simplex method obtains the optiml solution by repeating these procedures.

Soulution for non-standard linear programming problem
When unequality constraints are contained, they must be adjusted to the following equality constraints to transform them to a stndard linear programming problem.
Given

$$\sum_{j=1}^{n} a_{ij} x_j \le d_i, \quad i = 1,2,\dots, m_l$$

where

$$\left.\begin{array}{r} \displaystyle\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = d_i \\ x_{n+i} \ge 0 \end{array}\right\}, i = 1,2,\dots, m_l \qquad (4.13)$$

is assumed to be held.
Given

$$\sum_{j=1}^{n} a_{ij} x_j \le d_i, i = m_l + 1, m_l + 2, \dots, m_l + m_g$$

where

$$\left.\begin{array}{r} \displaystyle\sum_{j=1}^{n} a_{ij} x_j - x_{n+i} = d_i \\ x_{n+i} \ge 0 \end{array}\right\}, \quad \begin{array}{l} i = m_l + 1, m_l + 2, \\ \dots, m_l + m_g \end{array} \qquad (4.14)$$

is assumed to be held.

The varibles added in (4.13) and (4.14) are called slack variables. The maximization problem is transformed to the minization problem by multiplying objective functions by -1.

Obtaining the initial basic solution
When the initial feasible basic solution has not been obtained, the following problem must be solved before the given problem is solved:
Minimize the objective function

$$z_1 = \sum_{i=1}^{m} x_i^{(a)}$$

subfect to:

$$\begin{array}{l} \boldsymbol{A}^* \boldsymbol{x}^* + \boldsymbol{A}^{(a)} \boldsymbol{x}^{(a)} = \boldsymbol{d}, \\ \boldsymbol{x}^* \ge 0, \ \boldsymbol{x}^{(a)} \ge 0 \end{array} \qquad (4.15)$$

where $\boldsymbol{x}^*$ is an $(n + m_l + m_g)$-dimensional vector contains slack variables and $\boldsymbol{A}^*$ is the corrsponding coefficient matrix.
$x^{(a)}$ is an m-dimensional vector,
$\boldsymbol{x}^{(a)} = (x_1^{(a)}, x_2^{(a)}, \dots, x_m^{(a)})^{\mathrm{T}}$ and $A^{(a)}$ is a diagonal matrix of order m, $\boldsymbol{A}^{(a)} = \left(a_{ij}^{(a)}\right)$

$$\text{where } \boldsymbol{a}_{ij}^{(a)} = \begin{cases} 1 & \text{when } d_i \ge 0 \\ -1 & \text{when } d_i < 0 \end{cases} \qquad (4.16)$$

$\boldsymbol{x}_i^{(a)}$ are called artificial variables.

When the optimal solution is obtained of this problem, if, $z_1 = 0$, that is $\boldsymbol{x}^{(a)} = \boldsymbol{o}$ holds, the basic feasible solution for the given problem has been obtained. When $z_1 > 0$, the original problem is infeasible. If $z_1 = 0$, an artificial variable may remain in the basis. In this case, it must be handled so that the articial variable value is always zero when the subsequent problem is to be solved.

Computational procedures
When the initial fesible basis can be given (ISW=10 or ISW=11), this subroutine immediately executes procedure 2).

1) Initialization of phase 1
   Assuming $\boldsymbol{x}^{(a)}$ as a basis in (4.15) this subroutine defines the basic inverse matrix $\boldsymbol{B}^{-1}$, which is equal to (4.16).
   This subroutine determines the corresponding basic solution $\boldsymbol{g}$, simplex multiplier π and objective function value q form (4.11) and (4.12):

   $$g_i = |d_i|$$

   $$\pi_i = \left.\begin{cases} 1 & \text{when } d_i \ge 0 \\ -1 & \text{when } d_i < 0 \end{cases}\right\}, i = 1,2,\dots, m \qquad (4.17)$$

   $$q = \sum_{i=1}^{m} |d_i|$$

   This subroutine immediately executes procedure 4)
2) The basic inverse matrix corresponding to the initial feasible basis is computed by using the subroutines ALU and LUIV. If the inverse matrix cannot be obtained, this subroutine terminates abnormally after it sets a condition code of 20000 to ICON.
3) Initalizain of phase 2
   Simplex multiplier π is computed by using (4.11).
   Objective function value q is computed based on relationship shown in (4.12):

   $$q = \boldsymbol{C_B g} + c_0$$

   Where the element of $\boldsymbol{C_B}$ is assumed to be 0 corresponding to a slack variable and artificial variable.

4) Testing the optimality criterion
This suborutine obtains the p_j corresponding to the non-basic variables containing slack variables by using (4.12) and tests the optimality criterion shown in (4.7). If (4.8) is satisfied, this subroutine executes procedure 5) immediately.
If the optimality criterion is safisfied:
   - when ISW $\geq$ 10, the optimal solution has been obtained. This subroutine terminates normally after it set a condition code of 0 to ICON.
   - when ISW<10 and $q > 0$, the problem has no basic feasible solution. This suborutine terminates after it sets condition code of 20000 to ICON.
   - when ISW<10 and $q = 0$, an initial basic feasible solution has been obtained. This subroutine completes phase 1 at this stage. This subroutine increments the contents of ISW by 10 and executes procedure 3) again.

5) Testing the number of iterations If the number of terations has not reached the upper limit, this subroutine executes procedure 6) and continue iterations. When it has reahed the upper limit during phase 1, this subroutine terminates abnormally after it sets a condition code of 24000 to ICON. When it has reached the upper limit during phase 2, this subroutine terminates after it sets a condition code of 11000 to ICON.

6) Replacing bases
This subroutine computes $\boldsymbol{f_s}$ corresponding to x_s by using (4.12). If $f_{is} \leq 0$ $(i = 1,2,...,m)$ is satisfied, no optimal solution exists. When such a condition occurs during phase 1, this subroutine sets a condition code of 10000 to ICON and terminates. When it occurs during phase 2, this subroutine sets a condition code of 23000 to ICON and terminates. If $f_{is} > 0$, this subroutine determines base x_{kr} to be changed by using (4.9).

7) Computing basic inverse matrices for new bases
When the i-th row $(i = 1,2,...,m+1)$ of a matrix

$$\begin{bmatrix} \boldsymbol{B}^{-1} & \boldsymbol{g} \\ \boldsymbol{\pi} & q \end{bmatrix}$$

is expressed by $\boldsymbol{\beta}_i$, the matrix for new bases can be obtained as shown below:

$$\frac{1}{f_{rs}}\boldsymbol{\beta}_r \to \boldsymbol{\beta}_r \qquad (4.17)$$
$$\boldsymbol{\beta}_i - f_{is}\boldsymbol{\beta}_r \to \boldsymbol{\beta}_i, i \neq r$$

This is called pivot operation in which f_{rs} is assumed to be a pivot. This subroutine executes procedure 4) again after this procedure.

Convergence criterion

When a variable to be stored into a base is x_s and a variable to be fetched form the base during replacement of the base, the value of the objective function decreases by

$$p_s \frac{g_r}{f_{rs}} (> 0)$$

as seen from (4.10) if

$$g_r > 0 \qquad (4.18)$$

is satisfied.
The same basic feasible solution does not appear as far as (4.18) is satisfied during iteration. Since the number of basic feasible solutions is limited this subroutine can obtain the optimal soution while replacing base by the other - or it can find out that no optimal solution exists.
If $g_r = 0$ is satisfied, the value of the objective function does not vary. If this condition occurs repeatedly, a basic feasible solution which has appeared once may appear again - some feasible solution appear periodically. As a result no optimal solution cannot be obtained normally.
To avoid such a condition this subroutine imposed certain restriction on r and s:

- The minimum-valued j is regarded as s if $p_j > 0$ in (4.8)
- The r_0 which makes the minimum value of (4.9) zero, minimizes k_{r_0} and satisfies

$$\frac{g_{r_0}}{f_{r_0 s}} = \min_{i \in I^+}\left(\frac{g_i}{f_{is}}\right)$$

is regarded as r.

For further information, see Reference [41].

A52-21-0101 LSBIX, DLSBIX

A system of simultaneous linear equations with a real indefinite symmetric band matrix (block diagonal pivoting method)
CALL LSBIX (A, N, NH, MH, B, EPSZ, ISW, VW, IVW, ICON)

Function

This subroutine solve a system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

using the block diagonal pivoting method, where $\boldsymbol{A}$ is an $n \times n$ indefinite symmetric band matrix with band width h, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector (n>h≥0).(there are two similar methods, the one is analogous to Gaussian elimination method and the other is analogous to Crout method ;this subroutine uses the former.)

Paramters

A	Input. Coefficinet matrix $\boldsymbol{A}$. Compressed storage mode for symmetric band matrix. One-dimensional array of size $n(h+1)-h(h+1)/2$.
N	Input. The order n of coefficient matrix $\boldsymbol{A}$, constant vector $\boldsymbol{b}$, and solution vector $\boldsymbol{x}$.
NH	Input. Band width h of $\boldsymbol{A}$. The content is altered on output. (See“Comments on Use.”)
MH	Input Maximum tolerable band width h_m (N>MH≥NH) (See “Comments on Use.”)
B	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. One-dimensional array of size n.
EPSZ ..	Input. Tolerance (≥0.0) for relative zero test of pivots. The default value is used if 0.0 is specified. (See "Comments on Use.")
ISW....	Input. Control information. When l(≥1) systems of linear equations with the identical coefficient matirx are to be solved, ISW can be specified as follows: ISW=1 ... The first system is solved. ISW=2 ... The 2nd to l-th systems are solved. However, only parameter 2: $\boldsymbol{B}$ is specified for each consstant vector $\boldsymbol{b}$ of the systems, with the rest unchanged. (See "Notes".)
VW	Work area, One-dimensional array of size $n(h_m+1)-h_m(h_m+1)/2$. (See"Comments on Use.")
IVW ...	Work area. One-dimensional array of size $2n$.
ICON .	Output. Condition code. (See Table LSBIX-1.)

Table LSBIX-1 Condition codes

Code	Meaning	Processing
0	No error.	
20000	A relative zero pivot has been found. It is highly probale that the coefficient matrix is singular.	Bypassed.
25000	The bandwidth was exceeded the tolerable range during operation.	Bypassed.
30000	NH<0, NH>MH, MH≥N, EPSZ<0.0 or ISW ≠ 1 or 2.	Bypassed.

Comments on use

- Subprograms used
  SSL II...SBMDM, BMDMX, AMACH, MGSSL
  FORTRAN basic functions...AMAX1, ABS, IDIM

- Notes
  If 10^{-s} is specified as EPSZ, this is interperted as follows:
  If the loss of significant digits exceeds s decimal digits for the pivot value (a determinat of (1×1 or 2×2 matrix of the pivot) during the MDM$^{\mathrm{T}}$ decomposition by the block diagonal pivoting method, the pivot value is assumed to be relative zero, and ICON=20000 is set, then the processing is stopped.
  Let u be the unit round off, then the standard value of EPSZ is $16u$.
  It is posible to continue the decomposition, even if the pivot takes small value by specifying extreme small value, for the parameter EPSZ.
  In such case, however the result is not guarnteed.
  In case to solve several system of linear equations with identical coefficient matrix, the second and subsequent system can be solved efficiently by specifying as ISW = 2. Because this subroutine bypasses the procedure for MDM$^{\mathrm{T}}$ decomposition. When ISW=2.

For the method to obtain the determinant of matrix ***A***, see "Example" and explanations of subroutine SBMDM which is called in this subroutine.

If rows and columns are exchanged by pivoting, generally the band width of the matrix is increased. It is desirable to specify the maximum tolerable band width h_m so that the increment of band width can be allowed. If the band width exceeds h_m during $\mathrm{MDM}^{\mathrm{T}}$ decompositon, processing is stopped assuming that ICON=25000.

This subroutine permit to allocate the array A and VW to the same area.

- Example

Given l systems of linear equations with the identical coeffcient matrix, the solutions and the determinant are obtained as follows, where $n \le 100$ and $h \le h_m \le 50$.

```
C      **EXAMPLE**
       DIMENSION A(3825),B(100),IVW(200)
       READ(5,500) N,NH,MH
       WRITE(6,600) N,NH,MH
       NHP1=NH+1
       NT=(N+N-NH)*NHP1/2
       READ(5,510) (A(J),J=1,NT)
       READ(5,520) L
       EPSZ=1.0E-6
       ISW=1
       DO 10 K=1,L
       IF(K.GE.2) ISW=2
       READ(5,510) (B(I),I=1,N)
       CALL LSBIX(A,N,NH,MH,B,EPSZ,ISW,A,
      *           IVW,ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) STOP
       WRITE(6,620) (B(I),I=1,N)
   10 CONTINUE
       DET=1.0
       J=1
       I=1
   20 M=IVW(I)
       IF(M.NE.I) GO TO 30
       DET=A(J)*DET
       J=MIN0(I,MH)+1+J
       I=I+1
       GO TO 40
   30 JJ=MIN0(I,MH)+1+J
       DET=(A(J)*A(JJ)-A(JJ-1)*A(JJ-1))*DET
       J=MIN0(I+1,MH)+1+JJ
       I=I+2
   40 IF(I.LE.N) GO TO 20
       WRITE(6,630) DET
       STOP
  500 FORMAT(3I4)
  510 FORMAT(4E15.7)
  520 FORMAT(I4)
  600 FORMAT('1'/10X,'N=',I3,5X,'NH=',I3,5X,
      *'MH=',I3)
  610 FORMAT('0',10X,'ICON=',I5)
  620 FORMAT(11X,'SOLUTION VECTOR'
      */(15X,5E15.6))
  630 FORMAT('0',10X,
      *'DETERMINANT OF COEFFICIENT MATRIX=',
      *E15.6)
       END
```

Method

This subroutine solves a system of linear equation

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

as follows:

- $\mathrm{MDM}^{\mathrm{T}}$ decompositon of the coefficient matrix ***A*** (the block diagonal pivoting method)

The $\mathrm{MDM}^{\mathrm{T}}$ decomposition of coefficient matrix ***A*** is obtained by the block diagonal pivoting method as

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \tag{4.2}$$

where ***P*** denotes the permutation matrix to exchange rows in the pivoting operation, ***M*** is a unit lower-band matrix, and ***D*** is a symmetric block diagonal matrix comprising blocks at most of order 2. Subroutine SBMDM is used for this calculation.

- Solution

A system of linear equations (4.1) is equivalent to

$$\boldsymbol{P}^{-1}\boldsymbol{MDM}^{\mathrm{T}}\boldsymbol{P}^{-\mathrm{T}}\boldsymbol{x} = \boldsymbol{b} \tag{4.3}$$

This can be solved by forward and backward substitutions and other minor computations as follows:

$$\boldsymbol{Mx}^{(1)} = \boldsymbol{Pb} \tag{4.4}$$

$$\boldsymbol{Dx}^{(2)} = \boldsymbol{x}^{(1)} \tag{4.5}$$

$$\boldsymbol{M}^{\mathrm{T}}\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)} \tag{4.6}$$

$$\boldsymbol{P}^{-\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{(3)} \tag{4.7}$$

Subroutine BMDMX is used for these calculations. (See references [9] and [10] for details.)

A52-31-0101 LSBX, DLSBX

A system of linear equations with a positive-definite symmetric band matrix (Modified Cholesky's method)
CALL LSBX (A, N, NH, B, EPSZ, ISW, ICON)

Function

This subroutine solves a system of linear equations (1.1) by using the modified Cholesky's method,

$$\boldsymbol{Ax} = \boldsymbol{b}$$

Where $\boldsymbol{A}$ is an $n \times n$ real positive-definite symmetric band matrix with lower and upper band widths h, $\boldsymbol{b}$ is an n-dimensional real constant vector, $\boldsymbol{x}$ is an n-dimensional solution vector, and $n > h \geq 0$.

Parameters

A	Input. Coefficient matrix $\boldsymbol{A}$. The contents of A are altered on output. $\boldsymbol{A}$ is stored in one-dimensional array of size $n(h+1)-h(h+1)/2$ in the compressed mode for symmetric band matrices.
N	Input. Order n of coefficient matrix $\boldsymbol{A}$.
NH	Input. Lower and upper band width h.
B	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. One dimensional array of size n.
EPSZ..	Input. Tolerance for relative zero test of pivots in decomposition process of matrix $\boldsymbol{A}$($\geq$0.0). When this is 0.0, the standard value is used. (Refer to "Notes")
ISW...	Input. Control information. When solving $l(\geq 1)$ systems of linear equations with an identical coefficient matrix, ISW can be specified as follows: ISW=1...The first system is solved. ISW=2...The 2nd to lth system are solved. However, only parameter B is specified for each constant vector $\boldsymbol{b}$ of the systems with the rest unchanged. (Refer to "Notes".)
ICON ..	Output. Condition code. Refer to Table LSBX-1.

Table LSBX-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The negative pivot occurred. The coefficient matrix is not positive-definite.	Continued
20000	The relatively zero pivot occurred. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	NH<0, NH≥N, EPSZ<0.0 or ISW ≠ 1,2.	Bypassed

Comments on use

- Subprograms used
  SSL II ....SBDL, BDLX, AMACH, MGSSL
  FORTRAN basic function....ABS

- Notes
  The solution obtained by this subroutine can be refined in accuracy by successively calling subroutine LSBXR.
  Since this subroutine omits the operations concerning the elements out of the band, the processing speed is faster than subroutine LSX provided for positive-definite symmetric matrices.
  If EPSZ is set to 10^{-s}, this value has the following meaning; while performing the LU-decomposition by Crount's method, if the loss of over s significant digits occurred for the pivot, the LU-decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero. The standard value of EPSZ is $16u$, where u being the unit round off. If the processing is to proceed at a lower pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.
  When successively solving systems of linear equations with an identical coefficient matrix, computation can be performed by setting ISW = 2, the $\mathrm{LDL}^{\mathrm{T}}$ decomposition process for the coefficient matrix is bypassed so that the computation time can be reduced.
  If the negative pivot occured in the decomposition, the coefficient matrix is not positive-definite. In this case, this subroutine is continued with ICON=10000. However, it should be noted large calculation errors may occur since the pivoting is not performed.
  The determinant of the coefficient matrix can be obtained by multiplying all the n diagonal elements in array A after the subroutine has been executed and then by inverting the multiplied value. Note that array A is in the compressed mode for symmetric band matrices.
  The contents of the resultant A are identical to those on output of subroutine SBDL.

- Example
  l systems of linear equations in *n* unknown with an identical coefficient matrix are solved. $n \geq 100$ and $h \leq 50$.

```
C     **EXAMPLE**
      DIMENSION A(3825),B(100)
      READ(5,500) N,NH
      WRITE(6,600) N,NH
      NH1=NH+1
      NT=N*NH1-NH*NH1/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,500) L
      K=1
      ISW=1
      EPSZ=1.0E-6
   10 READ(5,510) (B(I),I=1,N)
      CALL LSBX(A,N,NH,B,EPSZ,ISW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,620) (B(I),I=1,N)
      IF(K.EQ.L) GO TO 20
      K=K+1
      ISW=2
      GO TO 10
   20 DET=A(1)
      K=1
      DO 30 I=2,N
      K=K+MIN0(I,NH1)
      DET=DET*A(K)
   30 CONTINUE
      DET=1.0/DET
      WRITE(6,630) DET
      STOP
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'/10X,'N=',I3,'NH=',I3)
  610 FORMAT('0',10X,'ICON=',I5)
  620 FORMAT(11X,'SOLUTION VECTOR'
     */(15X,5E17.8))
  630 FORMAT('0',10X,
     *'DETERMINANT OF COEFFICIENT MATRIX='
     *, E17.8)
      END
```

Method
A system of linear equations (4.1) with a real positive-definite symmetric band matrix ***A*** are solved in the following procedure.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

- LDL^T decomposition of the coefficient matrix ***A*** (Modified Cholesky's method) LDL^T decomposion is performed on coefficient matrix ***A*** by modified Cholesky's method.

$$\boldsymbol{A} = \boldsymbol{LDL}^T \tag{4.2}$$

  where ***L*** is a unit lower band matrix and ***D*** is a diagonal matrix. Subroutine SBDL is used for this operation.

- Solving $\boldsymbol{LDL}^T\boldsymbol{x} = \boldsymbol{b}$ (Forward and backward substitutions)

A system of linear equations

$$\boldsymbol{LDL}^T\boldsymbol{x} = \boldsymbol{b} \tag{4.3}$$

  is solved using subroutine BDLX. For details, see Reference [7].

A52-31-0401 LSBXR, DLSBXR

Iterative refinement of the solution to a system of linear equations with a positive-definite symmetric band matrix

CALL LSBXR (X, A, N, NH, FA, B, VW, ICON)

Function

Given an approximate solution to linear equations with an $n \times n$ positive definite symmetric band matrix of upper and lower band width h,

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

this subroutine refines the approximate solution by the method of iterative modification, where $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector.

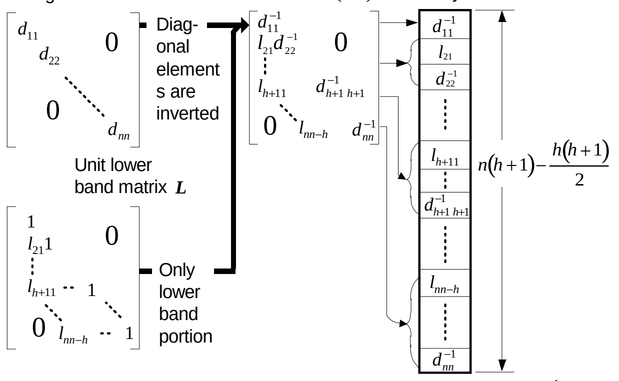
Prior to calling this subroutine, the coefficient matrix $\boldsymbol{A}$ must be LDL$^{\mathrm{T}}$-decomposed as shown in Eq. (1.2),

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \qquad (1.2)$$

where $\boldsymbol{L}$ and $\boldsymbol{D}$ are an $n \times n$ unit lower band matrix with lower band width h and an $n \times n$ diagonal matrix respectively. Also $n > h \geq 0$.

Parameters

X.....	Input. Approximate solution vector $\boldsymbol{x}$. Output. Refind solution vector $\boldsymbol{x}$. One dimensional array of size n.
A....	Input. Coefficient matrix $\boldsymbol{A}$. Matrix $\boldsymbol{A}$ is stored in one-dimensional array of the size $n(h+1)$-$h(h+1)/2$ in the compressed mode for a symmetric band matrix.
N....	Input. Order n of the matrix $\boldsymbol{A}$. (Refer to notes.)
NH....	Input. Lower band width h.
FA....	Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$. Refer to Fig. LSBXR-1. Matrices are stored in one -dimensional array of size $n(h+1)$-$h(h+1)/2$ in the compressed mode for a symmetric band matrix.
B....	Input. Constant vector $\boldsymbol{b}$. One-dimensional array of size n.
VW....	Work area. One-dimensional array of size n.
ICON..	Output. Condition code. Refer to Table LSBXR-1.


Diagonal matrix $\boldsymbol{D}$
Matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$
Array FA
Diag-onal elements are inverted
Unit lower band matrix $\boldsymbol{L}$
Only lower band portion
$n(h+1)-\frac{h(h+1)}{2}$

Note: The diagonal and the lower band portions in the matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$ are contained in array FA in the compressed mode for a symmetric band matrix.

Fig .LSBXR-1 Storage of matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$

Table LSBXR-1 Condition Codes

Code	Meaning	Processing
0	No error	
10000	Coefficient matrix was not positive-definite	Discontinued
25000	The convergence condition was not met because of an ill-conditioned coefficient matrix.	(Refer to Method (4) d for the convergence condition.)
30000	N=0,NH<0 or NH≥|N|	Bypassed

Comments on use

- subprograms used
  SSL II ... BDLX, MSBV, AMACH, MGSSL
  FORTRAN basic function ... ABS

- Notes
  This subroutines iteratively improves the approximate solution $\tilde{\boldsymbol{x}}$ obtained by subroutine LSBX to get solution $\boldsymbol{x}$ with refined accuracy.
  Therefore, prior to calling this subroutine, $\tilde{\boldsymbol{x}}$ must be obtained by subroutine LSBX and the results must be input as the parameters X and FA for this subroutine. In addition, the coefficient matrix $\boldsymbol{A}$ and constant vector $\boldsymbol{b}$ whitch are required for this subroutine must also be prepared and stored separately before valling subroutine LSBX.
  Refer to the example for details
  By specifying N = -n, an estimated accuracy (relative error) for the approximate solution $\tilde{\boldsymbol{x}}$ given by the subroutine LSBX can be obtained.
  When specified, this subroutine calculates the relative error and outputs it to work area VW(1) without performing the iterative refinement of accuracy.Refer to method for estimation of the accuracy.

- Example
  An approximate solution $\tilde{\boldsymbol{x}}$ for a system of linear equations in n unkowns is obtained by subroutine

LSBX, then it is iteratively refined by this subroutine. $n \leq 100$ and $h \leq 50$

```
C     **EXAMPLE**
       DIMENSION A(3825),FA(3825),X(100),
      *B(100),VW(100)
   10 READ(5,500) N,NH
      IF(N.EQ.0) STOP
      NH1=NH+1
      NT=N*NH1-NH*NH1/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (B(I),I=1,N)
      WRITE(6,610) (I,B(I),I=1,N)
      DO 20 I=1,N
   20 X(I)=B(I)
      DO 30 I=1,NT
   30 FA(I)=A(I)
      EPSZ=0.0E0
      ISW=1
      CALL LSBX(FA,N,NH,X,EPSZ,ISW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL LSBXR(X,A,N,NH,FA,B,VW,ICON)
      WRITE (6,630) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,640) (I,X(I),I=1,N)
      DET=FA(1)
      L=1
      DO 40 I=2,N
      L=L+MIN0(I,NH1)
   40 DET=DET*FA(L)
      DET=1.0/DET
      WRITE(6,650) DET
      STOP
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  610 FORMAT(///10X,'CONSTANT VECTOR'
     */(10X,4('(',I3,')',E17.8)))
  620 FORMAT('0','LSBX  ICON=',I5)
  630 FORMAT('0','LSBXR ICON=',I5)
  640 FORMAT('0',10X,'SOLUTION VECTOR'
     */(10X,4('(',I3,')',E17.8)))
  650 FORMAT(///10X,
     *'DETERMINANT OF COEFFICIENT MATRIX=',
     *E17.8)
      END
```

Method

Given an approximate solution, $\tilde{\boldsymbol{x}}$ to the linear equations,

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principle of iterative refinement
  The iterative refinement is a method to obtain a successively, improved approximate solution $\boldsymbol{x}^{(s+1)}$ to the linear equations (4.1) through use of the following euqtions starting with $\boldsymbol{x}^{(1)} = \tilde{\boldsymbol{x}}$.

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$

$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$

s=1,2,...

where $\boldsymbol{x}^s$ is the s-th approximate solution to equation(4.1).

If Eq.(4.2)is accurately computed ,a refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained.

If, however, the condition of coefficient matrix $\boldsymbol{A}$ is not suitable, an improved solution is not obtained. (Refer to 3.4(2)e.Iterative refinement of a solution.)

- Procedure performed in this subroutine
  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by subroutine LSBX.
  Then -this subroutine repeats the following steps:
  - The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq. (4.2). Subroutine MSBV is used for this operation.
  - The correction $\boldsymbol{d}^{(s)}$ is obtained next by using Eq.(4.3). Subroutine BDLX is used for this operation.
  - Finally the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq.(4.4).

The convergence of the iteration is tested as follows:
Considering u as a unit round off, the iteration refinement is assumed to converge if, at the s-th iteration step, the following relationship is satisfied

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty / \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2 \cdot u \tag{4.5}$$

The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as the final solution.
However, if the relationship,

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable. The iteration refinement is assumed not to converge, and consequently the processing is terminated with ICON = 25000.

- Accuracy estimation for approximate solution
  Suppose that the error for the approximate solution $\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}\left(=\boldsymbol{x}^{(1)}-\boldsymbol{x}\right)$,its relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty/\left\|\boldsymbol{x}^{(1)}\right\|_\infty$. If this iteration method converges, $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$, therefore the relative error for the approximate solution is estimated by $\left\|d^{(1)}\right\|_\infty/\left\|\boldsymbol{x}^{(1)}\right\|_\infty$.(Refer to ’’Accuracy estimation for approximate solution’’ in Section 3.4.)
  For further details, see References[1],[3]and[5].

A22-21-0101 LSIX,DLSIX

A system of linear equations with a real indefinite symmetric matrix (Block diagnoal pivoting method)
CALL LSIX (A,N,B,EPSZ,ISW,VW,IP,IVW,ICON)

Function

This subroutine solves a system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b}$$

using the block diagonal pivoting method(there are two similar methods which are called Croutlike method and Gaussianlike method respectively. This subroutine uses the former method.),where $\boldsymbol{A}$ is an $n\times n$ real indefinite symmetric matrix, $\boldsymbol{b}$ is an n-dimensional real constant vector, $\boldsymbol{x}$ is an n-dimensional solution vector, and $n \geq 1$.

Parameters

A ..... Input. Coefficient matrix $\boldsymbol{A}$.
The contents of A may be overridden after operation.
Compressed mode for symmetric matrix.
One-dimensional array of size $n(n+n)/2$.

N ..... Input. Order n of the coefficient matrix $\boldsymbol{A}$, constant vector $\boldsymbol{b}$ and solution vector $\boldsymbol{x}$.

B ..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

EPSZ .. Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ ($\geq$0.0).
If EPSZ = 0.0, a standard value is used.
(See Notes.)

ISW ... Input. Control information
When l ($\geq$1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows:
ISW=1 ... The first system is solved.
ISW=2 ... The 2-nd to n-th systems are solved. However, only parameter B is specified for each constant vector $\boldsymbol{b}$ of the systems, with the rest unchanged.
(See Notes.)

VW .... Work area. One-dimensional array of size $2n$.

IP .... Work area. One-dimensional array of size n.

IVW ... Work area. One-dimensional array of size n.

ICON ..... Output. Condition code. Refer to Table LSIX-1.

Table LSIX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The relatively zero pivot occurred. It is highly probable that the coefficient matrix is singular.	Aborted
30000	N<1, EPSZ<0.0 or ISW≠1, 2	Aborted

Comments on use

- Subprograms used
SSL II ... SMDM, MDMX, AMACH, MGSSL USCHA
FORTRAN basic functions ... ABS,SQRT, IABS,ISIGN

- Notes
If EPSZ is set to 10^{-s}, this value has the following meaning: while performing the $\mathrm{MDM}^{\mathrm{T}}$ -decomposition by the block diagonal pivoting method,
if the loss of over s significant digits occurred for the pivot value(i.e., determinant of a 1×1 or 2×2 matrix of the pivot), the $\mathrm{MDM}^{\mathrm{T}}$ -decomposition should be discontinued with ICON = 20000 regarding the pivot value as relatively zero.
Let u be the unit round off, then the standard value of EPSZ is 16 u.
If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.
When successively solving systems of linear equations with the identical coefficient matrix, after solving the first system, ISW should be set to 2.
With ISW=2, calculation time is reduced since the process in which the coefficient matrix $\boldsymbol{A}$ is $\mathrm{MDM}^{\mathrm{T}}$ decomposed is bypassed.
For how to obtain the determinant of matrix $\boldsymbol{A}$, refer to the example shown below or the corresponding description on the subroutine SMDM which is used in this subroutine.
This subroutine makes use of symmetric matrix characteristics also while decomposing in order to save the data storage area.

- Example
l systems of linear equations with the identical coefficient matrix are solved as well as the determinant of the coefficient matrix, where $n\leq$100.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100),VW(200),
     *IP(100),IVW(100)
      CHARACTER*4 IA,IB,IX
      DATA IA,IB,IX/'A   ','B   ','X   '/
      READ(5,500) N,L
      NT=(N*(N+1))/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N
      CALL PSM(IA,1,A,N)
      M=1
      ISW=1
      EPSZ=1.0E-6
   10 READ(5,510) (B(I),I=1,N)
      CALL PGM(IB,1,B,N,N,1)
      CALL LSIX(A,N,B,EPSZ,ISW,VW,IP,IVW,
     *          ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      CALL PGM(IX,1,B,N,N,1)
      IF(M.GE.L) GO TO 20
      M=M+1
      ISW=2
      GO TO 10
   20 DET=1.0
      I=1
      J=1
   30 IF(IP(J+1).GT.0) GO TO 40
      DET=DET*(A(I)*A(I+J+1)-A(I+J)*A(I+J))
      J=J+2
      I=I+J-1+J
      GO TO 50
   40 DET=DET*A(I)
      J=J+1
      I=I+J
   50 IF(J.LT.N) GO TO 30
      IF(J.EQ.N) DET=DET*A(I)
      WRITE(6,620) DET
      STOP
  500 FORMAT(2I3)
  510 FORMAT(4E15.7)
  600 FORMAT('1',
     * /6X,'LINEAR EQUATIONS AX=B'
     * /6X,'ORDER=',I4)
  610 FORMAT(' ',5X,'ICON OF LSIX=',I6)
  620 FORMAT(' ',5X,'DETERMINANT OF A=',
     * E14.7)
      END
```

The subroutines PSM and PGM that are both called in this example are used for printing a real symmetric matrix and a read general matrix, respectively. The descriptions on those subroutines can be found in the example for the subroutine MGSM.

Method

The linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

is solved using the following procedure:

- MDM$^{\mathrm{T}}$ - decomposition of the coefficient matrix $\boldsymbol{A}$ (block diagonal pivoting method).
  The coefficient matrix $\boldsymbol{A}$ is decomposed by the block diagonal pivoting method as follows.

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \tag{4.2}$$

where $\boldsymbol{P}$ is a permutation matrix that exchanges rows of the matrix $\boldsymbol{A}$ required in its pivoting, $\boldsymbol{M}$ is a unit lower triangular matrix, and $\boldsymbol{D}$ is a symmetric block diagonal matrix which consists only of the blocks, each at most of order 2. The subroutine SMDM is used for this operation.

- Solving $\boldsymbol{P}^{-1}\boldsymbol{MDM}^{\mathrm{T}}\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x} = \boldsymbol{b}$
  solving the linear equations (4.1)is reduced to solving

$$\boldsymbol{P}^{-1}\boldsymbol{MDM}^{\mathrm{T}}\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x} = \boldsymbol{b} \tag{4.3}$$

this equation (4.3) can be also reduced to equation (4.4) to (4.7).

$$\boldsymbol{Mx}^{(1)} = \boldsymbol{Pb} \tag{4.4}$$
$$\boldsymbol{Dx}^{(2)} = \boldsymbol{x}^{(1)} \tag{4.5}$$
$$\boldsymbol{M}^{\mathrm{T}}\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)} \tag{4.6}$$
$$\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x} = \boldsymbol{x}^{(3)} \tag{4.7}$$

Consequently, the solution is obtained using forward substitution and backward substitution, in addition to minor computation. The subroutine MDMX is used for this operation. For more details, see References [9] and [10].

A22-21-0401 LSIXR, DLSIXR

Iterative refinement of the solution to a system of linear equations with a real indefinite matrix
CALL LSIXR (X, A, N, FA, B, IP, VW, ICON)

Function

When an approximate solution $\tilde{x}$ is given to linear equations with an $n \times n$ symmetric matrix.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

This subroutine refines the approximate solution by the method of iterative modification, where ***b*** is an n-dimensional real constant vector, and ***x*** is an *n*-dimensional solution vector.

Prior to calling this subroutine, the coefficient matrix ***A*** must be MDM^T decomposed as shown in(1.2),

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \tag{1.2}$$

where, ***P*** is a permutation matrix which exchanges rows of the matrix ***A*** required in pivoting, ***M*** is a unit lower triangular matrix and ***D*** is a symmetric block diagonal matrix consist of a symmetric blocks of maximum order 2, furthermore, if $d_{k+1,k} \neq 0, m_{k+1,k} = 0$ .Where $\boldsymbol{M} = (m_{ij})$, $\boldsymbol{D} = (d_{ij})$ and $n \geq 1$.

Parameters

X ..... Input .Approximate solution vector $\tilde{x}$.
Output. Refined solution vector ***x***.
One-dimensional array of size *n*.

A ..... Input. Coefficient matrix ***A***.
Matrix ***A*** is stored in a one-dimensional array of size *n*(*n*+1)/2 in the compressed mode for a symmetric matrix.

N ..... Input. Order *n* of matrix ***A***.
See Notes.

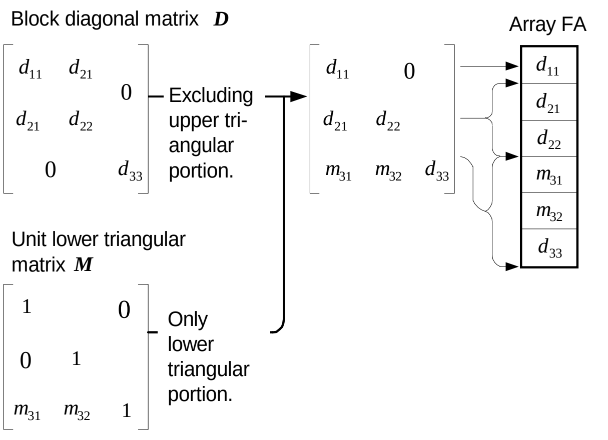
FA ..... Input. Matrices ***M*** and ***D***.
See Fig. LSIXR-1.Matrices are stored in a one-dimensional array of size *n*(*n*+1)/2.

B ..... Input. Constant vector ***b***.
One-dimensional array of size *n*.

IP ..... Input. Transposition vector indicating the history of exchanging rows of the matrix ***A*** required in pivoting.
One-dimensional array of size *n*.
(See Notes.)

VW ..... Work area. One-dimensional array of size *n*

ICON .. Output. Condition code.
See Table LSIXR-1.


Block diagonal matrix D
Array FA
Excluding upper tri-angular portion.
Unit lower triangular matrix M
Only lower triangular portion.
d_{11} d_{21} d_{22} m_{31} m_{32} d_{33}

Note: The diagonal and the lower triangular portions of the matrix ***D***+(***M***-***I***) are stored in one-dimensional array FA in the compressed mode for a symmetric matrix.
In this case *D* consists of block of order 2 and 1.

Fig. LSIXR-1 Storage method of matrices D and M

Table LSIXR-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Coefficient matrix was singular	Discontinued
25000	The convergence condition was not satisfied because of an ill-conditioned coefficient matrix.	Discontinued.(See " Method" for the convergence condition.
30000	N = 0	Bypassed

Comments on use

- Subprograms used
  SSL II ... MDMX,MSV,AMACH, MGSSL
  FORTRAN basic functions ... IABS, ABS

- Notes
  This subroutine iteratively improves to get solution the approximate solution $\tilde{x}$ obtained by subroutine LSIX $\tilde{x}$ with refined precision.
  Therefore, prior to calling this subroutine, $\tilde{x}$ must be obtained by subroutine LSIX and the results must be input as the parameters X, FA and IP for this subroutine. In addition, the coefficient matrix ***A*** and constant vector ***b*** which are required for this subroutine must also be prepared and stored separately before calling subroutine LSIX.
  See the Example for details.
  By specifying N = $-n$, an estimated accuracy (relative error) for the approximate solution $\tilde{x}$ given by subroutine LSIX can be obtained.
  When specified, this subroutine calculates the relative error and outputs it to work area VW(1) without performing the iterative refinement of accuracy.
  See the Method for estimation of the accuracy.

- Example

  An approximate solution $\tilde{x}$ for a system of linear equations of order *n* is obtained by subroutine LSIX, then it is iteratively refined by this subroutine.

  $n \leq 100$.

```
C      **EXAMPLE**
       DIMENSION A(5050),FA(5050),X(100),
      *B(100),VW(200),IP(100),IVW(100)
   10 READ(5,500) N
       IF(N.EQ.0) STOP
       NT=N*(N+1)/2
       READ(5,510) (A(I),I=1,NT)
       READ(5,510) (B(I),I=1,N)
       DO 20 I=1,N
       X(I)=B(I)
   20 CONTINUE
       DO 30 I=1,NT
   30 FA(I)=A(I)
       EPSZ=0.0E0
       ISW=1
       CALL LSIX(FA,N,X,EPSZ,ISW,VW,IP,IVW,
      *          ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) GO TO 10
       CALL LSIXR(X,A,N,FA,B,IP,VW,ICON)
       WRITE(6,630) ICON
       IF(ICON.GE.20000) STOP
       WRITE(6,640) (I,X(I),I=1,N)
       GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  620 FORMAT('0',10X,'LSIX  ICON=',I5)
  630 FORMAT('0',10X,'LSIXR ICON=',I5)
  640 FORMAT('0',10X,'SOLUTION VECTOR'
      */(10X,4('(',I3,')',E17.8)))
       END
```

Method

Given an approximate solution $\tilde{x}$ to the linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principle of iterative refinement

  The iterative refinement method is to obtain a succesively improved approximate solution $\boldsymbol{x}^{(s)}$ to the linear equations (4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)} = \tilde{\boldsymbol{x}}$

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$

$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$

$$s = 1,2, ...$$

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to (4.1).

If (4.2) is accurately computed, a refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained. If, however, the condition of coefficient matrix $\boldsymbol{A}$ is not suitable, an improved solution is not obtained. (See Section 3.4 "Iterative refinement of a solution.")

- Procedure performed in this subroutine

  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by subroutine LSIX.

  Then this subroutine repeats the following steps:
  - The residual $\boldsymbol{r}^{(s)}$ is computed by using (4.2). Subroutine MSV is used for this operation.
  - The correction $\boldsymbol{d}^{(s)}$ is obtained next by using (4.3). Subroutine MDMX is used for this operation.
  - Finally the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using (4.4).

  The convergence of the iteration is tested as follows: Considering u as a unit round-off, the iteration refinement is assumed to converge if, at the s-th iteration step, the following relationship is satisfied.

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty / \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2 \cdot u \tag{4.5}$$

  The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as the final solution. However, if the relationship,

$$\frac{\left\|\boldsymbol{d}^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|\boldsymbol{d}^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

  results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable. The iteration refinements is assumed not to converge, and consequently the processing is terminated with ICON=25000.

- Accuracy estimation for approximate solution

  Suppose that the error for the approximate solution $\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}$ $\left(= \boldsymbol{x}^{(1)} - \boldsymbol{x}\right)$,its relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$

  If this iteration method converges, $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$, therefore the relative error for the approximate solution is estimated by $\left\|\boldsymbol{d}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$. (See Section 3.4 for details.)

  For further details, see References [1], [3], and [5].

A52-31-0501 LSTX, DLSTX

A system of linear equations with a positive-definite symmetric tridiagonal matrix (Modified Cholesky's method)
CALL LSTX (D, SD, N, B, EPSZ, ISW, ICON)

Function

A system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

is solved by using the modified Cholesky's method, where $\boldsymbol{A}$ is an $n \times n$ positive definite symmetric tridiagonal matrix, $\boldsymbol{b}$ is an n-dimentional real constant vector and $\boldsymbol{x}$ is an n-dimentional solution vector. Also n≥1.

Parameters

D	Input. Diagonal portion of the coefficient matrix $\boldsymbol{A}$. After computation, the contents are destroyed. Refer to Fig. LSTX-1. One-dimensional array of size n.
SD	Input. Subdiagonal portion of the coefficient matrix $\boldsymbol{A}$. After computation, the contents are destroyed. Refer to Fig. LSTX-1. One-dimensional array of size n–1.
N	Input. Order n of the coefficient matrix $\boldsymbol{A}$, the constant vector $\boldsymbol{b}$ and the solution vector $\boldsymbol{x}$.
B	Input. Constant vector $\boldsymbol{b}$. Output. Solution vector $\boldsymbol{x}$. One-dimensional array of size n.
EPSZ	Input. Tolerance for relative zero test of pivots (≥0.0). If EPSZ=0.0, a standard value is used. See "Notes".
ISW	Input. Control information When l(≥1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows: ISW = 1 ... The first system is solved ISW = 2 ... The 2nd to l-th systems are solved. However, only parameter B is specified for the new constant vector $\boldsymbol{b}$ of the systems, with the rest unchanged. See "Notes".
ICON	Output. Condition code. See Table LSTX-1.

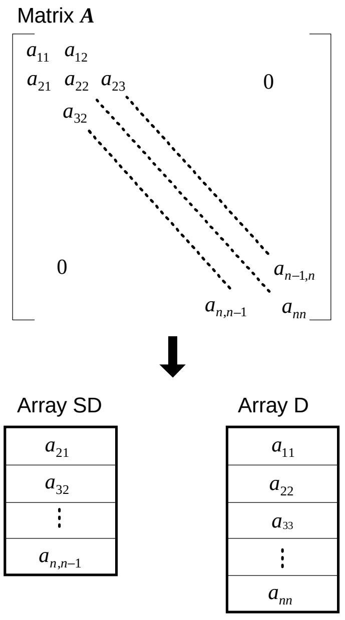


Fig. LSTX-1 Storing method for each element of matrix A into arrays SD and D

Table LSTX-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	A negative pivot occurred. The coefficient matrix is not positive-definite.	Continued
20000	The zero pivot occurred. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	N<1, EPSZ<0.0 or ISW ≠ 1, 2	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL
  FORTRAN basic function ... ABS

- Notes
  If EPSZ is set to 10^{-s}, this value has the following meaning; while performing the LDL^T-decomposition using the modified Cholesky method, if the loss of over s significant digits occurs for the pivot value, the LDL^T-decomposition is discontinued with ICON=20000 and the pivot is regarded as zero. Let u be the unit round off, then the standard value of EPSZ is $16 \cdot u$.

  If the processing is to proceed even at a low pivot value, EPSX has to be given the minimum value but the result is not always guaranteed.

  When successively solving systems of linear equations with the identical coefficient matrix, ISW should be set to 2 after solving the first system. With ISW=2, calculation time is reduced since the LU-decomposition of the matrix $\boldsymbol{A}$ is bypassed.

  The determinant of matrix $\boldsymbol{A}$ can be obtained by multiplying n array elements, D(i), i=1, ...,n.

  When the negative pivot occurs in the decomposition, the calculation error may possibly be large since no pivoting is performed in this subroutine.

This subroutine make use of the characteristics of a matrix when repetitive computations are carried out. As a result this subroutine has a faster processing speed compared to the normal modified Cholesky method although the number of computations are the same.

- Example

  This example solves *l* systems of linear equations in n unknown with an identical coefficient matrix, $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION D(100),SD(99),B(100)
      CHARACTER*4 IA,IB,IX
      DATA IA,IB,IX/'A   ','B   ','X   '/
      READ(5,500) N,L
      NM1=N-1
      READ(5,510) (D(I),I=1,N),(SD(I),
     *            I=1,NM1)
      WRITE(6,600) N
      CALL PTM(IA,1,SD,D,SD,N)
      ISW=1
      M=1
   10 READ(5,510) (B(I),I=1,N)
      CALL PGM(IB,1,B,N,N,1)
      CALL LSTX(D,SD,N,B,0.0,ISW,ICON)
      IF(ICON.EQ.0) GO TO 20
      WRITE(6,610) ICON
      STOP
   20 CALL PGM(IX,1,B,N,N,1)
      IF(M.EQ.L) GO TO 30
      M=M+1
      ISW=2
      GO TO 10
   30 WRITE(6,620)
      STOP
  500 FORMAT(2I3)
  510 FORMAT(4E15.7)
  600 FORMAT('1',5X,
     *  'LINEAR EQUATIONS AX=B(TRIDIAGONAL)'
     *  /6X,'(POSITIVE,SYMMETRIC)'
     *  /6X,'ORDER=',I5)
  610 FORMAT(' ',5X,'ICON OF LSTX=',I6)
  620 FORMAT(' ',5X,'* NORMA END *')
      END
```

Subroutines PTM and PGM used in this example print out a real tridiagonal matrix and a real general matrix, respectively. The descriptions of those programs are given in the examples for subroutines LTX and MGSM.

Method

A linear equations with a positive-definite symmetric tridiagonal matrix,

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

is solved by using the modified Cholesky's method.

Since matrix $\boldsymbol{A}$ is a positive-definite symmetric tridiagonal matrix, the matrix can always be decomposed into the LDL$^{\mathrm{T}}$ form by using the modified Cholesky method as shown in Eq. (4.2).

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}}$$

where $\boldsymbol{L}$ is a unit lower band matrix with band width 1 and $\boldsymbol{D}$ is a positive definite diagonal matrix.

Therefore, solving Eq. (4.1) results is solving Eqs. (4.3) and (4.4),

$$\boldsymbol{Ly} = \boldsymbol{b} \tag{4.3}$$

$$\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{y} \tag{4.4}$$

Eqs. (4.3) and (4.4) can be readily solved by using forward and backward substitutions.

- Modified Cholesky method

  If matrix $\boldsymbol{A}$ is positive-definite, the LDL$^{\mathrm{T}}$-decomposition as shown in (4.2) is always possible. The decomposition using the modified Cholesky method is given by

$$l_{ij}d_j = a_{ij} - \sum_{k=1}^{j-1} l_{ik}d_k l_{jk}, \quad j = 1,...,i-1 \tag{4.5}$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}d_k l_{ik}, \quad i = 1,...,n \tag{4.6}$$

where $\boldsymbol{A} = (a_{ij})$, $\boldsymbol{L} = (l_{ij})$ and $\boldsymbol{D} = \mathrm{diag}(d_i)$. Furthermore, since matrix $\boldsymbol{A}$ is a tridiagonal matrix,
Eqs. (4.5) and (4.6) can be rewritten into

$$l_{i,i-1}d_{i-1} = a_{i,i-1} \tag{4.7}$$

$$d_i = a_{ii} - l_{i,i-1}d_{i-1}l_{i,i-1} \tag{4.8}$$

- Procedure performed in this subroutine

  This subroutine obtains the solution by considering the characteristics of the matrix.

  – LDL$^{\mathrm{T}}$-decomposition

  Normally, when matrix $\boldsymbol{A}$ is LDL$^{\mathrm{T}}$-decomposed using the modified Cholesky method, elements $l_{i,i-1}$ and $d_i\,(i = 1,...,n)$ are successively obtained from Eqs. (4.7) and (4.8). However, this subroutine considers the characteristic of a symmetric tridiagonal matrix, and obtains the elements of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$ recursively, starting with the upper left hand elements together with the lower right hand elements, ending with the middle elements. This fact reduces the number of iterations.

  Elements $l_{i,i-1}, d_i, l_{n-i+1,n-i+2}$ and $d_{n-i+1}\,(i = 1,...,[(n+1)/2])$ are successively obtained by

$$l_{i,i-1}d_{i-1} = a_{i,i-1} \tag{4.9}$$

$$d_i = a_{ii} - l_{i,i-1}d_{i-1}l_{ij-1} \tag{4.10}$$

$$l_{n-i+1,n-i+2}d_{n-i+2} = a_{n-i+1,n-i+2} \tag{4.11}$$

$$d_{n-i+1} = a_{n-i+1,n-i+1} - l_{n-i+2,n-i+1} \quad d_{n-i+2} l_{n-i+2,n-i+1} \qquad (4.12)$$

– Solving $\boldsymbol{Ly} = \boldsymbol{b}$ (forward and backward substitutions)
Normally, the solution is obtained successively by

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k, \quad i = 1,...,n \qquad (4.13)$$

where $\boldsymbol{y}^{\mathrm{T}} = (y_1,...,y_n)$, and $\boldsymbol{b}^{\mathrm{T}} = (b_1,...,b_n)$. This subroutine successively obtains the solution by Eqs. (4.14) and (4.15).

$$y_i = b_i - l_{i,i-1} y_{i-1} \qquad (4.14)$$

$$y_{n-i+1} = b_{n-i+1} - l_{n-i+1,n-i+2}\, y_{n-i+2} \qquad (4.15)$$

$$i = 1,...,[(n+1)/2]$$

– Solving $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}$=$\boldsymbol{D}^{-1}\boldsymbol{y}$ (forward and backward substitutions)
Normally, the solution is obtained successively by

$$x_i = y_i d^{-1} - \sum_{k=i+1}^{n} l_{ki} x_k, \quad i = n,...,1 \qquad (4.16)$$

where, $\boldsymbol{x}^{\mathrm{T}} = (x_1,...,x_n)$, and $\boldsymbol{D}^{-1} = \mathrm{diag}\ (d_i^{-1})$. This subroutine successively obtains the solution by Eqs. (4.17), (4.18) and (4.19).

$$x_{[(n+1)/2]} = y_{[(n+1)/2]}\, d^{-1}{}_{[(n+1)/2]} \qquad (4.17)$$

$$x_i = y_i d_i^{-1} - l_{i+1,i}\, x_{i+1} \qquad (4.18)$$

$$x_{n-i+1} = y_{n-i+1} d_{n-i+1}^{-1} - l_{n-i+1,n-i}\, x_{n-i}$$

$$i = [(n+1)/2]-1,\ \ldots,\ 1 \qquad (4.19)$$

A22-51-0101 LSX, DLSX

A system of linear equations with a positive-definite symmetric matrix (Modified Cholesky's method)
CALL LSX (A, N, B, EPSZ, ISW, ICON)

Function

This subroutine solves a system of linear equations (1.1) using the modified (square root free) Cholesky's method.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

$\boldsymbol{A}$ is an $n \times n$ positive-definite symmetric matrix, $\boldsymbol{b}$ is an n -dimentional real constant vector, and $\boldsymbol{x}$ is the n-dimentional solution vector. $n \geq 1$.

Parameters

A ..... Input. Coefficient matrix $\boldsymbol{A}$.
The contents of $\boldsymbol{A}$ are overridden after operation. $\boldsymbol{A}$ is stored in a one-dimensional array of size $n(n+1)/2$ in the compressed mode for symmetric matrices.

N ..... Input. Order n of the coefficient matrix $\boldsymbol{A}$.

B ..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
B is a one-dimensional array of size n.

EPSZ ..... Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$(≥0.0). When EPSZ is 0.0, a standard value is used. (See Notes.)

ISW ..... Input. Control information
When l(≥1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows:
ISW=1 ... The first system is solved.
ISW=2 ... The 2nd to l-th systems are solved. However, only parameter B is specified for each constant vector $\boldsymbol{b}$ of the systems, with the unchanged (See Note.)

ICON ..... Output. Condition code.
See Table LSX-1.

Table LSX-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The negative pivot occured. The coefficient matrix is not positive-definite.	Continued
20000	The relatively zero pivot occurred. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	N<1, EPSX<0.0 or ISW ≠ 1, 2	Bypassed

Comments on use

- Subprograms used
  SSL II ... SLDL, LDLX, AMACH, and MGSSL
  FORTRAN basic function ... ABS
- Notes
  The solution obtained with this subroutine can be refined in accuracy by successively calling subroutine LSXR.

  If EPSZ is set to 10^{-s}, this value has the following meaning: while performing the $\mathrm{LDL^T}$ decomposition by modified Cholesky's method, if the loss of over s significant digits occurred for the pivot, the $\mathrm{LDL^T}$ decomposition should be discontinued with ICON=20000 regarding the pivot to be relatively zero.

  Let u be the unit round off, then the standard value of EPSZ is $16u$. If the processing is to proceed even at a low pivot value, EPSZ has to be given the minimum value but the result is not always guaranteed.

  When successively solving systems of linear equations with the identical coefficient matrix, after solving the first system, ISW. should be set to 2. With ISW=2, calculation time is reduced since the process in which the coefficient matrix A is $\mathrm{LDL^T}$ decomposed is bypassed.

  If the negative pivot occurs in the decomposition, the coefficient matrix is not a positive-definite. In this subroutine the condition code is set accordingly (ICON=10000) and processing is continued. However, it should be noted that large calculation error may occur since the pivoting is not performed.

  The determinant of the coefficient matrix can be obtained by multiplying the n diagonal elements (the diagonal elements of $\boldsymbol{D}^{-1}$) of the array A after the subroutine has been executed and then by determining the inverse number. Note that array A is in the compressed mode for symmetric matrices.

  When for a positive definite symmetric band matrix, subroutine LSBX processes faster than this subroutine because the operation for the element out of the band is omitted.
- Example
  l systems of linear equations in n unknown with the identical coefficient matrix are solved. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100)
      READ(5,500) N
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N
      READ(5,500) L
      M=1
      ISW=1
      EPSZ=1.0E-6
   10 READ(5,510) (B(I),I=1,N)
      CALL LSX(A,N,B,EPSZ,ISW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,620) (B(I),I=1,N)
      IF(L.EQ.M) GOTO 20
      M=M+1
      ISW=2
      GOTO 10
   20 DET=A(1)
      L=1
      DO 30 I=2,N
      L=L+I
      DET=DET*A(L)
   30 CONTINUE
      DET=1.0/DET
      WRITE(6,630) DET
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'/10X,'ORDER=',I5)
  610 FORMAT('0',10X,'ICON=',I5)
  620 FORMAT(11X,'SOLUTION VECTOR'
     */(15X,5E16.8))
  630 FORMAT('0',10X,
     *'DETERMINANT OF COEFFICIENT MATRIX='
     *,E16.8)
      END
```

Method

A system of linear equations (4.1) which has a real positive-definite symmetric coefficient matrix is solved using the following procedure.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

- LDL$^{\mathrm{T}}$ decomposition of coefficient matrix $\boldsymbol{A}$ (Modified Cholesky's method)
  The coefficient matrix A is LDL$^{\mathrm{T}}$ decomposed into the form (4.2).

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \tag{4.2}$$

where $\boldsymbol{L}$ is a unit lower triangular matric and $\boldsymbol{D}$ is a diagonal matrix. Subroutine SLDL is used for this operation.

- Solving $\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{b}$ (Forward and backward substitution)
  A system of linear equations

$$\boldsymbol{LDL}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{b} \tag{4.3}$$

is solved using subroutine LDLX.
For more information, see Reference [2].

A22-51-0401 LSXR, DLSXR

Iterative refinement of the solution to a system of linear equations with a positive-definite symmetric matrix
CALL LSXR (X, A, N, FA, B, VW, ICON)

Function

When an approximate solution $\tilde{x}$ is given to linear equations with an $n \times n$ positive-definite symmetric matrix $\boldsymbol{A}$ such as

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

this subroutne refines the approximate solution by the method of iterative modification, where $\boldsymbol{b}$ is an n-dimensional real constant vector and $\boldsymbol{x}$ is an n-dimensional solution vector.

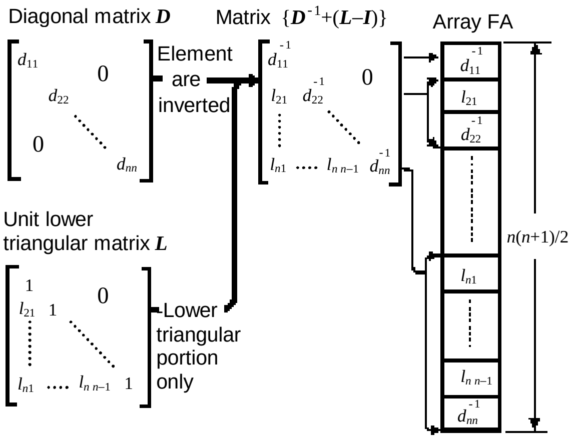
Prior to calling this subroutine, the coefficient matrix $\boldsymbol{A}$ must be LDL$^{\mathrm{T}}$ decomposed as shown in Eq. (1.2),

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \qquad (1.2)$$

where $\boldsymbol{L}$ and $\boldsymbol{D}$ are an $n \times n$ unit lower triangular matrix and a diagonal matrix respectively, also $n \geq 1$.

Parameters

X	Input. Approximate solution vector $\boldsymbol{x}$ Output. Refined solution vector $\boldsymbol{x}$ One-dimensional array of size n.
A	Input. Coefficient matrix $\boldsymbol{A}$. Matrix $\boldsymbol{A}$ is stored in one-dimensional array of size $n(n+1)/2$ in the compressed mode for a symmetric matrix.
N	Input. Order n of matrix $\boldsymbol{A}$. (See Notes.)
FA	Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$. Refer to Fig. LSXR-1. One-dimensional array of size $n(n+1)/2$ to contain symmetric band matrices in the compressed mode.
B	Input. Constant vector $\boldsymbol{b}$. One-dimensional array of size n.
VW	Work area. One-dimensional array of size n.
ICON	Output. Condition code. See table LSXR-1.


Diagonal matrix $\boldsymbol{D}$
Matrix $\{\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})\}$
Array FA
Element are inverted
Unit lower triangular matrix $\boldsymbol{L}$
Lower triangular portion only
$n(n+1)/2$

Note: The diagonal and lower triangular sections of the matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}\text{-}\boldsymbol{I})$ are contained in the one-dimensional array FA in the compressed mode for symmetric matrices.

Fig. LSXR-1 Storage of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$

Table LSXR-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The coefficient matrix was not positive-definite.	continued
25000	The convergence condition was not met because of a very ill-conditioned coefficient matrix.	Discontinued (Refer to " method" for convergence condition.)
30000	N=0	By passed

Comments on use

- Subprograms used
  SSL II ..... LDLX, MSV, AMACH, MGSSL
  FORTRAN basic function ..... ABS

- Notes

This subroutine iteratively improves the approximate solution $\boldsymbol{x}$ obtained by subroutine LSX to get solution $\boldsymbol{x}$ with refined precision. Therefore, prior to calling this subroutine, $\boldsymbol{x}$ must be obtained by subroutine LSX and the results must be input as the parameters X and FA to be used for this subroutine. In addition, the coefficient matrix $\boldsymbol{A}$ and constant vector $\boldsymbol{b}$ which are required for this subroutine must also be prepared and stored separately before calling subroutine LSX. Refer to the example for details.

If N= $-n$ is specified, an estimated accuracy (relative error) for the approximate solution $\boldsymbol{x}$ given by the subroutine LSX can be obtained. When specified, this subroutine calculates the relative error and outputs it to work area VW(1) without performing the iterative refinement of precision. See "Method" for estimation of accuracy.

- Example

An approximate solution $\boldsymbol{x}$ for a system of linear equations in n unknowns is obtained by subroutine LSX, then it is iteratively refined by this subroutine. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),FA(5050),X(100),
     *B(100),VW(100)
   10 READ(5,500) N
```

```
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (B(I),I=1,N)
      DO 20 I=1,N
      X(I)=B(I)
   20 CONTINUE
      DO 30 I=1,NT
   30 FA(I)=A(I)
      EPSZ=0.0E0
      ISW=1
      CALL LSX(FA,N,X,EPSZ,ISW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL LSXR(X,A,N,FA,B,VW,ICON)
      WRITE(6,630) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,640) (I,X(I),I=1,N)
      DET=FA(1)
      L=1
      DO 40 I=2,N
      L=L+1
   40 DET=DET*FA(L)
      DET=1.0/DET
      WRITE(6,650) DET
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  620 FORMAT('0',10X,'LSX  ICON=',I5)
  630 FORMAT('0',10X,'LSXR ICON=',I5)
  640 FORMAT('0',10X,'SOLUTION VECTOR'
     */(10X,4('(',I3,')',E17.8)))
  650 FORMAT(///10X,
     *'DETERMINANT OF COEFFICIENT MATRIX=',
     *E17.8)
      END
```

Method

Given an approximate solution, $\boldsymbol{x}$ to the linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

the solution is iteratively refined as follows:

- Principle of iterative refinement
  The iterative refinement is a method to obtain a successively improved approximate solution $\boldsymbol{x}^{(s+1)}$ to the linear equations (4.1) through use of the following equations starting with $\boldsymbol{x}^{(1)} = \tilde{\boldsymbol{x}}$

$$\boldsymbol{r}^{(s)} = \boldsymbol{b} - \boldsymbol{Ax}^{(s)} \tag{4.2}$$

$$\boldsymbol{Ad}^{(s)} = \boldsymbol{r}^{(s)} \tag{4.3}$$

$$\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \boldsymbol{d}^{(s)} \tag{4.4}$$

$s = 1, 2, \ldots$

where $\boldsymbol{x}^{(s)}$ is the s-th approximate solution to equation (4.1). If Eq. (4.2) is accurately computed, a refined solution of the approximate solution $\boldsymbol{x}^{(1)}$ is numerically obtained. If, however, the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable, no improved solution is obtained. (See Section 3.4 "Iterative refinement of a solution".)

- Procedure performed in this subroutine
  Suppose that the first approximate solution $\boldsymbol{x}^{(1)}$ has already been obtained by the subroutine repeats the following steps:
  - The residual $\boldsymbol{r}^{(s)}$ is computed by using Eq. (4.2) This is done by calling the subroutine MSV.
  - The correction $\boldsymbol{d}^{(s)}$ is obtained next by using Eq. (4.3). by calling subroutine LDLX.
  - Finally, the modified approximate solution $\boldsymbol{x}^{(s+1)}$ is obtained by using Eq. (4.4).

The convergence of the iteration is tested as follows: Considering u as a unit round off, the iteration refinement is assumed to converge if, at the s-th iteration step, the following relationship is satisfied.

$$\left\|\boldsymbol{d}^{(s)}\right\|_\infty / \left\|\boldsymbol{x}^{(s+1)}\right\|_\infty < 2 \cdot u \tag{4.5}$$

The obtained $\boldsymbol{x}^{(s+1)}$ is then taken as the final solution. However, if the relationship,

$$\frac{\left\|d^{(s)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s+1)}\right\|_\infty} > \frac{1}{2} \cdot \frac{\left\|d^{(s-1)}\right\|_\infty}{\left\|\boldsymbol{x}^{(s)}\right\|_\infty}$$

results, this indicates that the condition of the coefficient matrix $\boldsymbol{A}$ is not suitable. The iteration refinement is assumed not to converge, and consequently the processing is terminated with ICON = 25000.

- Accuracy estimation for approximate solution
  Suppose the error for the approximate solution $\boldsymbol{x}^{(1)}$ is $\boldsymbol{e}^{(1)}\left(= \boldsymbol{x}^{(1)} - \boldsymbol{x}\right)$, its relative error is represented by $\left\|\boldsymbol{e}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$ If this iteration method converges, $\boldsymbol{e}^{(1)}$ is assumed to be almost equal to $\boldsymbol{d}^{(1)}$. The relative error for the approximate solution is therefore estimated by $\left\|\boldsymbol{d}^{(1)}\right\|_\infty / \left\|\boldsymbol{x}^{(1)}\right\|_\infty$ (See Section 3.4 "Accuracy estimation for approximate solution".)
  For further details, see References [1], [3], and [5].

A52-11-0501 LTX, DLTX

A system of linear equations with a real tridiagonal matrix (Gaussian elimination method)
CALL LTX (SBD, D, SPD, N, B, EPSZ, ISW, IS, IP, VW, ICON)

Function

A system of linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

is solved by using the Gaussian elimination method, where $\boldsymbol{A}$ is an $n \times n$ real tridiagonal matrix, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. Here $n \geq 1$.

Parameters

SBD ..... Input. Lower sub-diagonal portion of the coefficient matrix $\boldsymbol{A}$.
The contents are destroyed after computation.
Refer to Fig. LTX-1. One-dimensional array of size n–1.

D ..... Input. Diagonal portion of the coefficient matrix $\boldsymbol{A}$. The contents are destroyed after computation.
See Fig. LTX-1. One-dimensional array of size n.

SPD ..... Input. Upper sub-diagonal portion of the coefficient matrix $\boldsymbol{A}$.
The contents are destroyed after computation.
See Fig. LTX-1.
One-dimensional array of size n -1.

N ..... Input. Order n of the coefficient matrix $\boldsymbol{A}$

B ..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

EPSZ ..... Input. Tolerance for relative zero test of pivots (≥0.0)
If EPSZ = 0.0, a standard value is used. (See “Notes”.)

ISW ..... Input. Control information.
When l(≥1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows:
ISW=1 ... The first system is solved
ISW=2 ... The 2nd to l-th systems are solved. However, only parameter B is specified for the new constant vector $\boldsymbol{b}$ of the system, with the rest unchanged.
(See “Notes”.)

IS ..... Output. Information used to obtain a determinant of matrix $\boldsymbol{A}$.
(See “Notes”.)

IP ..... Work area. One-dimensional array of size n.

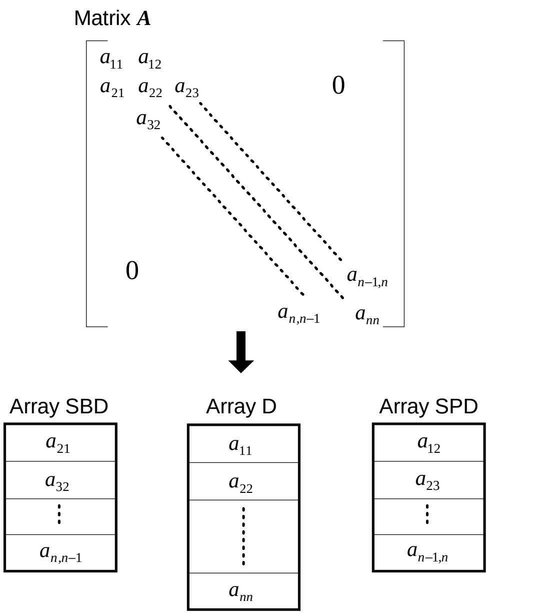


Fig. LTX-1 Storing method for each element of matrix $\boldsymbol{A}$ into arrays SBD, D and SPD

VW ..... Work area. One-dimensional array of size n.

ICON ..... Output. Condition code. Refer to Table LTX-1.

Table LTX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The negative pivot occurred. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	N<1, or EPSZ<0.0	By passed

Comments on use

- Subprograms used
SSL II ... AMACH, MGSSL
FORTRAN basic functions ... AMAX1, ABS
- Notes
This subroutine makes decisions on the relative zero of pivots by using a relational equation containing the value of EPSZ. For details, see “Method”.
Let u be the unit round off. The standard value of EPSZ is then $16 \cdot u$.
If the processing is to proceed even at a low pivot value, EPSZ has to be given the minimum value but the result is not always guaranteed.
When successively solving systems of linear equations with the identical coefficient matrix, after solving the first system, ISW should be set to 2.

With ISW=2, the calculation time is reduced since the process in which the coefficient matrix ***A*** is LU-decomposed is bypassed. The value of IS the same as when ISW=1.

The determinant of the matrix ***A*** is obtained by multiplying the *n* elements, D(*i*), *i*=1, ..., *n* by the value of IS.

- Example

L systems of linear equations with an identical coefficient matrix of order *n* are solved. *n*≤100.

```
C      **EXAMPLE**
       DIMENSION SBD(99),D(100),SPD(99),
      *B(100),IP(100),VW(100)
       CHARACTER*4 NT1(6),NT2(4),NT3(4)
       DATA NT1/'CO  ','EF  ','FI  ','CI  ',
      *         'EN  ','T   '/,
      *     NT2/'CO  ','NS  ','TA  ','NT  '/,
      *     NT3/'SO  ','LU  ','TI  ','ON  '/
       READ(5,500) N,L
       NM1=N-1
       READ(5,510) (SBD(I),I=1,NM1),
      *   (D(I),I=1,N),(SPD(I),I=1,NM1)
       WRITE(6,600) N
       CALL PTM(NT1,6,SBD,D,SPD,N)
       ISW=1
       M=1
    10 READ(5,510) (B(I),I=1,N)
       CALL PGM(NT2,4,B,N,N,1)
       CALL LTX(SBD,D,SPD,N,B,0.0,ISW,
      *  IS,IP,VW,ICON)
       WRITE(6,610)ICON
       IF(ICON.NE.0)STOP
       CALL PGM(NT3,4,B,N,N,1)
       IF(M.EQ.L) GO TO 20
       M=M+1
       ISW=2
       GO TO 10
    20 WRITE(6,620)
       STOP
   500 FORMAT(3I2)
   510 FORMAT(4E15.7)
   600 FORMAT('1',5X,
      *  'LINEAR EQUATIONS(TRIDIAGONAL)'
      *  /6X,'ORDER=',I5)
   610 FORMAT(' ',5X,'ICON OF LTX=',I5)
   620 FORMAT(' ',5X,'** NORMAL END')
       END

       SUBROUTINE PTM(ICOM,L,SBD,D,SPD,N)
       DIMENSION SBD(1),D(N),SPD(1)
       CHARACTER*4 ICOM(L)
       WRITE(6,600) (ICOM(I),I=1,L)
       DO 30 I=1,N
       IF(I.NE.1) GO TO 10
       IC=1
       WRITE(6,610) I,IC,D(I),SPD(I)
       GO TO 30
    10 IC=I-1
       IF(I.NE.N) GO TO 20
       WRITE(6,610) I,IC,SBD(IC),D(I)
       GO TO 30
    20 WRITE(6,610)I,IC,SBD(IC),D(I),SPD(I)
    30 CONTINUE
       RETURN
   600 FORMAT(/10X,35A2)
   610 FORMAT(/5X,2(1X,I3),3(3X,E14.7))
       END
```

The subroutines PTM and PGM are used only to print out a real tridiagonal matrix and a real general matrix. The subroutine PGM is described in the example for the subroutine MGSM.

Method

A system of linear equations with a real tridiagonal matrix ***A***

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.1}$$

is solved by using the Gaussian elimination method based on partial pivoting. ***A*** matrix can be normally decomposed into the product of a unit lower triangular matrix ***L*** and an upper triangular matrix ***U*** with partial pivoting.

$$\boldsymbol{A} = \boldsymbol{LU} \tag{4.2}$$

Consequently, solving Eq. (4.1) is equal to solving

$$\boldsymbol{Ly} = \boldsymbol{b} \tag{4.3}$$

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{4.4}$$

Since both ***L*** and ***U*** are triangular matrices, Eqs. (4.3) and (4.4) can be readily solved by using back-ward and forward substitutions.

- Guassian elimination method

Let $\boldsymbol{A}^{(k)}$ represent the matrix at the *k*-th step $(k = 1,...,n-1)$ of the Gaussian elimination method, where $\boldsymbol{A}^{(1)} = \boldsymbol{A}$. The *k*-th step of elimination process is represented by

$$\boldsymbol{A}^{(k+1)} = \boldsymbol{M}_k \boldsymbol{P}_k \boldsymbol{A}^{(k)} \tag{4.5}$$

where $\boldsymbol{P}_k$ is a permutation matrix to select a pivot row of the *k*-th column of the matrix $\boldsymbol{A}^{(k)}$. The $\boldsymbol{M}_k$ is a matrix to eliminate the element below the diagonal in the *k*-th column of the permuted matrix, and is given by

$$\boldsymbol{M}_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & 0 & \\ & & 1 & & \\ & & -m_{k+1,k} & 1 & \\ & 0 & & \ddots & \\ & & & & 1 \end{bmatrix} \tag{4.6}$$

$m_{k+1,k} = a_{k+1,k}^{(k)}/a_{kk}^{(k)}, \quad \boldsymbol{A}^{(k)} = \left(a_{ij}^{(k)}\right)$

After the elimination process is finished, by setting

$$\boldsymbol{U} = \boldsymbol{A}^{(n)} = \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1} \cdots \boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{A}^{(1)} \tag{4.7}$$

$$\boldsymbol{L} = \left(\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1} \cdots \boldsymbol{M}_1\boldsymbol{P}_1\right)^{-1} \tag{4.8}$$

matrix $\boldsymbol{A}$ is rewritten as

$\boldsymbol{A} = \boldsymbol{LU}$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are a unit lower triangular matrix and an upper triangular matrix, respectively.

- Procedure performed in this subroutine
  [Forward substitution]
  Eq. (4.9) results from Eq. (4.3) based on Eq. (4.8)

$$\boldsymbol{y} = \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1} \cdots \boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{b} \tag{4.9}$$

Eq. (4.9) is repeatedly calculated as follows:

$$\begin{aligned} &\boldsymbol{y}^{(1)} = \boldsymbol{b} \\ &\boldsymbol{y}^{(2)} = \boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{y}^{(1)} \\ &\quad\vdots \\ &\boldsymbol{y}^{(n)} = \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\boldsymbol{y}^{(n-1)} \\ &\boldsymbol{y} = \boldsymbol{y}^{(n)} \end{aligned}$$

This subroutine, at the k-th ($k = 1,...,n-1$) elimination step, obtains each element of the k-th column of matrix $\boldsymbol{L}$ and k-th row of matrix $\boldsymbol{U}$ by using Eqs. (4.10) and (4.11).

$$m_{k+1,k} = a_{k+1,k}^{(k)}/a_{kk}^{(k)} \tag{4.10}$$

$$u_{kj} = a_{kj}^{(k)}, \quad j = k, k+1, k+2 \tag{4.11}$$

The elements of the (k+1)-th row of matrix $\boldsymbol{A}^{(k+1)}$ are given by Eq. (4.12). The other elements of matrix $\boldsymbol{A}^{(k+1)}$ to which the (k+1)-th elimination step is applied are equal to the corresponding element of matrix $\boldsymbol{A}^{(k)}$.

$$a_{k+1,\ \ j}^{k+1} = a_{k+1}^{(k)} - a_{k,j}^{(k)} m_{k+1,k}^{(k)}$$
$$, j=k+1,\ k+2 \tag{4.12}$$

The pivot element $a_{kk}^{(k)}$ is Eq. (4.10) is selected as follows to minimize the computational errors before this elimination process. $\boldsymbol{a}_{\boldsymbol{lk}}^{(\boldsymbol{k})}$ that reflects scaling factor $\boldsymbol{V}_l$ as $V_l \cdot \left|a_{lk}^{(k)}\right| = \max\left(V_l \cdot \left|a_{lk}^{(k)}\right|\right)$ is chosen as the pivot element, where$\boldsymbol{V}_l$ is an inverse of the maximum absolute value element in the l -th row of coefficient matrix $\boldsymbol{A}$.

If the selected pivot element $a_{lk}^{(k)}$ satisfies

$\left|a_{lk}^{(k)}\right| < \max\left(\left|a_{ij}\right|\right) \cdot u$

where $\boldsymbol{A} = \left(a_{ij}\right)$, u unit round off then matrix $\boldsymbol{A}$ is assumed to be numerically singular and the processing is terminated with ICON=20000.

[Backward substitution]
Eq. (4.4) is obtained iteratively by

$$x_k = \left(y_k - \sum_{j=k+1}^{k+2} u_{kj}x_j\right)/u_{kk}, \quad k = n,...,1 \tag{4.13}$$

where $\boldsymbol{U} = \left(u_{ij}\right)$, and $\boldsymbol{x} = \left(x_1, x_2, ..., x_n\right)^{\mathrm{T}}$

A22-11-0602 LUIV, DLUIV

The inverse of a real general matrix decomposed into the factors ***L*** and ***U***
CALL LUIV (FA, K, N, IP, ICON)

Function

This subroutine computes the inverse $\boldsymbol{A}^{-1}$ of an $n \times n$ real general matrix ***A*** given in decomposed form $\boldsymbol{PA} = \boldsymbol{LU}$

$$\boldsymbol{A}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$$

L and ***U*** are respectively the $n \times n$ lower triangular and unit upper triangular matrices, and ***P*** is the permutation matrix which performs the row exchanges in partial pivoting for LU decomposition. $n \geq 1$.

Parameters

FA	Input. Matrix ***L*** and matrix ***U***. Output. Inverse $\boldsymbol{A}^{-1}$. FA is a two-dimensional array, FA (K, N). Refer to Fig. LUIV-1.
K	Input. Adjustable dimension of array FA (≥N).
N	Input. Order n of the matrices ***L*** and ***U***.
IP	Input. Transposition vector which indicates the history of row exchanges in partial pivoting. One-dimensional array of size *n*.
ICON	Output. Condition code. See Table LUIV-1.

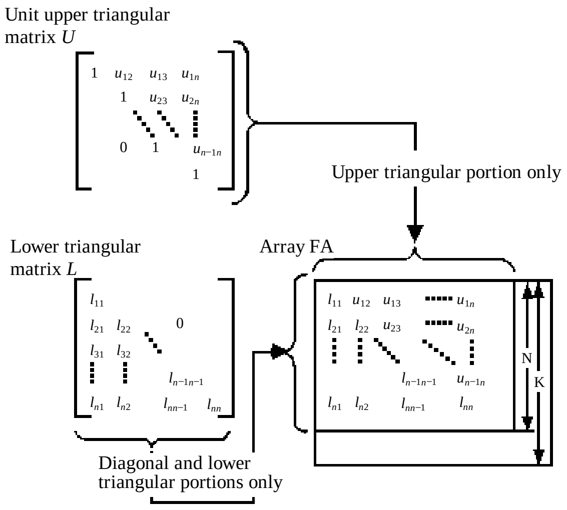

Unit upper triangular matrix U
Upper triangular portion only
Lower triangular matrix L
Array FA
N
K
Diagonal and lower triangular portions only

Fig. LUIV-1 Storage of the elements of *L* and *U* in array FA

Table LUIV-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	A real matrix was singular.	Discontinued
30000	K<N or N<1 or there was an error in IP.	Bypassed

Comments on use

- Subprograms used
  SSL II ..... MGSSL
  FORTRAN basic function ..... None

- Notes
  Prior to calling this subroutine, LU-decomposed matrix must be obtained by subroutine ALU and must be input as the parameters FA and IP to be used for this subroutine. The subroutine LAX should be used for solving linear equations. Obtaining the solution by first computing the inverse matrix requires more steps of calculation, so subroutine LUIV should be used only when the inverse matrix is inevitable. The transposition vector corresponds to the permutation matrix ***P*** of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

When performing LU decomposition with partial pivoting. Refer to the notes of the subroutine ALU.

- Example
  The inverse of an $n \times n$ real general matrix is obtained. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),VW(100),IP(100)
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),
     *             I=1,N)
      CALL ALU(A,100,N,0.0,IP,IS,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      CALL LUIV(A,100,N,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,630) ((I,J,A(I,J),I=1,N),
     *             J=1,N)
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT(//11X,'**INPUT MATRIX**'/12X,
     * 'ORDER=',I5/(2X,4('(',I3,',',I3,')',
     * E16.8)))
  610 FORMAT('0',10X,'CONDITION ',
     * 'CODE(ALU)=',I5)
  620 FORMAT('0',10X,'CONDITION ',
     * 'CODE(LUIV)=',I5)
  630 FORMAT('0',10X,'**INVERSE MATRIX**',
     * /(2X,4('(',I3,',',I3,')',E16.8)))
      END
```

Method
This subroutine computes the inverse of an $n \times n$ real general matrix, giving the LU-decomposed matrices ***L***, ***U*** and the permutation matrix ***P*** which indicates row exchanges in partial pivoting.

$$\boldsymbol{PA} = \boldsymbol{LU} \qquad (4.1)$$

then, the inverse of ***A*** can be represented using (4.1) as follows:
The inverse of ***L*** and ***U*** are computed and then the inverse of ***A*** is obtained as (4.2).

$$\boldsymbol{A}^{-1} = \left(\boldsymbol{P}^{-1}\boldsymbol{LU}\right)^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P} \qquad (4.2)$$

L and ***U*** are as shown in Eq. (4.3) for the following explanation.

$$\boldsymbol{L} = (l_{ij}) \text{ , } \boldsymbol{U} = (u_{ij}) \qquad (4.3)$$

- Calculating $\boldsymbol{L}^{-1}$
  Since the inverse $\boldsymbol{L}^{-1}$ of a lower triangular matrix ***L*** is also a lower triangular matrix, if we represent $\boldsymbol{L}^{-1}$ by

$$\boldsymbol{L}^{-1} = \left(\tilde{l}_{ij}\right) \qquad (4.4)$$

then Eq. (4.5) is obtained based on the relation $\boldsymbol{LL}^{-1} = \boldsymbol{I}$.

$$\sum_{k=1}^{n} l_{ik}\tilde{l}_{kj} = \delta_{ij} \text{ ,}$$
$$\delta_{ij} = \begin{cases} 1 \text{ ,} i = j \\ 0 \text{ ,} i \neq j \end{cases} \qquad (4.5)$$

(4.5) is rewritten as

$$\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} + l_{ii}\tilde{l}_{ij} = \delta_{ij}$$

and the elements $\tilde{l}_{ij}$ of the *j*-th column (*j*=1,...,*n*) of the matrix $\boldsymbol{L}^{-1}$ are obtained as follows:

$$\tilde{l}_{ij} = \left(-\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} \right) / l_{ii} \text{ ,} \quad i = j+1,...,n$$
$$\tilde{l}_{jj} = 1/l_{jj} \qquad (4.6)$$
$$\text{where, } l_{ii} \neq 0 (i = j,...,n)$$

- Calculation $\boldsymbol{U}^{-1}$
  Since the inverse $\boldsymbol{U}^{-1}$ of a unit upper triangular matrix ***U*** is also a unit upper triangular matrix, if we represent $\boldsymbol{U}^{-1}$ by

$$\boldsymbol{U}^{-1} = \left(\tilde{u}_{ij}\right) \qquad (4.7)$$

then Eq. (4.8) is obtained based on the relation $\boldsymbol{UU}^{-1} = \boldsymbol{I}$.

$$\sum_{k=1}^{n} u_{ik}\tilde{u}_{kj} = \delta_{ij} \text{ ,}$$
$$\delta_{ij} = \begin{cases} 1 \text{ ,} i = j \\ 0 \text{ ,} i \neq j \end{cases} \qquad (4.8)$$

Since $u_{ii} = 1$, (4.8) can be rewritten

$$\tilde{u}_{ij} + \sum_{k=i+1}^{j} u_{ik}\tilde{u}_{kj} = \delta_{ij}$$

Considering $\tilde{u}_{jj} = 1$, the elements $\tilde{u}_{ij}$ of the *j*-th column (*j* = *n*,...,2) of $\boldsymbol{U}^{-1}$ are obtained as follows:

$$\tilde{u}_{ij} = -u_{ij} - \sum_{k=i+1}^{j-1} u_{ik}\tilde{u}_{kj} \text{ ,} \quad i = j-1,...,1 \qquad (4.9)$$

- Calculating $\boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{P}$
  Let the product of matrices $\boldsymbol{U}^{-1}$ and $\boldsymbol{L}^{-1}$ be ***B***, then its elements b_{ij} are obtained by

$$b_{ij} = \sum_{k=j}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \text{ ,} \quad i = 1,..., j-1$$
$$b_{ij} = \sum_{k=i}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \text{ ,} \quad i = j,...,n$$

Considering $\tilde{u}_{ii}=1$, the element b_{ij} of the *j*-th column (*j*=1,...,*n*) of ***B*** are obtained by

$$b_{ij} = \sum_{k=j}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \text{ ,} \quad i = 1,..., j-1$$
$$\qquad (4.10)$$
$$b_{ij} = \tilde{l}_{ij} + \sum_{k=i+1}^{n} \tilde{u}_{ik}\tilde{l}_{kj} \text{ ,} \quad i = j,...,n$$

Next, matrix ***B*** is multiplied by the permutation matrix to obtain the inverse $\boldsymbol{A}^{-1}$. Actually however, based on the values of the transposition vector ***IP***, the elements of $\boldsymbol{A}^{-1}$ are obtained simply by exchanging the column in the matrix ***B***. The precision of the inner products in (4.6), (4.9)m and (4.10) has been raised to minimize the effect of rounding errors. For more information, see Reference [1].

A22-11-0302 LUX, DLUX

A system of linear equations with a real general matrix decomposed into the factors ***L*** and ***U***
CALL LUX (B, FA, K, N, ISW, IP, ICON)

Function

This subroutine solves a system of linear equations

$$\boldsymbol{LUx} = \boldsymbol{Pb} \qquad (1.1)$$

L and ***U*** are, respectively, the lower triangular and unit upper triangular matrices, ***P*** is a permutation matrix which performs row exchange with partial pivoting for LU decomposition of the coefficient matrix, ***b*** is an *n*-dimentional real constant vector, and ***x*** is an *n*-dimentional solution vector. Instead of equation (1.1), one of the following can be solved.

$$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad (1.2)$$
$$\boldsymbol{Uz} = \boldsymbol{b} \qquad (1.3)$$

Parameters

B	Input. constant vector ***b*** Output. One of solution vector ***x***, ***y*** or ***z*** ***B*** is a one-dimensional array of size *n* .
FA	Input. Matrix ***L*** and matrix ***U***. See Fig. LUX-1. FA is a two-dimensional array, FA (K, N).
K	Input. Adjustable dimension of array FA (≥N)
N	Input. The order *n* of matrices ***L*** and ***U***.
ISW	Input. Control information. ISW=1 ... ***x*** is obtained. ISW=2 ... ***y*** is obtained. ISW=3 ... ***z*** is obtained.
IP	Input. The transposition vector which indicates the history of the row exchange in partial pivoting. IP is a one-dimensional array of size *n* (See Notes of subroutine ALU).
ICON	Output. Condition code. See Table LUX-1.

Table LUX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The coefficient matrix was singular.	Discontinued
30000	K<N, N<, ISW≠1,2,3, or there was an error in IP.	Bypassed

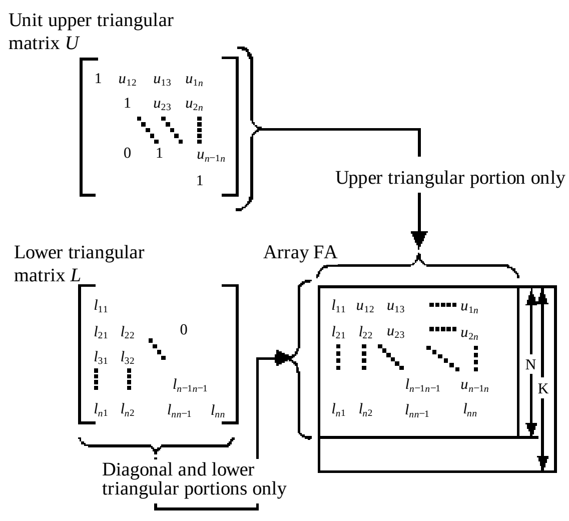

Unit upper triangular matrix U
Upper triangular portion only
Lower triangular matrix L
Array FA
Diagonal and lower triangular portions only
N
K

Fig. LUX-1 Storage of elements of ***L*** and ***U*** in array FA

Comments on use

- Subprograms used]
  SSL II ..... MGSSL
  FORTRAN basic function ..... none

- Notes
  A system of linear equations can be solved by first calling the subroutine ALU to decompose the coefficient matrix into ***L*** and ***U*** and by then calling this subroutine. However, instead of both these subroutines, the subroutine LAX can be called to solve such equations in one step.

- Example
  A system of linear equations is solved by first using subroutine ALU to decompose the $n \times n$ coefficient matrix into ***L*** and ***U***, $n \le 100$.

```
C      **EXAMPLE**
       DIMENSION A(100,100),B(100),
      *          VW(100),IP(100)
    10 READ(5,500) N
       IF(N.EQ.0) STOP
       READ(5,510) ((A(I,J),I=1,N),J=1,N)
       READ(5,510) (B(I),I=1,N)
       WRITE(6,600) N,((I,J,A(I,J),J=1,N),
      *             I=1,N)
       WRITE(6,610) (I,B(I),I=1,N)
       CALL ALU(A,100,N,1.0E-6,IP,IS,VW,ICON)
       WRITE(6,620) ICON
       IF(ICON.GE.20000) GO TO 10
       CALL LUX(B,A,100,N,1,IP,ICON)
       WRITE(6,630) ICON
       IF(ICON.GE.20000) GO TO 10
       WRITE(6,640) (I,B(I),I=1,N)
       GO TO 10
```

```
500 FORMAT(I5)
510 FORMAT(4E15.7)
600 FORMAT(///10X,'**COEFFICIENT MATRIX**'
   * /12X,'ORDER=',I5,/(10X,4('(',I3,',',
   * I3,')',E16.8)))
610 FORMAT('0',10X,'CONSTANT VECTOR'
   * /(10X,5('(',I3,')',E16.8)))
620 FORMAT('0',10X,'CONDITION(ALU)'
   * ,I5)
630 FORMAT('0',10X,'CONDITION(LUX)'
   * ,I5)
640 FORMAT('0',10X,'SOLUTION VECTOR'
   * /(10X,5('(',I3,')',E16.8)))
    END
```

Method
A system of linear equations

$$\boldsymbol{LUx} = \boldsymbol{Pb} \tag{4.1}$$

can be solved by solving following equations

$$\boldsymbol{Ly} = \boldsymbol{Pb} \tag{4.2}$$

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{4.3}$$

- Solving $\boldsymbol{Ly} = \boldsymbol{Pb}$ (forward substitution)
  $\boldsymbol{Ly} = \boldsymbol{Pb}$ can be serially solved using equation (4.4).

$$y_i = \left(b'_i - \sum_{k=1}^{i-1} l_{ik} y_k \right) \Big/ l_{ii}\,, \quad i = 1,...,n \tag{4.4}$$

  where $\boldsymbol{L}=(l_{ij})$, $\boldsymbol{y}^{\mathrm{T}} = (y_1, \dots, y_n)$, $(\boldsymbol{Pb})^{\mathrm{T}}=(b_1, \dots, b'_n)$.

- Solving $\boldsymbol{Ux} = \boldsymbol{y}$ (backward substitution)
  $\boldsymbol{Ux} = \boldsymbol{y}$ can be serially solved using equations (4.5).

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k\,, \quad i = n,...,1 \tag{4.5}$$

  where, $\boldsymbol{U}=(u_{ij})$, $\boldsymbol{x}^{\mathrm{T}}=(x_1, \dots, x_n)$.

Precision of the inner products in (4.4) and (4.5) has been raised to minimize the effect of rounding error. For more information, see References [1], [2], [3], and [4].

A21-13-0101 MAV, DMAV

Multiplication of a real matrix and a real vector.
CALL MAV (A, K, M, N, X Y, ICON)

Function

This subroutine performs multiplication of an $m \times n$ real matrix $\boldsymbol{A}$ and a vector $\boldsymbol{x}$.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

where, $\boldsymbol{x}$ is an n-dimensional vector and $\boldsymbol{y}$ is an m-dimensional vector, m and n≥1.

Parameters

A	Input. Matrix $\boldsymbol{A}$, two-dimensional array, A(K, N).
K	Input. The adjustable dimension of array A, (≥M).
M	Input. The number of rows of matrix $\boldsymbol{A}$.
N	Input. The number of columns of matrix $\boldsymbol{A}$.
X	Input. Vector $\boldsymbol{x}$, one dimensional array of size n.
Y	Output. Multiplication y of matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$, one-dimensional array of size m.
ICON	Output. Condition codes. See Table MAV-1.

Table MAV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M<1, N=0 or K<M	Bypassed

Comments on use

- Subprograms used
  SSL II ..... MGSSL
  FORTRAN basic function ... IABS
- Notes
  This subroutine mainly consists of the computation.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \tag{3.1}$$

but it can be changed to another type of computation,

$$\boldsymbol{y} = \boldsymbol{y}' - \boldsymbol{A}\boldsymbol{x} \tag{3.2}$$

by specifying N$=-n$ and giving an arbitrary vectory $\boldsymbol{y}'$ to the parameter Y.
This method can be used to compute a residual vector of linear equations such as

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \tag{3.3}$$

See the example in “Comments on use” below.

- Example
  This example shows the program that solves a linear equations (3.4) with subroutine LAX and that obtains a residual vector $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ through the solution, when $n \le 100$.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{3.4}$$

```
C     **EXAMPLE**
      DIMENSION A(100,100),X(100),Y(100),
     * VW(100),IP(100),W(100,100)
      READ(5,500) N
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      READ(5,510) (X(I),I=1,N)
      WRITE(6,600) N
      WRITE(6,610) ((I,J,A(I,J),J=1,N),
     * I=1,N)
      WRITE(6,620) (I,X(I),I=1,N)
      EPSZ=1.0E-6
      ISW=1
      DO 10 I=1,N
      Y(I)=X(I)
      DO 10 J=1,N
      W(J,I)=A(J,I)
   10 CONTINUE
      CALL LAX(A,100,N,X,EPSZ,ISW,IS,VW,IP,
     *ICON)
      WRITE(6,630) (I,X(I),I=1,N)
      CALL MAV(W,100,N,-N,X,Y,ICON)
      IF(ICON.NE.0) GOTO 30
      WRITE(6,640) (I,Y(I),I=1,N)
      DN=0.0
      DO 20 I=1,N
      DN=DN+Y(I)*Y(I)
   20 CONTINUE
      DN=SQRT(DN)
      WRITE(6,650) DN
   30 WRITE(6,660) ICON
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1','COEFFICIENT MATRIX'
     * /' ','ORDER=',I5)
  610 FORMAT(/4(' ','(',I3, ',',I3, ')',
     * E17.8))
  620 FORMAT(/' ','CONSTANT VECTOR'
     * /(10X,4('(',I3,')',E17.8)))
  630 FORMAT(/' ','SOLUTION VECTOR'
     * /(10X,4('(',I3,')',E17.8)))
  640 FORMAT(/' ','RESIDUAL VECTOR'
     * /(10X,4('(',I3,')',E17.8)))
  650 FORMAT(/' ','NORM=',E17.8)
  660 FORMAT(/' ','ICON=',I5)
      END
```

Method

This subroutine performs multiplication $\boldsymbol{y} = (y_i)$ of an $m \times n$ real matrix $\boldsymbol{A} = (a_{ij})$ and an n-dimensional vector $\boldsymbol{x} = (x_j)$ through using the equation (4.1).

$$y_i = \sum_{j=1}^{n} a_{ij} x_j , \quad i = 1, ..., m \qquad (4.1)$$

In this subroutine, precision of the sum of products in (4.1) has been raised to minimize the effect of rounding errors.

A51-11-0101 MBV, DMBV

Multiplication of a real band matrix and a real vector.
CALL MBV (A, N, NH1, NH2, X, Y, ICON)

Function

This subroutine performs multiplication of an $n \times n$ band matrix $\boldsymbol{A}$ with lower band width h_1 and upper band width h_2 by a vector $\boldsymbol{x}$

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{1.1}$$

where x and y are both an n-dimensional vectors.
Also, $n > h_1 \geq 0$ and $n > h_2 \geq 0$.

Parameters

A ..... Input. Matrix $\boldsymbol{A}$.
Compressed mode for a band matrix.
One-dimensional array of size $n \cdot \min(h_1+h_2+1, n)$.
N ..... Input. Order n of the matrix $\boldsymbol{A}$.
(See Notes.)
NH1 ..... Input Lower band width h_1.
NH2 ..... Input Upper band width h_2.
X ..... Input. Vector $\boldsymbol{x}$.
One-dimensional array of size n.
Y ..... Output. Vector $\boldsymbol{y}$
One-dimensional array of size n.
ICON ..... Output. Condition code. See the Table MBV-1.

Table MBV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N=0, $\lvert N \rvert \leq$ NH1, $\lvert N \rvert \leq$ NH2 , NH1 < 0 or NH2 < 0	Bypassed

Comments on use

- Subprograms used
  SSLII ... MGSSL
  FORTRAN basic functions ... IABS, MIN0
- Notes
  This subroutine mainly consists of the computation

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{3.1}$$

but it can be changed to another type of computation

$$\boldsymbol{y} = \boldsymbol{y}' - \boldsymbol{Ax}$$

by specifying N= n and giving an arbitary vector $\boldsymbol{y}'$ to the parameter Y.
In practice, this method can be used to compute a residual vector of linear equations (Refer to the example shown below).

- Example
  The linear equations with an $n \times n$ matrix of lower band with h_1 and upper band width h_2.

$$\boldsymbol{Ax} = \boldsymbol{b}$$

is solved by calling the subroutine LBX1, and then the residual vector $\boldsymbol{b} - \boldsymbol{Ax}$ is computed with the resultant. Here $n \leq 100$, $h_1 \leq 20$ and $h_2 \leq 20$.

```
C     **EXAMPLE**
      DIMENSION A(4100),X(100),IP(100),
     *     FL(1980),VW(100),Y(100),W(4100)
      CHARACTER*4 NT1(6),NT2(4),NT3(4),
     *     NT4(4)
      DATA NT1/'CO  ','EF  ','FI  ','CI  ',
     *         'EN  ','T   '/,
     *     NT2/'CO  ','NS  ','TA  ','NT  '/,
     *     NT3/'SO  ','LU  ','TI  ','ON  '/,
     *     NT4/'RE  ','SI  ','DU  ','AL  '/
      READ(5,500) N,NH1,NH2
      WRITE(6,600) N,NH1,NH2
      IF(N.LE.0.OR.NH1.GE.N.OR.NH2.GE.N)
     * STOP
      NT=N*MIN0(N,NH1+NH2+1)
      READ(5,510) (A(I),I=1,NT)
      CALL PBM(NT1,6,A,N,NH1,NH2)
      READ(5,510) (X(I),I=1,N)
      CALL PGM(NT2,4,X,N,N,1)
      DO 10 I=1,NT
   10 W(I)=A(I)
      DO 20 I=1,N
   20 Y(I)=X(I)
      CALL LBX1(A,N,NH1,NH2,X,0.0,1,IS,
     *  FL,VW,IP,ICON)
      IF(ICON.GE.20000) GO TO 30
      CALL PGM(NT3,4,X,N,N,1)
      CALL MBV(A,-N,NH1,NH2,X,Y,ICON)
      IF(ICON.NE.0) GO TO 30
      CALL PGM(NT4,4,Y,N,N,1)
      RN=0.0
      DO 25 I=1,N
   25 RN=RN+Y(I)*Y(I)
      RN=SQRT(RN)
      WRITE(6,610) RN
      STOP
   30 WRITE(6,620) ICON
      STOP
  500 FORMAT(3I5)
  510 FORMAT(10F8.3)
  600 FORMAT('1','BAND EQUATIONS'
     *  /5X,'ORDER=', I5
     *  /5X,'SUB-DIAGONAL,LINES=',I4
     *  /5X,'SUPER-DIAGONAL,LINES=',I4)
  610 FORMAT(' ',4X,'RESIDUAL NORM=',E17.8)
  620 FORMAT(' ',4X,'ICON=',I5)
      END
```

The subroutines PBM and PGM are used in this example only to print out a band matrix and a real general matrix respectively.

The description of the two programs are shown in the example for the subroutines LBX1 and MGSM, respectively.

Method

This subroutine performs multipuliation **y** = (y_i), of an $n \times n$ band matrix **A** = (a_{ij}) with lower band width h_1 and upper band width h_2 by an n-dimentional vector **x** = (x_j) through using the Eq. (4.1).

$$y_i = \sum_{j=1}^{n} a_{ij} x_j , \quad i = 1,...,n \tag{4.1}$$

However, this subroutine recognizes that the matrix is a band matrix and the actual computation is done by

$$y_i = \sum_{j=\max(1,i-h_1)}^{\min(i+h_2,n)} a_{ij} x_j , \quad i = 1,...,n \tag{4.2}$$

This subroutine performs the product sum calculation in Eq. (4.2) with higher precision in order to minimize the effect of rounding errors.

A21-15-0101 MCV, DMCV

Multiplication of a complex matrix and complex vector ***x***
CALL MCV (ZA, K, M, N, ZX, ZV, ICON)

Function

This subroutine performs multiplication of an $m \times n$ complex matrix ***A*** by a complex vector ***x***

$$\boldsymbol{y} = \boldsymbol{Ax} \qquad (1.1)$$

where, ***x*** is an *n*-dimensional complex vector, ***y*** is an *m*-dimensional complex vector and $m,\ n \geq 1$.

Parameters

ZA	Input. Matrix ***A*** A complex two-dimensional array, ZA (K, N)
K	Input. Adjustable dimension of array ZA (≥M)
M	Input. Row number *m* of matrix ***A***
N	Input. Column number *n* of matrix ***A***. (Refer to "Comments on use".)
ZX	Input. Complex vector ***x*** A complex one-dimensional array of size *n*.
ZY	Output. Multiplication ***y*** of matrix ***A*** complex vector ***x*** A complex one-dimensional array of size *m*
ICON	Output. Condition code. Refer to Table MCV-1.

Table MCV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M<1, n=0 or K<M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... IABS

- Notes
  This subroutine mainly consists of the computations,

$$\boldsymbol{y} = \boldsymbol{Ax} \qquad (3.1)$$

but it can be changed to another type of computation,

$$\boldsymbol{y} = \boldsymbol{y}' - \boldsymbol{Ax} \qquad (3.2)$$

by specifying $N = -n$ and giving an arbitrary vector ***y'*** to the parameter ZY.

This method can be used to compute a residual vector of linear equations.

Refer to the example in 'Comments on use' below.

- Example
  In this example, the n -dimensional linear equations with complex coefficients.

$$\boldsymbol{Ax} = \boldsymbol{b}$$

are solved by calling the subroutine LCX, and then the residual vector $\boldsymbol{b} - \boldsymbol{Ax}$ is obtained through the solution. Here $n \leq 50$.

```
C     **EXAMPLE**
      DIMENSION ZA(50,50),ZX(50),ZY(50),
     *  ZVW(50),IP(50),ZW(50,50)
      CHARACTER*4 NT1(6),NT2(4),
     *            NT3(4),NT4(4)
      COMPLEX ZA,ZX,ZY,ZVW,ZW
      DATA NT1/'CO  ','EF  ','FI  ','CI  ',
     *        'EN  ','T   '/,
     *     NT2/'CO  ','NS  ','TA  ','NT  '/,
     *     NT3/'SO  ','LU  ','TI  ','ON  '/,
     *     NT4/'RE  ','SI  ','DU  ','AL  '/
      READ(5,500) N
      IF(N.LE.0) STOP
      READ(5,510) ((ZA(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      CALL PCM(NT1,6,ZA,50,N,N)
      ISW=1
      EPSZ=1.0E-6
      READ(5,510) (ZX(I),I=1,N)
      CALL PCM(NT2,4,ZX,N,N,1)
      DO 10 I=1,N
      ZY(I)=ZX(I)
      DO 10 J=1,N
      ZW(J,I)=ZA(J,I)
   10 CONTINUE
      CALL LCX(ZA,50,N,ZX,EPSZ,ISW,IS,
     *ZVW,IP,ICON)
      IF(ICON.GE.20000) GO TO 30
      CALL PCM(NT3,4,ZX,N,N,1)
      CALL MCV(ZW,50,N,-N,ZX,ZY,ICON)
      IF(ICON.NE.0) GO TO 30
      CALL PCM(NT4,4,ZY,N,N,1)
      RN=0.0
      DO 20 I=1, N
      CR=REAL(ZY(I))
      CI=IMAG(ZY(I))
      RN=RN+CR*CR+CI*CI
   20 CONTINUE
      RN=SQRT(RN)
      WRITE(6,610) RN
      STOP
   30 WRITE(6,620) ICON
      STOP
  500 FORMAT(I5)
  510 FORMAT(5(2F8.3))
  600 FORMAT('1','COMPLEX LINEAR EQUATIONS'
     *  /5X,'ORDER=',I5)
  610 FORMAT(' ',4X,'RESIDUAL NORM=',E17.8)
  620 FORMAT(' ',4X,'ICON=',I5)
      END
      SUBROUTINE PCM(ICOM,L,ZA,K,M,N)
      DIMENSION ZA(K,N)
      CHARACTER*4 ICOM(L)
      COMPLEX ZA
      WRITE(6,600) (ICOM(I),I=1,L)
      DO 10 I=1,M
      WRITE(6,610) I,(J,ZA(I,J),J=1,N)
   10 CONTINUE
      RETURN
  600 FORMAT(' ',35A2)
  610 FORMAT(' ',1X,I3,2(I3,2E17.7)
     */(5X,2(I3,2E17.7)))
      END
```

The subroutine PCM is used in this example only to print out a complex matrix.

Method

Elements of $\boldsymbol{y} = (y_i)$, that is a resultant product of $m \times n$ complex matrix $\boldsymbol{A} = (a_{ij})$ by n-dimensional complex vector $\boldsymbol{x} = (x_i)$, are computed as shown in Eq. (4.1),

$$\boldsymbol{y}_i = \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1,...,m \tag{4.1}$$

This subroutine reduces rounding errors as much as possible by performing the inner product computation, Eq. (4.1), with higher precision.

A22-21-0302 MDMX, DMDMX

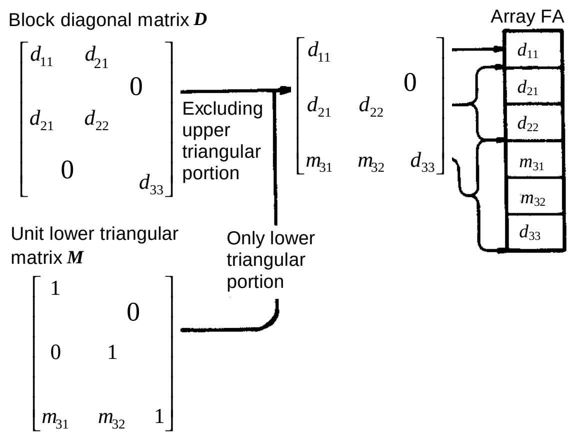
A system of linear equations with a real indefinite symmetric matrix decomposed into the factors ***M***, ***D*** and ***M***
CALL MDMX (B, FA, N, IP, ICON)

Function

This subroutine solves a system of linear equations with an MDM^T-decomposed real indefinite symmetric matrix, where ***M*** is a unit lower triangular matrix, ***D*** is a symmetric block diagonal matrix consisting of symmetric blocks at most of order 2, ***P*** is a permutation matrix (which exchanges rows of the coefficient matrix based on pivoting for MDM^T-decomposition), ***b*** is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. If $d_{k+1,k} \neq 0$, then, $m_{k+1,k} = 0$, and $n \geq 1$.

Parameters

B ..... Input. Constant vector ***b***.
Output. Solution vector ***x***.
One-dimensional array of size n.
FA ..... Input. Matrices ***M*** and ***D***
See Fig. MDMX-1
One-dimensional array of size $n(n+1)/2$.
N ..... Input. Order n of the matrices ***M*** and ***D***, constant vector ***b*** and solution vector ***x***.
IP ..... Input. Transposition vector that indicates the history exchanging rows based on pivoting.
One-dimensional array of size n.
ICON ..... Output. Condition code.
See Table MDMX-1.


Block diagonal matrix D
Excluding upper triangular portion
Array FA
Unit lower triangular matrix M
Only lower triangular portion
d11
d21
d22
m31
m32
d33

Note: The diagonal portion and the lower triangular portion of the matrix ***D***+(***M***-***I***) are stored in the one-dimensional array FA in compressed mode for a symmetrical matrix. In this case ***D*** consists of blocks of order 2 and 1.

Fig. MDMX-1 Storing method for matrices ***L*** and ***D***

Table MDMX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	Coefficient matrix was singular.	Discontinued
30000	N<1, or an error was found in IP.	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... IABS

- Notes
  A system of linear equations can be solved by calling the subroutine SMDM first to MDM^T-decompose the coefficient matrix prior to calling this subroutine. However, such equations can be solved by calling the subroutine LSIX in one step. The input parameters FA and IP to this subroutine are the same as the output parameters A and IP of the subroutine SMDM.

- Example
  A system of linear equations is solved after MDM^T-decomposing an $n \times n$ real symmetric matrix by calling the subroutines SMDM. Where $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100),
     * VW(200),IP(100),IVW(100)
      CHARACTER*4 IA,IB,IX
      DATA IA,IB,IX/'A   ','B   ','X   '/
      READ(5,500) N
      NT=(N*(N+1))/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N
      CALL PSM(IA,1,A,N)
      EPSZ=0.0
      CALL SMDM(A,N,EPSZ,IP,VW,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      READ(5,510) (B(I),I=1,N)
      CALL PGM(IB,1,B,N,N,1)
      CALL MDMX(B,A,N,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      CALL PGM(IX,1,B,N,N,1)
      STOP
  500 FORMAT(I3)
  510 FORMAT(4E15.7)
  600 FORMAT('1'
     *  /6X,'LINEAR EQUATIONS AX=B'
     *  /6X,'ORDER=',I4)
  610 FORMAT(' ',5X,'ICON OF SMDM=',I6)
  620 FORMAT(' ',5X,'ICON OF MDMX=',I6)
      END
```

The subroutines PSM and PGM in this example are used only to print out a real symmetric matrix and a real general matrix, respectively. These programs are described in the example for subroutine MGSM.

Method

Solving a system of linear equations with an MDM^T-decomposed real symmetric matrix.

$$\boldsymbol{P}^{-1}\boldsymbol{MDM}^{\mathrm{T}}\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x}=\boldsymbol{b} \tag{4.1}$$

is reduced into solving the following four equations:

$$\boldsymbol{Mx}^{(1)}=\boldsymbol{Pb} \tag{4.2}$$
$$\boldsymbol{Dx}^{(2)}=\boldsymbol{x}^{(1)} \tag{4.3}$$
$$\boldsymbol{M}^{\mathrm{T}}\boldsymbol{x}^{(3)}=\boldsymbol{x}^{(2)} \tag{4.4}$$
$$\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x}=\boldsymbol{x}^{(3)} \tag{4.5}$$

where ***M*** is a unit lower triangular matrix, ***D*** is a symmetric block diagonal matrix consisting of symmetric blocks at most of order 2, ***b*** is a constant vector, and ***x*** is a solution vector. This subroutine assumes that ***M*** and ***D*** are both decomposed by the block diagonal pivoting method, and ***P*** is a permutation matrix.
(For details, see “Method” for the subroutine SMDM.)

- Solving $\boldsymbol{Mx}^{(1)}=\boldsymbol{Pb}$ (back substitution)
  For a 1×1 pivot (i.e., if the order of the block of ***D*** is 1), it can be serially solved using Eq. (4.6).

$$x_i^{(1)}=b_i-\sum_{k=1}^{i-1}m_{ik}x_k^{(1)},\quad i=1,...,n \tag{4.6}$$

If, however, the *i*-th ineration uses a 2×2 pivot (i.e., the order of matrix D block is 2), $x_{i+1}^{(1)}$ is obtained using Eq. (4.7), preceded by $x_i^{(1)}$, and after that (*i*+2)-th step is computed.

$$x_{i+1}^{(1)}=b'_{i+1}-\sum_{k=1}^{i-1}m_{ik}x_k^{(1)} \tag{4.7}$$

where $\boldsymbol{M}=\left(m_{ij}\right), x^{(1)\mathrm{T}}=\left(x_1^{(1)},...,x_n^{(1)}\right), (\boldsymbol{Pb})^{\mathrm{T}}=\left(b'_1,...,b'_n\right)$

- Solving $\boldsymbol{Dx}^{(2)}=\boldsymbol{x}^{(1)}$
  For a 1×1 pivot, it can be serially solved using eq. (4.8).

$$x_i^{(2)}=x_i^{(1)}/d_{ii},\quad i=1,...,n \tag{4.8}$$

If, however, the *i*-th iteration uses a 2×2 pivot, $x_i^{(2)}$ and $x_{i+1}^{(2)}$ are both obtained using Eq. (4.9) and after that (*i*+2)-th step is computed.

$$\begin{aligned}x_i^{(2)}&=\left(x_i^{(1)}d_{i+1,i+1}-x_{i+1}^{(1)}d_{i+1,i}\right)/DET\\ x_{i+1}^{(2)}&=\left(x_{i+1}^{(1)}d_{ii}-x_i^{(1)}d_{i+1,i}\right)/DET\end{aligned} \tag{4.9}$$

where $DET=\det\begin{pmatrix}d_{ii} & d_{i,i+1}\\ d_{i+1,i} & d_{i+1,i+1}\end{pmatrix}, \boldsymbol{D}=\left(d_{ij}\right)$,

and $x^{(2)\mathrm{T}}=\left(x_1^{(2)},...,x_n^{(2)}\right)$

- Solving $M^{\mathrm{T}}x^{(3)}=x^{(2)}$ (forward substitution)

For a 1×1 pivot, it is serially solved using eq. (4.10)

$$x_i^{(3)}=x_i^{(2)}-\sum_{k=i+1}^{n}m_{ki}x_k^{(3)},\quad i=n,...,1 \tag{4.10}$$

If, however, the *i*-th iteration uses a 2×2 pivot, $x_{i-2}^{(3)}$ is obtained using Eq. (4.11), preceeded by $x_i^{(3)}$, and after that the (*i*–2)-th step is computed.

$$x_{i-1}^{(3)}=x_{i-1}^{(2)}-\sum_{k=i+1}^{n}m_{k,i-1}x_k^{(3)} \tag{4.11}$$

where $\boldsymbol{x}^{(3)\mathrm{T}}=(x_1^{(3)},...,x_n^{(3)})$

- Solving $\left(\boldsymbol{P}^{\mathrm{T}}\right)^{-1}\boldsymbol{x}=\boldsymbol{x}^{(3)}$
  The vector $\boldsymbol{x}^{(3)}$ is multipled by the permutation matrix to obtain the element x_1 of the solution vector ***x***. In practice, however, the elements of the vector $\boldsymbol{x}^{(3)}$ have only to be exchanged by referencing the value of the transposition vector ***IP***.
  Precision of the inner products in this subroutine has been raised to minimize the effect of rounding errors.

A21-11-0301 MGGM, DMGGM

Multiplication of two matrices (real general by real general)
CALL MGGM (A, KA, B, KB, C, KC, M, N, L, VW, ICON)

Function

This subroutine performs multiplication of an $m \times n$ real general matrix $\boldsymbol{A}$ by an $n \times l$ real general matrix $\boldsymbol{B}$.

$$\boldsymbol{C} = \boldsymbol{AB}$$

where, $\boldsymbol{C}$ is an $m \times l$ real matrix. $m, n, l \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, two-dimensional array, A(KA, N).
KA	Input. The adjustable dimension of array A, (≥M).
B	Input. Matrix $\boldsymbol{B}$, two-dimensional array, B(KB, L).
KB	Input. The adjustable dimension of array B, (≥N).
C	Output. Matrix $\boldsymbol{C}$, two-dimensional array, C(KC, L). (Refer to “Comments on use.”)
KC	Input. The adjustable dimension of array C, (≥M).
M	Input. The number of rows m in matrix $\boldsymbol{A}$ and $\boldsymbol{C}$.
N	Input. The number of columns n in matrix $\boldsymbol{A}$ and the number of rows n in matrix $\boldsymbol{B}$.
L	Input. The number of columns l in matrices $\boldsymbol{B}$ and $\boldsymbol{C}$.
VW	Work area. A one-dimensional array of size n.
ICON	Output. Condition codes. Refer to Table MGGM-1.

Table MGGM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M<1, N<1, L<1, KA<M, KB<N, or KC<M	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... None

- Notes
  Saving the storage area.
  The contents of array A are not required to be reserved, the subroutines can be called to save the storage area as follows:
  CALL MGGM (A, KA, B, KB, A, KA, M, N, L, VW, ICON)

  In this case, matrix $\boldsymbol{C}$ is stored in array A. However, user must declare the array A as A (KA, L) instead of A (KA, N).

- Example
  The following shows an example of obtaining the multiplication of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Here, $m \leq 50$, $n \leq 60$, and $l \leq 30$.

```
C      **EXAMPLE**
       DIMENSION A(50,60),B(60,30),C(50,30),
      *VW(60)
       CHARACTER*4 IA,IB,IC
       DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
       DATA KA/50/,KB/60/,KC/50/
    10 READ(5,100) M,N,L
       IF(M.EQ.0) STOP
       WRITE(6,150)
       READ(5,200) ((A(I,J),I=1,M),J=1,N)
       READ(5,200) ((B(I,J),I=1,N),J=1,L)
       CALL MGGM(A,KA,B,KB,C,KC,M,N,L,VW,
      *ICON)
       IF(ICON.NE.0) GOTO 10
       CALL PGM(IA,1,A,KA,M,N)
       CALL PGM(IB,1,B,KB,N,L)
       CALL PGM(IC,1,C,KC,M,L)
       GOTO 10
   100 FORMAT(3I5)
   200 FORMAT(4E15.7)
   150 FORMAT('1'///10X,
      *'** MATRIX MULTIPLICATION **')
       END
```

Subroutine PGM in the example is for printing a real matrix. This program is shown in the example for subroutine MGSM.

A21-11-0401 MGSM, DMGSM

Multiplication of two matrices (real general by real symmetric)
CALL MGSM (A, KA, B, C, KC, N, VW, ICON)

Function

This subroutine performs multiplication of an $n \times n$ real general matrix $\boldsymbol{A}$ by an $n \times n$ real symmetric matrix $\boldsymbol{B}$.

$$\boldsymbol{C} = \boldsymbol{AB}$$

where, $\boldsymbol{C}$ is an $n \times n$ real matrix, $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, two-dimensional array, A(KA, N).
KA	Input. The adjustable dimension of array A, (≥N).
B	Input. Matrix $\boldsymbol{B}$ stored in the compressed mode, one dimensional array of size $n(n+1)/2$.
C	Output. Matrix $\boldsymbol{C}$ two-dimensional array, C(KC, N). (See “Comments on use.”)
KC	Input. The adjustable dimension of array C, (≥N).
N	Input. The number of columns n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.
VW	Work area. One dimensional array of size n.
ICON	Output. Condition codes. See Table MGSM-1.

Table MGSM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N<1,KA<N or KC<N	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... none

- Notes
  Saving the storage area:
  When the contents of array A is not required to be reserved, the subroutines can be called to save the storage area as follows:

  CALL MGSM(A, KA, B, A, KA, N, VW, ICON)

  In this case, matrix $\boldsymbol{C}$ is stored in the general mode in array A.

- Example
  The following shows an example of obtaining the multiplication $\boldsymbol{C}$ of a real matrix $\boldsymbol{A}$ by a real symmetric matrix $\boldsymbol{B}$. Now, the matrix $\boldsymbol{C}$ is overwritten in the same area for $\boldsymbol{A}$. n<100.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(5050),VW(100)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
   10 READ(5,100) N
      IF(N.EQ.0) STOP
      WRITE(6,150)
      NT=N*(N+1)/2
      READ(5,200) ((A(I,J),J=1,N),I=1,N)
      READ(5,200) (B(I),I=1,NT)
      CALL PGM(IA,1,A,100,N,N)
      CALL PSM(IB,1,B,N)
      CALL MGSM(A,100,B,A,100,N,VW,ICON)
      WRITE(6,250) ICON
      IF(ICON.NE.0) GOTO 10
      CALL PGM(IC,1,A,100,N,N)
      GOTO 10
  100 FORMAT(I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** C=A*B  GENERAL BY SYMMETRIC **')
  250 FORMAT(//10X,'** MGSM ICON=',I5)
      END

C     ** MATRIX PRINT(REAL NON-SYMMETRIC) **
      SUBROUTINE PGM(ICOM,L,A,K,M,N)
      DIMENSION A(K,N)
      CHARACTER*4 ICOM(L)
      WRITE(6,600) (ICOM(I),I=1,L)
      DO 10 I=1,M
      WRITE(6,610) I,(J,A(I,J),J=1,N)
   10 CONTINUE
      RETURN
  600 FORMAT(/10X,35A2)
  610 FORMAT(/5X,I3,3(4X,I3,E17.7),
     *(/8X,3(4X,I3,E17.7)))
      END

C     ** MATRIX PRINT(REAL SYMMETRIC) **
      SUBROUTINE PSM(ICOM,L,A,N)
      DIMENSION A(1)
      CHARACTER*4 ICOM(L)
      WRITE(6,600) (ICOM(I),I=1,L)
      LS=1
      LE=0
      DO 10 I=1,N
      LE=LE+I
      WRITE(6,610) I,(A(J),J=LS,LE)
   10 LS=LE+1
      RETURN
  600 FORMAT(/10X,35A2)
  610 FORMAT(/5X,I3,3(4X,E17.7),
     *(/8X,3(4X,E17.7)))
      END
```

Subroutines PGM and PSM in the example are for printing the real and real symmetric matrices.

D11-10-0101 MINF1, DMINF1

Minimization of a function with several variables (Revised quasi-Newton method, using function values only)
CALL MINF1 (X, N, FUN, EPSR, MAX, F, G, H, VW, ICON)

Function

Given a real function $f(\boldsymbol{x})$ of n variables and an initial vector $\boldsymbol{x}_0$, the vector $\boldsymbol{x}^*$ which gives a local minimum of $f(\boldsymbol{x})$ and its function value $f(\boldsymbol{x}^*)$ are obtained by using the revised quasi-Newton method.

The $f(\boldsymbol{x})$ is assumed to have up to the second continuous partial derivative, and $n \geq 1$.

Parameters

X ..... Input. Initial vector $\boldsymbol{x}_0$.
Output. Vector $\boldsymbol{x}^*$.
One-dimensional array of size n.

N ..... Input. Number of variables n.

FUN ... Input. Name of function subprogram which calculates $f(\boldsymbol{x})$.
The form of subprogram is as follows:
FUNCTION FUN(X)
where
X .....Input. Arbitrary variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
The function FUN should be assigned with the value of $f(\boldsymbol{x})$.
(See the example below.)

EPSR ... Input. Convergence criterion ($\geq$0.0)
When EPSR=0.0, a standard value is used.
See "Note".

MAX ... Input. Upper limit or number of evaluations for the function ($\neq$0). See "Note".
Output. Number of times actually evaluated (>0).

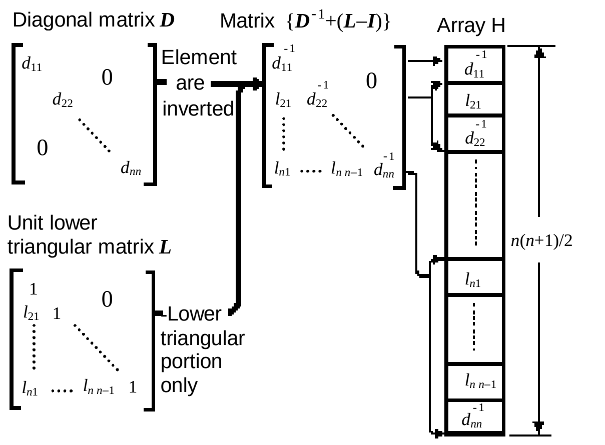
F ..... Output. Value of the function $f(\boldsymbol{x}^*)$.

G ..... Output. Gradient vector at $\boldsymbol{x}^*$.
One-dimensional array of size n.

H ..... Output. Hessian matrix at $\boldsymbol{x}^*$.
This is decomposed as LDL^T and stored in compressed storage mode for a symmetric matrix. See Fig. MINF1-1.
One-dimensional array of size $n(n+1)/2$.

VW ..... Work area. One-dimensional array of size $3n+1$.

ICON ..... Output. Condition code.
See Table MINF1-1.



Note: The approximate Hessian matrix is decomposed as LDL^T, and after computation, the diagonal and lower triangular portions of the matrix, $D^{-1}+(L-I)$, are stored into the one-dimensional array H in compressed storage mode for a symmetric matrix.

Fig. MINF1-1 Storage Hessian matrix

Table MINF1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Convergence condition was not satisfied within the specified number of evaluations of the function.	Parameters X, F, G and H each contains the last value obtained.
20000	During computation, $\mathbf{g}_k^T\mathbf{p}_k \geq 0$ occurred, so the local decrement of the function was not attained. See Eq. (4.5) in " Method" . EPSR was too small or the error of difference approximation for a gradient vector exceeded the limit.	Discontinued (Parameters X and F each contains the last value obtained.)
30000	N<1, EPSR<0.0 or MAX=0	Bypassed

Comments on use

- Subprograms used
  SSL II ... LDLX, UMLDL, AMACH and MGSSL
  FORTRAN basic functions ... ABS, SQRT, AMAX1 and AMIN1
- Notes
  – The program which calls this subroutine must have an EXTERNAL statement for the function program name that corresponds to the argument FUN.
  – Giving EPSR
  The subroutine tests convergence by

$$\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \leq \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot \text{EPSR}$$

for the iteration vector $\boldsymbol{x}_k$ and if the above condition is satisfied, $\boldsymbol{x}_{k+1}$ is taken as the local minimum point $\boldsymbol{x}^*$ and the iteration is terminated.
The subroutine assumes that function $f(\boldsymbol{x})$ is approximately quadratic in the region of the local minimum point $\boldsymbol{x}^*$. If the function value $f(\boldsymbol{x}^*)$ is to be obtained as accurate as the unit round off,

EPSR = $\sqrt{u}$, u is the unit round off

is satisfactory.
The standard value of EPSR is $2 \cdot \sqrt{u}$

– Giving MAX
The number of evaluations of a function is calculated by the number of $f(\boldsymbol{x})$ for variable vector x.
It corresponds to the number of calling subprogram FUN.
The number of evaluations of a function depends on characteristics of the function in addition to the initial vector and a convergence criterion. Generally, for good initial vector, if a standard value is used as the convergence criterion, MAX=$400 \cdot n$ is appropriate.
If the convergence condition is not satisfied within the specified number of evaluations and the subroutine is returned with ICON=10000, the iteration can be continued by calling the subroutine again.
In this case parameter MAX is specified with a negative value for an additional evaluation number, and the contents of other parameters must be kept intact.

- Example
The global minimum point $\boldsymbol{x}^*$ for
$f(\boldsymbol{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ is obtained with the initial vector $\boldsymbol{x}_0 = (-1.2, 1.0)^{\mathrm{T}}$ given.

```
C     **EXAMPLE**
      DIMENSION X(2),G(2),H(3),VW(7)
      EXTERNAL ROSEN
      X(1)=-1.2
      X(2)=1.0
      N=2
      EPSR=1.0E-3
      MAX=400*2
      CALL MINF1(X,N,ROSEN,EPSR,MAX,
     *           F,G,H,VW,ICON)
      WRITE(6,600) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,610) F,MAX
      WRITE(6,620) (I,X(I),I,G(I),I=1,N)
      STOP
  600 FORMAT('1','*ICON=',I5)
  610 FORMAT(' ','*F=',E15.7,' MAX=',I5/)
  620 FORMAT(/' X(',I2,')=',E15.7,2X,
     *        ' G(',I2,')=',E15.7)
      END
C     OBJECTIVE FUNCTION
      FUNCTION ROSEN(X)
      DIMENSION X(2)
      ROSEN=(1.0-X(1))**2+100.0*
     *      (X(2)-X(1)*X(1))**2
      RETURN
      END
```

Method

Given a real function $f(\boldsymbol{x})$ of n variables and an initial vector $\boldsymbol{x}_0$, the vector $\boldsymbol{x}^*$ which gives a local minimum of $f(\boldsymbol{x})$ and its function value $f(\boldsymbol{x}^*)$ are obtained by using the revised quasi-Newton method.
The subroutine obtains the gradient vector g of $f(\boldsymbol{x})$ by using difference formula.

- Revised quasi-Newton method
When the function $f(\boldsymbol{x})$ is quadratic, its Taylor series expansion in the region of the local minimum point $\boldsymbol{x}^*$ is given by

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}^*) \tag{4.1}$$

where $\boldsymbol{B}$ is a Hessian matrix of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}^*$. If the matrix $\boldsymbol{B}$ is positive definite, Eq. (4.1) has a global minimum. Let $\boldsymbol{x}_k$ be an arbitrary point in the region of $\boldsymbol{x}^*$ and let $\boldsymbol{g}_k$ be the gradient vector of f(x) at the point $\boldsymbol{x}_k$ then $\boldsymbol{x}^*$ can be obtained by using Eq. (4.1) as follows:

$$\boldsymbol{x}^* = \boldsymbol{x}_k - \boldsymbol{B}^{-1}\boldsymbol{g}_k \tag{4.2}$$

Even when function $f(\boldsymbol{x})$ is not quadratic, it can be assumed to approximate a quadratic function in the region of $\boldsymbol{x}^*$ and a iterative formula can be derived based on Eq. (4.2).
However, since obtaining an inverse of matrix $\boldsymbol{B}$ directly is not practical because of the great amount of computation, an approximate matrix to $\boldsymbol{B}$ is generally set and is modified while the iteration process is being carried out.
The revised quasi-Newton method obtains a local minimum point $\boldsymbol{x}^*$ by letting $\boldsymbol{B}_k$ be an approximation of the matrix $\boldsymbol{B}$ and using the following iterative formulae:

$$\left.\begin{aligned} \boldsymbol{B}_k \boldsymbol{p}_k &= -\boldsymbol{g}_k \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \\ \boldsymbol{B}_{k+1} &= \boldsymbol{B}_k + \boldsymbol{E}_k \end{aligned}\right\} \quad k = 0,1,... \tag{4.3}$$

where, $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$ and $\boldsymbol{B}_0$ are an arbitrary positive definite matrix, $\boldsymbol{p}_k$ is a vector denoting the search direction from $\boldsymbol{x}_k$ toward the local minimum point, and α_k is a (linear search) constant which is set so $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$ is the smallest locally.
$\boldsymbol{E}_k$ is a matrix of rank two which is used to improve the approximate Hessian matrix $\boldsymbol{B}_k + \boldsymbol{E}_k$ is defined assuming that function $f(\boldsymbol{x})$ is quadratic in the region of the local minimum point $\boldsymbol{x}^*$, and the secant condition is satisfied.

$$\left(\boldsymbol{g}_{k+1}-\boldsymbol{g}_k\right)=\boldsymbol{B}_{k+1}\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right) \tag{4.4}$$

For the search direction $\boldsymbol{p}_k$ to be downwards (i.e., the function $f(\boldsymbol{x})$. decreasing locally along the $\boldsymbol{p}_k$ direction at point $\boldsymbol{x}_k$) during the iteration process shown in (4.3), the following relation

$$\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p}_k\left(=-\boldsymbol{p}_k^{\mathrm{T}}\boldsymbol{B}_k\boldsymbol{p}_k\right)<0 \tag{4.5}$$

must be satisfied based on the sufficient condition of the second order that the function $f(\boldsymbol{x})$ has a minimum value at the point $\boldsymbol{x}^*$.

In other words, the iterative calculation requires that the approximate Hessian matrix $\boldsymbol{B}_k$ is positive definite.

In the revised quasi-Newton method, the approximate Hessian matrix $\boldsymbol{B}_k$ is expressed as being decomposed as LDL$^{\mathrm{T}}$ and the refinement by the $\boldsymbol{E}_k$ is accomplished as follows.

$$\boldsymbol{L}_{k+1}\boldsymbol{D}_{k+1}\boldsymbol{L}_{k+1}^{\mathrm{T}}=\boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^{\mathrm{T}}+\boldsymbol{E}_k \tag{4.6}$$

The characteristic of the revised quasi-Newton method is to guarantee that $\boldsymbol{D}_{k+1}$ is positive definite by keeping all the diagonal elements of $\boldsymbol{D}_{k+1}$ positive.

- Computational procedure in the subroutine
  - (a) Initializing the Hessian matrix ($\boldsymbol{B}_0=\boldsymbol{I}_n$)
  - (b) Computation of the gradient vector $\boldsymbol{g}_k$
  - (c) Determining the search vector $\boldsymbol{p}_k$ ($\boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^{\mathrm{T}}\boldsymbol{p}_k=-\boldsymbol{g}_k$) This equation is solved by calling the subroutine LDLX.
  - (d) Linear search ($\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\alpha_k\boldsymbol{p}_k$)
  - (e) Improvement of the approximate Hessian matrix ($\boldsymbol{L}_{k+1}\boldsymbol{D}_{k+1}\boldsymbol{L}_{k+1}^{\mathrm{T}}=\boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^{\mathrm{T}}+\boldsymbol{E}_k$)

The above steps, (b) to (e), are repeated for k=0, 1, ...

- Notes on each algorithm

(***a***) Computation of the gradient vector $\boldsymbol{g}_k$

The subroutine approximates $\boldsymbol{g}_k$ by using the forward difference (4.7) and the central difference (4.8),

$$g_k^i\approx\left(f\left(\boldsymbol{x}_k+h\boldsymbol{e}_i\right)-f\left(\boldsymbol{x}_k\right)\right)/h \tag{4.7}$$

$$g_k^i\approx\left(f\left(\boldsymbol{x}_k+h\boldsymbol{e}_i\right)-f\left(\boldsymbol{x}_k-h\boldsymbol{e}_i\right)\right)/2h \tag{4.8}$$

Where,

$\boldsymbol{g}_k=\left(g_k^1,g_k^2,\ldots,g_k^n\right)^{\mathrm{T}}$

$\boldsymbol{x}_i=\left(x_k^1,x_k^2,\ldots,x_k^n\right)^{\mathrm{T}}$

$\boldsymbol{e}_i$ is the i-th coordinate vector

$h=\sqrt{u}$, u unit round off

At the beginning of the iteration, $\boldsymbol{g}_k$ is approximated by using Eq. (4.7), but when the iteration is close to the convergence area, the approximation method is changed to (4.8).

(***b***) Linear search (selection of α_k)

The linear search obtains the minimum point of function $f(\boldsymbol{x})$ along the search direction $\boldsymbol{p}_k$ i.e. it obtains α_k which minimizes the function

$$\psi(\alpha)=f\left(\boldsymbol{x}_k+\alpha\boldsymbol{p}_k\right),\quad \alpha\geq 0 \tag{4.9}$$

The subroutine approximates $\psi(\alpha)$ by the quadratic interpolation and assumes α_k as follows:

$$\alpha_k\approx\min\{1,\ 2(f(\boldsymbol{x}_k)-f(\boldsymbol{x}_{k-1}))/\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p}_k\} \tag{4.10}$$

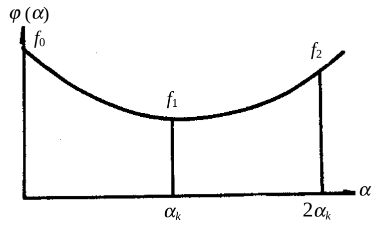
The above equation makes use of the quadratic convergency of the Newton method, setting α_k=1 in the final step of the iteration. In addition, the second term of Eq. (4.10) guarantees that $f(\boldsymbol{x}_{k+1})<f(\boldsymbol{x}_k)$ to prevent it being divergent during the initial iteration process. The second term of Eq. (4.10) assumes.

$$f\left(\boldsymbol{x}_{k+1}\right)-f\left(\boldsymbol{x}_k\right)\approx f\left(\boldsymbol{x}_k\right)-f\left(\boldsymbol{x}_{k-1}\right) \tag{4.11}$$

in the initial iteration step, and it is not an exact approximation made by the quadratic interpolation. Therefore the subroutine searches for the minimum point by using extrapolation and interpolation described as follows (linear search):

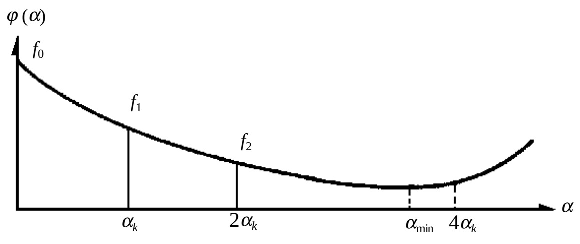
let f_0, f_1 and f_2 be the function values for the points $\boldsymbol{x}_{k+1}^{(0)}=\boldsymbol{x}_k$, $\boldsymbol{x}_{k+1}^{(1)}=\boldsymbol{x}_k+\alpha_k\boldsymbol{p}_k$ and $\boldsymbol{x}_{k+1}^{(2)}=\boldsymbol{x}_k+2\alpha_k\boldsymbol{p}_k$, respectively, then

(a) If $f_0>f_1$ and $f_1<f_2$, the search is terminated setting $\boldsymbol{x}_{k+1}^{(1)}$ as the minimum point.

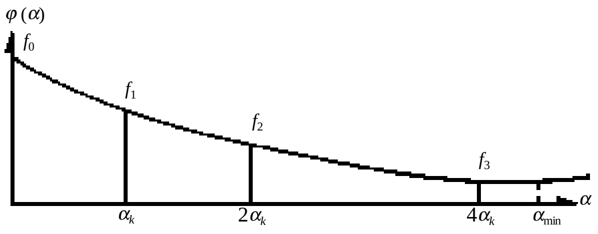


(b) If $f_0>f_1>f_2$, $\alpha_{\min}$ which gives the minimum point is extrapolated by using the quadratic interpolation based on those three points as follows:

- If $2\alpha_k < \alpha_{\min} < 4\alpha_k$, the search is terminated setting $x_{k+1}^{(2)}$ as the minimum point.

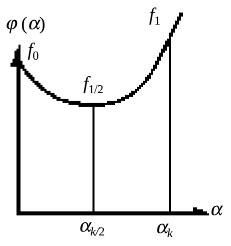


- If $4\alpha_k < \alpha_{\min}$ the search goes back to the beginning after setting $\alpha_k = 2\alpha_k$

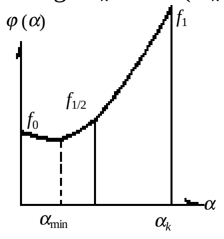


(c) If $f_0 < f_1$, the function value $f_{1/2}$ corresponding to $x_{k+1}^{(1/2)} = x_k + \frac{1}{2} \cdot \alpha_k p_k$ is obtained, and

- If $f_0 > f_{1/2}$ and $f_{1/2} < f_1$, the search is terminated setting $x_{k+1}^{(1/2)}$ as the minimum point.



- If $f_0 < f_{1/2} < f_1$, the search goes back to the beginning after interpolating $\alpha_{\min}$ which gives a minimum point by using the quadratic interpolation based on those three points and then setting $\alpha_k = \max(\alpha_k/10, \alpha_{\min})$



As described above, the linear search based on α_k is terminated, but if the function $f(x)$ keeps decreasing further at the point x_{k+1}, i.e.,

$$g_{k+1}^{\mathrm{T}} p_k < g_k^{\mathrm{T}} p_k \tag{4.12}$$

then the new search direction p_{k+1} as well as α_{k+1} are determined to repeat the linear search.
If Eq. (4.12) is not satisfied, the process moves to the next step to improve the approximate Hessian matrix.

(*c*) Convergence criterion
The subroutine terminates the iteration when Eq. (4.13) is satisfied in the iteration process (linear search) performed after the calculation of g_k is changed to the approximation by the central difference (4.8). The obtained x_{k+1} is assumed to be the local minimum point x^*.

$$\|x_{k+1} - x_k\|_\infty \le \max\left(1.0, \|x_k\|_\infty\right) \cdot \text{EPSR} \tag{4.13}$$

(*d*) Improving approximate Hessian matrix
The BFGS (Broyden - Fletcher - Goldfarb - Shanno) furmula (4.14) is used for the improvement.

$$B_{k+1} = B_k + \frac{r_k r_k^{\mathrm{T}}}{\delta_k^{\mathrm{T}} r_k} - \frac{B_k \delta_k \delta_k^{\mathrm{T}} B_k}{\delta_k^{\mathrm{T}} B_k \delta_k} \tag{4.14}$$

Where, $r_k = g_{k+1} - g_k$
$\delta_k = x_{k+1} - x_k$

The subroutine starts the iteration by setting a unit matrix to the initial approximate Hessian matrix B_0. The i-th step improvement for the Hessian matrix B_k is carried out in the form of being decomposed as LDL$^{\mathrm{T}}$.

$$\tilde{L}_k \tilde{D}_k \tilde{L}_k^{\mathrm{T}} = L_k D_k L_k^{\mathrm{T}} + \frac{r_k r_k^{\mathrm{T}}}{\alpha_k p_k r_k} \tag{4.15}$$

$$L_{k+1} D_{k+1} L_{k+1}^{\mathrm{T}} = \tilde{L}_k \tilde{D}_k \tilde{L}_k^{\mathrm{T}} + \frac{g_k g_k^{\mathrm{T}}}{p_k g_k} \tag{4.16}$$

Where the second terms of both (4.15) and (4.16) are rank one matrices.
For further details, refer to Reference [34].

D11-20-0101 MING1, DMING1

Minimization of a function with several variables (Quasi-Newton method, using function values and its derivatives)
CALL MING1 (X, N, FUN, GRAD, EPSR, MAX, F, G, H, VW, ICON)

Function

Given a real function $f(\boldsymbol{x})$ of n variables, its derivative $\boldsymbol{g}(\boldsymbol{x})$ and an initial vector $\boldsymbol{x}_0$, the vector $\boldsymbol{x}^*$ which gives a local minimum of $f(\boldsymbol{x})$ and its function value $f(\boldsymbol{x}^*)$ are obtained by using the quasi-Newton method.

$f(\boldsymbol{x})$ is assumed to have up to the second continuous partial derivative, where $\boldsymbol{x} = (x_1, x_2,..., x_n)^T$ and $n \geq 1$.

Parameters

X ..... Input. Initial vector $\boldsymbol{x}_0$.
Output. Vector $\boldsymbol{x}^*$
One-dimensional array of size n.

N ..... Input. Number of variables n.

FUN ..... Input. Name of function subprogram which calculates $f(\boldsymbol{x})$.
The form of subprogram is as follows:
FUNCTION FUN(X)
where,
X ..... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
The function FUN should be assigned with the value of $f(\boldsymbol{x})$.
(See Example.)

GRAD .. Input. Name of subroutine subprogram which computes $\boldsymbol{g}(\boldsymbol{x})$.
The form of subprogram is as follows:
SUBROUTINE GRAD(X, G)
where,
X ..... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
G ..... Output. One-dimensional array of size n which has a correspondence
$\mathrm{G}(1) = \partial f / \partial x_1, ..., \mathrm{G}(\mathrm{N}) = \partial f / \partial x_n$
(See Example.)

EPSR ..... Input. Convergence criterion (≥0.0)
When EPSR=0.0, a default value is used.
(See Notes.)

MAX ..... Input. The upper limit (≠0) of the number of evaluations of functions $f(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$.
(See Notes.)
Output. The number of evaluations in which $f(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are actually evaluated (>0).

F ..... Output. Function value $f(\boldsymbol{x}^*)$.

G ..... Output. Gradiant vector $\boldsymbol{g}(\boldsymbol{x}^*)$.
One-dimentional array of size n.

H ..... Output. Inverse matrix of a Hessian matrix at $\boldsymbol{x}^*$.
This is stored in the compressed mode for symmetric matrix.
One-dimensional array of size $n(n+1)/2$.

VW ..... Work area. One-dimensional array of size $3n+1$.

ICON .... Output. Condition code.
See Table MING1-1.

Table MING1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Convergence condition was not satisfied within the specified number of evaluations.	The last value is stored in parameters X, F, G, and H.
20000	During computation, $g_k^T p_k \geq 0$ occurred, so the local decrement of the function was not attained (See (4.5) in Method). EPSR was too small.	Discontinued (The last value is stored in parameters X, and F).
25000 0	The function is monotonically decreasing along searching direction.	Discontinued
30000	N<1, EPSR<0.0 or MAX=0	Bypassed.

Comments on use

- Subprograms used
SSL II ... AMACH, MGSSL, MSV, AFMAX
FORTRAN basic functions ... ABS, SQRT
- Notes
The program which calls this subroutine must have an EXTERNAL statement for the subprogram name that corresponds to the arguments FUN and GRAD.

Giving EPSR:
The subroutine tests convergence of the iteration vector $\boldsymbol{x}_k$ by

$$\left\| \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \right\|_\infty \leq \max\left(1.0, \left\| \boldsymbol{x}_k \right\|_\infty\right) \cdot \mathrm{EPSR}$$

and if the above condition is satisfied, $\boldsymbol{x}_{k+1}$ is taken as the local minimum point $\boldsymbol{x}^*$ and the iteration is terminated.

The subroutine assumes that function $f(\boldsymbol{x})$ is approximately quadratic in the region of the local minimum point $\boldsymbol{x}^*$.

If the function value $f(\boldsymbol{x}^*)$ is to be obtained as accurate as the unit round off, EPSR $= \sqrt{u}$, where u is the unit round off is satisfactory. The default value is $\sqrt{u} / 8.0$

Giving MAX:
For a variable vector $\boldsymbol{x}$, the total number of evaluations is calculated by adding the number of computation for $f(\boldsymbol{x})$, (i.e., 1), and the number of computations for $\boldsymbol{g}(\boldsymbol{x})$, (i.e., n).

The number of evaluations of functions depends on characteristics of the functions in addition to the initial vector and the convergence criterion. Generally, if the default value is used as the convergence criterion and a good initial vector are used, MAX=$400 \cdot n$ is appropriate. If the convergence condition is not satisfied within the specified number of evaluations and the subroutine returned with ICON=10000, the iteration can be continued by calling the subroutine again. In this case, parameter MAX is specified with a negative value for an additional evaluation number, and the contents of other parameters must be kept intact.

- Example
  The global minimum point $\boldsymbol{x}^*$ for $f(\boldsymbol{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ is obtained with the initial vector $\boldsymbol{x}_0 = (-1.2, 1.0)^{\mathrm{T}}$.

```
C     **EXAMPLE**
      DIMENSION X(2),G(2),H(3),VW(7)
      EXTERNAL ROSEN,ROSENG
      X(1)=-1.2
      X(2)=1.0
      N=2
      EPSR=1.0E-4
      MAX=400*2
      CALL MING1(X,N,ROSEN,ROSENG,EPSR,
     *           MAX,F,G,H,VW,ICON)
      WRITE(6,600) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,610) F,MAX
      WRITE(6,620) (I,X(I),I,G(I),I=1,N)
      STOP
  600 FORMAT('1','*ICON=',I5)
  610 FORMAT(' ','*F=',E15.5,' MAX=',I5/)
  620 FORMAT(/' X(',I2,')=',E15.7,2X,
     *        ' G(',I2,')=',E15.7)
      END

C     OBJECTIVE FUNCTION
      FUNCTION ROSEN(X)
      DIMENSION X(2)
      ROSEN=(1.0-X(1))**2+100.0*
     *      (X(2)-X(1)*X(1))**2
      RETURN
      END
C     GRADIENT VECTOR
      SUBROUTINE ROSENG(X,G)
      DIMENSION X(2),G(2)
      G(1)=-2.0*(1.0-X(1))
     *-400.0*X(1)*(X(2)-X(1)*X(1))
      G(2)=200.0*(X(2)-X(1)*X(1))
      RETURN
      END
```

Method
Given a real function $f(\boldsymbol{x})$ of n variables, its derivative $\boldsymbol{g}(\boldsymbol{x})$ and initial vector $\boldsymbol{x}_0$, the vector $\boldsymbol{x}^*$ which gives a local minimum of $f(\boldsymbol{x})$ and its function value $f(\boldsymbol{x}^*)$ are obtained by using the quasi-Newton method.

- Quasi-Newton method
  When the function $f(\boldsymbol{x})$ is quadratic, its Taylor series expansion in the region of the local minimum point $\boldsymbol{x}^*$ is given by

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^{\mathrm{T}} \boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}^*) \tag{4.1}$$

where $\boldsymbol{B}$ is a Hessian matrix of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}^*$. If the matrix $\boldsymbol{B}$ is positive definite, (4.1) has a global minimum. Let $\boldsymbol{x}_k$ be an arbitrary point in the region of $\boldsymbol{x}^*$ and let $\boldsymbol{g}_k$ be the gradient vector of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}_k$ then $\boldsymbol{x}^*$ can be obtained using (4.1) as follows:

$$\boldsymbol{x}^* = \boldsymbol{x}_k - \boldsymbol{H}\boldsymbol{g}_k \tag{4.2}$$

where, $\boldsymbol{H}$ is an inverse matrix of Hessian matrix $\boldsymbol{B}$.
Even when function $f(\boldsymbol{x})$ is not quadratic, it can be assumed to approximate a quadratic function in the region of $\boldsymbol{x}^*$ and a iterative formula can be derived based on (4.2). However, since obtaining an inverse of matrix $\boldsymbol{B}$ directly is not practical because of the great amount of computation, an approximate matrix to $\boldsymbol{B}$ is generally set and is modified while the iteration process is being carried out.
The quasi-Newton method obtains a local minimum point $\boldsymbol{x}^*$ by letting $\boldsymbol{H}_k$ be an approximation of the matrix $\boldsymbol{H}$ and using the following iterative formula (4.3) and (4.4)

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \tag{4.3}$$

$$\begin{aligned}\boldsymbol{H}_{k+1} = \boldsymbol{H}_k &+ \left(1 + \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H}^k \boldsymbol{r}^k / \delta_k^{\mathrm{T}} \boldsymbol{r}^k\right)\left(\delta_k \delta_k^{\mathrm{T}} / \delta_k^{\mathrm{T}} \boldsymbol{r}_k\right) \\ &- \left(\boldsymbol{H}_k \boldsymbol{r}_k \delta_k^{\mathrm{T}} + \delta_k \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H}_k\right) / \delta_k^{\mathrm{T}} \boldsymbol{r}_k\end{aligned} \tag{4.4}$$

where,

$$\boldsymbol{p}_k = -\boldsymbol{H}_k \boldsymbol{g}_k, \quad \boldsymbol{r}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k,$$
$$\delta_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k,$$
$$k = 0,1,2,...$$

where, $\boldsymbol{H}_0$ is an arbitary positive definite matrix, in (4.3) $\boldsymbol{p}_k$ is a vector denoting the search direction from $\boldsymbol{x}_k$, toward the local minimum point and α_k is a (linear search) constant which is set so that $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)$ is locally smallest.
For the search direction $\boldsymbol{p}_k$ to be downwards (the function $f(\boldsymbol{x})$ decreasing locally along the $\boldsymbol{p}_k$ direction) during the iteration process shown in (4.3) and (4.4), the following relation

$$\boldsymbol{g}_k^{\mathrm{T}} \boldsymbol{p}_k < 0 \tag{4.5}$$

must be satisfied.

- Computational procedure in the subroutine

1) Initializing the approximate inverse matrix of the Hessian matrix ($\boldsymbol{H}_0 = \boldsymbol{I}_n$).
2) Computation of gradient vector $\boldsymbol{g}_k$
3) Computation of search vector $\boldsymbol{p}_k (\boldsymbol{p}_k = -\boldsymbol{H}_k \boldsymbol{g}_k)$
4) Linear search ($\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$)
5) Improvement of the approximate inverse matrix $\boldsymbol{H}_k$ ($\boldsymbol{H}_{k+1}$ is obtained from (4.4))

The above steps 2 through 5 are repeated for k=0, 1, ...

- Note on each algorithm

– Initializing the approximate inverse matrix

This subroutine uses unit matrix $\boldsymbol{I}_n$ as initial approximate inverse matrix $\boldsymbol{H}_0$ corresponding to a Hessian matrix. Inverse matrix $\boldsymbol{H}_k$ is improved by (4.4) each time it is iterated.

$\boldsymbol{H}_1$, however, is obtained by (4.4) after resetting

$$\boldsymbol{H}_0 = s\boldsymbol{I}_n$$

where

$$s = \boldsymbol{\delta}^{\mathrm{T}} \boldsymbol{r}_0 / \boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0$$

– Linear search (selection of α_k)

The linear search obtains the minimum point of function $f(\boldsymbol{x})$ along the search direction $\boldsymbol{p}_k$ that is α_k which minimizes the function $\varphi(\alpha)$

$$\varphi(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k), \quad \alpha \geq 0 \tag{4.6}$$

The subroutine approximates $\varphi(\alpha)$ by quadratic interpolation and assumes α_k to be:

$$\alpha_k \approx \min\left\{1, 2\left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})\right) / \boldsymbol{g}_k^{\mathrm{T}} \boldsymbol{p}_k\right\} \tag{4.7}$$

The above equation makes use of the quadratic convergency of the Newton method, setting α_k= 1 in the final step of the iteration. In addition, the second term of (4.7) guarantees that $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ to prevent it from diverging during the initial iteration process.
The second term of (4.7) assumes variable ratio of the function as

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1}) \tag{4.8}$$

in the initial iteration step, and it is not an exact approximation made by the quadratic interpolation. Therefore the subroutine searches for the minimum point by using extrapolation and interpolation described as follows (linear search):

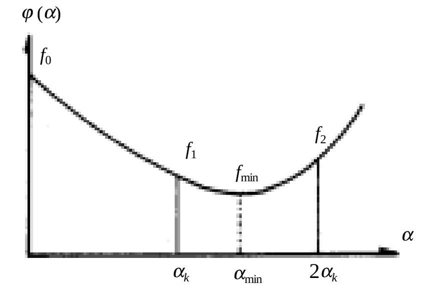
(a) α_k is obtained from (4.7).

(b) The function values corresponding to points $\boldsymbol{x}_{k+1}^{(0)} = \boldsymbol{x}_k, \boldsymbol{x}_{k+1}^{(1)} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \boldsymbol{x}_{k+1}^{(2)} = \boldsymbol{x}_k + 2\alpha_k \boldsymbol{p}_k$ are obtained and they are assumed to be f_0, f_1 and f_2 respectively.

(c) If $f_0 > f_1$ and $f_1 < f_2$, point $\alpha_{\min}$ is interpolated by using a quadratic interpolation based on these three points by

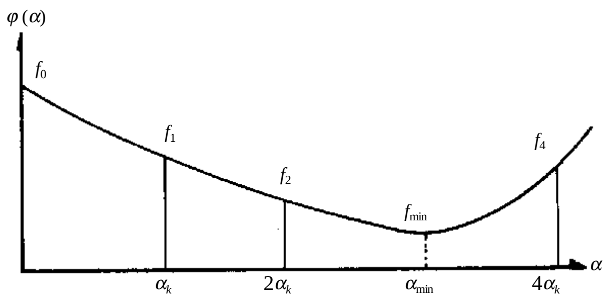
$$\alpha_{\min} = \alpha_k - \frac{\alpha_k}{2} \cdot \frac{f_2 - f_0}{f_0 - 2f_1 + f_2} \tag{4.9}$$

The search is terminated by setting $\alpha_{\min}$ as the final α_k and the function value is assumed to be a minimum value.

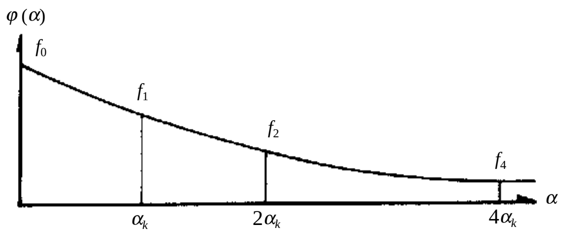


(d) If $f_0 > f_1 > f_2$, the function values corresponding to $\boldsymbol{x}_{k+1}^{(4)} = \boldsymbol{x}_k + 4\alpha \boldsymbol{p}_k$ is obtained and assumed to be f_4.

- If $f_2 < f_4$, $\alpha_{\min}$ is extrapolated by using quadratic interpolation (4.9) based on these three points f_0, f_1 and f_2.
  If $4\alpha_k < \alpha_{\min}$, the search goes back to (b) after setting $\alpha_k = 2\alpha_k$.
  If $2\alpha_k < \alpha_{\min} < 4\alpha_k$, this subroutine terminates the search assuming this $\alpha_{\min}$ to be final α_k and function value $f_{\min}$ to be the minimum value.



- If $f_2 > f_4$, this subroutine goes back to (b) after setting $\alpha_k = 2\alpha_k$.

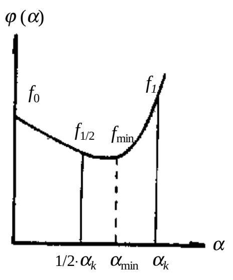


(e) If $f_0 \le f_1$, the function value corresponding to point $x^{(1/2)}_{k+1} = x_k + \frac{1}{2}\alpha_k p_k$ is obtained and sets it as $f_{1/2}$.

- If $f_0 > f_{1/2}$ and $f_{1/2} < f_1$, $\alpha_{\min}$ is interpolated by using the quadratic interpolation based on these three points.

$$\alpha_{\min} = \frac{\alpha_k}{2} - \frac{\alpha_k}{4} \cdot \frac{f_1 - f_0}{f_0 - 2f_{1/2} + f_1} \tag{4.10}$$

The search is terminated setting $\alpha_{\min}$ as α_k and function value $f_{\min}$ is assumed to be the minimum value.



- If $f_0 < f_{1/2}$, this subroutine goes back to the beginning of (e) after setting $f_1 = f_{1/2}$ and $\alpha_k = \frac{1}{2}\alpha_k$.

Thus, this subroutine terminates the linear search based on α_k. If function $f(x)$ continues to decrement at x_{k+1}, that is, if

$$g^{\mathrm{T}}_{k+1} p_k < g^{\mathrm{T}}_k p_k \tag{4.11}$$

a new search direction p_{k+1} and α_{k+1} are determined and the linear search is iterated.
If(4.11) is not satisfied, this subroutine goes the next step in which approximate inverse matrix of a Hessian matrix is improved.

– Convergence criterion
This subroutine terminates the iteration when the iterative vector x_k and x_{k+1} satisfy

$$\|x_{k+1} - x_k\|_\infty = \max\left(1.0, \|x_k\|_\infty\right) \cdot \mathrm{EPSR}$$

The obtained x_{k+1} is assumed to be the minimum point x^*.

For further details, see Reference [35].

A51-14-0101 MSBV, DMSBV

Multiplication of a real symmetric band matrix and a real vector.
CALL MSBV (A, N, NH, X, Y, ICON)

Function

This subroutine performs multiplication of an $n \times n$ real symmetric band matrix $\boldsymbol{A}$ with upper and lower band widths h by a vector $\boldsymbol{x}$

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{1.1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are both n-dimensional vectors, and $n > h \geq 0$.

Parameters

A	Input. Matrix $\boldsymbol{A}$. Matrix $\boldsymbol{A}$ is stored in one-dimensional array of size $n(h+1)-h(h+1)/2$ in the compressed mode for symmetric band matrices.
N	Input. Order n of the matrix $\boldsymbol{A}$. (See Notes.)
NH	Input. Upper and lower band widths h.
X	Input. Vector $\boldsymbol{x}$. One-dimensional array of size n.
Y	Output. Vector $\boldsymbol{y}$. One-dimensional array of size n.
ICON	Output. Condition code. See Table MSBV-1.

Table MSBV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N = 0, NH < 0 or NH ≥ |N|	Bypassed

Comments on use

- Subprograms used
  SSL II ..... MGSSL
  FORTRAN basic function .... IABS
- Notes
  This subroutine mainly consists of the computation

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{3.1}$$

but it can be changed to another type of computation,

$$\boldsymbol{y} = \boldsymbol{y}' - \boldsymbol{Ax} \tag{3.2}$$

by specifying N = $-n$ and giving an arbitrary vector $\boldsymbol{y}'$ to the parameter Y.

In practice, this method can be used to compute a residual vector of linear equations. (Refer to the example shown below.)

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax} \tag{3.3}$$

- Example
  The linear equations with an $n \times n$ real positive-definite symmetric band matrix

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.4}$$

is solved using subroutine LSBX and then the residual vector $\boldsymbol{b} - \boldsymbol{Ax}$ is compute with the resultant. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),X(100),
     *          Y(100),W(5050)
      READ(5,500) N,NH
      IF(N.EQ.0) STOP
      NH1=NH+1
      NT=N*NH1-NH*NH1/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (X(I),I=1,N)
      WRITE(6,600) N,NH
      L=1
      LE=0
      DO 10 I=1,N
      LE=LE+MIN0(I,NH1)
      JS=MAX0(1,I-NH1)
      WRITE(6,610) I,JS,(A(J),J=L,LE)
      L=LE+1
   10 CONTINUE
      WRITE(6,620) (I,X(I),I=1,N)
      EPSZ=1.0E-6
      ISW=1
      DO 20 I=1,N
      Y(I)=X(I)
   20 CONTINUE
      DO 30 I=1,NT
      W(I)=A(I)
   30 CONTINUE
      CALL LSBX(A,N,NH,X,EPSZ,ISW,ICON)
      IF(ICON.GE.20000) GOTO 50
      WRITE(6,630) (I,X(I),I=1,N)
      CALL MSBV(W,-N,NH,X,Y,ICON)
      IF(ICON.NE.0) GOTO 50
      WRITE(6,640) (I,Y(I),I=1,N)
      DN=0.0
      DO 40 I=1,N
      DN=DN+Y(I)*Y(I)
   40 CONTINUE
      DN=SQRT(DN)
      WRITE(6,650) DN
      STOP
   50 WRITE(6,660) ICON
      STOP
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1','COEFFICIENT MATRIX'
     */' ','N=',I5,3X,'NH=',I5)
  610 FORMAT(/5X,'(',I3,',',I3,')',4E17.8
     */(10X,4E17.8))
  620 FORMAT(/' ','CONSTANT VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  630 FORMAT(/' ','SOLUTION VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  640 FORMAT(/' ','RESIDUAL VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  650 FORMAT(/' ','NORM=',E17.8)
  660 FORMAT(/' ','ICON=',I5)
      END
```

Method

The multiplication $\boldsymbol{Y} = (\boldsymbol{y}_{ij})$ of a matrix $\boldsymbol{A}$ by a vector $\boldsymbol{X}$ is computed:

$$y_i = \sum_{j=1}^{n} a_{ij} x_j , \quad i = 1,...,n \qquad (4.1)$$

where $\boldsymbol{A}$ is $n \times n$ real symmetric band matrix with lower and upper band widths h and $\boldsymbol{x}$ is an n dimensional vector. While, this subroutine computes the multiplication using equation (4.2) instead of equation (4.1) by making use of symmetric band matrix characteristics.

$$y_i = \sum_{j=\max(1,i-h)}^{\min(i+h,n)} a_{ij} x_i , \quad i = 1,...,n \qquad (4.2)$$

This subroutine increases the precision of the inner products in equation (4.2) so that the effects of running error are minimized.

A-21-12-0401 MSGM, DMSGM

Multiplication of matrices (real symmetric by real general)
CALL MSGM (A, B, KB, C, KC, N, VW, ICON)

Function

This subroutine performs multiplication of an $n \times n$ real symmetric matrix $\boldsymbol{A}$ and an $n \times n$ real matrix $\boldsymbol{B}$.

$$\boldsymbol{C} = \boldsymbol{AB}$$

where, $\boldsymbol{C}$ is an $n \times n$ real matrix $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
B	Input. Matrix $\boldsymbol{B}$, two-dimensional array, B(KB, N)
KB	Input. The adjustable dimension of array B, (≥ N).
C	Output. Matrix $\boldsymbol{C}$, two-dimensional array, C(KC, N). (See "Comment on use".)
KC	Input. The adjustable dimension of array C, (≥ N).
N	Input. The order n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.
VW	Work area. One-dimensional array of size n
ICON	Output. Condition codes. See Table MSGM-1.

Table MSGM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N<1, KB<N or KC<N	Bypassed

Comments on use

- Subprograms used
  SSL II ... CSGM, MGGM, MGSSL
  FORTRAN basic function ... None
- Notes
  Saving the storage area:
  If there is no need to keep the contents on the array A, more storage area can be saved by using the EQUIVALENCE statement as follows:

  EQUIVALENCE (A(1), C(1.1))

  Refer to the example shown in "Comments on use" below.
- Example
  The following shows an example of obtaining the multiplication of a real symmetric matrix $\boldsymbol{A}$ by a real matrix $\boldsymbol{B}$, Here, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100,100),
     *C(100,100),VW(100)
      EQUIVALENCE (A(1),C(1,1))
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
   10 READ(5,100) N
      IF(N.EQ.0) STOP
      WRITE(6,150)
      NT=N*(N+1)/2
      READ(5,200) (A(I),I=1,NT)
      READ(5,200) ((B(I,J),J=1,N),I=1,N)
      CALL PSM(IA,1,A,N)
      CALL PGM(IB,1,B,100,N,N)
      CALL MSGM(A,B,100,C,100,N,VW,ICON)
      WRITE(6,250) ICON
      IF(ICON.NE.0) GOTO 10
      CALL PGM(IC,1,C,100,N,N)
      GOTO 10
  100 FORMAT(I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX MULTIPLICATION **')
  250 FORMAT(//10X,'** MSGM ICON=',I5)
      END
```

Subroutines PSM and PGM in the example are for printing the real symmetric and real matrices. These programs are shown in the example for subroutine MGSM.

A21-12-0301 MSSM, DMSSM

Multiplication of two matrices (real symmetric by real symmetric)
CALL MSSM (A, B, C, KC, N, VW, ICON)

Function

The subroutine performs multiplication of two $n \times n$ real symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

$$\boldsymbol{C} = \boldsymbol{AB}$$

where, C is an $n \times n$ real matrix, $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
B	Input. Matrix $\boldsymbol{B}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
C	Output. Matrix $\boldsymbol{C}$, two-dimensional array, C(KC, N). (See "Notes".)
KC	Input. The adjustable dimension of array C, (≥ N).
N	Input. The order n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.
VW	Work area. One-dimensional array of size n.
ICON	Output. Condition codes. See Table MSSM-1.

Table MSSM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N<1 or KC<N	Bypassed

Comments on use

- Subprograms used
  SSL II ... CSGM, MGSM, and MGSSL.
  FORTRAN basic function ... None

- Notes
  Saving the storage area:
  If there is no need to keep the contents on the array A, more storage area can be saved by using the EQUIVALENCE statement as follows:

  EQUIVALENCE (A(1), C(1,1))

  Refer to the example shown in "Comments on use" below.

- Example
  The following shows an example of obtaining the multiplication real symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.
  Here, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),C(100,100),
     *VW(100)
      EQUIVALENCE (A(1),C(1,1))
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
   10 READ(5,100) N
      IF(N.EQ.0) STOP
      WRITE(6,150)
      NT=N*(N+1)/2
      READ(5,200) (A(I),I=1,NT)
      READ(5,200) (B(I),I=1,NT)
      CALL MSSM(A,B,C,100,N,VW,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PSM(IA,1,A,N)
      CALL PSM(IB,1,B,N)
      CALL PGM(IC,1,C,100,N,N)
      GOTO 10
  100 FORMAT(I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX MULTIPLICATION **')
      END
```

The subroutines PSM and PGM in the example are for printing the real symmetric and real matrices. These programs are shown in the example for subroutine MGSM.

A21-14-0101 MSV, DMSV

Multiplication of a real symmetric matrix and a real vector.
CALL MSV (A, N, X, Y, ICON)

Function

This subroutine performs multiplication of an $n \times n$ real symmetric matrix $\boldsymbol{A}$ and a vector $\boldsymbol{x}$.

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{1.1}$$

where, $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-dimensional vectors, $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$, in the compressed mode, one-dimensional array of size $n(n+1)/2$.
N	Input. The order n of matrix $\boldsymbol{A}$.
X	Input. Vector $\boldsymbol{x}$, one-dimensional array of size n.
Y	Output. Multiplication $\boldsymbol{y}$ of matrix $\boldsymbol{A}$ and vector $\boldsymbol{x}$, one-dimensional array of size n.
ICON	Output. Condition codes. See Table MSV-1.

Table MSV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N = 0	Bypassed

Comments on use

- Subprograms used
  SSL II .... MGSSL
  FORTRAN basic function ....IABS.
- Notes
  This subroutine mainly consists of the computation,

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{3.1}$$

but it can be changed to another type of computation,

$$\boldsymbol{y} = \boldsymbol{y}' - \boldsymbol{Ax} \tag{3.2}$$

by specifying N = – n and giving an arbitrary vector $\boldsymbol{y}'$ to the parameter Y.
This method can be used to compute a residual vector of linear equations such as

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax} \tag{3.3}$$

Refer to the example in “Comments on use” below.

- Example
  This example shows the program to solve a system of linear equations (3.4) by subroutine LSX and to obtain a residual vector $\boldsymbol{b} - \boldsymbol{Ax}$ based on the solution. Where $n \leq 100$.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3.4}$$

```
C     **EXAMPLE**
      DIMENSION A(5050),X(100),
     *          Y(100),W(5050)
      READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (X(I),I=1,N)
      WRITE(6,600) N
      L=1
      LE=0
      DO 10 I=1,N
      LE=LE+I
      WRITE(6,610) I,(A(J),J=L,LE)
      L=LE+1
   10 CONTINUE
      WRITE(6,620) (I,X(I),I=1,N)
      EPSZ=1.0E-6
      ISW=1
      DO 20 I=1,N
      Y(I)=X(I)
   20 CONTINUE
      DO 30 I=1,NT
      W(I)=A(I)
   30 CONTINUE
      CALL LSX(A,N,X,EPSZ,ISW,ICON)
      WRITE(6,630) (I,X(I),I=1,N)
      CALL MSV(W,-N,X,Y,ICON)
      IF (ICON.NE.0) GO TO 50
      WRITE(6,640) (I,Y(I),I=1,N)
      DN=0.0
      DO 40 I=1,N
      DN=DN+Y(I)*Y(I)
   40 CONTINUE
      DN=SQRT(DN)
      WRITE(6,650) DN
   50 WRITE(6,660) ICON
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1','COEFFICIENT MATRIX'
     */' ','ORDER=',I5)
  610 FORMAT(/5X,'(',I3,')',4E17.8/
     *(10X,4E17.8))
  620 FORMAT(/' ','CONSTANT VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  630 FORMAT(/' ','SOLUTION VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  640 FORMAT(/' ','RESIDUAL VECTOR'
     */(5X,4('(',I3,')',E17.8,5X)))
  650 FORMAT(/' ','NORM=',E17.8)
  660 FORMAT(/' ','ICON=',I5)
      END
```

Method

This subroutine performs multiplication $\boldsymbol{y} = (y_i)$ of an $n \times n$ real matrix $\boldsymbol{A} = (a_{ij})$ and an n dimensional vector $\boldsymbol{x}$ = (x_j) through using the equation (4.1).

$$\boldsymbol{y}_i = \sum_{j=1}^{n} a_{ij} x_j , \quad i = 1,...,n \tag{4.1}$$

In this subroutine, precision of the inner products in (4.1) has been raised to minimize the effect of rounding errors.

I11-91-0101 NDF, DNDF

Normal distribution function $\phi(x)$
CALL NDF (X, F, ICON)

Function

This subroutine computes the value of normal distribution function $\phi(x)=\frac{1}{\sqrt{2\pi}}\int_0^x e^{-\frac{t^2}{2}}dt$ by the relation.

$$\phi(x)=\operatorname{erf}\left(x/\sqrt{2}\right)/2 \tag{1.1}$$

Parameters

X	Input. Independent variable ***x***.
F	Output. Function value $\phi(x)$
ICON ..	Output. Condition code See Table NDF-1.

Table NDF-1 Condition code

Code	Meaning	Processing
0	No error	

Comments on use

- Subprograms used
  SSL II ...MGSSL
  FORTRAN basic function ...ERF

- Notes
  There is no restriction with respect to the range of argument X.
  Using the relationship between normal distribution function $\phi(x)$ and complementary normal distribution function $\psi(x)$

$$\phi(x)=1/2-\psi(x) \tag{3.1}$$

the value of $\phi(x)$ can be computed by using subroutine NDFC. Note that in the range of $|x|>2$, however, this leads to less accurate and less efficient computation than calling NDF.

- Example
  The following example generates a table of $\phi(x)$ in which *x* varies from 0.0 to 10.0 with increment 0.1.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,101
      X=FLOAT(K-1)/10.0
      CALL NDF(X,F,ICON)
      WRITE(6,610) X,F
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF NORMAL DISTRI',
     *'BUTION FUNCTION'//
     *6X,'X',7X,'NDF(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
      END
```

Method

Normal distribution function:

$$\phi(x)=\frac{1}{\sqrt{2\pi}}\int_0^x e^{-\frac{t^2}{2}}dt \tag{4.1}$$

can be written with variable transformation $t/\sqrt{2}=u$ as

$$\phi(x)=\frac{1}{\sqrt{2\pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-u^2}\sqrt{2}du$$
$$=\frac{1}{2}\frac{2}{\sqrt{\pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-u^2}du$$

Therefore from $\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-u^2}du$

the following holds.

$$\phi(x)=\frac{1}{2}\operatorname{erf}\left(x/\sqrt{2}\right) \tag{4.2}$$

This subroutine computes $\phi(x)$ from (4.2) by using FORTRAN function ERF.

I11-91-0201 NDFC, DNDFC

Complementary normal distribution function $\psi(x)$
CALL NDFC (X, F, ICON)

Function

This subroutine computes the value of complementary normal distribution function.

$$\psi(x)=\frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-\frac{t^2}{2}}dt$$

by the relation ship,

$$\psi(x)=\operatorname{erfc}\left(x/\sqrt{2}\right)/2 \qquad (1.1)$$

Parameters

X	Input. Independent variable *x*.
F	Output. Function value $\psi(x)$
ICON ..	Output. Condition code. See Table NDFC-1.

Table NDFC-1 Condition code

Code	Meaning	Processing
0	No error	

Comments on use

- Subprogram used
  SSL II ... MGSSL
  FORTRAN basic function ... ERFC

- Notes
  There is no restrictions in the range of argument X.
  Using the relationship between normal distribution function $\phi(x)$ and complementary normal distribution function $\psi(x)$.

$$\psi(x)=\frac{1}{2}-\phi(x) \qquad (3.1)$$

the value of $\psi(x)$ can be computed by using subroutine NDF. Note that in the range of $|x|>2$, however, this leads to less accurate and less efficient computation that calling NDFC.

- Example
  The following example generates a table of $\psi(x)$ in which *x* varies from 0.0 to 10.0 with increment 0.1.

```
C     **EXAMPLE**
      WRITE(6,600)
      DO 10 K=1,101
      X=FLOAT(K-1)/10.0
      CALL NDFC(X,F,ICON)
      WRITE(6,610) X,F
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF COMPLEMENTARY',
     *' NORMAL DISTRIBUTION FUNCTION'
     *//6X,'X',7X,'NDFC(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
      END
```

Method

Complementary normal distribution function:

$$\psi(x)=\frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-\frac{t^2}{2}}dt \qquad (4.1)$$

can be written with variable transformation $t/\sqrt{2}=u$, as

$$\begin{aligned}\psi(x)&=\frac{1}{\sqrt{2\pi}}\int_{\frac{x}{\sqrt{2}}}^\infty e^{-u^2}\sqrt{2}du\\&=\frac{1}{2}\frac{2}{\sqrt{\pi}}\int_{\frac{x}{\sqrt{2}}}^\infty e^{-u^2}du\end{aligned}$$

Therefore from $\operatorname{erfc}(x)=\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-u^2}du$ the following holds.

$$\psi(x)=\frac{1}{2}\operatorname{erfc}\left(x/\sqrt{2}\right) \qquad (4.2)$$

This subroutine computes $\psi(x)$ from (4.2) by using FORTRAN function ERFC.

D31-20-0101-NLPG1, DNLPG1

Nonlinear programming (Powell's method using function values and its derivatives)
CALL NLPG1 (X, N, FUN, GRAD, FUNC, JAC, M, EPSR, MAX, F, VW, K, IVW, ICON)

Function

Given an n-variable real function $f(\boldsymbol{x})$, its derivative $\boldsymbol{g}(\boldsymbol{x})$, and initial vector $\boldsymbol{x}_0$, vector $\boldsymbol{x}^*$ which minimizes $f(\boldsymbol{x})$ and the value of function $f(\boldsymbol{x}^*)$ are obtained subject to the constrains

$$c_i(\boldsymbol{x}) = 0, \quad i = 1, 2,...,m_1 \qquad (1.1)$$
$$c_i(\boldsymbol{x}) \geq 0, \quad i = m_1+1, m_1+2,..., m_1+m_2 \qquad (1.2)$$

The Jacobian $\boldsymbol{J}(\boldsymbol{x})$ of $\{c_i(\boldsymbol{x})\}$ is given as a function and $f(\boldsymbol{x})$ is assumed to have up to the second continuous derivative.
Further, $\boldsymbol{x} = (x_1, x_2,...,x_n)^T$ and m_1 and m_2 are the number of the equality and inequality constraints, where $n \geq 1$, $m_1 \geq 0$, $m_2 \geq 0$, and $m \geq 1$ $(m = m_1 + m_2)$.

Parameters

X ..... Input. Initial vector $\boldsymbol{x}_0$.
Output. Vector $\boldsymbol{x}^*$.
One-dimensional array of size n.

N ..... Input. Number n of variables.

FUN ..... Input. Name of function subprogram which calculates $f(\boldsymbol{x})$
The form of subprogram is as follows:
FUNCTION FUN (X)
Parameters
X ... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
Substitute the value of $f(\boldsymbol{x})$ in function FUN.
(See "Example.")

GRAD. Input. Name of subroutine subprogram which calculates $\boldsymbol{g}(\boldsymbol{x})$.
The form of subprogram is as follows:
SUBROUTINE GRAD (X, G)
Parameters
X ... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
G ... Output. One-dimensional array of size n, where G(1) = $\partial f/\partial x_1$,..., G(N) = $\partial f/\partial x_n$
(See "Example.")

FUNC. Input. Name of subroutine subprogram which calculates $c_i(\boldsymbol{x})$.
The form of subprogram is as follows:
SUBROUTINE FUNC (X, C)
Parameters
X ... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
C ... Output. One-dimensional array of size m, where C(1) = $c_1(\boldsymbol{x})$, ..., C(M(1)) = $c_{m_1}(\boldsymbol{x})$, ..., C(M(1) + M(2)) = $c_m(\boldsymbol{x})$.
(See "Example.")

JAC .... Input. Name of subroutine subprogram which calculates $\boldsymbol{J}(\boldsymbol{x})$.
The form of subprogram is as follows:
SUBROUTINE JAC (X, CJ, K)
Parameters
X … Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
CJ ... Output. Jacobian matrix.
Two-dimensional array, CJ (K, N), where CJ(I,J) = $\partial c_i/\partial x_j$
K ... Input. Adjustable dimension of array CJ.
(See "Example.")

M ..... Input. The number of constraints.
One-dimensional array of size 2, where M(1) = m_1 and M(2) = m_2.

EPSR .. Input. Convergence criterion (≥ 0.0).
The default value is used if 0.0 is specified.
(See "Comments on Use.")

MAX .. Input. The upper limit (≠0) of number of evaluations for functions $f(\boldsymbol{x})$, $\boldsymbol{g}(\boldsymbol{x})$, $\boldsymbol{c}(\boldsymbol{x})$, and $\boldsymbol{J}(\boldsymbol{x})$.
(See "Comments on Use.")
Output. The number (>0) of actual evaluations

F ..... Output. The value of function $f(\boldsymbol{x}^*)$

VW ..... Work area. VW is two-dimensional array, VW(K, M(1)+M(2)+2×N+12).

K .... Input. Adjustable dimension (≥ M(1)+M(2)+N+4) of array VW.

IVW ..... Work area. One-dimensional array of size 2×(M(1)+M(2)+N+4).

ICON ..... Output. Condition code.
(See Table NLPG1-1)

Table NLPG1-1 Condition codes

Code	Meaning	Processing
0	No error.	
10000	The convergence condition has not been satisfied within the specified function evaluation count.	The last values obtained are stored in X and F.
20000	Local decrement of the function was not satisfied during the calculation. (See " Method.") The value of EPSR is too small.	Bypassed. (The last values obtained are stored in X and F.)
21000	There may not be a solution that satisfies the constraints, or the initial value x_0, is not appropriate. Retry with a different initial value x_0.	Bypassed.
30000	N < 1, EPSR < 0.0, M(1) < 0, M(2) <0, K < M (1) + M(2) + N + 4, MAX = 0.	Bypassed.

Comments on use

- Subprograms used
  SSL II ... AMACH, UQP, UNLPG, MGSSL
  FORTRAN basic functions ... ABS, AMAX1,

- Notes
  An EXTERNAL statement is necessary to declare the subprogram names correspond to parameters FUN, GRAD, FUNC and JAC in the calling program.

  EPSR
  In this subroutine, the convergence condition is checked as follows: During iteration, if

  $$\left\|x_{k+1}-x_k\right\|_\infty \le \max\left(1.0,\left\|x_k\right\|_\infty\right)\cdot \text{EPSR}$$

  is satisfied, point $\boldsymbol{x}_{k+1}$ is assumed to be minimum point $\boldsymbol{x}^*$ and iteration is stopped.
  Since $f(\boldsymbol{x})$ is assumed to be approximately a quadratic function in the vicinity of point $\boldsymbol{x}^*$, it is appropriate to specify EPSR as EPSR $\approx \sqrt{u}$, where u is the unit round off to obtain the value of function $f(\boldsymbol{x}^*)$ as accurate as the rounding error. The default value of EPSR is $2\sqrt{u}$

  MAX
  The number of function evaluation is incremented by one every time $f(\boldsymbol{x})$ is evaluated, by n every time $\boldsymbol{g}(\boldsymbol{x})$ is evaluated, by m every time $\boldsymbol{c}(\boldsymbol{x})$ is evaluated, and by mn every time $\boldsymbol{J}(\boldsymbol{x})$ is evaluated.
  The number depends on characteristics of the functions, initial vector, and convergence criterion.
  Generally, when an appropriate initial vector is specified and the default value is used for the convergence criterion, it is adequate to specify MAX = $800 \cdot mn$.
  Even if the convergence condition is not satisfied within the specified evaluation count and the subroutine is returned with ICON = 10000, iteration can be resumed by calling this subroutine again. In this case, the user must specify a negative value as the additional evaluation count in the parameter MAX and retain other parameters unchanged.

- Example
  Given the following 2-variable real function

$$f(x_1,x_2)=x_1^2-2x_1x_2+2x_2^2-10x_1+x_2$$

the vector which minimizes function and the value of $f(\boldsymbol{x}^*)$ are obtained subject to the following constraints:

$$c_1(x_1,x_2)=0.5x_1^2+1.5x_2^2-2=0$$
$$c_1(x_1,x_2)=-x_1+x_2\ge 0$$

where the initial vector is $\boldsymbol{x}_0 = (-2, 2)^T$

```
C     **EXAMPLE**
      DIMENSION X(2),M(2),VW(8,18),
     *          IVW(16)
      EXTERNAL TEST,GRAD,TESTC,JAC
      X(1)=-2.0
      X(2)=2.0
      N=2
      M(1)=1
      M(2)=1
      EPSR=1.0E-3
      MAX=800*2*2
      K=8
      CALL NLPG1(X,N,TEST,GRAD,TESTC,
     *     JAC,M,EPSR,MAX,F,VW,K,IVW,ICON)
      WRITE(6,600) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,610) F,MAX
      WRITE(6,620) (I,X(I),I=1,N)
      STOP
  600 FORMAT('1','*ICON=',I5)
  610 FORMAT(' ','*F=',E15.7,1X,'MAX=',I5)
  620 FORMAT('0',(/2X,'X(',I2,')=',E15.7))
      END
C     OBJECTIVE FUNCTION
      FUNCTION TEST(X)
      DIMENSION X(2)
      TEST=(X(1)-2.0*X(2)-10.0)*X(1)+
     *    (2.0*X(2)+1.0)*X(2)
      RETURN
      END
C     DERIVATIVE
      SUBROUTINE GRAD(X,G)
      DIMENSION X(2),G(2)
      G(1)=2.0*X(1)-2.0*X(2)-10.0
      G(2)=-2.0*X(1)+4.0*X(2)+1.0
      RETURN
      END
C     CONSTRANTS
      SUBROUTINE TESTC(X,C)
      DIMENSION X(2),C(2)
      C(1)=0.5*X(1)*X(1)+1.5*X(2)*X(2)-2.0
      C(2)=-X(1)+X(2)
      RETURN
      END
```

```
C      JACOBIAN
       SUBROUTINE JAC(X,CJ,K)
       DIMENSION X(2),CJ(K,2)
       CJ(1,1)=X(1)
       CJ(2,1)=-1.0
       CJ(1,2)=3.0*X(2)
       CJ(2,2)=1.0
       RETURN
       END
```

Method

This subroutine solves a nonlinear programming problem given as

$$f(x) \longrightarrow \text{minimize} \tag{4.1}$$

subject to the constraints

$$c_i(\boldsymbol{x}) = 0\ ,\quad i = 1, 2, ..., m_1 \tag{4.2}$$

(equality constraints)

$$c_i(\boldsymbol{x}) \geq 0\ ,\quad i = m_1+1,...,m_1+m_2 \tag{4.3}$$

(inequality constraints)

using Powell's variable metric method. Let us introduce the following symbols for simplicity:

M1 ..... Set $(1,2,...,m_1)$ of subscripts of the equality constraints. (This may be an empty set.)

M2 ..... Set $(m_1+1,...,m_1+m_2)$ of subscripts of the inequality constraints. (This may be and empty set.)

M ..... Set $(1, 2,...,m_1+m_2)$ of subscripts of constraints. (This must not be an empty set.)

$\boldsymbol{g}$ ..... Gradient vector of f

∇c_i .. Gradient vector of c_i.

Let us explain outline of the algorithm for the problem by comparing with that for unconstrained minimization problem.

The revised quasi-Newton method, that is, the variable metric method to minimize the objective function $f(\boldsymbol{x})$ without constraints such as (4.2) and (4.3), is described as follows. Function $f(\boldsymbol{x})$ can be approximated by quadratic function at an arbitrary point $\boldsymbol{x_k}$, in the region of the minimum point as follows:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x_k}) + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{g}(\boldsymbol{x_k})\frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{By} \tag{4.4}$$

where

$$\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{x}_k \tag{4.5}$$

$\boldsymbol{B}$ is the Hessian matrix of $f(\boldsymbol{x})$ for $\boldsymbol{x}_{\mathrm{k}}$. The value of $\boldsymbol{y}$ that minimizes function (4.4) is computed and the solution is defined as $\boldsymbol{y}_{\mathrm{k}}$. Then, the linear search is a applied to obtain the value of a that satisfies

$$\min_{\alpha} f(\boldsymbol{x}_k + \alpha\boldsymbol{y}_k) \tag{4.6}$$

this value is defined as α_k Substituting this value α_k , in expression

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{y}_k \tag{4.7}$$

the better approximation $\boldsymbol{x}_{k+1}$ is obtained.

On such process, Hessian matrix $\boldsymbol{B}$ is not calculated directly but is approximated using vectors $\boldsymbol{g}(\boldsymbol{x}_{k+1}) - \boldsymbol{g}(\boldsymbol{x}_k)$ and $\alpha_k \boldsymbol{y}_k$ during iteration.

On the other hand, for the minimization with constraints such as (4.2) and (4.3), the value of $\boldsymbol{y}$ is obtained under the constraints. For (4.2), the condition of the linear approximation.

$$c_i(\boldsymbol{x}) = c_i(\boldsymbol{x}_k) + \boldsymbol{y}^{\mathrm{T}}\nabla c_i(\boldsymbol{x}_k) = 0,\quad i \in M_1 \tag{4.8}$$

is imposed, whereas for (4.3), the condition

$$c_i(\boldsymbol{x}) = c_i(\boldsymbol{x}_k) + \boldsymbol{y}^{\mathrm{T}}\nabla c_i(\boldsymbol{x}_k) \geq 0,\quad i \in M_2 \tag{4.9}$$

is imposed.

It is therefore necessary to obtain the value y that minimizes (4.4) satisfying the conditions of (4.8) and (4.9) in order to solve the nonlinear programming problem given by (4.1), (4.2), and (4.3). This is a quadratic programming problem with respect to $\boldsymbol{y}$. Concerning the linear search, the penalty function

$$W(\boldsymbol{x}) = f(\boldsymbol{x}) + P[\boldsymbol{c}(\boldsymbol{x})] \tag{4.10}$$

should be applied instead of (4.6), namely the value of α that minimizes $W(\boldsymbol{x})$ is obtained. Where the function P[$\boldsymbol{c}(\boldsymbol{x})$] takes zero if all constraints are satisfied; otherwise, it takes a positive value. This will be explained precisely later. Further, information about not only $f(\boldsymbol{x})$ but also $\boldsymbol{c}(\boldsymbol{x})$ are incorporated to update approximation matrix $\boldsymbol{B}_k$ of the Hessian matrix.

If only the equality constraint is imposed, the following must be satisfied at the minimum point:

$$\boldsymbol{g}(\boldsymbol{x}) - \sum_{i \in M_1} \lambda_i \nabla c_i(\boldsymbol{x}) = 0 \tag{4.11}$$

where λ_i , is the Lagrange multiplier.

To solve (4.11) which is a simultaneous nonlinear equations of order $n+m_1$, the second partial derivatives of

$$\phi(\boldsymbol{x},\lambda) = f(\boldsymbol{x}) - \sum_{i \in M_1} \lambda_i c_i(\boldsymbol{x}) \tag{4.12}$$

are necessary. This means that (4.11) cannot be solved with only the partial derivatives of $f(\boldsymbol{x})$.

Updating of $\boldsymbol{B}_k$ is therefore performed based on the matrix of rank 2 obtained from vectors $\alpha_k \boldsymbol{y}_k$ and

$$\boldsymbol{\gamma}_k = \nabla_x \phi(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}) - \nabla_x \phi(\boldsymbol{x}_k, \boldsymbol{\lambda}) \qquad (4.13)$$

Where $\nabla_x \phi$ is the gradient vector of ϕ with respect to $\boldsymbol{x}$. If the problem has nonlinear equality constraints, the exact linear search for $W(\boldsymbol{x})$ causes extreme slow step size of $\boldsymbol{x}_k$, as a result, it does not converge to the actual minimum point in some cases; therefore, a watchdog should be introduced to watch behavior of solution $\boldsymbol{x}_k$.

Explanations about calculation procedures of this subroutine are given in the next subsection, where the functions are used which are defined as follows:

- Penalty function

$$\begin{aligned} W(\boldsymbol{x}) = f(\boldsymbol{x}) &+ \sum_{i \in M_1} \mu_i |c_i(\boldsymbol{x})| \\ &+ \sum_{i \in M_2} \mu_i |\min(0, c_i(\boldsymbol{x}))| \end{aligned} \qquad (4.14)$$

The function increases effect of the penalty term as the point $\boldsymbol{x}$ takes out of the constraints. Coefficient μ_i is determined according to the method explained later.

- Linear approximation of the penalty function this is defined as linear approximation at an arbitrary point $\boldsymbol{x}_k$, in the region of the minimum point

$$\begin{aligned} W_k(\boldsymbol{x}) = {} & f(\boldsymbol{x}_k) + (\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}_k) \\ & + \sum_{i \in M_1} \mu_i \left| c_i(\boldsymbol{x}_k) + (\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k) \right| \\ & + \sum_{i \in M_2} \mu_i \left| \min\left(0, c_i(\boldsymbol{x}_k) + (\boldsymbol{x} - \boldsymbol{x}_k)^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k)\right) \right| \end{aligned}$$

(4.15)

Lagrangian function

$$L_k(\boldsymbol{x}) = f(\boldsymbol{x}) - \sum_{i \in M_1} \lambda_i c_i(\boldsymbol{x}) \qquad (4.16)$$

where λ_i is the Lagrange multiplier determined from $\boldsymbol{x}_k$, as explained later.

Calculation procedures

Step 1 (initial value setting)

1) Sets the following values:
   $k = 0$ (iteration count)
   $\boldsymbol{H}_0 = \boldsymbol{I}_n$
   $l = 0$ (watchdog location)
   $lt = 5$ (interval to check watchdog location)
   $\theta = 0.25$

$$\varepsilon = \begin{cases} \text{EPSR, EPSR} \neq 0 & \text{(conversion judg-} \\ 2\sqrt{u}, \text{ EPSR} = 0 & \text{ment criterion)} \end{cases}$$

Step 2 (quadratic programming problem)

2) Solves the quadratic programming problem QP$_k$ with respect to $\boldsymbol{y}$ as follows:

$$\left.\begin{array}{l} \frac{1}{2}\boldsymbol{y}^{\mathrm{T}} \boldsymbol{B}_k \boldsymbol{y} + \boldsymbol{y}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}_k) \longrightarrow \text{Minimize} \\ \text{Subject to constraints} \\ \boldsymbol{y}^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k) + c_i(\boldsymbol{x}_k) = 0 \quad , i \in M_1 \\ \boldsymbol{y}^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k) + c_i(\boldsymbol{x}_k) \geq 0 \quad , i \in M_2 \end{array}\right\} \qquad (4.17)$$

3) When there is an optimal solution $\boldsymbol{y}_k$ for QP$_k$, if

$$\|y_k\|_\infty < \max\left(1.0, \|x_k\|_\infty\right) \cdot \varepsilon \qquad (4.18)$$

is satisfied, where $\boldsymbol{x}_k$, is a feasible point, assumes $\boldsymbol{x}_k$ to be $\boldsymbol{x}^*$ and $f(\boldsymbol{x}_k)$ to be $f(\boldsymbol{x}^*)$, and sets ICON = 0, then stops processing; otherwise, the linear search for $W(\boldsymbol{x})$ is performed to obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha^* \boldsymbol{y}_k \qquad (4.19)$$

in the $\boldsymbol{y}_k$ direction.

If $\left\|\alpha^* \boldsymbol{y}_k\right\|$ is small enough to be considered as convergence and if $\boldsymbol{x}_k$ is a feasible point, ICON = 20000 is set and processing is stopped. This is because the convergence criterion ε is too small, but the point can be considered to be the minimum point. If $\boldsymbol{x}_k$ is not a feasible point, ICON = 21000 is set and processing is stopped.

If $\left\|\alpha^* \boldsymbol{y}_k\right\|$ is large, proceeds to step 3.

4) When there is no feasible solution for QP$_k$: The problem is modified to the following quadratic programming problem:

QP$_k$:

$$\left.\begin{array}{l} \frac{1}{2}\boldsymbol{y}^{\mathrm{T}} B_k \boldsymbol{y} + \boldsymbol{y}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}_k) - \beta z \rightarrow \text{Minimize} \\ \text{Subject to constraints} \\ \boldsymbol{y}^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k) + c_i(\boldsymbol{x}_k) z = 0, \quad i \in M_1 \\ \boldsymbol{y}^{\mathrm{T}} \nabla c_i(\boldsymbol{x}_k) + c_i(\boldsymbol{x}_k) z_i \geq 0, \quad i \in M_2 \\ \text{where,} \\ 0 \leq z \leq 1, \\ z_i = \begin{cases} 1, & c_i(\boldsymbol{x}_k) > 0, \\ z, & c_i(\boldsymbol{x}_k) < 0, \end{cases} \end{array}\right\} \qquad (4.20)$$

β is a sufficiently large positive value

Denotes the optimal solution as $\boldsymbol{y}_k$ and $\bar{z}$.

If $\bar{z} < \varepsilon$ and if $\boldsymbol{x}_k$ is a feasible point assumes $\boldsymbol{x}_k$ to be $\boldsymbol{x}^*$ and $f(\boldsymbol{x}_k)$ to be $f(\boldsymbol{x}^*)$ and sets ICON =0, then stops processing.

If $\bar{z} < \varepsilon$ and if $\boldsymbol{x}_k$ is not a feasible point, there is no feasible solution for the nolinear programming problem given as (4.1), (4.2), and (4.3) (constraints conflict with each other), or the initial value $\boldsymbol{x}_0$ is not appropriate, thus ICON = 21000 is set and processing is stopped.

If $\bar{z} \geq \varepsilon$ and if

$$\|\boldsymbol{y}_k\|_\infty < \max\left(1.0, \|\boldsymbol{x}_k\|_\infty\right) \cdot \varepsilon \tag{4.21}$$

is satisfied, and if $\boldsymbol{x}_k$ is a feasible point; assumes $\boldsymbol{x}_k$ to be $\boldsymbol{x}^*$ and $f(\boldsymbol{x}_k)$ to be $f(\boldsymbol{x}^*)$, and sets ICON = 0, then stops processing. If (4.21) is not satisfied, proceeds to step 3.

Step 3 (watchdog processing)

5) Assume $k = k + 1$
   Checks behavior of solution $\boldsymbol{x}_k$ to judge whether the step size of $\boldsymbol{x}_k$ is appropriate.
   If

$$W\left(\boldsymbol{x}_k\right) \leq W\left(\boldsymbol{x}_l\right) - \theta\left(W\left(\boldsymbol{x}_l\right) - W_l\left(\boldsymbol{x}_{l+1}\right)\right) \tag{4.22}$$

   is satisfied, proceeds to step 5.

Step 4(watchdog and $\boldsymbol{B}_k$ updating)

6) If

$$W\left(\boldsymbol{x}_k\right) \leq W\left(\boldsymbol{x}_l\right)$$

   is satisfied, assumes $l = k$ and $\boldsymbol{x}_l = \boldsymbol{x}_k$.
7) If the value of k is a multiple of lt, assumes $\boldsymbol{x}_k = \boldsymbol{x}_l$ and $l = k$.
8) Updates $\boldsymbol{B}_k$ as follows. Using

$$\left.\begin{aligned} &\boldsymbol{\delta} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}\left(= \alpha^* \boldsymbol{y}_{k-1}\right) \\ &\boldsymbol{\gamma} = \boldsymbol{g}_k - \boldsymbol{g}_{k-1} \\ &\qquad - \sum_{i \in M} \lambda_i \left(\nabla c_i\left(\boldsymbol{x}_k\right) - \nabla c_i\left(\boldsymbol{x}_{k-1}\right)\right) \end{aligned}\right\} \tag{4.23}$$

   determines

$$\xi = \begin{cases} 1 & , \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma} \geq 0.2\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}\boldsymbol{\delta} \\ 0.8\dfrac{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}\boldsymbol{\delta}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}\boldsymbol{\delta} - \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma}} & , \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma} < 0.2\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}\boldsymbol{\delta} \end{cases} \tag{4.24}$$

   with the value obtained as ξ, calculates $\boldsymbol{\eta}$ as follows:

$$\boldsymbol{\eta} = \xi\boldsymbol{\gamma} + (1 - \xi)\boldsymbol{B}_{k-1}\boldsymbol{\delta} \tag{4.25}$$

   Substituting these values in (4.26), $\boldsymbol{B}_k$ is obtained from

$$\boldsymbol{B}_k = \boldsymbol{B}_{k-1} - \frac{\boldsymbol{B}_{k-1}\boldsymbol{\delta}\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{B}_{k-1}\boldsymbol{\delta}} + \frac{\boldsymbol{\eta}\boldsymbol{\eta}^{\mathrm{T}}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\eta}} \tag{4.26}$$

   Then, returns to step 2.

Step 5 (modification of step size)

9) Checks the value of α^* obtained at 3) as follows:
   Obtains $\boldsymbol{x}$ that satisfies the following in the $\boldsymbol{y}_k$ direction:

$$\left.\begin{aligned} & W(\boldsymbol{x}) \leq W(\boldsymbol{x}_{k-1}) \\ \text{or} \quad & \\ & L(\boldsymbol{x}) \geq L(\boldsymbol{x}_{k-1}) \end{aligned}\right\} \tag{4.27}$$

   If there is no $\boldsymbol{x}$ that satisfies (4.27), returns to step 4.
   If there is an $\boldsymbol{x}$ that satisfies (4.27),
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \tilde{\alpha}\,\boldsymbol{y}_k$ is assumed.
   If $\tilde{\alpha} = \alpha^*$, returns to step 4; otherwise returns to step 3.

Notes on algorithm

1) Determination of λ_i
   Determines λ_i, from solutions of QP$_k$ using the Lagrange multiplier for the constraints.
2) Determination of μ_i
3) The initial value is specified as μ_i=2|λ_i|, then μ_i=2|λ_i| is set for μ_i, that satisfies μ_i<1.5|λ_i| (the current value is used for λ_i) thereafter.

(See references [94] and [95] for details.)

C24-11-0101 NOLBR, DNOLBR

Solution of a system of nonlinear equations (Brent's Method)
CALL NOLBR (X, N, FUN, EPSZ, EPST, FC, M, FNOR, VW, ICON)

Function

This subroutine solves a system of nonlinear equations (1.1) by Brent's method.

$$\left.\begin{aligned} f_1(x_1,x_2,...,x_n)=0 \\ f_2(x_1,x_2,...,x_n)=0 \\ : \\ f_n(x_1,x_2,...,x_n)=0 \end{aligned}\right\} \tag{1.1}$$

That is, let $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}),..., f_n(\boldsymbol{x}))^T$ and $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$, then eq. (1.2) is solved from the initial value x_0.

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \tag{1.2}$$

where $\boldsymbol{0}$ is an n-order zero vector.

Parameters

X ..... Input. An initial vector $\boldsymbol{x}_0$ to solve equation (1.2).
Output. Solution vector. One dimensional array of size n.

N ..... Input. Dimension n of the system.

FUN ..... Input. The name of the function subprogram which evaluates the function $f_k(\boldsymbol{x})$. FUN must be declared as EXTERNAL in the program from which this subroutine is called. Its specification is:
FUNCTION FUN (X, K)
Parameters
X ..... Input. Vector variable, $\boldsymbol{x}$ One dimensional array of size n.
K ..... Input. An integer such that $f_k(\boldsymbol{x})$ is evaluated ($1 \le$ K $\le n$). (See example)

EPSZ ..... Input. The tolerance (≥ 0.0). The search for a solution vector is terminated when $\|\boldsymbol{f}(\boldsymbol{x}_i)\|_\infty \le$ EPSZ (Refer to notes)

EPST ..... Input. The tolerance (≥ 0.0). The iteration is considered to have converged when $\|\boldsymbol{x}_i - \boldsymbol{x}_{k-1}\|_\infty \le \text{EPSZ} \cdot \|\boldsymbol{x}_i\|_\infty$ (Refer to notes)

FC ..... Input. A value to indicate the range of search for the solution vector (≥ 0.0).
The search for a solution vector is terminated when $\|\boldsymbol{x}_i\|_\infty > \text{FC} \cdot \max(\|\boldsymbol{x}_0\|_\infty, 1.0)$ (Refer to notes)

M ..... Input. The upper limit of iterations (> 0) (See notes)
Output. The number of iterations executed.

FNOR … Output. The value of $\|\boldsymbol{f}(\boldsymbol{x}_i)\|_\infty$ for the solution vector obtained.

VW ..... Work area. One dimensional array of size $n(n + 3)$

ICON ..... Output. Condition codes. See table NOLBR-1.

Table NOLBR-1 Condition codes

Code	Meaning	Processing
1	Convergence criterion $\|f(x_i)\| \le$ EPSZ was satisfied.	Normal
2	Convergence criterion $\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|_\infty \le \text{EPSZ} \cdot \|\boldsymbol{x}_i\|_\infty$ was satisfied.	Normal
10000	The specified convergence conditions were not satisfied during the given number of iterations.	The last $\mathbf{x}_i$ is returned in X.
20000	A solution vector was not found within the search range (See parameter FC.).	Bypassed.
25000	The Jacobian of f(x) reduced to 0 during iterations.	The last $\mathbf{x}_i$ is returned in X.
30000	N≤0, EPSZ<0, EPST<0, FC≤0, or M≤0	Bypassed

Comments on use

- Subprogram used
SSL II ... AMACH and MGSSSL
FORTRAN basic functions ... ABS, AMAX1, SIGN, SQRT

- Notes
ZFUN must be declared as EXTERNAL in the program from which this subroutine is called.
Setting of EPSZ and EPST
Two convergence criteria are used in this subroutine. When either of two is met, the iteration terminates. If the user wishes to cancel one of the criteria, he has only to set the corresponding tolerance equal to 0.0. That is, when:
a) $\text{EPSZ} = \varepsilon_A (> 0)$ and $\text{EPST} = 0$.
Unless $\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|_\infty = 0$ is satisfied, the iteration is repeated until $\|f(\boldsymbol{x}_i)\|_\infty \le \varepsilon_A$ is satisfied, or until M times iterations have been completed.
b) $\text{EPSZ} = 0$ and $\text{EPST} = \varepsilon_B (> 0)$
Unless $\|f(\boldsymbol{x}_i)\|_\infty = 0$ is satisfied, the iteration is repeated until $\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|_\infty \le \varepsilon_B \|\boldsymbol{x}_i\|_\infty$ is satisfied until M times iteration have been completed.
c) EPSZ = 0 and EPST = 0
Unless $\|\boldsymbol{f}(\boldsymbol{x}_i)\|_\infty = 0$ or $\|\boldsymbol{x}_i - \boldsymbol{x}_{i-1}\|_\infty = 0$, the iteration is repeated M times. This setting c) is useful for executing M times-iterations.

[The meaning of FC]
Sometimes a solution vector cannot be found in the neighbourhood of the initial vector $\boldsymbol{x}_0$. When this happens, $\boldsymbol{x}_i$ diverges from $\boldsymbol{x}_0$, as a result, numerical difficulties such as overflows may occur in evaluating $\boldsymbol{f}(\boldsymbol{x})$. Parameter FC is set to make sure that these anomalies may not occur by limiting the range of search for solution. Standard value of FC is around 100.0.
[Setting of M]
The number of iterations needed for convergence to the solution vector depends on the nature of the equations and the magnitude of tolerances. When an initial vector is improperly set or the tolerances are too narrowly set, parameter M should be set to a large number. As a rule of thumb, M is set to around 50 for n = 10.
Single precision/double precision setting
Usually, double precision subroutine can often solve those nonlinear equation. While single precision subroutine may fail.

- Example
  Non-linear simultaneous equations with two unknowns

$$\left.\begin{aligned} x_1\cdot\left(1-x_2^2\right)&=2.25\\ x_1\cdot\left(1-x_2^3\right)&=2.625\end{aligned}\right\}$$

are solved with initial vector $x_0 = (5.0,\ 0.8)^{\mathrm{T}}$
The solutions are $\boldsymbol{x} = (3.0,\ 0.5)^{\mathrm{T}}$ and $\boldsymbol{x} = (81/32,-1/3)^{\mathrm{T}}$

```
C      **EXAMPLE**
       DIMENSION X(2),VW(10)
       EXTERNAL FUN
       X(1)=5.0
       X(2)=0.8
       N=2
       EPSZ=1.0E-5
       EPST=0.0
       FC=100.0
       M=20
       CALL NOLBR(X,N,FUN,EPSZ,EPST,FC,
      *           M,FNOR,VW,ICON)
       WRITE(6,600) ICON,M,FNOR,(I,X(I),
      *             I=1,N)
       STOP
  600 FORMAT(' ','ICON=',I5/' ','M=',I5/
     *      ' ','FNOR=',E15.7/
     *     (' ','X(',I2,')=',E15.7))
       END

       FUNCTION FUN(X,K)
       DIMENSION X(2)
       GO TO(10,20),K
   10  FUN=X(1)*(1.0-X(2)**2)-2.25
       RETURN
   20  FUN=X(1)*(1.0-X(2)**3)-2.625
       RETURN
       END
```

Method
A system of non-linear equations

$$\boldsymbol{f}(\boldsymbol{x}) = 0 \tag{4.1}$$

is solved by Brent's method in this subroutine. At a typical step, starting from $\boldsymbol{y}_1 = \boldsymbol{x}_{i-1}$, a set of intermediate approximations $\boldsymbol{y}_2, \boldsymbol{y}_3, \ldots \boldsymbol{y}_n, \boldsymbol{y}_{n+1}$ are calculated and $\boldsymbol{y}_{n+1}$ is taken as $\boldsymbol{x}_i$ which can be considered as better approximation than $\boldsymbol{x}_{i-1}$. Each of $\boldsymbol{y}_{k+1}$ ($1 \le k \le n$) is selected in the way that the Taylor expansion of $f_i(\boldsymbol{y})$ up to the first order term at $\boldsymbol{y}_i$ should be zero.

$$f_j(\boldsymbol{y}) \approx f_j(\boldsymbol{y}_j) + \boldsymbol{g}_j^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{y}_j), \quad j = 1,2,\ldots,k$$

, where

$$\boldsymbol{g}_j^{\mathrm{T}} = \left(\frac{\partial f_j}{\partial \boldsymbol{x}_1},\ldots,\frac{\partial f_j}{\partial \boldsymbol{x}_n}\right)_{x=y_j} \tag{4.2}$$

- Procedure of Brent's method
  The procedure to obtain $\boldsymbol{x}_i$ from $\boldsymbol{x}_{i-1}$ is discussed here. Assume that an orthogonal matrix $\boldsymbol{Q}_1$ is given ($\boldsymbol{Q}_1 = \boldsymbol{I}$ when $\boldsymbol{x}_1$ is to be obtained from $\boldsymbol{x}_0$)
  (a) First step
  Let $\boldsymbol{y}_1 = \boldsymbol{x}_{i-1}$ and expand $f_1(\boldsymbol{y})$ at $\boldsymbol{y}_1$ in Taylor's series and approximate it by taking up to the first order term.

$$f_1(\boldsymbol{y}) \approx f_1(\boldsymbol{y}_1) + \boldsymbol{g}_1^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{y}_1) \tag{4.3}$$

The first step is to obtain y which satisfies the equation

$$f_1(\boldsymbol{y}_1) + \boldsymbol{g}_1^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{y}_1) = 0$$

and to let it be $\boldsymbol{y}_2$.
This is performed according to the following procedure.
Let $\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{Q}_1 = \boldsymbol{w}_1^{\mathrm{T}}$ then an orthogonal matrix.
$\boldsymbol{P}_1$, is obtained by Householder method to satisfy the following condition

$$\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{P}_1 = \alpha_1\boldsymbol{e}_1^{\mathrm{T}}, \quad \alpha_1 = \pm\|\boldsymbol{w}_1\|_2, \quad \boldsymbol{e}_1^{\mathrm{T}} = (1,0,\ldots,0)$$

where one of the double sign is selected to be equal to that of the first element of $\boldsymbol{w}_1^{\mathrm{T}}$. The, $\boldsymbol{y}_2$ is calculated as follows;

$$\boldsymbol{y}_2 = \boldsymbol{y}_1 - \frac{f_1(\boldsymbol{y}_1)}{\alpha_1}\boldsymbol{Q}_2\boldsymbol{e}_1, \quad \boldsymbol{Q}_2 = \boldsymbol{Q}_1 P_1 \tag{4.4}$$

(b) Second step
:
:
(c) k-th step
Again using Taylor's series of $f_k(\boldsymbol{y})$ at $\boldsymbol{y}_k$:

$$f_k(\boldsymbol{y}) \approx f_k(\boldsymbol{y}_k) + \boldsymbol{g}_k^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{y}_k) \qquad (4.5)$$

From the row vector $\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{Q}_k$, we obtain vector $\boldsymbol{w}_k^{\mathrm{T}}$ by replacing the first(k-1) elements in $\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{Q}_k$ with zeros

$$\boldsymbol{w}_k^{\mathrm{T}} = (\overbrace{0,...,0}^{k-1},*,*,...,*)$$

Next, we obtain an orthogonal matrix $\boldsymbol{P}_k$ by Householder method to satisfy the following condition.

$$\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{P}_k = \alpha_k \boldsymbol{e}_k^{\mathrm{T}}, \quad \alpha_k = \pm\|\boldsymbol{w}_k\|_2,$$

$$\boldsymbol{e}_k^{\mathrm{T}} = (\overbrace{0,...,0}^{k-1},1,0,...,0)$$

Here, the double sign is selected to be the same sign as the k-th element in $\boldsymbol{w}_k^{\mathrm{T}}$. The matrix $\boldsymbol{P}_k$ has the form:

$$\boldsymbol{P}_k = \begin{array}{c} k-1\left\{\begin{array}{c}\\ \\ \\ \end{array}\right. \\ \\ \\ \end{array} \left[\begin{array}{c|c} \begin{matrix}1 & & 0\\ & \ddots & \\ 0 & & 1\end{matrix} & 0 \\ \hline 0 & \hat{\boldsymbol{P}}_k \end{array}\right]$$

Then $\boldsymbol{y}_{k+1}$ is calculated as follows.

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k - \frac{f_k(\boldsymbol{y}_k)}{\alpha_k}\boldsymbol{Q}_{k+1}\boldsymbol{e}_k \quad ,\boldsymbol{Q}_{k+1} = \boldsymbol{Q}_k\boldsymbol{P}_k \qquad (4.6)$$

It can be shown that $\boldsymbol{y}_{k+1}$, according to (4.6), satisfies the conditions (4.7)

$$\left.\begin{array}{l} f_1(\boldsymbol{y}_1) + \boldsymbol{g}_1^{\mathrm{T}}(\boldsymbol{y}_{k+1} - \boldsymbol{y}_1) = 0 \\ f_2(\boldsymbol{y}_2) + \boldsymbol{g}_2^{\mathrm{T}}(\boldsymbol{y}_{k+1} - \boldsymbol{y}_2) = 0 \\ \quad : \\ f_k(\boldsymbol{y}_k) + \boldsymbol{g}_k^{\mathrm{T}}(\boldsymbol{y}_{k+1} - \boldsymbol{y}_k) = 0 \end{array}\right\} \qquad (4.7)$$

Letting $\boldsymbol{y}_{n+1}$, which is obtained at n-th step (k= n), be $\boldsymbol{x}_i$, if $\boldsymbol{x}_i$ satisfies the convergence criterion, the iteration terminates and $\boldsymbol{x}_i$ is taken as the solution vector.

If not, the above steps are repeated from the first step with the next starting value of iteration vector $\boldsymbol{y}_1 = \boldsymbol{x}_i$ and $\boldsymbol{Q}_1 = \boldsymbol{Q}_{n+1}$.

- Considerations on Algorithm
  - (a) Approximation of partial derivatives

    In calculating $\boldsymbol{w}_k$, the j-th element (k≤j≤n) of $\boldsymbol{w}_k$ is $\boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{Q}_k\boldsymbol{e}_j$. This is equal to the derivative of f_k at $\boldsymbol{y}_k$ along the direction indicated by $\boldsymbol{Q}_k\boldsymbol{e}_j$ (the j-th column of $\boldsymbol{Q}_k$). This subroutine approximates this derivative by the divided difference as follows:

$$\boldsymbol{w}_k \approx \frac{1}{h_i} \begin{array}{c} k-1\left\{\begin{array}{c}\\ \\ \\ \end{array}\right. \\ \\ \\ \\ \end{array} \begin{bmatrix} 0 \\ : \\ 0 \\ f_k(\boldsymbol{y}_k + h_i\boldsymbol{Q}_k\boldsymbol{e}_k) - f_k(\boldsymbol{y}_k) \\ : \\ f_k(\boldsymbol{y}_k + h_i\boldsymbol{Q}_k\boldsymbol{e}_k) - f_k(\boldsymbol{y}_k) \end{bmatrix} \qquad (4.8)$$

where $h_i = \|\boldsymbol{x}_{i-1}\|_\infty \cdot \sqrt{u}$ (4.9)

and u is the round-off unit.

  - (b) When $\alpha_k = 0$

    In the k-th step above, if

$$\alpha_k = 0 \qquad (4.10)$$

it is impossible to calculate $\boldsymbol{y}_{k+1}$. In this case, the subroutine automatically sets $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k$. This means that $\boldsymbol{y}_k$ will not be modified at all. If, further

$$\alpha_k = 0, k = 1,2,...,n \qquad (4.11)$$

then, y_1 (= $\boldsymbol{x}_{i-1}$) will never be modified during the steps. This can happen when the Jacobian of $f(\boldsymbol{x})$.

$$J = \left(\frac{\partial f_i}{\partial x_j}\right) \qquad (4.12)$$

is nearly singular. In this case, processing fails with ICON = 25000. The condition in (4.10) is determined by testing if the following condition is satisfied.

$$\|f_k(\boldsymbol{y}_k + h_i\boldsymbol{Q}_k\boldsymbol{e}_j) - f_k(\boldsymbol{y}_k)\|_\infty \le u \cdot \|f_k(\boldsymbol{y}_k)\|_\infty \qquad (4.13)$$
$$,j = k, k+1, ..., n$$

Further details should be referred to Reference [33].

D15-10-0101 NOLF1, DNOLF1

Minimization of the sum of squares of functions. (Revised Marquardt method, using function values only)
CALL NOLF1 (X, N, FUN, M, EPSR, MAX, F, SUMS, VW, K, ICON)

Function

Given *m* real functions $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}),..., f_m(\boldsymbol{x})$ of *n* variables and initial vector $\boldsymbol{x}_0$, this subroutine obtains vector $\boldsymbol{x}^*$ which gives a local minimum of

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} \{f_i(\boldsymbol{x})\}^2 \tag{1.1}$$

and its function value $F(\boldsymbol{x}^*)$ by using the revised Marquardt method (Levenberg-Marquardt-Morrison method (LMM method)). This subroutine does not require derivative of $F(\boldsymbol{x})$. However, the $f_i(\boldsymbol{x})$ is assumed to have up to the first continuous partial derivative, and $m \geq n \geq 1$.

Parameters

X .... Input. Initial vector $\boldsymbol{x}_0$.
Output. Vector $\boldsymbol{x}^*$.
One-dimensional array of size *n*.

N ..... Input. Number of variables *n*.

FUN ..... Input. Name of subroutine subprogram which calculates $f_i(\boldsymbol{x})$.
The form of subprogram is as follows:
SUBROUTINE FUN (X, Y)
where,
X ..... Input. Variable vector *x*.
One-dimensional array of size *n*.
Y .... Output. Function value $f_i(\boldsymbol{x})$ corresponding to variable vector $\boldsymbol{x}$.
One-dimensional array of size *m*, with correspondence,

F(1)= $f_1(\boldsymbol{x})$,...,F(M)= $f_m(\boldsymbol{x})$

M ..... Input. Number of functions *m*.

EPSR .. Input. Convergence criterion (≥ 0.0).
When EPSR = 0.0 is specified, a default value is used. (See Notes.)

MAX ... Input. The upper limit (≠0) of the number of evaluations of function (See Notes.)
Output. Number of evaluation actually performed (> 0).

F ..... Output. Function value $f_i(\boldsymbol{x}^*)$
One-dimensional array of size m, with correspondence,

F(1)= $f_1(\boldsymbol{x}^*)$,...,F(M)= $f_m(\boldsymbol{x}^*)$

SUMS .. Output. Value of the sum of squares $F(\boldsymbol{x}^*)$

VW ..... Work area. Two-dimensional array, VW (K, N + 2).

K ..... Input. Adjustable dimension (≥ *m* + *n*) of array VW.

ICON ..... Output. Condition code.
See Table NOLF1-1.

Table NOLF1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The convergence condition was not satisfied within the specified number of evaluations.	The last value is stored in parameters X, F and SUMS.
20000	During computation, Marquardt number v_k exceeded the upper limit (See (4.14) in Method). EPSR was too small or the error of difference approximation of a Jacobian matrix exceeded the limit of computation.	Discontinued (The last value is stored parameters X, F and SUMS.)
30000	N < 1, M < N, EPSR < 0.0, MAX = 0 or K< m+n	Bypassed

Comments on use

- Subprograms used
SSL II ... AMACH, MGSSL
FORTRAN basic functions ... ABS, SQRT, FLOAT

- Notes
The program which calls this subroutine must have an EXTERNAL statement for the subprogram name that correspondents to the argument FUN.
[Giving EPSR]
This subroutine assumes that $F(\boldsymbol{x})$ is approximately quadratic in the region of the local minimum point $\boldsymbol{x}^*$. To obtain $F(\boldsymbol{x}^*)$ as accurate as the unit round off, EPSR should be given as EPSR $\approx \sqrt{u}$ when *u* is the unit round-off. The default value is $2 \cdot \sqrt{u}$
[Giving MAX]
The number of evaluations of a function is counted by the number of computation of $f_i(\boldsymbol{x})$ for a variable vector $\boldsymbol{x}$, i = 1, ..., m. This corresponds to the number of callings of subprogram FUN.
The number of evaluations of a function depends on characteristics of equation $\{ f_i(\boldsymbol{x})\}$ in addition to the initial vector and a convergence criterion.
Generally, if the default values is used as the convergence criterion, and a good initial vector is used, MAX = $100 \cdot n \cdot m$ is appropriate.
If the convergence condition is not satisfied within the specified number of evaluations and the subroutine is returned with ICON = 10000, the iteration can be continued by calling the subroutine again. In this case, parameter MAX is specified with a negative

value for an additional evaluation number and the contents of other parameters must be kept intact.

- Example

The minimum point $\boldsymbol{x}^*$ for

$$F(x_1, x_2) = f_1^2(x_1,x_2) + f_2^2(x_1,x_2)$$

where

$$f_1(x_1, x_2) = 1 - x_1$$
$$f_2(x_1, x_2) = 10\left(x_2 - x_1^2\right)$$

is obtained with the initial vector $\boldsymbol{x}_0 = (-1.2, 1.0)^{\mathrm{T}}$.

```
C     **EXAMPLE**
      DIMENSION X(2),F(2),VW(4,4)
      EXTERNAL ROSEN
      X(1)=-1.2
      X(2)=1.0
      N=2
      M=2
      EPSR=1.0E-3
      MAX=100*2*2
      CALL NOLF1(X,N,ROSEN,M,EPSR,MAX,
     *           F,SUMS,VW,4,ICON)
      WRITE(6,600) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,610) SUMS,MAX
      WRITE(6,620) (I,X(I),I=1,N)
      WRITE(6,630) (I,F(I),I=1,M)
      STOP
  600 FORMAT('1','*ICON=',I5)
  610 FORMAT(' ',' SUM OF SQUARES= ',
     *E15.7,' MAX= ',I5/)
  620 FORMAT(1X,' X(',I2,')=',E15.7)
  630 FORMAT(1X,' F(',I2,')=',E15.7)
      END
C     OBJECTIVE FUNCTION
      SUBROUTINE ROSEN (X,Y)
      DIMENSION X(2),Y(2)
      Y(1)=1.0-X(1)
      Y(2)=(X(2)-X(1)*X(1))*10.0
      RETURN
      END
```

Method

This subroutine obtains vector $\boldsymbol{x}^*$ which gives a local minimum of

$$F(\boldsymbol{x}) = \|\boldsymbol{f}(\boldsymbol{x})\|^2 = \boldsymbol{f}^T(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{m}\{f_i(\boldsymbol{x})\}^2 \qquad (4.1)$$

and function value $F(\boldsymbol{x}^*)$ corresponding to m real functions $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})$ of n variables.

Where,

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^{\mathrm{T}}$$
$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \qquad (4.2)$$

This subroutine solves this problem by using the LMM method. The Levenberg-Marquardt method, the Newton-Gauss method and the steepest descent method are explained below.

Suppose that the approximate vector $\boldsymbol{x}_k$ of vector $\boldsymbol{x}^*$ which gives a local minimum is obtained and following relation is satisfied.

$$\boldsymbol{x}^* = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k \qquad (4.3)$$

$\boldsymbol{f}(\boldsymbol{x})$ is expanded up to the first order in the region of $\boldsymbol{x}_k$ by using the Taylor series which results in (4.4).

$$\boldsymbol{f}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k \qquad (4.4)$$

where $\boldsymbol{J}(\boldsymbol{x}_k)$ is a Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$ shown in (4.5)

$$\boldsymbol{J}(\boldsymbol{x}_k) \left[\begin{array}{c} \dfrac{\partial f_1}{\partial x_1} \cdots\cdots\cdots \dfrac{\partial f_1}{\partial x_n} \\ \vdots \qquad\qquad \vdots \\ \vdots \qquad\qquad \vdots \\ \dfrac{\partial f_m}{\partial x_1} \cdots\cdots\cdots \dfrac{\partial f_m}{\partial x_n} \end{array}\right]_{\boldsymbol{x}=\boldsymbol{x}_k} \qquad (4.5)$$

$\boldsymbol{J}(\boldsymbol{x}_k)$ is subsequently expressed as $\boldsymbol{J}_k$.

From (4.4), the value of $F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)$ can be approximated by (4.6) if $|F(\boldsymbol{x}_k)|$ is sufficiently small.

$$\begin{aligned} &\boldsymbol{F}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \\ &= \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \\ &\approx \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k) + 2\boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{J}_k\Delta\boldsymbol{x}_k \\ &\quad + \Delta\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\Delta\boldsymbol{x}_k \end{aligned} \qquad (4.6)$$

The value of $\Delta\boldsymbol{x}_k$ which minimizes this value is given as the solution of the system of linear equations (4.7) obtained when the right side of (4.6) is differentiated for $\Delta\boldsymbol{x}_k$.

$$\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\Delta\boldsymbol{x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \qquad (4.7)$$

(4.7) is called a normal equations.

In the Newton-Gauss method, $\Delta\boldsymbol{x}_k$ is used for iterations as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$$

In this method, $\Delta\boldsymbol{x}_k$ denotes the descent direction of $F(\boldsymbol{x})$, but $\Delta\boldsymbol{x}_k$ may diverge itself.

On the other hand, the gradient vector $\nabla F(\boldsymbol{x}_k)$ of $F(\boldsymbol{x})$ at $\boldsymbol{x}_k$ can be given by

$$\nabla F(\boldsymbol{x}_k) = 2\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \qquad (4.8)$$

$-\nabla F(\boldsymbol{x}_k)$ is the direction of the steepest descent of $F(\boldsymbol{x})$ at $\boldsymbol{x}_k$. In the steepest descent method, $\Delta\boldsymbol{x}_k$ is used as

$$\Delta\boldsymbol{x}_k = -\nabla\boldsymbol{F}(\boldsymbol{x}_k) \tag{4.9}$$

Although $\Delta\boldsymbol{x}_k$ in (4.9) surely guarantees a decrement of $F(\boldsymbol{x})$, it is noted that if iteration is repeated, $F(\boldsymbol{x})$ starts to zigzag, as many computational practices have been reported.

Therefore, to decrease these demerits, Levenberg, Marquardt and Morrison proposed to determine $\Delta\boldsymbol{x}_k$ by the following equations:

$$\left\{\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k + v_k^2\boldsymbol{I}\right\}\Delta\boldsymbol{x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \tag{4.10}$$

where, v_k is a positive value (called the Marquardt number). $\Delta\boldsymbol{x}_k$ which is determined by (4.10) depends on the value of v_k. As $v_k \to 0$, the direction of $\Delta\boldsymbol{x}_k$ is that of the Newton-Gauss method. On the other hand, the $\|\Delta\boldsymbol{x}_k\|$ decreases monotonically in proportion as v_k increases from 0, and the angle between $\Delta\boldsymbol{x}_k$ and the steepest descent direction $-\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k)$ decreases monotonically along with the increment of v_k. As $v_k \to \infty$, the direction of $\Delta\boldsymbol{x}_k$ is that of the steepest descent method.

The characteristics of the Levenberg-Marquardt method is to determine the value of v_k adaptively during iteration and to minimize $F(\boldsymbol{x})$ efficiently.

- LMM method
  In the method by (4.10), normal equations are explicitly constructed, so it is not numerically stable. Equation (4.10) is equivalent to the least squares problem corresponding to the following:

$$\begin{bmatrix} \boldsymbol{J}_k \\ \hdashline \mathrm{v}_k\,\boldsymbol{I} \end{bmatrix}\Delta\boldsymbol{x}_k = -\begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_k) \\ \hdashline \boldsymbol{O} \end{bmatrix} \tag{4.11}$$

That is, the minimization of sum of squares of residual in (4.11) can be expressed by (4.10).

In the LMM method, the normal equations are not explicitly constructed. The LMM method obtains $\Delta\boldsymbol{x}_k$ by the least squares method applying the orthogonal transformation which is numerically stable, to (4.11).

This subroutine obtains $\Delta\boldsymbol{x}$ by (4.11), and then

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$$

and iterates to satisfy

$$F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$$

to obtain minimum point $\boldsymbol{x}^*$.

- Computational procedures in this subroutine

1) Initialization
   Set Marquardt number v_0.
   Obtains $\boldsymbol{f}(\boldsymbol{x}_0)$ and $F(\boldsymbol{x}_0)$.
   Set $k = 0$
2) Obtain $\boldsymbol{J}_k$ by difference approximation.
3) Solve (4.11) by the least squares method to obtain $\Delta\boldsymbol{x}_k$
   Let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$ and obtain

$$\boldsymbol{f}(\boldsymbol{x}_{k+1}),\ F(\boldsymbol{x}_{k+1})$$

4) Test whether or not $F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$ is satisfied.
   When satisfied, go to step 8).
5) Convergence criterion
   When the convergence condition is satisfied, the subroutine terminates processing with ICON = 0 assuming $\boldsymbol{x}_k$ to be minimum point $\boldsymbol{x}^*$.
6) Increase the Marquardt value, that is, let

$$v_k = 1.5\ v_k$$

7) Test the upper limit of the Marquardt number by

$$v_k \le 1/u, \text{ where } u \text{ is the unit round off.} \tag{4.14}$$

   When (4.14) is satisfied, go to step 3) and continue iteration. When not satisfied, this subroutine terminates processing with ICON = 20000.
8) Convergence criterion
   When the convergence condition is satisfied, this subroutine terminates processing with ICON = 0 assuming $\boldsymbol{x}_{k+1}$ to be minimum point $\boldsymbol{x}^*$.
9) If this subroutine does not execute step 6), the Marquardt number is decreased, that is, let,

$$v_k = 0.5\ v_k$$

   Then setting k as $k = k + 1$ and proceed to step 2) to continuation.

- Notes on each algorithm

1) Setting Marquardt number v_0
   The norm of the Jacobian matrix at $\boldsymbol{x}_0$ is used as the initial value of the Marquardt number,

$$v_0 = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\partial f_i / \partial x_j\right)^2 / (m \cdot n)} \tag{4.15}$$

2) To compute the difference approximation of Jacobian matrix $\boldsymbol{J}_k$,

$$\boldsymbol{J}_k = \left[\begin{array}{ccc} \frac{\partial f_1}{\partial x_1} & \cdots\cdots & \frac{\partial f_n}{\partial x_n} \\ : & & : \\ : & & : \\ \frac{\partial f_m}{\partial x_1} & \cdots\cdots & \frac{\partial f_m}{\partial x_n} \end{array}\right]_{\boldsymbol{x}=\boldsymbol{x}_k} \quad (4.16)$$

the forward difference (4.17) are used.

$$\frac{\partial f_i}{\partial x_j} \approx \left\{ f_i\left(\boldsymbol{x}_k + h\boldsymbol{e}_j\right) - f_i\left(\boldsymbol{x}_k\right)\right\} / h \quad (4.17)$$

where, $\boldsymbol{e}_j$ is the j-th coordinate vector $h = \sqrt{u}$, where u is the unit round off.

3) Computing $\Delta\boldsymbol{x}_k$ by the least squares method
This subroutine uses the Householder method to obtain $\Delta\boldsymbol{x}_k$ solving (4.11) by the least squares method.
That is, the left side of (4.11) is multiplied from the left by the orthogonal matrix $\boldsymbol{Q}$ of the Householder transformation to obtain upper triangular matrix.

$$\boldsymbol{Q}\begin{bmatrix} \boldsymbol{J}_k \\ \cdots\cdots \\ v_k \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} \\ \cdots \\ \boldsymbol{O} \end{bmatrix} \quad (4.18)$$

Where, $\boldsymbol{R}$ is the upper triangular matrix of $n \times n$.
The orthogonal transformation is performed for the right side of (4.11).

$$-\boldsymbol{Q}\begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_k) \\ \cdots\cdots\cdots \\ \boldsymbol{O} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{g}_1 \\ \cdots\cdots \\ \boldsymbol{g}_2 \end{bmatrix} \quad (4.19)$$

Where, $\boldsymbol{g}_1$ is the n-dimensional vector and $\boldsymbol{g}_2$ is the m-dimensional vector. Since the norm is unitarily invariant for orthogonal transformation, least squares solution $\Delta\boldsymbol{x}_k$ in (4.11) is obtained by (4.20).

$$\boldsymbol{R}\Delta\boldsymbol{x}_k = -\boldsymbol{g}_1 \quad (4.20)$$

Since $\boldsymbol{R}$ is an upper triangular matrix, (4.20) can be computed by backwards substitution.

4) Convergence criterion
This subroutine test the convergence during iteration as follows:
   - When $F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$ and
     $\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \le \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot$ EPSR
     as satisfied, $\boldsymbol{x}_{k+1}$ is
     assumed to be minimum point $\boldsymbol{x}$*.
   - When $F(\boldsymbol{x}_{k+1}) \ge F(\boldsymbol{x}_k)$ and
     $\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \le \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot$ EPSR
     are satisfied, $\boldsymbol{x}_k$ is assumed to be minimum point $\boldsymbol{x}^*$.

For further details, refer to References [36] and [37].

D15-20-0101 NOLG1, DNOLG1

Minimization of the sum of squares of functions (Revised Marquardt method using function values and their derivatives)
CALL NOLG1 (X, N, FUN, JAC, M, EPSR, MAX, F, SUMS, VW, K, ICON)

Function

Given m real functions $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$,..., $f_m(\boldsymbol{x})$ with n variables, its Jacobian $\boldsymbol{J}(\boldsymbol{x})$, and initial vector $\boldsymbol{x}_0$, this subroutine obtains vector which gives a local minimum of

$$F(x)=\sum_{i=1}^{m}\{f_i(x)\}^2 \tag{1.1}$$

and its function value $F(\boldsymbol{x}^*)$ using the revised Marquardt method, that is, Levenberg-Marquardt-Morrison (LMM) method:

In this subroutine, $f_i(\boldsymbol{x})$, $i = 1$,..., m is assumed to have up to the first continuous partial derivative, and $m \geq n \geq 1$.

Parameters

X ..... Input. Initial vector $\boldsymbol{x}_0$.
Output. Vector $\boldsymbol{x}^*$.
One-dimensional array of size n.

N ..... Input. Number of variables n.

FUN..... Input. Name of subroutine subprogram that calculates $f_i(\boldsymbol{x})$
The form of subprogram is as follows:
SUBROUTINE FUN (X, Y)
Parameters
X ..... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
Y ..... Output. Function values $f_i(\boldsymbol{x})$ for variable vector $\boldsymbol{x}$.
One-dimensional array of size m where F(1)= $f_1(\boldsymbol{x})$, ... , F(M) = $f_m(\boldsymbol{x})$

JAC ... Input. Name of subroutine subprogram that calculates $\boldsymbol{J}(\boldsymbol{x})$
The form of subprogram is as follows:
SUBROUTINE JAC (X, G, K)
Parameters
X ..... Input. Variable vector $\boldsymbol{x}$.
One-dimensional array of size n.
G ..... Output. Jacobian matrix for variable vector $\boldsymbol{x}$.
Two-dimensional array G (K, N) where
$G(I,J)=\partial f_i/\partial x_j$
K ..... Input. Adjustable dimension of array G.

M ..... Input. Number m of the functions.

EPSR .. Input. Convergence criterion (≥ 0.0).
The default value is used if 0.0 is specified.
(See "Comments on Use.")

MAX ..... Input. The upper limit ($\neq 0$) of the count of function evaluation.
Output. The count (> 0) of actual evaluation

F ..... Output. The value of function $f_i(\boldsymbol{x}^*)$.
One-dimensional array of size m,
where F(1)=$f_1(\boldsymbol{x}^*)$, …., F(M)=$f_m(\boldsymbol{x}^*)$.

SUMS . Output. The value of the sum of squares $F(\boldsymbol{x}^*)$.

VW ..... Work area. Two-dimensional array of VW (K, N + 2)

K ..... Input. The adjustable dimension ($\geq m + n$) of array VW.

ICON ..... Condition code. (See Table NOLG1-1.)

Table NOLG1-1 Condition codes

Code	Meaning	Processing
0	No error.	
10000	The convergence condition was not satisfied within the specified number of interation.	The last values obtained are stored in X, F and SUMS.
20000	Marquardt number (v_k) exceeded the upper limit during calculation. (See (4.14) in " Method.") EPSR was too small or the error of the difference approximation of the Jacobian matrix exceeded the limit of calculation.	Bypassed. (The last values obtained are stored in X, F and SUMS.)
30000	N < 1, M < N, EPSR < 0.0, MAX = 0, or K < m+n	Bypassed.

Comments on use

- Subprograms used
  SSL II .. AMACH, MGSSL
  FORTRAN basic functions ... ABS, SORT, FLOAT

- Notes
  An EXTERNAL statement is necessary to declare the subprogram names correspond to parameters FUN and JAC in the calling program.
  EPSR
  Since $F(\boldsymbol{x})$ is assumed to be approximately a quadratic function in the vicinity of point $\boldsymbol{x}^*$, it is appropriate to specify EPSR as $\text{EPSR}\approx\sqrt{u}$, where u is the unit round off to obtain the value of function $F(\boldsymbol{x}^*)$ as accurate as the rounding error.

The default value of EPSR is $2\sqrt{u}$

MAX
The function evaluation count is incremented by one every time $f_i(\boldsymbol{x})$, $i = 1,...,m$ is calculated and by n every time $\boldsymbol{J}(\boldsymbol{x})$ is calculated for variable vector $\boldsymbol{x}$.

The function evaluation count depends on characteristics of function $\{f_i(\boldsymbol{x})\}$, initial vector, and convergence criterion.

Generally, when an appropriate initial value is specified and the default value is used for the convergence criterion, it is adequate to specify MAX = $100 \cdot n\,m$.

Even if the convergence condition is not satisfied within the specified evaluation count and the subroutine is returned with ICON = 10000, iteration can be resumed by calling this subroutine again. In this case, the user must specify a negative value as the additional evaluation count in the parameter MAX and retain other parameters unchanged.

- Example

Given the following function:

$$F(x_1,x_2) = f_1^2(x_1,x_2) + f_2^2(x_1,x_2)$$

where $f_1(x_1,x_2) = 1 - x_1$

$$f_2(x_1,x_2) = 10\left(x_2 - x_1^2\right)$$

minimum point $\boldsymbol{x}^*$ is obtained using value $\boldsymbol{x}_0 = (-1{,}2, 1.0)^{\mathrm{T}}$ as the initial value

```
C      **EXAMPLE**
       DIMENSION X(2),F(2),VW(4,4)
       EXTERNAL ROSEN,ROSENJ
       X(1)=-1.2
       X(2)=1.0
       N=2
       M=2
       EPSR=1.0E-3
       MAX=100*2*2
       CALL NOLG1(X,N,ROSEN,ROSENJ,M,EPSR,
      *           MAX,F,SUMS,VW,4,ICON)
       WRITE(6,600) ICON
       IF(ICON.GE.20000) STOP
       WRITE(6,610) SUMS,MAX
       WRITE(6,620) (I,X(I),I=1,N)
       WRITE(6,630) (I,F(I),I=1,M)
       STOP
   600 FORMAT('1','*ICON=',I5)
   610 FORMAT(' ',' SUM OF SQUARES= ',
      * E15.7,' MAX= ',I5/)
   620 FORMAT(1X,' X(',I2,')=',E15.7)
   630 FORMAT(1X,' F(',I2,')=',E15.7)
       END
C      OBJECTIVE FUNCTION
       SUBROUTINE ROSEN (X,Y)
       DIMENSION X(2),Y(2)
       Y(1)=1.0-X(1)
       Y(2)=(X(2)-X(1)*X(1))*10.0
       RETURN
       END
C      JACOBIAN
       SUBROUTINE ROSENJ(X,G,K)
       DIMENSION X(2),G(K,2)
       G(1,1)=-1.0
       G(2,2)=-20.0*X(1)
       G(1,2)=0.0
       G(2,2)=10.0
       RETURN
       END
```

Method

Given m real functions $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$, ..., $f_m(\boldsymbol{x})$ with n variable:

$$\begin{aligned} F(\boldsymbol{x}) &= \|f(\boldsymbol{x})\|^2 = f^{\mathrm{T}}(\boldsymbol{x}) f(\boldsymbol{x}) \\ &= \sum_{i=1}^{m} \{f_i(\boldsymbol{x})\}^2 \end{aligned} \tag{4.1}$$

vector $\boldsymbol{x}^*$ which gives a local minimum of function $F(\boldsymbol{x})$ and its function value $F(\boldsymbol{x}^*)$ are obtained. Where,

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &= (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}),..., f_m(\boldsymbol{x}))^{\mathrm{T}} \\ \boldsymbol{x} &= (x_1, x_2,..., x_n)^{\mathrm{T}} \end{aligned} \tag{4.2}$$

This subroutine uses the revised Marquardt method, that is, the Levenberg-Marquardt-Morrison (LMM) method. To explain this method, let us review the Levenberg-Marquardt, Newton-Gauss, and steepest descent methods.

Suppose that the approximate vector $\boldsymbol{x}_k$ of vector $\boldsymbol{x}^*$ that gives a local minimum is given and expressed as

$$\boldsymbol{x}^* = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k \tag{4.3}$$

Expanding $\boldsymbol{f}(\boldsymbol{x})$ to a Taylor series of the first order in the vicinity of $\boldsymbol{x}_k$, we obtain

$$\boldsymbol{f}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k \tag{4.4}$$

where $\boldsymbol{J}(\boldsymbol{x}_k)$ is the Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{x}_k) = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots\cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots\cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}\right|_{\boldsymbol{x} = \boldsymbol{x}_k} \tag{4.5}$$

$\boldsymbol{J}(\boldsymbol{x}_k)$ is referred to as $\boldsymbol{J}_k$ hereafter.

From (4.4), the function $F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)$ is approximated by (4.6) if $|F(\boldsymbol{x}_k)|$ is sufficiently small:

$$\begin{aligned} &F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \\ &= \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \\ &\approx \boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k) + 2\boldsymbol{f}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{J}_k\Delta\boldsymbol{x}_k \\ &+ \Delta\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\Delta\boldsymbol{x}_k \end{aligned} \tag{4.6}$$

The value of $\Delta\boldsymbol{x}_k$ which minimizes $F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)$ is the solution of the system of linear equations (4.7) obtained by differentiating the right side of (4.6) with respect to $\Delta\boldsymbol{x}_k$:

$$\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k\Delta\boldsymbol{x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \qquad (4.7)$$

This is called as the normal equation.
In the Newton-Gauss method, $\Delta\boldsymbol{x}_k$ is used for iterations as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$$

The $\Delta\boldsymbol{x}_k$ direction indicates the descent direction, but $\Delta\boldsymbol{x}_k$ may diverge in some case.
Gradient vector $\nabla F(\boldsymbol{x}_k)$ of $F(\boldsymbol{x})$ for $\boldsymbol{x}_k$ is

$$\nabla F(\boldsymbol{x}_k) = 2\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \qquad (4.8)$$

and $-\nabla F(\boldsymbol{x}_k)$ is the steepest descent direction of $F(\boldsymbol{x})$ for $\boldsymbol{x}_k$. In the steepest descent method,

$$\Delta\boldsymbol{x}_k = -\nabla F(\boldsymbol{x}_k) \qquad (4.9)$$

is used. Although decrement of $F(\boldsymbol{x})$ is guaranteed by $\Delta\boldsymbol{x}_k$ of (4.9), many computational practices have shown that the value of $F(\boldsymbol{x})$ starts oscillation during iterations.

To eliminate these disadvantages, that is, divergence of $\Delta\boldsymbol{x}_k$ and oscillation of $F(\boldsymbol{x})$, Levenberg, Marquardt, and Morrison have proposed to obtain $\Delta\boldsymbol{x}_k$ using

$$\left\{\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{J}_k + v_k^2\boldsymbol{I}\right\}\Delta\boldsymbol{x}_k = -\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k) \qquad (4.10)$$

where v_k is a positive number called a Marquardt number.

The value of $\Delta\boldsymbol{x}_k$ obtained from (4.10) apparently depends on the value of v_k : The direction of $\Delta\boldsymbol{x}_k$ for $v_k \to 0$ is that used in the Newton-Gauss method, where $\|\Delta\boldsymbol{x}_k\|$ monotonically decreases as the value of v_k increases beginning from 0 and the angle between $\Delta\boldsymbol{x}_k$ and the steepest descent direction $-\boldsymbol{J}_k^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}_k)$ monotonically decreases as v_k further increases.
If v_k approaches infinity, the direction of $\Delta\boldsymbol{x}_k$ becomes equal to that used in the steepest descent method.
Advantageous features of the Levenberg-Marquardt method are to determine the most suitable value of v_k dynamically during iterations to minimize the value of $F(\boldsymbol{x})$ efficiently.

LMM method
The method in which expression (4.10) is used does not have sufficient numerical stability because the normal equation system is explicitly constructed. Equation (4.10) is equivalent to the least squares problem for

$$\begin{bmatrix}\boldsymbol{J}_k \\ \cdots \\ v_k\boldsymbol{I}\end{bmatrix}\Delta\boldsymbol{x}_k = -\begin{bmatrix}\boldsymbol{f}(\boldsymbol{x}_k) \\ \cdots \\ \boldsymbol{O}\end{bmatrix} \qquad (4.11)$$

This means that the minimization of sum of squares of residual in (4.11) can be expressed by (4.10).

The LMM method obtains $\Delta\boldsymbol{x}_k$ without generating a normal equation system, but it includes the least squares method in which the orthogonal transformation having a high numerical stability is applied to (4.11).

In this subroutine, $\Delta\boldsymbol{x}$ is obtained from (4.11); then using

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$$

minimum point $\boldsymbol{x}^*$ is obtained through iterations in which

$$F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$$

is satisfied.

Computational procedures

1) Initialization
   Sets Marquardt number v_0.
   Obtains $\boldsymbol{f}(\boldsymbol{x}_0)$ and $F(\boldsymbol{x}_0)$.
   Sets $k = 0$.
2) Obtains $\boldsymbol{J}_k$.
3) Solves (4.11) using the least squares method to obtain $\Delta\boldsymbol{x}_k$
   Sets $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$
   Obtains $\boldsymbol{f}(\boldsymbol{x}_{k+1})$ and $F(\boldsymbol{x}_{k+1})$.
4) Checks whether $F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$ is satisfied;
   if so, proceeds to 8)
5) Checks convergence; if the convergence condition is satisfied, assumes $\boldsymbol{x}_k$ as to be minimum point $\boldsymbol{x}^*$, sets ICON = 0, then stops processing.
6) Increases Marquardt number as

   $$v_k = 1.5\, v_k$$

7) Checks upper limit of Marquardt number. If

   $$v_k \le 1/u\text{, where u is the unit round off} \qquad (4.14)$$

   is satisfied, returns to 3) to continue iterations; otherwise, sets ICON = 20000, then stops processing.
8) Checks convergence; if the convergence condition is satisfied, assumes $\boldsymbol{x}_{k+1}$ as to be minimal point $\boldsymbol{x}^*$, sets ICON = 0, then stops processing.
9) If 6) has been bypassed, decreases Marquardt number: $v_k = 0.5v_k$.
   Sets $k = k + 1$, then returns to 2).

Notes on algorithms

1) Marquardt number v_0 setting
   The norm of the Jacobian matrix for $\boldsymbol{x}_0$ is used as the initial value of Marquardt number.

$$v_0 = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\partial f_i / \partial x_j\right)^2 / (mn)} \qquad (4.15)$$

2) Calculation of $\Delta \boldsymbol{x}_k$ by the least squares method

The Householder method is used to solve (4.11) and obtain $\Delta \boldsymbol{x}_k$ by the least squares method.

The orthogonal matrix $\boldsymbol{Q}$ of the Householder transformation is multiplied by the left side of (4.11) to obtain the upper triangular matrix.

$$\boldsymbol{Q}\begin{bmatrix} \boldsymbol{J}_k \\ \cdots\cdots \\ v_k \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} \\ \cdots \\ \boldsymbol{O} \end{bmatrix} \qquad (4.16)$$

where $\boldsymbol{R}$ is the $n \times n$ upper triangular matrix. The orthogonal transformation is also multiplied on the right side of (4.11) to obtain

$$-\boldsymbol{Q}\begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_k) \\ \cdots\cdots\cdots \\ \boldsymbol{O} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{g}_1 \\ \cdots\cdots \\ \boldsymbol{g}_2 \end{bmatrix} \qquad (4.17)$$

where $\boldsymbol{g}_1$ is an n-dimensional vector and $\boldsymbol{g}_2$ is an m-dimensional vector. Since the norm is invariant for the orthogonal transformation, the least squares solution of $\Delta \boldsymbol{x}_k$ for (4.10) is obtained from

$$\boldsymbol{R}\Delta \boldsymbol{x}_k = -\boldsymbol{g}_1 \qquad (4.18)$$

Since R is an upper triangular matrix, (4.18) is solved using backward substitution.

3) Convergence check

The convergence condition is checked as follows:

If $F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$ and

$\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \le \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot \mathrm{EPSR}$

are satisfied, assumes $\boldsymbol{x}_{k+1}$ to be minimum point $\boldsymbol{x}^*$.

If $F(\boldsymbol{x}_{k+1}) \ge F(\boldsymbol{x}_k)$

and $\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right\|_\infty \le \max\left(1.0, \left\|\boldsymbol{x}_k\right\|_\infty\right) \cdot \mathrm{EPSR}$

are satisfied, assumes $\boldsymbol{x}_k$ to be minimum point $\boldsymbol{x}^*$. (See reference [36] and [37] for details.)

B21-11-0702 NRML, DNRML

Normalization of eigenvectors of a real matrix
CALL NRML (EV, K, N, IND, M, MODE, ICON)

Function

Eigenvectors $\boldsymbol{y}_i$ are formed by normalizing m eigenvectors $\boldsymbol{x}_i$ (i = 1, ... , m) of an n-order real matrix. Either (1.1) or (1.2) is used.

$$\boldsymbol{y}_i = \boldsymbol{x}_i / \left\| \boldsymbol{x}_i \right\|_\infty \tag{1.1}$$

$$\boldsymbol{y}_i = \boldsymbol{x}_i / \left\| \boldsymbol{x}_i \right\|_2 \tag{1.2}$$

$n \geq 1.$

Parameters

EV..... Input. m eigenvectors $\boldsymbol{x}_i$ (i = 1, ... , m).
(See "Comments on use")
EV (K,M) is a two-dimensional array.
Output Normalized eigenvectors $\boldsymbol{y}_i$.

K..... Input. Adjustable dimenson of array EV. (≥ n)

N..... Input. Order n of the real matrix.

IND..... Input. For each eigenvector in EV, indicates whether eigenvector is real or complex.
If the Jth column of EV contains a real eigenvector, IND(J) = 1; if it contains the real part of a complex eigenvector, IND(J) = −1, and if it contains the imaginary part of a complex eigenvector, IND(J) = 0.
IND is a one-dimensional array of size M.

M..... Input. Size of the array IND.

MODE.....Input. Indicate the method of normalization
MODE = 1... (1.1) is used.
MODE = 2... (1.2) is used.

ICON..... Output. Condition code
See Table NRML-1.

Comments on use

- Subprograms used
  SSL II.....MGSSL
  FORTRAN basic functions..... ABS and SQRT

Table NRML-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	EV (1,1) =1.0
30000	N<M, M<1, K<N, MODE was not 1 and 2 or an error was found in IND	Bypassed

When the eigenvectors of a real symmetric matrix are to be normalized, all of IND informations are 1.

- Notes
  Eigenvectors are stored in EV such that each real eigenvector occupies one column and each complex eigenvector occupies two consecutive columns (one for the real part and one for the imaginary part). Refer to Fig. NRML-1.
  If subroutine HVEC or HBK1 are called before this subroutine, parameters EV, IND, and M can be used as input parameters to this routine.

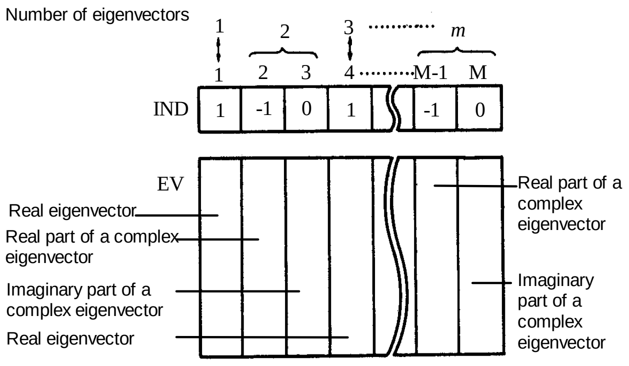

Number of eigenvectors
1
2
3
m
1
2
3
4
M-1
M
IND
1
-1
0
1
-1
0
EV
Real eigenvector
Real part of a complex eigenvector
Imaginary part of a complex eigenvector
Real eigenvector
Real part of a complex eigenvector
Imaginary part of a complex eigenvector

Fig. NRML-1 Relationship between IND and EV

- Example
  Subroutine EIG1 is called to obtain the eigenvectors of an n-order real matrix, and then this routine is used to normalize the resultant eigenvalues such that $\left\|\boldsymbol{x}\right\|_\infty = 1.$ $n \leq 100.$

```
C     **EXAMPLE**
      DIMENSION A(100,100),ER(100),EI(100),
     *EV(100,100),VW(100),IND(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N
      DO 20 I=1,N
      WRITE(6,610) (I,J,A(I,J),J=1,N)
   20 CONTINUE
      CALL EIG1(A,100,N,0,ER,EI,EV,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GT.10000) GO TO 10
      DO 30 I=1,N
   30 IND(I)=1
      DO 40 I=1,N
      IF(EI(I).EQ.0.0) GO TO 40
      IF(IND(I).EQ.0) GO TO 40
      IND(I)=-1
      IND(I+1)=0
   40 CONTINUE
      CALL NRML(E,V,100,N,IND,N,1,ICON)
      CALL EPRT(ER,EI,EV,IND,100,N,N)
      GO TO 10
```

```
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',
     *5X,'N=',I3/)
  610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
     *E14.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

In this example, subroutine EPRT is used to print the eigenvalues and corresponding eigenvectors of the real matrix. For details refer to the Example in section EIG1.

Method

Given m eigenvectors $\boldsymbol{x}_i(i = 1, ..., m)$ of an n-order real matrix, normalized eigenvectors $\boldsymbol{y}_i$ are computed. Where $\boldsymbol{x}_i = (x_{1i}, ..., x_{ni})^{\mathrm{T}}$. When MODE = 1 is specified, each vector $\boldsymbol{x}_i$ is normalized such that the maximum absolute value among its elements becomes 1.

$$\boldsymbol{y}_i = \boldsymbol{x}_i / \|\boldsymbol{x}_i\|_\infty , \|\boldsymbol{x}_i\|_\infty = \max_k |x_{ki}| \tag{4.1}$$

When MODE = 2 is specified, each vector $\boldsymbol{x}_i$ is normalized such that the sum of the square of absolute values corresponding to its elements is 1.

$$\boldsymbol{y}_i = \boldsymbol{x}_i / \|\boldsymbol{x}_i\|_2 , \quad \|\boldsymbol{x}_i\|_2 = \sqrt{\sum_{k=1}^{n} |x_{ki}|^2} \tag{4.2}$$

H11-20-0141 ODAM, DODAM

A system of first order ordinary differential equations (variable-step, variable-order Adams method, step output, final value output).
CALL ODAM (X, Y, FUN, N, XEND, ISW, EPSA, EPSR, VW, IVW, ICON)

Function

This subroutine solves a system of first order ordinary differential equations of the form:

$$\left.\begin{array}{ll} y_1' = f_1(x, y_1, y_2, ..., y_N), & y_1(x_0) = y_{10} \\ y_2' = f_2(x, y_1, y_2, ..., y_N), & y_2(x_0) = y_{20} \\ \quad : & \quad : \\ y_N' = f_N(x, y_1, y_2, ..., y_N), & y_N(x_0) = y_{N0} \end{array}\right\} \qquad (1.1)$$

by the Adams method, when function

$$f_1, f_2, ..., f_N$$

and initial values $x_0, y_{10}, y_{20}, ..., y_{N0}$, , and the final value of x, (x_e), are given.
That is, it obtains the solutions

$$(y_{1m}, y_{2m}, ..., y_{Nm})$$

at points $x_m = x_0 + \sum_{j=1}^{m} h_j ; m = 1, 2, ..., e$
(See Fig. ODAM-1).

The step size h_j is controlled so that solutions satisfy the desired accuracy.

This subroutine provides two types of output mode as shown below. The user can select the appropriate mode according to his purpose.

- Final value output: Returns to the user program when the solution at final value x_e is obtained.
- Step output: Returns to the user program each time the solution at x_1, x_2, ...is obtained.

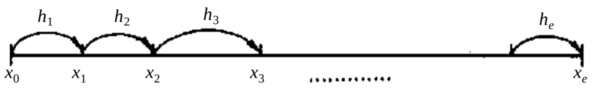


Fig. ODAM-1 Solution output point x_m ($x_0 < x_e$)

Parameters

X ..... Input. Starting point x_0.
Output. Final value x_e. When the step output is specified, an interim point x_m to which the solutions are advanced a single step.

Y ..... Input. Initial values $y_{10}, y_{20}, ..., y_{N0}$. They must be given in order of Y(1) = y_{10}, Y(2) = y_{20}, ..., Y(N) = y_{N0}.
One-dimensional array of size N.
Output. Solution vector at final value x_e.
When the step output is specified, the solution vector at $x = x_m$.

FUN ..... Input. The name of the subprogram which evaluates $f_i(i=1, 2, ..., N)$ in (1.1).
The form of the subroutine is as follows:
SUBROUTINE FUN (X, Y, YP)
Where
X: Input. Independent variable x.
Y: Input, One-dimensional array of size N, with corresponding Y(1) = y_1, Y(2) = y_2, ... , Y(N) = y_N.
YP: Output. One-dimensional array of size N, with corresponding

$$\mathrm{YP}(1) = f_1(x, y_1, y_2, ..., y_N),$$
$$\mathrm{YP}(2) = f_2(x, y_1, y_2, ..., y_N), ...,$$
$$\mathrm{YP}(N) = f_N(x, y_1, y_2, ..., y_N)$$

N ..... Input. Number of equations in the system.

XEND ... Input. Final point x_e to which the system should be solved.

ISW ..... Input. Integer variable to specify conditions in integration.
ISW is non-negative integer having three decimal digits, which can be expressed as

$$\mathrm{ISW} = 100d_3 + 10d_2 + d_1$$

Each d_i should be specified as follows

d_1: Specifies whether or not this is the first call.
0 ....First call
1 ....Successive call
The first call means that this subroutine is called for the first time for the given differential equations.

d_2: Specifies the output mode.
0 ...Final value output
1 ...Step output

d_3: Indicates whether or not functions $f_1, f_2, ..., f_N$ can be evaluated beyond the final value x_e.
0 ....Permissible
1 ....Not permissible
The situation in which the user sets $d_3 = 1$ is when derivatives are not defined beyond x_e, or there is a discontinuity there. However, the user should be careful that if this is not the case specifying $d_3 = 1$ leads to unexpectedly inefficient computation.

Output. When this subroutine returns to the user program after obtaining the solutions at x_e or the solutions at each step, d_1 and d_3 are altered as follows:

d_1: Set to 1. On subsequent calls, d_1 should not be altered by the user. Resetting $d_1 = 0$ is needed only when the user starts to solve another equations.

d_3: When $d_3 = 1$ on input, change it to $d_3 = 0$ when the solution at x_e is obtained.

EPSA ..... Input. Absolute error tolerance. (≥ 0.0)
Output. If a value smaller than the allowable value EPSA is specified at entry time, this value is appropriately changed. (See Notes.)

EPSR ..... Input. Relative error tolerance.
Output. If EPSR is too small, the value is changed to an appropriate value. (See Notes.)

VW ..... Work area. One-dimensional array of size 21N + 110. When calling this subroutine repeatedly, the contents should not be changed.

IVW ..... Work area. One-dimensional array of size 11. When calling this subroutine repeatedly, the contents should not be changed.

ICON ..... Output. Condition code.
See Table ODAM-1.

Table ODAM-1 Condition Codes

Code	Meaning	Processing
0	(In step output) A single step has been taken.	Subsequent calling is possible.
10	Solution at XEND was obtained.	Subsequent calling is possible after changing XEND.
100	A single step has been taken. It is detected that more than 500 steps are required to reach XEND.	To continue, just call again. The function counter will be reset to 0.
200	A single step has been taken. However it is detected the equations being processed have strong stiffness.	Although subsequent calling is possible, it is better to use a subroutine for stiff equations.
10000	EPSR and EPSA were too small for the arithmetic precision.	EPSR and EPSA were set to larger, acceptable vaiues. Subsequent calling is possible. (The increased values of EPSR and EPSA should be checked by the user)
30000	One of the following occurred: 1 N≤ 0 2 X = XEND 3 An erroneous ISW was specified. 4 EPSA < 0 or EPSR < 0 5 The contents of IVW was changed (in subsequent calling).	Processing terminates.

Comments on use

- Subprograms used
SSL II ... MGSSL, AMACH, UDE, USTE1, UNIT1
FORTRAN basic functions ... MOD, AMAX1, AMIX1, AMINI, ABS, SQRT, SIGN

- Notes
This subroutine is a standard program to solve non-stiff and mildly stiff differential equations along with Runge-Kutta subroutine ODRK1.
If in the following situation, this subroutine can be used more effectively.
– It takes much time for computing function $f_1, f_2,...,f_N$
– Highly accurate solution is required.
– The solutions at many points are required, for example to make a table of the solutions.
– Derivatives, $f_1, f_2,...,f_N$ have discontinuities.
If it is known beforehand, that the equations are stiff subroutine ODGE should be used.
The name of the subroutine associated with parameter FUN must be declared as EXTERNAL in the calling program.
[Judgment by ICON]
When the user specifies the final value output by setting the second digit of ISW to 0, he can obtain the solution at x_e only when ICON is 10. However, this subroutine may return the control to the user program when ICON is 100, 200 or 10000 before reaching x_e.
When the step output is specified by setting the second digit of ISW to 1, the user can receive the solution at each step not only when ICON is 0, but also when ICON is 100 or 200. ICON = 10 indicates that the solution at x_e has been obtained.
[EPSA and EPSR]
Suppose the elements of the solution of differential equations are Y(L) and the error (local error) is l_e(L), this subroutine controls the error to satisfy the following for L = 1, 2, ..., N.

$$|l_e(\mathrm{L})| \le \mathrm{EPSR} \times |\mathrm{Y(L)}| + \mathrm{EPSA} \qquad (3.1)$$

When EPSA = 0.0 is specified in (3.1), the relative error is tested. When EPSR = 0.0 is specified in (3.1), the absolute error is tested. The user should read the following notes in using these criterions:
– It is desirable to use the relative error criterion for the problem in which the magnitude of the solution varies substantially.
– The absolute error criterion may be used for the problem in which the magnitude of the solution does not vary so much or the small solution is not required.
– Specifying EPSA≠0 and EPSA≠0 result in stable and useful criterion. In this case, relative errors are tested for the larger solution and absolute errors are tested for the smaller solution.

If the maximum accuracy attainable by this subroutine is desired, specify EPSA and EPSR smaller than required. Then their values are appropriately increased in the subroutine.
ICON = 10000 notifies the user of this behavior. The user should call the subroutine successively after receiving ICON = 10000. When EPSA = EPSR = 0.0 is specified, it should be changed to EPSR = 16*u*. *u* is a unit of round-off errors.
[Parameter XEND]
If the solutions at a sequence of output points are required, this subroutine should be successively called changing XEND sequentially. It is assumed that this subroutine is called repeatedly and thus it sets the value of parameters required for subsequent calls when returning to the user program.
Therefore, the user can call the subroutine only by changing XEND. Only EPSA and EPSR can be changed on an as-required basis.

Discontinuous points of derivatives and non-defined domain.
If the solution or its derivatives has discontinuous points the points have to be detected to obtain satisfactory accuracy.

Example:
The equation,

$$y' = \begin{cases} y, & 0 \le x \le 1 \\ -y, & 1 < x \le 2 \end{cases} \quad , y(0) = 1$$

has the solution $y = e^x$ for $0 \le x \le 1$, and $y = e^{2-x}$ for $1 \le x \le 2$. Therefore, first-order derivative has a jump at $x = 1$.

This subroutine automatically detects discontinuous points and performs appropriate computation. The user needs not recognize the discontinuous points. However, if the user specifies the location of the discontinuous points in a way below, the time required for detection is shortened and computation may be accurate.

– Call this subroutine with XEND set to a discontinuous point and with setting the third digit of ISW to 1. Setting the third digit of ISW to 1 without changing the lower two digits of ISW can be performed by:

  ISW = MOD (ISW, 100) + 100

– When the solution at the discontinuous point has been obtained, the subroutine returns the control to the user program after setting the third digit to 0. Set the first digit of ISW to 0 on the next call, after advancing XEND appropriately.
  This can be performed by:

  ISW = (ISW / 10)*10

  By setting ISW in this way, the subroutine is told as if the solution at discontinous points were a new initial value and other differential equations were to be solved.

- Example
  Simultaneous second-order ordinary differential equations

$$\begin{cases} y_1'' = -\dfrac{y_1}{r^3}, y_1(0) = 1, y_1'(0) = 0 \\ y_2'' = -\dfrac{y_2}{r^3}, y_2(0) = 0, y_2'(0) = 1 \end{cases}$$

are solved, where $r = \left(y_1^2 + y_2^2\right)^{1/2}$. These second-order equations can be rewritten into first-order equations by replacing $y'_1$by y_3 and y'_2 by y_4 as follows:

$$\begin{cases} y_1' = y_3, & y_1(0) = 1 \\ y_2' = y_4, & y_2(0) = 0 \\ y_3' = -\dfrac{y_1}{r^3}, & y_3(0) = 0 \\ y_4' = -\dfrac{y_2}{r^3}, & y_4(0) = 1 \end{cases} \qquad (3.2)$$

The following example shows the solution on interval [0,2π] with EPSA =10^{-8} and EPSR = 10^{-5} in (3.2).
The solutions are to be output at each point of the following:

$$x_j = \frac{2\pi}{64} \cdot j, j = 1,2,...,64$$

Therefore, this subroutine is called repeatedly increasing the value of parameter XEND by 2π/64.

```
C     **EXAMPLE**
      DIMENSION Y(4),VW(200),IVW(11)
      EXTERNAL FUN
      X=0.0
      Y(1)=1.0
      Y(2)=0.0
      Y(3)=0.0
      Y(4)=1.0
      N=4
      EPSA=1.0E-8
      EPSR=1.0E-5
      ISW=0
      PAI=4.0*ATAN(1.0)
      DX=PAI/32.0
C
      WRITE(6,600)
C
      DO 30 I=1,64
      XEND=DX*FLOAT(I)
   10 CALL ODAM(X,Y,FUN,N,XEND,ISW,EPSA,
     *EPSR,VW,IVW,ICON)
      IF(ICON.EQ.10) GO TO 20
      IF(ICON.EQ.100) WRITE(6,620)
      IF(ICON.EQ.200) WRITE(6,630)
      IF(ICON.EQ.10000) WRITE(6,640) EPSA,
     *EPSR
```

```
      IF(ICON.EQ.30000) STOP
      GO TO 10
   20 WRITE(6,610) X,(Y(L),L=1,4)
   30 CONTINUE
      STOP
  600 FORMAT('1',12X,'X',22X,'Y(1)',
     *16X,'Y(2)',16X,'Y(3)',16X,'Y(4)'/)
  610 FORMAT(6X,E15.8,10X,4(E15.8,5X))
  620 FORMAT(10X,'TOO MANY STEPS')
  630 FORMAT(10X,'THE EQUATIONS',
     *1X,'APPEAR TO BE STIFF')
  640 FORMAT(10X,'TOLERANCE RESET',
     *5X,'EPSA=',E12.5,5X,'EPSR=',E12.5)
  650 FORMAT(10X,'INVALID INPUT')
      END

      SUBROUTINE FUN(X,Y,YP)
      DIMENSION Y(4),YP(4)
      R3=(Y(1)*Y(1)+Y(2)*Y(2))**1.5
      YP(1)=Y(3)
      YP(2)=Y(4)
      YP(3)=-Y(1)/R3
      YP(4)=-Y(2)/R3
      RETURN
      END
```

Method

This subroutine uses the Adams method with step-size control and order control. It is a standard subroutine to solve non-stiff or mildly-stiff initial value problems. This subroutine is most suitable for problems in which derivatives $f_1, f_2, ..., f_N$ in(1.1)are complicated and it takes much time to evaluate them.

For convenience, we consider a single equation for some time, and we write it as

$$y' = f(x, y), \; y(x_0) = y_0 \tag{4.1}$$

The solution at x_m is expressed by y_m and the exact solution is expressed by $y(x_m)$. The value of $f(x_m, y_m)$ is expressed by f_m to distinguish from $f(x_m, y_m)$.

1) Principle of the Adams method

Suppose solution $y_0, y_1, ..., y_m$ have already been obtained, and we are going to obtain the solution y_{m+1} at

$x_{m+1} = x_m + h_{m+1}$

From (4.1), the following equation holds:

$$y(x_{m+1}) = y(x_m) + \int_{x_m}^{x_{m+1}} f(x, y)dx \tag{4.2}$$

If the integrand $f(x, y)$ in the right side is approximated by the polynomial interpolation using some derivatives already computed we can get a formula.

Now, we consider a interpolation polynomial $P_{m,k}(x)$ based on k derivatives, which satisfies:

$$P_{k,m}(x_{m+1-j}) = f_{m+1-j}, \; j = 1,2,...,k \tag{4.3}$$

If this $P_{k,m}(x)$ is used for approximation to $f(x,y)$ in (4.2) and y_m is used instead of $y(x_m)$, we get a solution p_{m+1}:

$$p_{m+1} = y_m + \int_{x_m}^{x_{m+1}} P_{k,m}(x)dx \tag{4.4}$$

This is called k th-order Adams-Bashforth formula. Among various forms to represent $P_{k,m}(x)$, the Newton form is used here. Interpolation points, generally, x_{m+1-j} (j = 1,2, ..., k) are unequally spaced because of step-size control. But, for simplicity, we suppose they are equally spaced with the interval of h.

Newton backward difference representation for $P_{k,m}(x)$ is expressed by:

$$P_{k,m}(x) = f_m + \frac{x - x_m}{h}\nabla f_m + ... + \frac{(x - x_m)...(x - x_{m+2-k})}{h^{k-1}(k-1)!}\nabla^{k-1} f_m$$

Substituting this into (4.4), we get

$$\left.\begin{aligned}
&p_{m+1} = y_m + h\sum_{i=1}^{k}\gamma_{i-1}\nabla^{i-1} f_m \\
&\text{where } \nabla^0 f_m = f_m \\
&\qquad \nabla^i f_m = \nabla^{i-1} f_m - \nabla^{i-1} f_{m-1}, \quad i \geq 1 \\
&\qquad \gamma_0 = 1 \\
&\qquad \gamma_i = \frac{1}{i!h}\int_{x_m}^{x_{m+1}} \frac{(x - x_m)...(x - x_{m+1-i})}{h^{i-1}}dx \\
&\qquad\quad = \frac{1}{i!}\int_0^1 s(s+1)...(s+i-1)ds, \; i \geq 1
\end{aligned}\right\} \tag{4.5}$$

When k is 1, $P_{1,m}(x) = f_m$ and the following is obtained:

$$p_{m+1} = y_m + hf_m$$

This is Euler's method.

When $f(x_{m+1}, p_{m+1})$ is computed using p_{m+1} in (4.4), the approximation to derivative at x_{m+1} can be obtained. Using this, if we integrate again the approximation after correcting $P_{k,m}(x)$ in (4.4), a more accurate solution can be obtained. Based upon this idea, p_{m+1} in (4.4) is called a predictor and the corrected solution is called a corrector which is expressed by c_{m+1}. Suppose $P^*_{k,m}(x)$ to be a k-1 order polynomial interpolation satisfying the following:

$$P^*_{k,m}(x_{m+1-j}) = f_{m+1-j}, \ j = 1,2,\ldots,k-1$$
$$P^*_{k,m}(x_{m+1}) = f(x_{m+1}, p_{m+1})$$

In this interpolation, $f(x_{m+1}, p_{m+1})$ is used and the oldest f_{m+1-k} is removed. The corrector c_{m+1} is computed by:

$$c_{m+1} = y_m + \int_{x_m}^{x_{m+1}} P^*_{k,m}(x)dx \qquad (4.6)$$

This is called the k th order Adams-Moulton formula. If $P^*_{k,m}(x)$ is expressed in the form of Newton backward difference interpolation, (4.6) can be expressed as follows:

$$\left.\begin{aligned} & c_{m+1} = y_m + h\sum_{i=1}^{k} \gamma^*_{i-1} \nabla^{i-1} f^p_{m+1} \\ & \text{where, } \nabla^0 f^p_{m+1} = f^p_{m+1} \equiv f(x_{m+1}, p_{m+1}) \\ & \qquad \nabla^i f^p_{m+1} = \nabla^{i-1} f^p_{m+1} - \nabla^{i-1} f_m, i \geq 1 \\ & \qquad \gamma^*_0 = 1 \\ & \qquad \gamma^*_i = \frac{1}{i!}\int_0^1 (s-1)(s)\ldots(s+i-2)ds, i \geq 1 \end{aligned}\right\} \qquad (4.7)$$

In particular, in the case $k = 1$, since $P^*_{1,m}(x) = f_p^{m+1}$, the following holds:

$$c_{m+1} = y_m + hf^p_{m+1} \qquad (4.8)$$

This is called the Backward Euler's method.

These are the principles of the Adams method.

Since, in actual computation, points $\{x_{m+1-j}\}$ are spaced unequally due to step-size control $P_{k,m}(x)$ and $P^*_{k,m}(x)$ are expressed in the form of a modified divided difference instead of using (4.5) an (4.7).

The error of the corrector c_{m+1} is estimated as indicated below.

Note that f_{m+1-k} was not used in constructing $P_{k,m}(x)$. If, however, we use f_{m+1-k} and integrate the resulting interpolation polynomial of degree higher by one, which we donote by $P^*_{k+1,m}(x)$, we can obtain another corrector, say $c_{m+1}(k+1)$, of order higher by one. According to the error analysis,

$$E_{m+1} \equiv c_{m+1}(k+1) - c_{m+1} \qquad (4.9)$$

can be used as an estimate of the local error of c_{m+1}. Consequently, if the magnitude of E_{m+1} is within a tolerance, this subroutine consider c_{m+1} to meet the required accuracy, and take c_{m+1} $(k + 1)$, instead of c_{m+1}, as a solution to be output.

In what follows, $y_{m+1}(k)$ stands for c_{m+1}, and y_{m+1} for c_{m+1} $(k + 1)$.

The procedures to obtain the solution at x_{m+1} are summarized as follows:

- Prediction ... p_{m+1}
- Evaluation ... $f(x_{m+1},p_{m+1})$
- Correction ... y_{m+1}
- Evaluation... $f(x_{m+1},y_{m+1})$

This is often called the PECE method.

2) Adams method based upon modified divided differences

Since points$\{x_{m+1-j}\}$ are unequally spaced, the computation for a predictor (4.4) and corrector (4.6) in that situation are described concretely.

$P_{k,m}(x)$ can be expressed using divided differences as follows:

$$\begin{aligned} P_{k,m}(x) = {} & f[x_m] + (x - x_m)f[x_m, x_{m-1}] + \ldots \\ & + (x-x_m)(x-x_{m-1})\ldots \\ & (x - x_{m+2-k})f[x_m, x_{m-1}, \ldots, x_{m+1-k}] \end{aligned} \qquad (4.10)$$

where,

$$f[x_m] = f_m$$
$$f[x_m, x_{m-1}, \ldots, x_{m-j}] = \frac{f[x_{m-1}, \ldots, x_{m-j}] - f[x_m, \ldots, x_{m+1-j}]}{x_m - x_{m-j}}$$
$$, j = 1,2,\ldots,k-1$$

In the integration formula based upon the expression in (4.10), points $\{x_{m+1-j}\}$ and divided differences are used in the program. However, this subroutine uses a sequence of step sizes$\{h_{m+1-j}\}$ instead of $\{x_{m+1-j}\}$, and modified devided differences instead of divided differences. The formula to be described below is reduced to (4.5) if the step size is constant. We introduce notations to be used later.

$$\begin{aligned} & h_i = x_i - x_{i-1} \\ & s = (x - x_m)/h_{m+1} \\ & \Psi_i(m+1) = h_{m+1} + h_m + \ldots + h_{m+2-i}, \quad i = 1,2,\ldots \\ & \alpha_i(m+1) = h_{m+1}/\Psi_i(m+1), \quad i = 1,2,\ldots \\ & \beta_1(m+1) = 1 \\ & \beta_i(m+1) = \frac{\Psi_1(m+1)\Psi_2(m+1)\ldots\Psi_{i-1}(m+1)}{\Psi_1(m)\Psi_2(m)\ldots\Psi_{i-1}(m)}, \quad i = 2,3,\ldots \\ & \Phi_1(m) = f[x_m] = f_m \\ & \Phi_i(m) = \Psi_1(m)\Psi_2(m)\ldots\Psi_{i-1}(m)f[x_m, x_{m-1}, \ldots, x_{m+1-i}] \\ & \qquad i = 2,3,\ldots \end{aligned} \qquad (4.11)$$

Here, $\Phi_i(m)$ is called a modified divided difference. When the step size is constant,

$\Psi_i(m+1) = ih, \quad \alpha_i(m+1) = 1/i$ and $\beta_i(m+1) = 1$

therefore $\Phi_i(m) = \nabla^{i-1} f_m$

Using above notations, the general term in (4.10) can be expressed by:

$$(x-x_m)(x-x_{-1})\ldots(x-x_{m+2-i})f[x_m, x_{m-1},\ldots,x_{m+1-i}]$$
$$=\left(\frac{sh_{m+1}}{\Psi_1(m+1)}\right)\cdot\left(\frac{sh_{m+1}+\Psi_1(m)}{\Psi_2(m+1)}\right)\ldots$$
$$\left(\frac{sh_{m+1}+\Psi_{i-2}(m)}{\Psi_{i-1}(m+1)}\right)\beta_i(m+1)\Phi_i(m) \tag{4.12}$$

To simplify the right hand side of this equation we introduce

$$\Phi_i^*(m) = \beta_i(m+1)\Phi_i(m)$$

and

$$c_{i,m}(s) = \begin{cases} 1 & , i = 1 \\ \dfrac{sh_{m+1}}{\Psi_1(m+1)} = s & , i = 2 \\ \left(\dfrac{sh_{m+1}}{\Psi_1(m+1)}\right)\cdot\left(\dfrac{sh_{m+1}+\Psi_1(m)}{\Psi_2(m+1)}\right)\ldots & \\ \quad\left(\dfrac{sh_{m+1}+\Psi_{i-2}(m)}{\Psi_{i-1}(m+1)}\right) & , i = 3,4,\ldots \end{cases}$$

Then (4.12) can be expressed by $c_{i,m}(s)\Phi_i^*(m)$.
Therefore we get

$$P_{k,m}(x) = \sum_{i=1}^{k} c_{i,m}(s)\Phi_i^*(m) \tag{4.13}$$

where $c_{i,m}(s)$ is a polynomial with respect to s of degree i -1, which is determined depending upon only the distribution of points $\{x_{m+1-j}\}$.
Substituting (4.13) into (4.4), and changing the integration variable x to s, the following can be obtained.

$$p_{m+1} = y_m + h_{m+1}\sum_{i=1}^{k}\left(\int_0^1 c_{i,m}(s)ds\right)\Phi_i^*(m) \tag{4.14}$$

This corresponds to (4.5) in which the step size is constant. The integral for $c_{i,m}(x)$ can be obtained by repeated integration by parts. The results are mentioned below, where sequence $\{g_{i,q}\}$ is produced by (4.15) for $i \geq 1$ and $q \geq 1$.

$$g_{1,q} = 1/q, g_{2,q} = 1/(q(q+1))$$
$$g_{i,q} = g_{i-1,q} - \alpha_{i-1}(m+1)g_{i-1,q+1} \quad , i \geq 3 \tag{4.15}$$
$$g_{i,1} = \int_0^1 c_{i,m}(s)ds$$

The following shows the triangular table for $g_{i,q}$ in (4.15) when the step size is constant.

$q \downarrow$ \ $i \rightarrow$	1	2	3	4	$\cdots$
1	1	1/2	5/12	3/8	
2	1/2	1/6	1/8		
3	1/3	1/12			
4	1/4				
:					

The values placed on the first row in this table correspond to $\{g_{i,1}\}$.
Therefore (4.14) is reduced to

$$p_{m+1} = y_m + h_{m+1}\sum_{i=1}^{k} g_{i,1}\Phi_i^*(m) \tag{4.16}$$

Next, the computation of corrector is described. For corrector y_{m+1}, the formula of order k + 1 which is one order higher than the predictor is used. The corrector y_{m+1} is based upon:

$$y_{m+1} = y_m + \int_{x_m}^{x_{m+1}} P_{k+1,m}^*(x)dx \tag{4.17}$$

Where, $P^*_{k+1,m}(x)$ is the interpolation polynomial satisfying not only the interpolation conditions for $P_{k,m}(x)$ but also

$$P_{k+1,m}^*(x_{m+1}) = f_{m+1}^p = f(x_{m+1}, p_{m+1})$$

Using a divided difference we can write.

$$P_{k+1,m}^*(x) = P_{k,m}(x) + (x-x_m)(x-x_{m-1})\ldots(x-x_{m+1-k})$$
$$\cdot f^p[x_{m+1},\ldots,x_{m+1-k}] \tag{4.18}$$

The index p indicates that in the divided difference f_{m+1}^p should be used instead of $f_{m+1} = f(x_{m+1}, y_{m+1})$.
Substituting (4.18) into (4.17), we get,

$$y_{m+1} = p_{m+1} + h_{m+1}g_{k+1,1}\Phi_{k+1}^p(m+1) \tag{4.19}$$

So the corrector can be expressed by adding the correction term to predictor p_{m+1}. Here, the index p also indicates that in the divided difference of $\Phi_{k+1}(m+1)$, f_{m+1}^p should be used instead of $f_{m+1} = f(x_{m+1}, y_{m+1})$. The k th order corrector $y_{m+1}(k)$ can be computed in the same way as follows:

$$y_{m+1}(k) = p_{m+1} + h_{m+1}g_{k,1}\Phi_{k+1}^p(m+1) \tag{4.20}$$

3) Solution at any point
Since the step size is taken as long as the error is within a tolerance generally the output point x_e satisfies

$$x_m < x_e \leq x_{m+1} \tag{4.21}$$

In obtaining the solution at x_e, although we could restrict the step size so as to h it x_e, this subroutine integrates beyond x_e with a step size optimally determined to get the solution at x_{m+1}, unless a particular situation is posed on the problem, e.g. when $f(x,y)$ is not defined beyond x_e. Then the solution at x_e is obtained as follows. Let $P_{k+1,m+1}(x)$ be an interpolation polynomial of degree k satisfying:

$$P_{k+1,m+1}(x_{m+2-j}) = f_{m+2-j}\,,\ j = 1,2,...,k+1$$

The solution at x_e is computed by

$$y_e = y_{m+1} + \int_{x_{m+1}}^{x_e} P_{k+1,m+1}(x)dx \qquad (4.22)$$

If the solution at x_{m+1} satisfies the required accuracy, y_e also satisfies the required accuracy.

To compute (4.22), $P_{k+1,m+1}(x)$ should be expressed using modified divided differences.

First letting,

$$h_I = x_e - x_{m+1}\,,\ s = (x - x_{m+1})/h_I$$

$P_{k+1,m+1}(x)$ can be expressed as:

$$\begin{aligned}P_{k+1,m+1}(x) &= f[x_{m+1}] + (x - x_{m+1})f[x_{m+1}, x_m] + ...\\ &\quad + (x - x_{m+1})...(x - x_{m+2-k})f[x_{m+1},...,x_{m+1-k}]\end{aligned}$$

The general term is:

$$\begin{aligned}&(x - x_{m+1})(x - x_m)...(x - x_{m+3-i})f[x_{m+1}, x_m,..., x_{m+2-i}]\\ &= \left(\frac{sh_I}{\Psi_1(m+1)}\right)\cdot\left(\frac{sh_I + \Psi_1(m+1)}{\Psi_2(m+1)}\right)...\\ &\quad \left(\frac{sh_I + \Psi_{i-2}(m+1)}{\Psi_{i-1}(m+1)}\right)\Phi_i(m+1)\end{aligned}$$

After expressing $P_{k+1,m+1}(x)$ using modified divided differences and substituting it into (4.22), the following is obtained:

$$y_e = y_{m+1} + h_I \sum_{i=1}^{k+1} g_{i,1}^I \Phi_i(m+1) \qquad (4.23)$$

where $g_{i,1}^I$ are generated by the following recurrence equation:

$$\begin{aligned}g_{1,q}^I &= 1/q\\ g_{i,q}^I &= \xi_{i-1} g_{i-1,q}^I - \eta_{i-1} g_{i-1,q+1}^I\,, i \geq 2\end{aligned} \qquad (4.24)$$

where

$$\xi_i = \begin{cases} h_I/\Psi_1(m+1) & ,i = 1\\ \dfrac{h_I + \Psi_{i-1}(m+1)}{\Psi_i(m+1)} & ,i \geq 2\end{cases}$$

$$\eta_i = h_I/\Psi_i(m+1) \quad ,i \geq 1$$

4) Acceptance of solutions
Suppose local error of k th order corrector $y_{m+1}(k)$ to be $le_{m+1}(k)$
It can be estimated by the difference between(k + 1)th order corrector y_{m+1} and $y_{m+1}(k)$.
From (4.19) and (4.20), we can see the following:

$$|le_{m+1}(k)| \approx \left|h_{m+1}(g_{k+1,1} - g_{k,1})\Phi_{k+1}^p(m+1)\right| \qquad (4.25)$$

Defining the right side of the equation as ERR, if for a given tolerance ε.

$$\text{ERR} \leq \varepsilon \qquad (4.26)$$

is satisfied, the subroutine accepts y_{m+1} as the solution at x_{m+1}, then evaluates $f(x_{m+1},y_{m+1})$.
This completes the current step.

5) Order control
Control of the order is done at each step by selecting the order of the formula of the Adams method. This means to select the degree of the interpolation polynomial which approximates $f(x,y)$. This selection is performed before the selection of the step size.
Suppose the solution y_{m+1} at x_{m+1} has been accepted by the k th order Adams method and the order for the next step is going to be selected. The local error at x_{m+2} can be determined according to the step size h_{m+2} to be selected later and the derivative at x_{m+2}. Since these are not known when the order is selected, the error cannot be known correctively. Therefore using only those values which are available so far, the subroutine estimates the local errors at x_{m+2} of order k –1, k, and k + 1 as ERKM1, ERK, and ERKP1 below.

$$\left.\begin{aligned}\text{ERKM1} &= \left|h\gamma_{k-1}^*\sigma_k(m+1)\Phi_k^p(m+1)\right|\\ \text{ERK} &= \left|h\gamma_k^*\sigma_{k+1}(m+1)\Phi_{k+1}^p(m+1)\right|\\ \text{ERKP1} &= \left|h\gamma_{k+1}^*\Phi_{k+2}(m+1)\right|\end{aligned}\right\} \qquad (4.27)$$

where,

$$\sigma_i(m+1) = \begin{cases} 1 & ,i = 1\\ \dfrac{h\cdot 2h...(i-1)h}{\Psi_1(m+1)\Psi_2(m+1)...\Psi_{i-1}(m+1)} & ,i = 2,3,4,...\end{cases}$$

$$h = h_{m+1} = x_{m+1} - x_m$$

γ_i^* is the coefficient defined in (4.7).

ERKP1 is estimated only when the preceding k + 1 steps have been taken with a constant step size h. This is because if the step size is not constant, the value of ERKP1 may not be reliable.

In addition to (4.27), the local error of order k –2 is also estimated as ERKM2 below.

$$\text{ERKM2} = \left|h\gamma_{k-2}^*\sigma_{k-1}(m+1)\Phi_{k-1}^p(m+1)\right|$$

The order is controlled based on the above introduced values.

a) When one of the following is satisfied, reduce the order to k-1.

– $k > 2$ and max (ERKM1, ERKM2) ≤ ERK
– $k = 2$ and ERKM1 ≤ 0.5ERK
– The preceding $k + 1$ steps have been taken with a constant step size and ERKM1 ≤ min (ERK, ERKP1)

b) When the preceding $k + 1$ steps have been taken with a constant step size and one of the following is satisfied, increase the order to $k + 1$.
– $1 < k < 12$ and ERKP1 < ERK< max (ERKM1, ERKM2)
– $k = 1$ and ERKP1 < 0.5ERK

c) The order is not changed at all if neither a) or b) above are satisfied.

At each step, the above mentioned a), b) and c) are tested sequentially.

6) Step size Control
The step size is controlled at each step after the order has been selected. Suppose order k for the next step has been selected and the most recently used step size is h. If the next step is processed by step size $r \cdot h$, the local error can be estimated by:

$$r^{k+1}\ \mathrm{ERK} \qquad (4.28)$$

Therefore, the ratio r may be taken a value as large as possible while the value in (4.28) does not exceed the error tolerance ε. However, for safety, 0.5 ε is used instead of ε, and r is determined with the following conditions;

$$r^{k+1}\ \mathrm{ERK} \le 0.5\,\varepsilon \qquad (4.29)$$

If the step size varies, it will after the coefficients of predictor and corrector formulas. If the step size varies much, it will possibly cause undesirable error propagation property of the formula. Taking this into consideration, when increasing the step size, it should be increased by a factor of two at most, and when decreasing the step size, it should be halved at most. The step size is controlled as follows:

a) When increasing the step size, if

$$2^{k+1}\ \mathrm{ERK} \le 0.5\,\varepsilon$$

the step size is increased by a factor of 2.

b) When decreasing the step size, if

$$\mathrm{ERK} > 0.5\,\varepsilon$$

the actual rate is determined with $r = (0.5 / \mathrm{ERK})^{1/(k+1)}$ according to

$$r' = \min\,(0.9, \max(1/2, r))$$

c) If neither a) nor b) are satisfied, the step size is not changed at all.

7) When the solution could not be accepted
When the criterion (4.26) for accepting the solution is not satisfied, the step is retried. The order is selected by testing a) in item 5, that is, the order will either be decreased or not be changed at all. On the other hand, the step size is halved unconditionally.

If the step is tried three times and (4.26) has still not been satisfied, the first order Adams method, that is, Euler's method, is used and the step size is halved until (4.26) is satisfied.

8) Stating procedure
At the first step in solving the given initial value problem, the first order Adams method is used. Here, the method of determining the initial step size and the method of increasing the order and step size are described.

When we assume the initial step size is h_1, the local error in Euler's method at points x_1 ($= x_0 + h_1$) is estimated by

$$h_1^2 \left| f(x_0, y_0) \right|$$

h_1 can be selected so as not to exceed $0.5\,\varepsilon$. However, taking into consideration the possibility of $f(x_0,y_0) = 0$, and also for safety, the following is used:

$$h_1 = \min\left(0.25 \left| \frac{0.5\varepsilon}{f(x_0, y_0)} \right|^{1/2}, H \right) \qquad (4.30)$$

where,
$$H = \max\left(4u|x_0|, |x_e - x_0|\right)$$
u: unit of round-off error

At starting step, the order and step-size is controlled as follows:
The order is increased by one for subsequent steps and the step size is increased twice each time a step is processed. This is because the order should be increased rapidly for effectiveness, and also because h_1 of (4.30) tends to be conservative when ε is small. This control is terminated when the solution at a certain step does not meet the criterion, or when the order comes to 12, or a) in item 5) is satisfied. The subsequent orders and step sizes are controlled according to the previously mentioned general rules.

9) Extension to a system of differential equations
For a system of differential equations, the above mentioned procedures can be applied to each component of the solution vector.

However, each error estimate of ERR, ERK, ERKM1, ERKM2 and ERKP1 is defined as follows instead of defining to each component: From the user specified EPSA and EPSR, we introduce

$$\left.\begin{array}{l} \mathrm{EPS} = \max(\mathrm{EPSA}, \mathrm{EPSR}) \\ \mathrm{W(I)} = \left(|\mathrm{Y(I)}|\mathrm{EPSR} + \mathrm{EPSA}\right)/\mathrm{EPS} \end{array}\right\} \qquad (4.31)$$

then ERR is defined as

$$\mathrm{ERR} = \left(\sum_{I=1}^{N} \left(\frac{le(I)}{W(I)} \right)^2 \right)^{1/2} \qquad (4.32)$$

where, Y(I) and l_e(I) are the I-th component of the solution vector and its local error respectively. ERK and ERKM1 are defined similarly. The criterion used for accepting the solution is

$$\mathrm{ERR} \le \mathrm{EPS}$$

Note that when this is satisfied, from (4.31) and (4.32), we can see

$$le(I) \le |Y(I)|\mathrm{EPSR} + \mathrm{EPSA}$$
$$, I = 1,2,\ldots,\mathrm{N}$$

This subroutine is based on the code (Reference [71]) by L.F. Shampine and M.K. Gordon.

H11-20-0151 ODGE, DODGE

A stiff system of first order ordinary differential equations (Gear's method)
CALL ODGE (X, Y, FUN, N, XEND, ISW, EPSV, EPSR, MF, H, JAC, VW, IVW, ICON)

Function

This subroutine solves a system of first order ordinary differential equations of the form:

$$\left.\begin{array}{l} y_1' = f_1(x, y_1, y_2, ..., y_N), y_1(x_0) = y_{10} \\ y_2' = f_2(x, y_1, y_2, ..., y_N), y_2(x_0) = y_{20} \\ \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ y_N' = f_N(x, y_1, y_2, ..., y_N), y_N(x_0) = y_{N0} \end{array}\right\} \quad (1.1)$$

by Gear's method or Adams method, when functions f_1, f_2,...,f_N and initial values x_0, y_{10}, y_{20}, ..., y_{N0} and the final value of x, x_e are given. That is, it obtains the solutions (y_{1m}, y_{2m}, ..., y_{Nm}) at points $x_m = x_0 + \sum_{j=1}^{m} h_j ; m = 1, 2, ..., e$ (See fig. ODGE-1). The step size is controlled so that solutions satisfy the desired accuracy.

Gear's method is suitable for stiff equations, whereas Adams method is suitable for nonstiff equations. The user may select either of these methods depending on stiffness of the equations. This subroutines provides two types of output mode as shown below. The user can select the appropriate mode according to his purpose:

Final value output ... Returns to the user program when the solution at final value x_e is obtained.

Step output ... Returns to the user program each time the solution at x_1, x_2, ... is obtained.

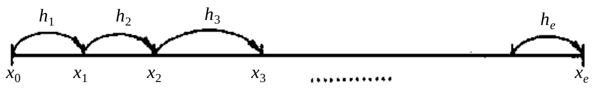


Fig. ODGE-1 Solution output point x_m (in the case $x_0 < x_e$)

Parameters

X ..... Input. Starting point x_0.
Output. Final value x_e. When the step output is specified, an interim point to which the solution is advanced a single step.

Y ..... Input.
Initial values y_{10},y_{20}, ..., y_{N0}. They must be given in the order of Y(1) = y_{10}, Y(2) = y_{20}, ..., Y(N) = y_{N0}
One-dimensional array of size N
Output: Solution vector at final value x_e. When the step output is specified, the solution vector at $x = x_m$.

FUN ..... Input. The name of subroutine subprogram which evaluates f_i, i=1, 2,...,N in (1.1).
The form of the subroutine is as follows:
SUBROUTINE FUN (X, Y,YP)
where
X ..... Input. Independent variable x.
Y ..... Input. One-dimensional array of size N, with the correspondence Y(1) = y_1, Y(2) = y_2, ..., Y(N) = y_N
YP .....Output. One-dimensional array of size N, with the correspondence
$\mathrm{YP}(1) = f_1(x, y_1, y_2, ..., y_N)$,
$\mathrm{YP}(2) = f_2(x, y_1, y_2, ..., y_N)$,...,
$\mathrm{YP}(\mathrm{N}) = f_N(x, y_1, y_2, ..., y_N)$

N ..... Input. Number of equations in the system.

XEND. Input. Final point x_e to which the system should be solved.

ISW ..... Input. Integer variable to specify conditions in integration. ISW is a nonnegative integer having four decimal digits, which can be expressed as
ISW= $1000d_4 + 100d_3 + 10d_2 + d_1$
Each d_i should be specified as follows.
d_1 ..... Specifies whether or not this is the first call
0 ..... First call.
1 ..... Successive call
The first call means that this subroutine is called for the first time for the given differential equations.
d_2 ..... Specifies the output mode.
0 ..... Final value output.
1 ..... Step output.
d_3 ..... Indicates whether or not functions f_1, f_2,...,f_N can be evaluated beyond the final value x_e.
0 ..... Permissible.
1 ..... Not permissible. The situation in which the user sets d_3 = 1 is when derivatives are not defined beyond x_e, or there is a discontinuity there. However, the user should be careful that if this is not the case specifying d_3 = 1 leads to unexpectedly inefficient computation.
d_4 ..... Indicates whether or not the user has altered some of the values of MF, EPSV, EPSR and N:
0 .... Not altered.
1 ..... Altered. See 6 and 7 in

"Comments on Use" for specification.

Output. When the solutions at x_e or at an interim point are reterned to the user program, the values of d_1, d_3, and d_4 are altered as follows.

d_1 ..... Set to 1: On subsequent calls, d_1 should not be altered by the user. Resetting $d_1 = 0$ is needed only when the user starts solving another system of equations.

d_3 ..... When $d_3 = 1$ on input, change it to $d_3 = 0$ when the solution at x_e is obtained.

d_4 ..... When $d_4 = 1$ on input, change it to $d_4 = 0$.

EPSV ..... Input. Absolute error tolerances for components of the solution. One-dimensional array of size N, where EPSV(L) ≥ 0.0, L = 1,2, ..., N
Output. If EPSV(L) is too small, the value is changed to an appropriate value. (See 4 in "Comments on Use.")

EPSR ..... Input Relative error tolerance.
Output. If EPSR is too small, the value is changed to an appropriate value. (See 4 in "Comments on Use.")

MF ..... Input. Method indicator. MF is an integer with two decimal digits represented as MF = 10*METH + ITER.
METH and ITER are the basic method indicator and the corrector iteration method respectively, with the following values and meanings.

METH... 1 for Gear's method. This is suitable for stiff equations
2 for Adams method. This is suitable for nonstiff equations.

ITER ... 0 for Newton method in which the analytical Jacobian matrix $\boldsymbol{J} = \left(\partial f_i / \partial y_j\right)$ is used. The user must prepare a subroutine to calculate the Jacobian matrix. (See explanations about parameter JAC.)
1 for Newton method in which the Jacobian matrix is internally approximated by finite difference.
2 for Same as ITER = 1 except the Jacobian matrix is approximated by a diagonal matrix.
3 for Function iteration in which the Jacobian matrix is not used.
For stiff equations, specify ITER = 0, 1 or 2.0 is the most suitable value, but if the analytical Jacobian matrix cannot be prepared, specify 1. Specify ITER = 2 if it is known that the Jacobian matrix is a diagonally dominant matrix.
For nonstiff equations, specify ITER = 3

H ..... Input. Initial step size (H≠0) to be attempted for the first step on the first call
The sign of H must be the same as that of $x_e - x_0$. A typical value of |H| is

$$|\mathrm{H}| = \min\left(10^{-5}, \max\left(10^{-4}|x_0|, |x_e - x_0|\right)\right)$$

The value of H is controlled to satisfy the required accuracy.
Output. The step size last used.

JAC ... Input. The name of subroutine subprogram which evaluates the analytical Jacobian matrix:

$$\mathrm{J} = \begin{bmatrix} \dfrac{\partial f_1}{\partial y_1} & \dfrac{\partial f_1}{\partial y_2} \cdots & \dfrac{\partial f_1}{\partial y_N} \\ \dfrac{\partial f_2}{\partial y_1} & \dfrac{\partial f_2}{\partial y_2} \cdots & \dfrac{\partial f_2}{\partial y_N} \\ \vdots & \vdots & \vdots \\ \dfrac{\partial f_N}{\partial y_1} & \dfrac{\partial f_N}{\partial y_2} \cdots & \dfrac{\partial f_N}{\partial y_N} \end{bmatrix}$$

The form of the subroutine is as follows.
SUBROUTINE JAC (X, Y, PD, K)
where,

X ..... Input. Independent variable *x*.

Y ..... Input. One-dimensional array of size N, with the correspondance Y(1) = y_1, Y(2) =y_2, ..., Y(N) = y_N

PD ... Output. Jacobian matrix stored in a two-dimensional array PD (K, K), where

$$\mathrm{PD}(i, j) = \frac{\partial f_i}{\partial y_j}$$

$1 \le i \le N$, $1 \le j \le N$

K ..... Input. Adjustable dimension of array PD. (See "Example.")

Even if the user specifies ITER ≠ 0, he must prepare a dummy subroutine for the JAC parameter as follows:

```
SUBROUTINE JAC (X, Y, PD, K)
RETURN
END
```

VW ..... Work area. One-dimensional array of size N(N +17) + 70. The contents of VW must not be altered on subsequent calls.

IVW ..... Work area. One-dimensional array of size N + 25. The contents of IVW must not be altered on subsequent calls.

ICON .... Output. Condition code.
(See Table ODGE-1.)

Table ODGE-1 Condition codes

Code	Meaning	Processing
0	A single steg has been taken (in step output).	Subsequent calls are possible.
10	Solution at XEND was obtained.	Subsequent calls are possible after changing XEND.
10000	EPSR and EPSV(L): L = 1, 2, .., N are too small for the arithmetic precision.	EPSR and EPSV(L): L = 1, 2, ..., N were increased to proper values. (Check the in EPSR and EPSV.)
15000	The requested accuracy could not be achieved even with a step size 10^{-10} times the initial step size.	
16000	The corrector iteration did not converge even with a step size 10^{-10} times the initial step size.	The methods specified through parameter MF may not be appropriate for the given equations. Change the MF parameter, then retry.
30000	One of the following occurred, 1) N≤ 0 2) X = XEND. 3) ISW specification error. 4) EPSR < 0 or there exists I such that EPSV(I) < 0. 5) (XEND – X)∗H≤0. 6) The IVW was changed (on the second or subsequent calls).	Bypassed

Comments on use

- Subprograms used

  SSL ... MGSSL, AMACH, USDE, UNIT2, USTE2, USETC, USETP, USOL, UADJU, UDEC

  FORTRAN basic functions ... MOD, FLOAT, AMAX1, AMIN1, ABS, SORT

- Notes

  This subroutine can be effectively used for stiff equations or those which is initially nonstiff but change to be stiff in the integration interval. For nonstiff equations, use subroutine ODAM or ODRK1 for efficiency.

  The names of the subroutines associated with parameter FUN and JAC must be declared as EXTERNAL in the calling program.

  Judgement by ICON

  When the user specifies the final value output, he can obtain the solution at x_e only when ICON = 10.

  When the step output is specified, the user can obtain the solution at each step when ICON = 0. ICON = 10 indicates that the solution at x_e has been obtained.

Parameters EPSV and EPSR

Let Y(L) be the L-th component of the solution vector and l_e(L) be its local error, then this subroutines controls the error so that

$$|l_e(L)| \le \mathrm{EPSR} \times |\mathrm{Y(L)}| + \mathrm{EPSV(L)} \qquad (3.1)$$

is satisfied for L = 1, 2, ..., N. When EPSV(L) = 0 is specified, the relative error is tested. If EPSR = 0, the absolute error is tested with different tolerances between components.

Note the following in specifying EPSV(l) and EPSR:

Relative error test is suitable for the components which range over different orders in the integration interval.

Absolute error test may be used for the components which vary with constant orders or so small as to be of no interest.

However, it would be most stable and economical to specify:

EPSV(L)≠0, L=1, 2, ...,N

EPSR≠0

In this case, relative errors are tested for the large components and absolute errors are tested for the small components. In the case of stiff equations the orders of magnitude of the components might be greatly different, it is recomendable to specify different EPSV(L) between the components.

If EPSR = 0 and some of the elements of EPSV are zero on input, the subroutine after EPSR to 16*u*, where *u* is the round off unit.

Parameter XEND

If the solutions at a sequence of values of the independent variable are required, this subroutine should be successively called changing XEND. For this purpose, the subroutine sets the values of parameters required for the next call when returning to the user program. Therefore the user can call the subroutine only by changing XEND.

Changing MF during the solution

If the given equations are non-stiff initially and stiff later in the integration interval, it is desirable to change the value of MF from 23 to 10 (or 11 or 12) as follows when the equations become stiff:

Set d_4 of ISW to 1. This can be done with the statement ISW = ISW + 1000.

Change the value of MF.

Set XEND to the value of the next output point, then call this subroutine.

This subroutine clears the d_4 of ISW by

ISW= MOD (ISW, 1000)

indicating that the user's request has been completely accomplished on return to the user program.

However, if the solution at XEND can be readily obtained without changing MF, MF is not changed on return. This means that the value of MF is changed only when the value of XEND reaches a value where the method should be changed.

Changing parameters N, EPSV, and EPSR during the solution
The user can change the values of parameters N, EPSV, and EPSR during the solution.
However, considerable knowledge about the behavior of the solution would be required for changing the value of N as follow:

In the solution of stiff equations, some components of the solution do not change so much as compared with the other components or some components become small enough to be neglected. If these components are of no interest to the user it will be practical to regard these components as constants thereafter then to integrate only the remainder. This reduces the amount of computations. To change the value of parameter N means to reduce the number of components to be intergrated. In this subroutine, some of the last components of the system (1.1) are removed. Therefore, the user must arrange the components prior to the solution.

Based on the above, reduction of parameter N can be done in the manner described below:

Suppose that initial value of N is N_0 and changed to N_c ($<N_0$) during the solution, ($N_0 – N_c$) equations of the last potion of the system are removed, so the reduced system of equations are solved. In this case, the components Y(L): L = $N_c + 1, N_c + 2, ..., N_0$, are kept constant in the system reduced.

In the user-prepared subroutines, FUN and JAC, derivatives and Jacobian matrices only for N_c equations need be calculated. To identify the change of N in these routines, specify parameter N in COMMON statements.

Values of EPSV and EPSR can also be changed if necessary in the same way as parameter MF explained above.

- Example

The following example solves the system:

$$\begin{cases} y_1' = y_2 & , y_1(0) = 1 \\ y_2' = -11y_2 - 10y_1 & , y_2(0) = -1 \end{cases}$$

in the interval [0, 100]. The system is stiff because the eigenvalues of the Jacobian matrix are –1 and –10. In the following example, MF = 10, EPSR = 10^{-4} and EPSV(1) = EPSV(2) = 0 are used. The solution at the following points are to be obtained.

$x_j = 10^{-3+j}$, j = 1, 2, ..., 5

For this purpose, this subroutine is called repeatedly by setting XEND to x_j.

```
C     **EXAMPLE**
      DIMENSION Y(2),EPSV(2),VW(110),IVW(30)
      EXTERNAL FUN,JAC
      X=0.0
      Y(1)=1.0
      Y(2)=-1.0
      N=2
      EPSR=1.0E-4
      EPSV(1)=0.0
      EPSV(2)=0.0
      MF=10
      ISW=0
      H=1.0E-5
C
      WRITE(6,600)
C
      XEND=1.0E-3
      DO 40 I=1,5
      XEND=XEND*10.0
   10 CALL ODGE(X,Y,FUN,N,XEND,ISW,EPSV,
     *EPSR,MF,H,JAC,VW,IVW,ICON)
      IF(ICON.EQ.10) GO TO 30
      IF(ICON.EQ.10000.OR.ICON.EQ.15000)
     *GO TO 20
      IF(ICON.EQ.16000) WRITE(6,630)
      STOP
   20 WRITE(6,620) EPSR,EPSV(1),EPSV(2)
      GO TO 10
   30 WRITE(6,610) X,Y(1),Y(2)
   40 CONTINUE
      STOP
  600 FORMAT('1',12X,'X',22X,'Y(1)',
     *16X,'Y(2)'/)
  610 FORMAT(6X,E15.8,10X,2(E15.8,5X))
  620 FORMAT(10X,'TOLERANCE RESET',5X,
     *'EPSR=',E12.5,5X,'EPSV(1)=',E12.5,
     *5X,'EPSV(2)=',E12.5)
  630 FORMAT(10X,'NO CONVERGENCE IN',
     *' CORRECTOR ITERATION')
  640 FORMAT(10X,'INVALID INPUT')
      END

      SUBROUTINE FUN(X,Y,YP)
      DIMENSION Y(2),YP(2)
      YP(1)=Y(2)
      YP(2)=-11.0*Y(2)-10.0*Y(1)
      RETURN
      END

      SUBROUTINE JAC(X,Y,PD,K)
      DIMENSION Y(2),PD(K,K)
      PD(1,1)=0.0
      PD(1,2)=1.0
      PD(2,1)=-10.0
      PD(2,2)=-11.0
      RETURN
      END
```

Method

Both Gear's and Adams methods with step size and order controls are used in this subroutine.

Gear's method is suitable for stiff equations, whereas Adams method is suitable for non-stiff equations.

Both methods are multistep methods. The subroutine employs the identical strategies for storing the solutions at the past points and for controling the step size and order between the two methods. Gear's method will be described below first, then the step size control and order control will follow. The modifications necessary to be made when Adams method is used are also presented.

For simplicity, consider a single equation

$$y'=f(x,y)\ ,\ y(x_0)=y_0$$

for which the computed solution at x_m (referred to as the solution hereafter) is denoted by y_m and the true solution is referred to as $y(x_m)$. The value of $f(x_m, y_m)$ is represented shortly as f_m, which must be distinguished from $f(x_m, y(x_m))$.

1) Principle of Gear's method

Assume that $y_0, y_1, ..., y_m$ are known and the solution y_{m+1} at

$$x_{m+1} = x_m + h_{m+1}$$

is being obtained.

The fundamental ideas of Gear's method are best stated as follows:

Now, we consider the interpolation polynomial $\pi_{m+1}(x)$ of degree k satisfying $k + 1$ conditions

$$\pi_{m+1}(x_{m+1-j}) = y_{m+1-j} \quad , j = 0,1,2,...,k \tag{4.1}$$

Equation (4.1) contains solution y_{m+1} as an unknown parameter. The value of y_{m+1} is determined so that the derivative of the polynomial $\pi_{m+1}(x)$ at x_{m+1} is equal to $f(x_{m+1}, y_{m+1})$, that is, to satisfy

$$\pi'_{m+1}(x_{m+1}) = f(x_{m+1}, y_{m+1}) \tag{4.2}$$

If we represent $\pi_{m+1}(x)$ as a Lagrangian form, and differentiate it at $x = x_{m+1}$, (4.2) gives

$$h_{m+1} f(x_{m+1}, y_{m+1}) = -\sum_{j=0}^{k} \alpha_{m+1,j}\, y_{m+1-j} \tag{4.3}$$

where $\alpha_{m+1,j}$ are constants determined by the distribution of $\{x_{m+1-j}\}$. Generally, since $f(x_{m+1}, y_{m+1})$is nonlinear with respect to y_{m+1}, (4.3) becomes nonlinear with respect to y_{m+1}; therefore, y_{m+1} is calculated using an iteration method with an appropriate initial value. An initial value can be determined as follows:

Suppose the polynomial $\pi_m(x)$to be an interpolation polynomial of degree k satisfying.

$$\left.\begin{aligned} &\pi_m(x_{m+1-j}) = y_{m+1-j}\ ,\ j = 1,2,...k \\ &\pi'_m(x_m) = f(x_m, y_m) \end{aligned}\right\} \tag{4.4}$$

Then, p_{m+1} obtained from

$$p_{m+1} = \pi_m(x_{m+1}) \tag{4.5}$$

is regarded as an approximation at x_{m+1} and is used as the initial value for the iteration.

In Gear's method explained above, p_{m+1} is referred to as the predictor and y_{m+1} obtained in the iteration method is referred to as the corrector. Especially, in the Gear's method when k = 1, that is, of order one

$$p_{m+1} = y_m + h_{m+1} f(x_m, y_m) \tag{4.6}$$

is used as the predictor, and the solution y_{m+1} is obtained from

$$h_{m+1} f(x_{m+1}, y_{m+1}) = y_{m+1} - y_m \tag{4.7}$$

(4.6) and (4.7) are called the Euler method and the backward Euler method, respectively.

Since Gear's method is based on the backward differentiation formula as expressed in (4.3), it is referred to as the backward differentiation formula (BDF) method.

2) Nordsieck form for Gear's method

The predictor is calculated from $\pi_m(x)$, whereas the corrector is calculated from $\pi_{m+1}(x)$ as explained above. Since the corrector obtained determines the $\pi_{m+1}(x)$, the calculation of the corrector means to generate $\pi_{m+1}(x)$. The determined $\pi_{m+1}(x)$ is used to calculate the predictor in the next step. This means that one integration step consists of generating $\pi_{m+1}(x)$ from $\pi_m(x)$; $\pi_m(x)$ can be expanded into a Taylor series as

$$\pi_m(x) = y_m + (x - x_m)y'_m + (x - x_m)^2 \frac{y''_m}{2!} + \dots + (x - x_m)^k \frac{y_m^{(k)}}{k!} \tag{4.8}$$

$$\text{where } y_m^{(q)} = \frac{d^q}{dx^q}\pi_m(x_m)$$

In this subroutine, coefficients of (4.8) and step size $h = h_{m+1} = x_{m+1} - x_m$ are used to express the row vector

$$z_m = \left(y_m, hy'_m, ..., h^k y_m^{(k)}/k!\right) \tag{4.9}$$

This is referred to as the Nordsieck expression for the history of solution and is used to express $\pi_m(x)$; therefore, the problem to generate $\pi_{m+1}(x)$ from $\pi_m(x)$ is reduced to obtain the following from z_m:

$$z_{m+1} = \left(y_{m+1}, hy'_{m+1}, ..., h^k y^{(k)}_{m+1} / k!\right) \qquad (4.10)$$

Using (4.8), we can express predictor p_{m+1} as

$$p_{m+1} = \pi_m(x_{m+1}) = \sum_{i=0}^{k} h^i y^{(i)}_m / i! \qquad (4.11)$$

The right-hand side is the sum of elements of z_m. To simplify conversion from z_m to z_{m+1} in the prediction, the following is calculated for (4.11)

$$\boldsymbol{z}_{m+1(0)} = \boldsymbol{z}_m \boldsymbol{A} \qquad (4.12)$$

where $\boldsymbol{A}$ is the $(k + 1) \times (k + 1)$ unit lower triangular matrix defined as

$$a_{ij} = \begin{cases} 0 & ,i < j \\ \begin{pmatrix} i \\ j \end{pmatrix} = \dfrac{i!}{i!(i-j)!} & ,i \geq j \end{cases}$$

where $\boldsymbol{A} = (a_{ij})$. For example, $\boldsymbol{A}$ is expressed as follows when $k = 5$:

$$A = \begin{bmatrix} 1 & & & & & \\ 1 & 1 & & & 0 & \\ 1 & 2 & 1 & & & \\ 1 & 3 & 3 & 1 & & \\ 1 & 4 & 6 & 4 & 1 & \\ 1 & 5 & 10 & 10 & 5 & 1 \end{bmatrix}$$

Since the lower triangular portion of A is the Pascal's triangle, this is referred to as the Pascal's triangle matrix.

Let the first element of $z_{m+1(0)}$ obtained from (4.12) be $y_{m+1(0)}$, then p_{m+1}

$$y_{m+1(0)} = y_m + hy'_m + ... + h^k y^{(k)}_m / k!$$

which is equal to p_{m+1}. So $y_{m+1(0)}$ is used hereafter for p_{m+1}. The $(i + 1)$-th element of $z_{m+1(0)}$is equal to $h^i \pi^{(i)}_m(x_{m+1})/i!$, so calculation of (4.12) means to predict the solution and higher order derivatives at x_{m+1}.

On the other hand, the corrector is calculated from a relation between $z_{m+1(0)}$ and z_{m+1}, which is derived below: The $(i + 1)$ -th element of $z_{m+1} - z_{m+1(0)}$ is obtained from

$$\begin{aligned} &\frac{h^i \pi^{(i)}_{m+1}(x_{m+1})}{i!} - \frac{h^i \pi^{(i)}_m(x_{m+1})}{i!} \\ &= \frac{h^i}{i!} \frac{d^i}{dx^i} \left\{\pi_{m+1}(x) - \pi_m(x)\right\}_{x = x_{m+1}} \end{aligned} \qquad (4.13)$$

Since the polynomial $\Delta_{m+1}(x) \equiv \pi_{m+1}(x) - \pi_m(x)$ has degree k and satisfies conditions

$$\Delta_{m+1}(x_{m+1-j}) = \begin{cases} 0 & , j = 1,2,...,k \\ y_{m+1} - y_{m+1(0)} & , j = 0 \end{cases}$$

$\Delta_{m+1}(x)$ is determined uniquely and can be expanded at x_{m+1} in to a Taylor series

$$\Delta_{m+1}(x) = \left(y_{m+1} - y_{m+1(0)}\right) \sum_{q=0}^{k} l_q \left(x - x_{m+1}\right)^q / h^q \qquad (4.14)$$

where l_q is determined depending on the distribution of $\{x_{m+1-j}\}$and more precisely, it is the coefficient of t^q in the expression

$$\left(1 + \frac{t}{s_1}\right)\left(1 + \frac{t}{s_2}\right) \cdots \left(1 + \frac{t}{s_k}\right) \qquad (4.15)$$

where $s_j = (x_{m+1} - x_{m+1-j})/h$. Although l_q also depends on values k and m, they are omitted in this expression. From (4.14), we can reduce the right-hand side of (4.13) to $(y_{m+1} - y_{m+1(0)})l_i$, therefore, by introducing

$$\boldsymbol{l} = \left(l_0, l_1, ..., l_k\right) \qquad (4.16)$$

the relation between z_{m+1} and $z_{m+1(0)}$ is expressed as

$$\boldsymbol{z}_{m+1} = \boldsymbol{z}_{m+1(0)} + \left(y_{m+1} - y_{m+1(0)}\right)\boldsymbol{l} \qquad (4.17)$$

This gives how $\boldsymbol{z}_{m+1}$ is generated after the corrector y_{m+1} is determined. The y_{m+1} can be calculated based on the relation between the second elements of both sides of (4.19), that is,

$$\left.\begin{aligned} hy'_{m+1} &= hy'_{m+1(0)} + \left(y_{m+1} - y_{m+1(0)}\right)l_1 \\ \text{where } \ y'_{m+1} &= f\left(x_{m+1}, y_{m+1}\right) \end{aligned}\right\} \qquad (4.18)$$

This can be written as

$$\left(y_{m+1} - y_{m+1(0)}\right) - \frac{h}{l_1}\left(f\left(x_{m+1}, y_{m+1}\right) - y'_{m+1(0)}\right) = 0 \qquad (4.19)$$

This means that y_{m+1} is just the zero of

$$G(u) \equiv \left(u - y_{m+1(0)}\right) - \frac{h}{l_1}\left(f\left(x_{m+1}, u\right) - y'_{m+1(0)}\right) \qquad (4.20)$$

, so it is obtained using the Newton's method (See later).

3) Solutions at an arbitrary value of the independent variable

Since the step size is taken as large as possible within the tolerable error, the following situation is typical about the point x_e at which the solution is to be obtained

$$x_m < x_e \leq x_{m+1} \qquad (4.21)$$

In this subroutine, after solution y_{m+1} at

x_{m+1} is determined, solution y_e at x_e is calculated using elements of z_{m+1} as follows:

$$\begin{aligned} y_e &= \pi_{m+1}(x_e) \\ &= \sum_{i=0}^{k} \{(x_e - x_{m+1})/h\}^i \left(h^i y_{m+1}^{(i)}/i!\right) \qquad (4.22) \\ &, h = h_{m+1} \end{aligned}$$

4) Acceptance of solutions
Let $le_{m+1}(k)$ denote the local error of the corrector y_{m+1}, then the subroutine estimate it as follows using the difference between the corrector and the predictor:

$$le_{m+1}(k) = -\frac{1}{l_1}\left\{1+\prod_{j=2}^{k}\left(\frac{x_{m+1}-x_{m+1-j}}{x_m - x_{m+1-j}}\right)\right\}^{-1}\left(y_{m+1}-y_{m+1(0)}\right) \qquad (4.23)$$

If, for some tolerance ε

$$|le_{m+1}(k)| \le \varepsilon \qquad (4.24)$$

is satisfied y_{m+1} is accepted as the solution at x_{m+1}, then z_{m+1} is generated from (4.17), to proceed to the next step.

5) Order and step size controls
This subroutine uses Gear's method of order1), 2), 3), 4), or 5) depending on the behavior of the solution. Suppose the order of Gear's method used at the previous step to be k, then the order at the current stop is either $k – 1$, k or $k + 1$.
Suppose that solution y_{m+1} at x_{m+1} has been accepted by the method of order k, and the order for the next step is to be determined. For that purpose, the subroutine estimates the local errors $le_{m+1}(k)$, $le_{m+1}(k – 1)$ as well as $le_{m+1}(k)$, where $le_{m+1}(k – 1)$ $le_{m+1}(k + 1)$ means the local errors at x_{m+1} if the methods of order $k – 1$, $k + 1$ respectively would have been used.
This subroutine estimates $le_{m+1}(k – 1)$ and $le_{m+1}(k + 1)$ from

$$le_{m+1}(k-1) = -\left(s_1 \cdot s_2 \ldots s_{k-1}/l_{k-1,1}\right)\left(h^k y_{m+1}^{(k)}/k!\right) \qquad (4.25)$$

$$le_{m+1}(k+1) = \frac{-s_{k+1}\left(e_{m+1} - Q_{m+1}\cdot e_m\right)}{(k+2)l_{k+1,1}\left\{1+\prod_{2}^{k}\left(\frac{x_{m+1}-x_{m+1-j}}{x_m - x_{m+1-j}}\right)\right\}} \qquad (4.26)$$

where $l_{k-1,1}$ and $l_{k+1,1}$ are coefficients of t^1 with k replaced by $k – 1$ and $k + 1$ in (4.15), and

$$\begin{aligned} e_{m+1} &= y_{m+1} - y_{m+1(0)} \\ Q_{m+1} &= (c_{m+1}/c_m)(h/h_m)^{k+1} \\ c_{m+1} &= s_1 \cdot s_2 \ldots s_k \left\{1+\prod_{2}^{k}\left(\frac{x_{m+1}-x_{m+1-j}}{x_m - x_{m+1-j}}\right)\right\}/(k+1)! \end{aligned}$$

Using the $le_{m+1}(q)$, $q = k – 1, k, k + 1$, the order is selected as follows. First from the tolerance ε, we calculate

$$\eta_q = \left(\varepsilon/|le_{m+1}(q)|\right)^{1/(q+1)}, q = k-1, k, k+1 \qquad (4.27)$$

where η_q means the tolerable scaling factor of the step size when order q is to be used for the next step.
Suppose

$$\eta_{q'} = \max_q (\eta_q) \qquad (4.28)$$

then q' is adopted as the order for the next step. This means that the step that results in the largest step size is adopted.
So, the next step size is determined as

$$\eta_{q'} \cdot h$$

If the solution is not accepted because the condition(4.24) is not satisfied, the step is retaken by reducing the step size to $\eta_k \cdot h$ without changing the order.

6) Adams method
Although Adams method is explained under "Method" of subroutine ODAM, this subroutine employs other computational procedures than those of ODAM. That is, while Adams method is represented in terms of the modified divided differences in subroutine ODAM, the Nordsieck form is used in this subroutine. Much of computational procedures are shared between Gear's and Adams methods in this subroutine.
In Adams method, however, values introduced above in Gear's method are modified as follows:

Nordsieck form in Adams method
In Adams method, the true solution $y(x)$ is approximated by the interpolation polynomial $P_{m+1}(x)$ of degree k given by the conditions

$$\left.\begin{aligned} P'_{m+1}(x_{m+1-j}) &= f(x_{m+1}, y_{m+1-j}) \\ j &= 0,1,\ldots,k-1 \\ P_{m+1}(x_m) &= y_m \end{aligned}\right\} \qquad (4.29)$$

(4.29) contains y_{m+1} as an unknown parameter, and y_{m+1} is determined so that

$$y_{m+1} = y_m + \int_{x_m}^{x_{m+1}} P'_{m+1}(x)dx \qquad (4.30)$$

Since(4.30) contains $f(x_{m+1}, y_{m+1})$ on the right-hand side, it generally becomes a nonlinear equation with respect to y_{m+1}.
To solve this equation, an initial value is calculated from the polynomial $P_m(x)$ of degree k given by the conditions

$$\left.\begin{array}{l} P_m'\left(x_{m+1-j}\right)=f\left(x_{m+1},y_{m+1-j}\right) \\ \qquad\qquad j=1,2,\ldots,k \\ P_m\left(x_m\right)=y_m \end{array}\right\} \tag{4.31}$$

as

$$p_{m+1}=P_m\left(x_{m+1}\right)=y_m+\int_{x_m}^{x_{m+1}}P_m'(x)dx \tag{4.32}$$

From the above, it follows that Adams method consists of generating $P_{m+1}(x)$ from $P_m(x)$. $P_m(x)$ can be expanded to a Taylor series at x_m

$$P_m(x)=y_m+\left(x-x_m\right)y_m'+\ldots+\left(x-x_m\right)^k y_m^{(k)}/k! \tag{4.33}$$

where $y_m^{(q)}$ q=0,1,..., k means $P_m^{(q)}(x_m)$. Using coefficients of (4.33) and step size $h=h_{m+1}=x_{m+1}-x_m$, we introduce a row vector

$$z_m=\left(y_m,hy_m',\ldots,h^k y_m^{(k)}/k!\right) \tag{4.34}$$

This is the Nordsieck form of polynomial $P_m(x)$ in Adams method.

Calculation of $z_{m+1(0)}$
The $z_{m+1(0)}$ can be calculated in the same way by (4.12) using z_m obtained from (4.34).

Relation between $z_{m+1(0)}$ and z_{m+1}
The (i +1)-th element of $z_{m+1}-z_{m+1(0)}$ is

$$\frac{h^i}{i!}\frac{d^i}{dx^i}\left\{P_{m+1}(x)-P_m(x)\right\}_{x=x_m+1} \tag{4.35}$$

Expanding the polynomial $\Delta_{m+1}(x)\equiv P_{m+1}(x)-P_m(x)$ to a Taylor series, we get

$$\Delta_{m+1}(x)=\left(y_{m+1}-y_{m+1(0)}\right)\sum_{k=0}^{k}l_q\left(x-x_{m+1}\right)^q/h^q \tag{4.36}$$

and by substituting this into (4.35), we can express (4.35) as $(y_{m+1}-y_{m+1(0)})l_i$, where l_q is the coefficient of t^q of the polynomial of degree k given by

$$\int_{-1}^{t}\prod_{1}^{k-1}\left(u+s_j\right)du\Bigg/\int_{-1}^{0}\prod_{1}^{k-1}\left(u+s_j\right)du \tag{4.37}$$

where $s_j=(x_{m+1}-x_{m+1-j})/h_{m+1}$. Using l_q, we obtain the relation between $z_{m+1(0)}$ and z_{m+1} which has the same from as (4.17).

Local error
Also, in Adams method, local errors at order $k-1$, k, and $k+1$ are estimated for the acceptance of the solution, step size and order control. Let these errors be $le_{m+1}(k-1)$, $le_{m+1}(k)$, $le_{m+1}(k+1)$, then these values are estimated as follows:

$$le_{m+1}(k-1)=\left\{k\int_{-1}^{0}\prod_{0}^{k-2}\left(u+s_j\right)du\right\}\left(h^k y_{m+1}^{(k)}/k!\right) \tag{4.38}$$

$$le_{m+1}(k)=\left\{kl_k\int_{-1}^{0}\prod_{0}^{k-1}\left(u+s_j\right)du\Big/s_k\right\}e_{m+1} \tag{4.39}$$

$$le_{m+1}(k+1)=\left\{kl_k\int_{-1}^{0}\prod_{0}^{k}\left(u+s_j\right)du\Big/Ls_k\right\}\left(e_{m+1}-Q_{m+1}e_m\right) \tag{4.40}$$

where

$$\begin{aligned} &e_{m+1}=y_{m+1}-y_{m+1(0)},L=k+1, \\ &Q_{m+1}=\frac{s_k\cdot l_k(m)}{s_k(m)\cdot l_q}\left(h/h_m\right)^{k+1} \end{aligned} \tag{4.41}$$

In (4.41), $s_k(m)$ and $l_k(m)$ are obtained by substituting m for m+1 in their definitions. Equations (4.38) to (4.40) are used to check the solution and to control the order and step size in the same way as Gear's method.

7) Extension to a system of differential equations
The above discussion can be applied to each component of the solution in the case of systems of differential equations; however, the error estimates, such as $le_{m+1}(k)$, must be defined as the norms of vector as follows.
From the user-specified EPSV(I), I = 1, 2, ..., N and EPSR we introduce
EPS = max (EPSR, max (EPSV(I)))
W(I)=(|Y(I)|·EPSR+EPSV(I))/EPS
and define ERK as

$$\mathrm{ERK}=\left(\sum_{I=1}^{N}\left(\frac{le(\mathrm{I})}{\mathrm{W(I)}}\right)^2\right)^{1/2} \tag{4.42}$$

where Y(I) and le(I) are the I-th component of the solution and its local error respectively. Under these conditions, the subroutine tests

ERK ≤ EPS

If this is satisfied,

$$\left|le(\mathrm{I})\right|\le\left|\mathrm{Y(I)}\right|\cdot\mathrm{EPSR}+\mathrm{EPSV(I)},\quad \mathrm{I}=1,2,\ldots,\mathrm{N}$$

is automatically satisfied.
Moreover, norms ERKM1 and ERKP1 which correspond to $le_{m+1}(k-1)$ and $le_{m+1}(k+1)$ respectively in the case of single equations are defined in the same way as (4.42). The order and step size are controlled based on (4.27) but with the following substitutions:

$$\begin{array}{ll} \varepsilon & \to \mathrm{EPS} \\ |\,le_{m+1}(k)\,\| & \to \mathrm{ERK} \\ |\,le_{m+1}(k-1)\,| & \to \mathrm{ERKM1} \\ |\,le_{m+1}(k+1)\,| & \to \mathrm{ERKP1} \end{array}$$

8) Corrector iteration
This subroutine calculates the corrector as a zero of a nonlinear algebraic equation as mentioned above.
This section summanzes how to obtain the zero in the case of systems of differential equations. In this case (4.20) is expressed as

$$\boldsymbol{G}(\boldsymbol{u}) \equiv \left(\boldsymbol{u} - \boldsymbol{y}_{m+1(0)}\right) - \frac{h}{l_1}\left(\boldsymbol{f}\left(x_{m+1}, \boldsymbol{u}\right) - \boldsymbol{y}'_{m+1(0)}\right) \qquad (4.43)$$

This subroutine solves (4.43) by the Newton's method expressed as

$$\left.\begin{aligned} &\boldsymbol{u}_{r+1} = \boldsymbol{u}_r - \boldsymbol{P}_{m+1,r}^{-1}\boldsymbol{G}(\boldsymbol{u}_r) \\ &\qquad , r = 0,1,2,..., \\ &\text{where} \quad \boldsymbol{u}_0 = \boldsymbol{y}_{m+1(0)} \\ &\boldsymbol{P}_{m+1,r} = \left.\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}_r} = \boldsymbol{I} - \frac{h}{l_1}\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}_r} \end{aligned}\right\} \qquad (4.44)$$

In the subroutine, $\boldsymbol{P}_{m+1.r}$ is not evaluated at each iteration, but

$$\boldsymbol{P}_{m+1,0} = \boldsymbol{I} - \frac{h}{l_1}\boldsymbol{J}_{m+1} \text{ where } \boldsymbol{J}_{m+1} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{y}_{m+1(0)}} \qquad (4.45)$$

is used instead.
This subroutine contains several methods for evaluating $\boldsymbol{P}_{m+1,0}$, For $\boldsymbol{P}_{m+1,0}$ calculation, there are several methods, one of which the user can select through the parameter MF. Let MF = 10*METH + ITER, then ITER is the indicator on how to evaluate $\boldsymbol{P}_{m+1,0}$, with the following values and meanings:

0: to evaluate $\boldsymbol{P}_{m+1,0}$ by the analytical Jacobian matrix $\boldsymbol{J}_{m+1}$. In this case, the user must prepare subroutine JAC to evaluate $\boldsymbol{J}_{m+1}$.

1. to approximate $\boldsymbol{P}_{m+1,0}$ by finite differences.

2: to approximate $\boldsymbol{P}_{m+1,0}$ with a diagonal matrix by finite differences

$$\boldsymbol{D}_{m+1} = \text{diag}\,(d_1, d_2, ..., d_N) \qquad (4.46)$$

where

$$d_i = [f_i(x_{m+1}, \boldsymbol{y}_{m+1(0)} + \boldsymbol{v}) - f_i(x_{m+1}, \boldsymbol{y}_{m+1(0)})]\,/v_i$$

$$\begin{aligned} \boldsymbol{v} &= -\,0.1\,\boldsymbol{G}(y_{m+1(0)}) \\ &= (v_1, v_2, ..., v_N)^T \end{aligned}$$

The approximation is effective only when $\boldsymbol{J}_{m+1}$ is diagonally dominant matrix.

3: to approximate $\boldsymbol{P}_{m+1,0}$ with the identity matrix $\boldsymbol{I}$. In this case (4.44) becomes a simple recurrence formula

$$\boldsymbol{u}_{r+1} = \boldsymbol{u}_r - \boldsymbol{G}(\boldsymbol{u}_r) \qquad (4.47)$$

This is sufficient to solve nonstiff equations.
Convergence of (4.44) is tested by

$$\left(\sum_{I=1}^{N}\left(\frac{\boldsymbol{u}_{r+1}^I - \boldsymbol{u}_r^I}{\mathrm{W}(I)}\right)^2\right)^{1/2} \le c^* \cdot \text{EPS} \qquad (4.48)$$

where $\boldsymbol{u}^I$ is the I-th element of vector $\boldsymbol{u}$ and c^* is a constant specific to the problem.
This subroutine is based on the code written by A.C. Hindmarsh and G.D.Byrne (References [76] and [77]).

H11-20-0131 ODRK1, DODRK1

A system of first order ordinary differential equations (Runge-Kutta-Verner method, step output, final value output)
CALL ODRK1 (X, Y, FUN, N, XEND, ISW, EPSA, EPSR, VW, IVW, ICON)

Function

This subroutine solves a system of first order ordinary differential equations of the form:

$$\left.\begin{array}{l} y_1' = f_1(x, y_1, y_2, ..., y_N), y_1(x_0) = y_{10} \\ y_2' = f_2(x, y_1, y_2, ..., y_N), y_2(x_0) = y_{20} \\ \quad : \qquad : \\ y_N' = f_N(x, y_1, y_2, ..., y_N), y_N(x_0) = y_{N0} \end{array}\right\} \qquad (1.1)$$

by Rung-Kutta-Verner method, when functions f_1, f_2 , ... ,f_N and initial values x_0, y_{10},y_{20}, ..., y_{N0} and the final value x_e are given, i.e. obtains the solution (y_{1m}, y_{2m}, ..., y_{Nm}) at $x_m = x_0 + \sum_{j=1}^{m} h$ (m=1,2,..., e)

(See Fig. ODRK 1-1). The step size h_j is controlled so that solutions satisfy the desired accuracy.

This subroutine provides two types of output mode as shown below. The user can select the appropriate mode according to this purposes.

- Final value output ... Returns to the user program when the solution at final value x_e is obtained.
- Step output ... Returns to the user program each time the solutions at x_1, x_2, ... are obtained.

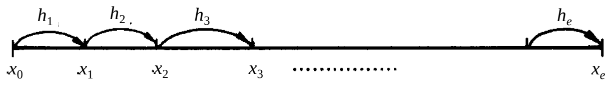


Fig. ODRK1-1 Solution output point x_m (in the case $x_0 < x_e$)

Parameters

X ..... Input. Starting point x_0.
Output. Final value x_e. When the step output is specified, an interim point x_m to which the solutions are advanced a single step.

Y ..... Input. Initial values y_{10}, y_{20}, ... y_{N0}. They must be given in order of Y(1) = y_{10}, Y(2) = y_{20}, ..., Y(N) = y_{N0}.
One-dimensional array of size N.
Output. Solution vector at final value x_e.
When the step output is specified, the solution vector at $x = x_m$.

FUN ..... Input. The name of the subprogram which evaluates f_i (i = 1, 2, ..., N) in (1.1).
The form of the subroutine is as follows:
SUBROUTINE FUN (X, Y, YP)
where
X: Input. Independent variable x.
Y: Input. One-dimensional array of size N, with corresponding Y(1) = y_1, Y(2) = y_2, ... , Y(N) = y_N.
YP: Output. One-dimensional array of size N, with corrosponding YP(1)=$f_1(x, y_1, y_2, ..., y_N)$, YP(2)=$f_2(x, y_1, y_2, ..., y_N)$, YP(N)=$f_N(x, y_1, y_2, ..., y_N)$.

N ..... Input. Number of equations in the system.

XEND .. Input. Final point x_e to which the system should be solved.

ISW ... Input. Integer variable to specify conditions in integration.
ISW is a non-negative integer having two decimal digits, which can be expressed as

ISW = 10d_2 + d_1

Each d_i should be specified as follows:

d_1: Specifies whether or not this is the first call.
0: First call
1: Successive call
The first call means that this subroutine is called for the first time for the given differential equations.

d_2: Indicator for the output mode
0: Final value output
1: Step output

Output. When this subroutine returns to the user program after obtaining the solutions at x_e or the solutions at each step, d_1 is set as d_1 = 1.

When this subroutine is called repeatedly, d_1 should not be altered.

The user has to set d_1 = 0 again only when he starts to solve other equations.

EPSA ..... Input. Absolute error tolerance See Method.

EPSR ..... Input. Relative error tolerance. Output. If EPSR is too small, the value is changed to an appropriate value. (See Notes.)

VW ..... Work area. One-dimensional array of size 9 N + 40.
When calling this subroutine repeatedly, the contents should not be changed.

IVW .... Work area. One-dimensional array of size 5.
When calling this subroutine repeatedly, the contents should not be changed.

ICON .. Output. Condition code. See Table ODRK1-1.

Table ODRK1-1 Condition codes

Code	Meaning	Processing
0	(In step output) A single step has been taken.	Normal, Successive calling is possible.
10	Solution at XEND was obtained.	Normal, Successive calling is possible after changing XEND
10000	Integration was not completed because EPSR was too small in comparison with the arithmetic precision of the computer used (See Comments on use.)	Return to user program before continuing the integration. Successive calling is possible.
11000	Integration was not completed because more than 4000 derivative evaluations were needed to reach XEND.	Return to user program before continuing the integration. The function counter will be reset to 0 on successive call.
15000	Integration was not completed because requested accuracy could not be achieved using smallest alloable stepsize.	Return to user program before continuing the integration. The user must increase EPSA or EPSR before calling again.
16000	(When EPSA = 0) Integration was not completed because solution vanished, making a pure relative error test impossible.	Return to user program before continuing the integration. The user must increase EPSA before calling again.
30000	Some of the following occurred: 1. N ≤ 0 2 X = XEND 3 ISW was set to an improper value. 4 EPSA < 0 or EPSR < 0 5 After ICON = 15000 or 16000 is put out, successive calling is done without changing EPSA or EPSR.	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, MGSSL, URKV, UVER
  FORTRAN basic functions ... ABS, SIGN, AMAX1, AMIN1
- Notes
  This subroutine may be used to solve non-stiff and mildly stiff differential equations when derivative evaluations are inexpensive but cannot be used if high accuracy is desired.

  The name of the subroutine associated with parameter FUN must be declared as EXTERNAL in the calling program.

  Solutions may be acceptable only when ICON is 0 or 10.

When ICON = 10000 to 11000, the subroutine returns control to the user program before continuing the integration. The user can call this subroutine successively after identifying occurrences. When ICON = 15000 to 16000, the subroutine returns control to the user program before continuing the integration. In these cases, however, the user must increase EPSA or EPSR then he can call this subroutine successively (See the example).

Relative error tolerance EPSR is required to satisfy

$$\mathrm{EPSR} \geq \varepsilon_{r\min} = 10^{-12} + 2u$$

where u is the round-off unit. When EPSR does not satisfy the above condition, the subroutine increases EPSR as

$$\mathrm{EPSR} = \varepsilon_{r\min}$$

and returns control to the user program with ICON = 10000. To continue the integration, the user may call the subroutine successively.

In this subroutine, the smallest stepsize h_{min} is defined to satisfy

$$h_{\min} = 26u \cdot \max(|x|, |d|)$$

, where x is independent variable, and $d = (x_e - x_0)/100$. When the desired accuracy is not achieved using the smallest stepsize, the subroutine returns control to the user program with ICON = 15000. To continue the integration, the user may call the subroutine again after increasing EPSA or EPSR to an appropriate value.

- Example
  A system of first order ordinary differential equations

$$\begin{cases} y_1' = y_1^2 y_2, & y_1(0) = 1.0 \\ y_2' = -1/y_1, & y_2(0) = 1.0 \end{cases}$$

is integrated from $x_0 = 0.0$ to $x_e = 4.0$, under EPSA = 0.0, EPSR = 10^{-5}. Solutions are put out at each step.

```
C     **EXAMPLE**
      DIMENSION Y(2),VW(58),IVW(5)
      EXTERNAL FUN
      X=0.0
      Y(1)=1.0
      Y(2)=1.0
      N=2
      XEND=4.0
      EPSA=0.0
      EPSR=1.0E-5
      ISW=10
   10 CALL ODRK1(X,Y,FUN,N,XEND,ISW,EPSA,
     *EPSR,VW,IVW,ICON)
      IF(ICON.EQ.0.OR.ICON.EQ.10) GO TO 20
      IF(ICON.EQ.10000) GO TO 30
      IF(ICON.EQ.11000) GO TO 40
      IF(ICON.EQ.15000) GO TO 50
      IF(ICON.EQ.16000) GO TO 60
      IF(ICON.EQ.30000) GO TO 70
   20 WRITE(6,600) X,Y(1),Y(2)
      IF(ICON.NE.10) GO TO 10
      STOP
   30 WRITE(6,610)
      GO TO 10
   40 WRITE(6,620)
      GO TO 10
   50 WRITE(6,630)
      EPSR=10.0*EPSR
      GO TO 10
   60 WRITE(6,630)
      EPSA=1.0E-5
      GO TO 10
   70 WRITE(6,640)
      STOP
  600 FORMAT('0',10X,'X=',E15.7,10X,
     *'Y(1)=',E15.7,10X,'Y(2)=',E15.7)
  610 FORMAT('0',10X,'RELATIVE ERROR',
     *' TOLERANCE TOO SMALL')
  620 FORMAT('0',10X,'TOO MANY STEPS')
  630 FORMAT('0',10X,'TOLERANCE RESET')
  640 FORMAT('0',10X,'INVALID INPUT')
      END
      SUBROUTINE FUN(X,Y,YP)
      DIMENSION Y(2),YP(2)
      YP(1)=Y(1)**2*Y(2)
      YP(2)=-1.0/Y(1)
      RETURN
      END
```

Method

Defining solution vector $\boldsymbol{y}(x)$, function vector $\boldsymbol{f}(x,\boldsymbol{y})$, initial vector $\boldsymbol{y}_0$ as

$$\begin{aligned}
&\boldsymbol{y}(x)=(y_1,y_2,...,y_N)^{\mathrm{T}},\\
&\boldsymbol{f}(x,\boldsymbol{y})=(f_1(x,\boldsymbol{y}),f_2(x,\boldsymbol{y}),...,f_N(x,\boldsymbol{y}))^{\mathrm{T}},\\
&\boldsymbol{y}_0=(y_{10},y_{20},...,y_{N0})^{\mathrm{T}}
\end{aligned}\tag{4.1}$$

the initial value problem of a system of first order ordinary differential equations (1.1) can be rewritten as:

$$\boldsymbol{y}'(x)=\boldsymbol{f}(x,\boldsymbol{y}),\ \ \boldsymbol{y}(x_0)=\boldsymbol{y}_0 \tag{4.2}$$

- Runge-Kutta-Verner method

  This subroutine uses the Runge-Kutta-Verner method with estimates of the truncation error as shown below (The excellence of this method are described in Reference [73]).

  Solutions at point $x_{m+1}=x_m+h_{m+1}$ are obtained by using the formulas below

$$\left\{\begin{aligned}
&\boldsymbol{k}_1=h_{m+1}\boldsymbol{f}(x_m,\boldsymbol{y}_m)\\
&\boldsymbol{k}_2=h_{m+1}\boldsymbol{f}\left(x_m+\frac{1}{18}h_{m+1},\boldsymbol{y}_m+\frac{1}{18}\boldsymbol{k}_1\right)\\
&\boldsymbol{k}_3=h_{m+1}\boldsymbol{f}\left(x_m+\frac{1}{6}h_{m+1},\boldsymbol{y}_m-\frac{1}{12}\boldsymbol{k}_1+\frac{1}{4}\boldsymbol{k}_2\right)\\
&\boldsymbol{k}_4=h_{m+1}\boldsymbol{f}\left(x_m+\frac{2}{9}h_{m+1},\boldsymbol{y}_m-\frac{2}{81}\boldsymbol{k}_1+\frac{4}{27}\boldsymbol{k}_2+\frac{8}{81}\boldsymbol{k}_3\right)\\
&\boldsymbol{k}_5=h_{m+1}\boldsymbol{f}\left(x_m+\frac{2}{3}h_{m+1},\boldsymbol{y}_m+\frac{40}{33}\boldsymbol{k}_1-\frac{4}{11}\boldsymbol{k}_2+\frac{56}{11}\boldsymbol{k}_3+\frac{54}{11}\boldsymbol{k}_4\right)\\
&\boldsymbol{k}_6=h_{m+1}\boldsymbol{f}\left(x_m+h_{m+1},\boldsymbol{y}_m-\frac{369}{73}\boldsymbol{k}_1+\frac{72}{73}\boldsymbol{k}_2+\frac{5380}{219}\boldsymbol{k}_3\right.\\
&\qquad\left.-\frac{12285}{584}\boldsymbol{k}_4+\frac{2695}{1752}\boldsymbol{k}_5\right)\\
&\boldsymbol{k}_7=h_{m+1}\boldsymbol{f}\left(x_m+\frac{8}{9}h_{m+1},\boldsymbol{y}_m-\frac{8716}{891}\boldsymbol{k}_1+\frac{656}{297}\boldsymbol{k}_2+\frac{39520}{891}\boldsymbol{k}_3\right.\\
&\qquad\left.-\frac{416}{11}\boldsymbol{k}_4+\frac{52}{27}\boldsymbol{k}_5\right)\\
&\boldsymbol{k}_8=h_{m+1}\boldsymbol{f}\left(x_m+h_{m+1},\boldsymbol{y}_m+\frac{3015}{256}\boldsymbol{k}_1-\frac{9}{4}\boldsymbol{k}_2-\frac{4219}{78}\boldsymbol{k}_3\right.\\
&\qquad\left.+\frac{5985}{128}\boldsymbol{k}_4-\frac{539}{384}\boldsymbol{k}_5+\frac{693}{3328}\boldsymbol{k}_7\right)\\
&\boldsymbol{y}^*_{m+1}=\boldsymbol{y}_m+\frac{3}{80}\boldsymbol{k}_1+\frac{4}{25}\boldsymbol{k}_3+\frac{243}{1120}\boldsymbol{k}_4+\frac{77}{160}\boldsymbol{k}_5+\frac{73}{700}\boldsymbol{k}_6\\
&\boldsymbol{y}_{m+1}=\boldsymbol{y}_m+\frac{57}{640}\boldsymbol{k}_1-\frac{16}{65}\boldsymbol{k}_3+\frac{1377}{2240}\boldsymbol{k}_4+\frac{121}{320}\boldsymbol{k}_5+\frac{891}{8320}\boldsymbol{k}_7\\
&\qquad+\frac{2}{35}\boldsymbol{k}_8\\
&\boldsymbol{T}=\boldsymbol{y}_{m+1}-\boldsymbol{y}^*_{m+1}\\
&\ \ =\frac{33}{640}\boldsymbol{k}_1-\frac{132}{325}\boldsymbol{k}_3+\frac{891}{2240}\boldsymbol{k}_4-\frac{33}{320}\boldsymbol{k}_5-\frac{73}{700}\boldsymbol{k}_6\\
&\qquad+\frac{891}{8320}\boldsymbol{k}_7+\frac{2}{35}\boldsymbol{k}_8
\end{aligned}\right.$$

(4.3)

In the above formula, $\boldsymbol{y}^*_{m+1}$ and $\boldsymbol{y}_{m+1}$ are approximations with 5th and 6th order truncation error respectively, and $\boldsymbol{T}$ is an estimate of the local truncation error in $\boldsymbol{y}^*_{m+1}$. In this subroutine, the approximation $\boldsymbol{y}_{m+1}$ with higher accuracy is accepted as the solution when $\boldsymbol{y}^*_{m+1}$ satisfies the desired accuracy.

- Stepsize control

  Initial stepsize determination:

  Since $\boldsymbol{y}^*_{m+1}$ given by (4.3) is a 5th order approximation, the local truncation error at $x_1=x_0+h$ is estimated by h^5 times $h\boldsymbol{f}(x_0,\boldsymbol{y}_0)$ which is the term of degree one in h in the Taylor expansion of the solution $\boldsymbol{y}(x_0+h)$ at x_0. So the initial stepsize h_1 is determined by

$$h_1=\max\{h_1',26u\cdot\max(|x_0|,|d|)\} \tag{4.4}$$

where

$$h' = \min_{1\le i\le N}\left\{h_i \left| h_i^6 \left| f_i(x_0, \boldsymbol{y}_0)\right| = \text{EPSA} + \text{EPSR}\cdot\left|y_i(x_0)\right|\right.\right\} \quad (4.5)$$

$$d = (x_e - x_0)/100 \quad ,\text{EPSR} \ge 10^{-12} + 2u \quad (4.6)$$

(u is the round-off error unit)

Solution acceptance and rejection:
When an estimate $\boldsymbol{T}$ of local truncation error given by (4.3) satisfies the following condition, the solution $\boldsymbol{y}_{m+1}$ is accepted.

$$|T_i| \le \text{EPSA} + \text{EPSR}\cdot\frac{|y_i(x_m)| + |y_i(x_{m-1})|}{2} \quad ,i = 1,2,\ldots,N \quad (4.7)$$

This test becomes a relative error test when EPSA = 0. Such pure relative error test is recommendable if the user wants to be sure of accuracy. If EPSR = 0.0, it is corrected to EPSR = $10^{-12}+2u$ automatically in the subroutine. At this time, unless the absolute value of the solution is so large, EPSA becomes the upper bound of the absolute error. If the absolute value of the solution is large, the second term of the right hand side in (4.7) becomes the upper bound of the absolute error, so (4.7) is essentially relative error test.
Stepsize control:
The dominant term of the truncation error term in i-th component of $\boldsymbol{y}^*_{m+1}$ can be expressed as h^6C_i with h being a stepsize, C_i being a constant. If h is sufficiently small,

$$h^6C_i \approx T_i \quad (4.8)$$

holds. If the stepsize h is changed to sh, the estimated truncation error changes from T_i to $s^6\, T_i$.

- If the previous stepsize is unsuccessful i.e., if

$$|T_{i0}| > \text{EPSA} + \text{EPSR}\cdot\frac{|y_{i0}(x_m)| + |y_{i0}(x_{m+1})|}{2} \quad (4.9)$$

holds for some integer i_0, the stepsize for the next trial is determined as follows. In order that i-th component of the solution may be accepted, the magnifying rate s_i for stepsize must satisfy the condition

$$s_i^6|T_i| = \text{EPSA} + \text{EPSR}\cdot\frac{|y_i(x_m)| + |y_i(x_{m+1})|}{2} \quad (4.10)$$

Taking the minimum of s_i for i = 1, ..., N and then multiplying it by a safety constant 0.9, the rate can be obtained as

$$s = 0.9\sqrt[6]{\min_{1\le i\le N}\frac{\text{EPSA} + \text{EPSR}\cdot\dfrac{|y_i(x_m)| + |y_i(x_{m+1})|}{2}}{|T_i|}} \quad (4.11)$$

If s determined by (4.11) is equal to or less than 0.1, s = 0.1 is assumed. If s < 0.1, that is,

$$\max_{1\le i\le N}\frac{|T_i|}{\text{EPSA} + \text{EPSR}\cdot\dfrac{|y_i(x_m)| + |y_i(x_{m+1})|}{2}} > 9^6 \quad (4.12)$$

s is calculated from (4.11).

- If the previous stepsize is successful
i.e. if s determined by (4.11) is equal to or greater than 5, s = 5.0 is assumed. If s < 5.0, that is,

$$\max_{1\le i\le N}\frac{|T_i|}{\text{EPSA} + \text{EPSR}\cdot\dfrac{|y_i(x_m)| + |y_i(x_{m+1})|}{2}} > \left(\frac{9}{50}\right)^6$$

s is calculated from (4.11).

For a detailed description of the Runge-Kutta-Verner method, see Reference [72].

F12-15-0402 PNR, DPNR

Permutation of data (reverse binary transformation
CALL PNR (A, B, NT, N, NS, ISN, ICON)

Function

When complex data $\{\alpha_k\}$ of dimension n is given, this subroutine determines $\{\tilde{\alpha}_k\}$ using reverse binary transformation. Also if $\{\tilde{\alpha}_k\}$ is given, determines $\{\alpha_k\}$. n must be a number expressed as $n = 2^l$ (l: 0 or a positive integer).

The reverse binary transformation means that the element of $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$ located

$$k = k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1} \tag{1.1}$$

is moved to location

$$\tilde{k} = k_{l-1} + k_{l-2} \cdot 2 + ... + k_0 \cdot 2^{l-1} \tag{1.2}$$

This routine performs the data permutation required in Fast Fourier Transform method.

Parameters

A	Input. Real parts of $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$. Output. Real parts of $\{\tilde{\alpha}_k\}$ or $\{\alpha_k\}$. One-dimensional array of size NT.
B	Input. Imaginary parts of $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$. Output. Imaginary parts of $\{\tilde{\alpha}_k\}$ or $\{\alpha_k\}$. One-dimensional array of size NT.
NT	Input. Total number of data (≥N) including the $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$ to be permuted. Normally, NT = N is specified. (See "Notes".)
N	Input. Dimension n.
NS	Input. The interval of the consecutive data $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$ to be permuted of dimension n in the NT data (≥ 1 and ≤ NT). Normally, NS = 1 is specified. (See "Notes".)
ISN	Input. Interval (≠0) of the NT data. Normally, ISN = 1 is specified. (See "Notes".)
ICON	Output. Condition code. See Table PNR-1.

Table PNR-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	ISN = 0, NS < 1, NT < N, NT < NS, or N≠2^l (l : 0 or a positive integer)	Bypassed

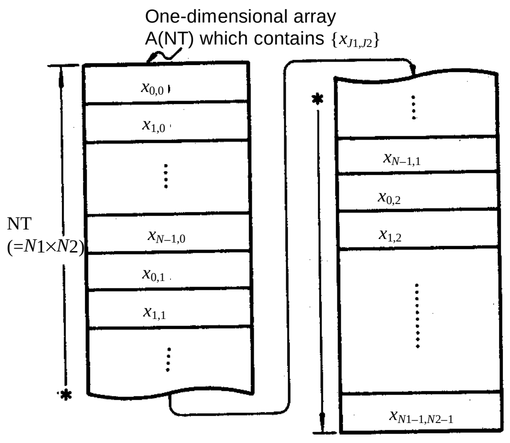


Fig. PNR-1 Storage of $\{x_{J1,J2}\}$

Comments on use

- Subprograms used
  SSL II ..... MGSSL
  FORTRAN basic functions ..... ALOG and IABS

- Notes
  Use I:
  This subroutine is usually used with subroutine CFTN or subroutine CFTR. Discrete complex Fourier transform and inverse Fourier transform are defined generally

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega^{-jk}, k = 0,1,...,n-1 \tag{3.1}$$

and

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega^{jk}, j = 0,1,...,n-1 \tag{3.2}$$

In CFTN, the transform in (3.3) or (3.4), corresponding to (3.1) or (3.2), is performed.

$$n\tilde{\alpha}_k = \sum_{j=0}^{n-1} x_j \omega^{-jk}, k = 0,1,...,n-1 \tag{3.3}$$

$$\tilde{x}_j = \sum_{k=0}^{n-1} \alpha_k \omega^{jk}, j = 0,1,...,n-1 \tag{3.4}$$

In CFTR, the transform in (3.5) or (3.6), corresponding to (3.1) or (3.2) is performed.

$$n\alpha_k = \sum_{j=0}^{n-1} \tilde{x}_j \omega^{-jk}, k = 0,1,...,n-1 \tag{3.5}$$

$$x_j = \sum_{k=0}^{n-1} \tilde{\alpha}_k \omega^{jk}, j = 0,1,...,n-1 \tag{3.6}$$

Thus, this subroutine is used with CFTN, after the transformation of (3.3) or (3.4), to permute $\{\tilde{\alpha}_k\}$ or $\{\tilde{x}_j\}$ into $\{\alpha_k\}$ or $\{x_j\}$.
When used with CFTR, just before the transformation of (3.5) or (3.6) is performed, this routine permutes $\{x_j\}$ or $\{\alpha_k\}$ into $\{\tilde{x}_j\}$ or $\{\tilde{\alpha}_k\}$. Since the parameters of this subroutine are essentially the same as with CFTN and CFTR, their specifications are the same.
Refer to Examples (a) and (b).
Use II:
This subroutine can be also used when performing a multi-variate Fourier transform or inverse Fourier transform with CFTN or CFTR.
A multi-variate discrete complex Fourier transform is defined generally for two variate

$$\alpha_{K1,K2} = \frac{1}{N1\cdot N2}\sum_{J1=0}^{N1-1}\sum_{J2=0}^{N2-1} x_{J1,J2}\,\omega_1^{-J1,K1}\omega_2^{-J2,K2} \quad (3.7)$$
$$,K1 = 0,1,...,N1-1, K2 = 0,1,...,N2-1$$

With CFTN, the transform in (3.8), corresponding to (3.7) can be done.

$$N1\cdot N2\tilde{\alpha}_{K1,K2} = \sum_{J1=0}^{N1-1}\sum_{J2=0}^{N2-1} x_{J1,J2}\,\omega_1^{-J1\cdot K1}\omega_2^{-J2\cdot K2} \quad (3.8)$$
$$,K1 = 0,1,...,N1-1, K2 = 0,1,...,N2-1$$

With CFTR, the transform in (3.9), corresponding to (3.7) can be done.

$$N1\cdot N2\alpha_{K1,K2} = \sum_{J1=0}^{N1-1}\sum_{J2=0}^{N2-1} \tilde{x}_{J1,J2}\,\omega_1^{-J1,K1}\omega_2^{-J2,K2} \quad (3.9)$$
$$,K1 = 0,1,...,N1-1, K2 = 0,1,...,N2-1$$

For an inverse transform, a transform similar to a one-variable transform can be performed. When this subroutine is used with CFTN, after the transformation of (3.8) is performed, $\{N1.N2\ \tilde{\alpha}_{k1.k2}\}$ is permuted. If used with CFTR, just before the transformation of (3.9),$\{x_{j1,j2}\}$ is permuted.
Refer to Example (c).
Specifying ISN:
If NT real parts and imaginary parts of $\{\alpha_k\}$ or $\{\tilde{\alpha}_k\}$ are each stored in areas of size NT·I in intervals of I, the following specification is made.

ISN = I

The permuted results are also stored in intervals of I.

- Examples
  (a) Permutation after a one-variable transform Given complex time series data $\{x_j\}$ of dimension n, after a Fourier transform is performed using subroutine CFTN, the results $\{n\tilde{\alpha}_k\}$ are permuted with this subroutine and scaled to obtain$\{\alpha_k\}$.
  In case of $n \le 1024$ $(= 2^{10})$.

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024)
       READ(5,500) N,(A(I),B(I),I=1,N)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
       CALL CFTN(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       CALL PNR(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       DO 10 I=1,N
       A(I)=A(I)/FLOAT(N)
       B(I)=B(I)/FLOAT(N)
    10 CONTINUE
       WRITE(6,620) (I,A(I),B(I),I=1,N)
       STOP
   500 FORMAT(I5/(2E20.7))
   600 FORMAT('0',10X,'INPUT DATA N=',I5/
      *       /(15X,I5,2E20.7))
   610 FORMAT('0',10X,'RESULT ICON=',I5)
   620 FORMAT(15X,I5,2E20.7)
       END
```

  (b) Permutation before a one-variable transform
  Given complex time series data $\{x_j\}$ of dimension n, before performing a Fourier transform, the data is permuted using this subroutine, and a transform is performed on the results $\{\tilde{x}_j\}$ using subroutine CFTR, and then scaling is performed to obtain $\{\alpha_k\}$.
  In case of $n \le 1024$ $(= 2^{10})$.

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024)
       READ(5,500) N,(A(I),B(I),I=1,N)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
       CALL PNR(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       CALL CFTR(A,B,N,N,1,1,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       DO 10 I=1,N
       A(I)=A(I)/FLOAT(N)
       B(I)=B(I)/FLOAT(N)
    10 CONTINUE
       WRITE(6,620) (I,A(I),B(I),I=1,N)
       STOP
   500 FORMAT(I5/(2E20.7))
   600 FORMAT('0',10X,'INPUT DATA N=',I5/
      *       /(15X,I5,2E20.7))
   610 FORMAT('0',10X,'RESULT ICON=',I5)
   620 FORMAT(15X,I5,2E20.7)
       END
```

(c) Permutation after a two-variate transform
Given complex time series data $\{x_{j1,j2}\}$ of dimension N1 and N2, a Fourier transform is performed with the subroutine CFTN, then the results $\{N1 \cdot N2\ \tilde{\alpha}_{k1.k2}\}$ are permuted by this subroutine and scaled to obtain $\{\alpha_{k1.k2}\}$.
In case of $N1 \cdot N2 \le 1024$ ($= 2^{10}$). The data $\{x_{j1,j2}\}$ can be stored as shown in Fig. PNR-1.

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024),N(2)
       READ(5,500) (N(I),I=1,2)
       NT=N(1)*N(2)
       READ(5,510) (A(I),B(I),I=1,NT)
       WRITE(6,600) N,(I,A(I),B(I),I=1,NT)
       NS=1
       DO 10 I=1,2
       CALL CFTN(A,B,NT,N(I),NS,1,ICON)
       IF(ICON.NE.0) STOP
       CALL PNR(A,B,NT,N(I),NS,1,ICON)
       NS=NS*N(I)
    10 CONTINUE
       DO 20 I=1,NT
       A(I)=A(I)/FLOAT(NT)
       B(I)=B(I)/FLOAT(NT)
    20 CONTINUE
       WRITE(6,610) (I,A(I),B(I),I=1,NT)
       STOP
   500 FORMAT(2I5)
   510 FORMAT(2E20.7)
   600 FORMAT('0',10X,'INPUT DATA N=',2I5/
      *       /(15X,I5,2E20.7))
   610 FORMAT('0',10X,'OUTPUT DATA'/
      *       /(15X,I5,2E20.7))
       END
```

Method
As is necessary with the radix 2 Fast Fourier Transform, this subroutine performs permutation such that the result is in reverse binary order against input order.
First, let a discrete complex Fourier transform be defined as

$$\alpha_k = \sum_{j=0}^{n-1} x_j \exp\left(-2\pi i \frac{jk}{n}\right), k = 0,1,...,n-1 \qquad (4.1)$$

In (4.1) the scaling factor $1/n$ is omitted.
The use of Fast Fourier Transform method is considered for $n = 2^l$. (Refer to the section on subroutine CFT for the principles of the Fast Fourier Transform method).

When transforming in an area with only $\{x_j\}$ specified, data must be permuted immediately before or after transformation. In other words, when k and j of (4.1) are expressed as

$$\begin{aligned} k &= k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1}, k_0,...,k_{l-1} = 0,1 \\ j &= j_0 + j_1 \cdot 2 + ... + j_{l-1} \cdot 2^{l-1}, j_0,..., j_{l-1} = 0,1 \end{aligned} \qquad (4.2)$$

the Fast Fourier Transform results become $\tilde{\alpha}$ $(k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1})$ against $\tilde{\alpha}$ $(k_{l-1} + k_{l-2} \cdot 2 + ... + k_0 \cdot 2^{l-1})$
In this case α $(k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1}) \equiv \alpha_{k_0+k_1 2+...k_{l-1}2^{l-1}}$ can be understood to be the order of data $\{\alpha_k\}$ expressed in reverse binary order.
Therefore, the data order of $\{\tilde{\alpha}_k\}$ against $\{\alpha_k\}$ is in reverse binary order, and final data permutation becomes necessary. On the other hand, if $\{\tilde{\alpha}_k\}$ and j are expressed as

$$\begin{aligned} k &= k_{l-1} + k_{l-2} \cdot 2 + ... + k_0 \cdot 2^{l-1}, k_0,...,k_{l-1} = 0,1 \\ j &= j_{l-1} + j_{l-2} \cdot 2 + ... + j_0 \cdot 2^{l-1}, j_0,..., j_{l-1} = 0,1 \end{aligned} \qquad (4.3)$$

the results of the Fast Fourier Transform are in normal order without permutation as

$$\alpha\left(k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1}\right)$$

If the order of the data $\{x_j\}$ is permuted in a reverse binary order as shown in j of (4.3) before transformation, final data permutation is not needed. As previously mentioned, with the Fast Fourier Transform for certain data$\{\alpha\}$, transposition of data in location shown in (4.4) and (4.5) is required.

$$k = k_0 + k_1 \cdot 2 + ... + k_{l-1} \cdot 2^{l-1} \qquad (4.4)$$
$$\tilde{k} = k_{l-1} + k_{l-2} \cdot 2 + ... + k_0 \cdot 2^{l-1} \qquad (4.5)$$

Given $\{\alpha_k\}$ or$\{\tilde{\alpha}_k\}$, this subroutine permutes the data by reverse binary order of transformation to obtain $\{\tilde{\alpha}_k\}$ or $\{\alpha_k\}$.
Basic conditions for permutation are:

- For $k = \tilde{k}$, transposition is unnecessary
- For $k \ne \tilde{k}$, $\alpha(k)$ and $\alpha(\tilde{k})$ (or, $\alpha(\tilde{k})$ and $\tilde{\alpha}(\tilde{k})$)are permuted.

For further information, refer to References [55], [56], [57].

J12-20-0101 RANB2

Generation of binomial random integers
CALL RANB2 (M, P, IX, IA, N, VW, IVW, ICON)

Function

This subroutine generates a sequence of *n* pseudo random integers from the probability density function (1.1) of binomial distribution with moduli *m* and *p*.

$$P_k = \binom{m}{k} p^k (1-p)^{m-k} \qquad (1.1)$$
$$,0 < p < 1, k = 0,1,...,m, m = 1,2,...$$

where $n \geq 1$.

Parameters

M	Input. Modulus *m*.
P	Input. Modulus *p*.
IX	Input. Starting value of non-negative integer (must be INTEGER*4) Output. Starting value for next call of RANB2. See Notes.
IA	Output. *n* binomial pseudo random integers.
N	Input. Number of *n* of binomial pseudo random integers to be generated.
VW	Work area. One-dimensional array of size *m* + 1.
IVW	Work area. One-dimensional array of size *m* + 1.
ICON	Output. Condition code. See Table RANB2-1.

Table RANB2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M < 1, P ≤ 0, P ≥ 1, IX < 0 or N < 1	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic functions .... ALOG, EXP, DMOD, FLOAT

- Note
  – Starting value IX
    This subroutine transforms uniform pseudo random numbers into binomial random integers. Parameter IX is given as the starting value to generate uniform pseudo random numbers. It is handled in the same way as RANU2.
    See comments on use for RANU2.
  – The contents of VW and IVW should not be changed as long as the same value has been specified in parameter M and P.

- Example
  Generating 10000 pseudo random numbers from a binomial distribution with *m* = 20 and *p* = 0.75, the frequency distribution histogram is plotted.

```
C     **EXAMPLE**
      DIMENSION IA(10000),VW(21),IVW(21),
     *          HSUM(21)
      INTEGER*4 IX
      DATA X1,X2/-0.5,20.5/
      DATA NINT,XINT/21,1.0/
      DATA M,P,IX,N/20,0.75,0,10000/
      DO 10 I=1,21
   10 HSUM(I)=0.0
      CALL RANB2(M,P,IX,IA,N,VW,IVW,ICON)
C     SUM NOS. IN HISTGRAM FORM
      ISW=1
      DO 20 I=1,N
      X=IA(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,HINT,HSUM,
     *         ISW)
      STOP
  600 FORMAT('1',10X,'BINOMIAL RANDOM',
     *' NUMBER DISTRIBUTION'//)
      END
```

For detailed information on subroutine HIST, see the Example in subroutine RANU2.

Method

Binomial random integers are generated as indicated below. When integer *l* is determined so that a pseudo random number *u* generated from uniform distribution in the interval (0, 1) satisfies equation (4. 1) *l* is a binomial random integer.

$$F_{l-1} \leq u < F_l \qquad (4.1)$$

where F_l is a binomial cumulative distribution function shown in (4.2).

$$F_{-1} = 0$$
$$F_l = \sum_{k=0}^{l} \binom{m}{k} p^k (1-p)^{m-k}, l = 0,1,...,m \qquad (4.2)$$

Figure RANB2-1 represents(4.1) where *m* = 6 and *p* = 0.5. For example when *u* = 0.74321, *l* = 4.

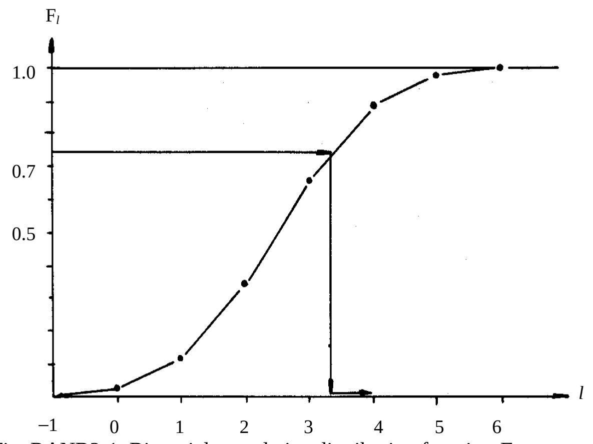


Fig. RANB2-1 Binomial cumulative distribution function F_l

Since this cumulative distribution is unique when *m* and *p* are determined, it is time consuming to compute (4.2) each time for a binomial random integer. Therefore if the cumulative distribution table is generated once, it can be used for reference for the subsequent computation. Parameter VW is used for these reference. If *u* is a value close to 1, checking l = 1, 2, ... in (4.1) is also time consuming. Parameter IVW is used as an index table to start l at an appropriate value depending upon value *u*.

For further information, see Reference [93].

J11-30-0101 RANE2

Generation of exponential pseudo random numbers
CALL RANE2 (AM, IX, A, N, ICON)

Function

This subroutine generates a sequence of *n* pseudo random numbers form the probability density function (1.1) of exponential distribution with mean value *m*.

$$g(x) = \frac{1}{m} e^{-\frac{x}{m}} \tag{1.1}$$

Where, $x \geq 0, m > 0,$ and $n \geq 1$

Parameters

AM.....	Input. Mean value of exponential distribution, *m*.
IX.....	Input. Starting value of non-negative integer (must be INTEGER *4). Output. Starting value for next call of RANE2. See Comments on use below.
A.....	Output. *n* random numbers. One-dimensional array of size, *n*.
N.....	Input. Number of pseudo-random numbers to be generated.
ICON..	Output. Condition codes. See Table RANE2-1.

Table RANE2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	AM ≤ 0, IX < 0 or N < 1	Bypassed

Comments on use

- Subprograms used
  SSL II ... RANU2 and MGSSL
  FORTRAN basic functions ... ALOG and DMOD

- Note
  Starting value IX
  This subroutine transforms uniform pseudo random numbers generated by RANU2 into exponential pseudo random numbers. Parameter IX is given as the starting value to generate uniform pseudo random numbers.
  See comments on use for RANU2.

- Example
  10,000 exponential pseudo random numbers from exponential distribution with mean value 1.0 are generated and the frequency distribution histogram is plotted.

```
C     **EXAMPLE**
      DIMENSION A(10000),HSUM(12)
      INTEGER*4 IX
      DATA X1,X2,NINT,XINT/0.0,6.0,12,0.5/
      DATA AM,IX,N/1.0,0,10000/
      DO 10 I=1,12
   10 HSUM(I)=0.0
      CALL RANE2(AM,IX,A,N,ICON)
C     SUM NOS. IN HISTGRAM FROM
      ISW=1
      DO 20 I=1,N
      X=A(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
      STOP
  600 FORMAT('1',10X,'EXPONENTIAL',
     * ' RANDOM NUMBER DISTRIBUTION'//)
      END
```

See the example in RANU2 for subroutine HIST.

Method

Exponential pseudo random numbers {*y*} are generated by

$$y = -m \log u \tag{4.1}$$

where, {*u*} is a sequence of uniform (0, 1) pseudo random numbers generated and *m* is the mean value.
The function, (4.1), can be derived as follow:
The cumulative exponential distribution function *F*(*y*) will be obtained from (1.1)

$$F(y) = \int_0^y g(x)dx = \int_0^y \frac{1}{m} e^{-\frac{1}{m}x} dx = 1 - e^{-\frac{1}{m}y} \tag{4.2}$$

Let u_1 be one of the uniform pseudo random numbers, (4.3) is obtained from (4.2) based on the relation $u = F(y)$.

$$y_1 = -m \log(1 - u_1) \tag{4.3}$$

Thus, exponential pseudo random numbers $y_1, y_2, y_3,$ ..... can be transformed one to one from uniform pseudo random numbers, $u_1, u_2, u_3,$ ..... .

J11-20-0301 RANN1

Fast normal pseudo random numbers
CALL RANN1 (AM, SD, IX, A, N, ICON)

Function

This subroutine generates n pseudo random numbers from a given probability density function (1.1) of normal distribution with mean value m and standard deviation σ:

$$g(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-m)^2/2\sigma^2} \qquad (1.1)$$

where $n \geq 1$

Parameters

AM.....	Input. Mean value m of the normal distribution.
SD.....	Input. Standard deviation σ of the normal distribution.
IX.....	Input. Initial value of nonnegative integer (must be INTEGER*4). Output. Initial value for the next call of this subroutine. (See "Comments on Use".)
A.....	Output. n pseudo random numbers. One-dimensional array of size n.
N.....	Input. Number of pseudo random numbers n to be generated.
ICON..	Output. Condition code. (See Table RANN-1).

Table RANN-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	IX < 0 or N < 1.	Bypassed.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic functions ... SQRT, ALOG, ABS, SIGN, DFLOAT, DINT

- Notes
  Initial value IX
  This subroutine generates uniform pseudo random numbers and then transforms them to normal pseudo random numbers.
  Parameter IX is specified as the initial value to generate uniform pseudo random numbers and is processed in the same way as for RANU2. (See "Comments on Use" for RANU2.)

  This subroutine generates normal pseudo random numbers faster than subroutine RANU2.

- Example
  Given a normal distribution having mean value 0 and standard deviation 1.0, this subroutine generates 10,000 pseudo random numbers and a frequency distribution histogram is plotted.

```
C     **EXAMPLE**
      DIMENSION A(10000),HSUM(12)
      INTEGER*4 IX
      DATA X1,X2/-3.0,3.0/
      DATA NINT,XINT/12,0.5/
      DATA AM,SD,IX,N/0.0,1.0,0,10000/
      DO 10 I=1,12
   10 HSUM(I)=0.0
      CALL RANN1(AM,SD,IX,A,N,ICON)
C     SUM NOS. IN HISTGRAM FORM
      ISW=1
      DO 20 I=1,N
      X=A(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
      STOP
  600 FORMAT('1',10X,'NORMAL RANDOM',
     *' NUMBER DISTRIBUTION'//)
      END
```

See "Example" for subroutine RANU2 for subroutine HIST in this example.

Method

The inverse function method is used to generate normal pseudo random numbers. For uniform pseudo random numbers u_i (i = 1, 2, ..., n) in interval (0, 1) transformation.

$$z_i = G^{-1}(u_i)\sigma + m \qquad (4.1)$$

is applied to generate normal pseudo random numbers z_i (i = 1, 2, ..., n), where $G^{-1}(u)$ is an inverse function of cumulative normal distribution function

$$G(z)=\int_{-\infty}^{x} g(x)dx$$

This subroutine uses Ninomiya's best approximation to realize high-speed $G^{-1}(u)$ calculation.

1) For $|u-0.5| \leq 0.46875$

$$G^{-1}(u)=cx\left(e/\left(x^2+d\right)+x^2+b\right)$$

where $x= u - 0.5$ and the theoretical absolute error is $7.9 \cdot 10^{-4}$.

2) For $|u - 0.5| > 0.46875$

$$G^{-1}(u) = \mathrm{sign}(u\text{-}0.5) \cdot p(v+q+r/v)$$

where $v = \sqrt{-\log\left(0.5 - |u - 0.5|\right)}$
and the theoretical absolute error is $9.3 \cdot 10^{-4}$.

Since the formula of 1) is used in most cases (probability is 15/16), high-speed calculation is realized.

J11-20-0101 RANN2

Generation of normal pseudo random numbers
CALL RANN2 (AM, SD, IX, A, N, ICON)

Function

This subroutine generates a sequence of n pseudo random numbers from the probability density function (1.1) of normal distribution with mean value m and standard deviation σ.

$$g(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-m)^2/2\sigma^2} \tag{1.1}$$

where $n \geq 1$

Parameters

AM.....	Input. Mean value of the normal distribution, m.
SD.....	Input. Standard deviation of the normal distribution, σ.
IX.....	Input. Starting value of non-negative integer (must be INTEGER*4) Output. Starting value for the next call of RANN2. See comments on use below.
A.....	Output. n pseudo random numbers. One-dimensional array of size n.
N.....	Input. Number of pseudo-random numbers to be generated.
ICON..	Output. Condition codes. See Table RANN2-1.

Table RANN2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	IX < 0 or N < 1	Bypassed

Comments on use

- Subprograms used
  SSL II ... RANN2 and MGSSL
  FORTRAN basic functions ... SQRT, ALOG, SIN, COS and DMOD

- Notes
  Starting value IX
  This subroutine transforms uniform pseudo random numbers generated by RANU2 into normal pseudo random numbers. Parameter, IX is given as the starting value to generate uniform pseudo random numbers. See Comments on use for RANU2.

- Example
  10,000 normal pseudo random numbers are generated from normal distribution with mean value 0 and standard deviation 1.0 and the frequency distribution histogram is plotted.

```
C     **EXAMPLE**
      DIMENSION A(10000),HSUM(12)
      INTEGER*4 IX
      DATA X1,X2/-3.0,3.0/
      DATA NINT,XINT/12,0.5/
      DATA AM,SD,IX,N/0.0,1.0,0,10000/
      DO 10 I=1,12
   10 HSUM(I)=0.0
      CALL RANN2(AM,SD,IX,A,N,ICON)
C     SUM NOS. IN HISTGRAM FORM
      ISW=1
      DO 20 I=1,N
      X=A(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
      STOP
  600 FORMAT('1',10X,'NORMAL RANDOM',
     *' NUMBER DISTRIBUTION'//)
      END
```

Refer to the example in RANU2 for subroutine HIST.

Method

Normal pseudo random numbers are generated using the Box and Müller method, according to (4.1) and (4.2):

$$z_i = \sigma(-2\log u_i)^{\frac{1}{2}} \cos 2\pi u_{i+1} + m \tag{4.1}$$

$$z_{i+1} = \sigma(-2\log u_i)^{\frac{1}{2}} \sin 2\pi u_{i+1} + m \tag{4.2}$$

Where m and σ are the mean value and standard deviation of normal distribution respectively.

Thus, normal pseudo random number (z_1, z_2, z_3, ...) are calculated from as many uniform pseudo random number (u_1, u_2, u_3, ...). Here, when the odd number of normal pseudo random number is to be generated, a pair of uniform random numbers is required for the last one.

J12-10-0101 RANP2

Generation of Poisson pseudo random integers
CALL RANP2 (AM, IX, IA, N, VW, IVW, ICON)

Function

This subroutine generates a sequence of *n* pseudo random integers from the probability density function (1.1) of Poisson distribution with mean value *m*.

$$p_k = e^{-m}\frac{m^k}{k!} \qquad (1.1)$$

where, $m < 0$, k is non-negative integer, and $n \geq 1$.
(Refer to Fig. RANP2-1)

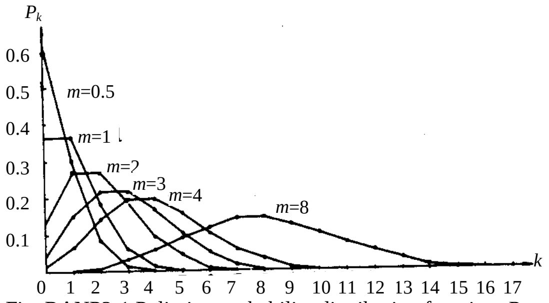


Fig. RANP2-1 Polission probability distribution function, P_k

Parameters

AM..... Input. Mean value of Poisson distribution, *m*. See Comments on use below.
IX..... Input. Starting value of non-negative integer (must be INTEGER*4).
Output. Starting value of next call of RANP2. See Comments on use below.
IA..... Output. Poisson pseudo random integers. One-dimensional array of size *n*.
N..... Input. Number *n* of Position pseudo random integers to be generated.
VW.... Work area. One-dimensional array of size [2*m* + 10].
IVW... Work area. One-dimensional array of size [2*m* + 10].
ICON.. Output. Condition codes. See Table RANP2-1.

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic functions ... EXP and DMOD

Table RANP2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	AM ≤ 0, AM > log(fl_{max}) IX < 0 or N < 1	Aborted

- Notes
  AM ≤log(fl_{max})
  Because, for AM > log(fl_{max}), the value e^{-m} underflows in computation of F_n in (4.2).
  For a large AM (≥ 20), the Poisson pseudo random integers may be approximated by the normal pseudo random numbers with the mean value *m* and standard deviation *m*.
  See Methods.
  Starting value IX
  This subroutine transforms uniform pseudo random numbers generated by RANU2 into Poisson Pseudo random integers. Parameter, IX is given as the starting value to generate the uniform pseudo random numbers. See Comment on use for RANU2.
  Both VW and IVW must not be altered while parameter AM remains the same.

- Example
  10,000 Poisson pseudo random integers from Poisson distribution with mean value 1.0 are generated and the frequency distribution histogram is plotted.

```
C     **EXAMPLE**
      DIMENSION IA(10000),VW(12),
     *          IVW(12),HSUM(6)
      INTEGER*4 IX
      DATA X1,X2/-0.5,5.5/
      DATA NINT,XINT/6,1.0/
      DATA AM,IX,N/1.0,0,10000/
      DO 10 I=1,6
   10 HSUM(I)=0.0
      CALL RANP2(AM,IX,IA,N,VW,IVW,ICON)
C     SUM NOS. IN HISTGRAM FORM
      ISW=1
      DO 20 I=1,N
      X=IA(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
      STOP
  600 FORMAT('1',10X,'POISSON RANDOM',
     *' NUMBER DISTRIBUTION'//)
      END
```

Refer to the example in RANU2 for subroutine HIST.

Methods

Poisson pseudo random integer is determined as l when pseudo random number l generated from uniform distribution on the range (0, 1) satisfies the relationship.

$$F_{l-1} \le u < F_l \tag{4.1}$$

where F_1 is the Poisson cumulative distribution function defined in (4.2)

$$\begin{aligned} &F_{-1} = 0 \\ &F_l = \sum_{k=0}^{l} p_k = \sum_{k=0}^{l} e^{-m} \frac{m^k}{k!}, l = 0,1,2,... \end{aligned} \tag{4.2}$$

Fig. RANP2-2 demonstrates the generation of Poisson random integers according to the transformation in (4.1) where, u = 0.84321 and l = 2.

Since m determines the form of cumulative distribution function F_l if a table of cumulative distribution function for given value of m is once determined, repetitive computations of (4.2) will be eliminated.

Parameter VW is used to support this table.

Further, u gets closer to 1, it would be of no use to search this table step by step starting from 0 in ascending order. Here again, parameter IVW is used to give a proper index to search this table depending upon the value of u.

[The effect of truncation in computation of F_l]

Since Poisson distribution (4.2) continues infinitively, computation of F_l must be stopped at an appropriate value of n.

This value depends upon the precision of computation in (4.2). Once F_{l-1} for l = 1, 2, 3, ..., further computation in (4.2) is meaningless. When $m \ge 20$, because of the above affection, Poisson pseudo-random integers may be better approximated by the normal pseudo-random integers with (mean value m and standard deviation m).

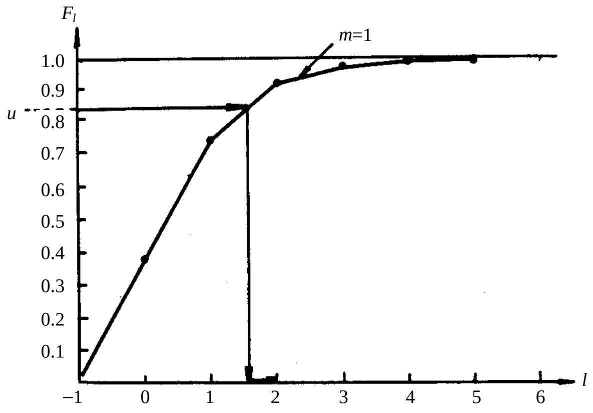


Fig. RANP2-2 Poissons cumulative distribution functionF_l

J11-10-0101 RANU2

Generation of uniform (0, 1) pseudo random numbers
CALL RANU2 (IX, A, N, ICON)

Function

This subroutine generates, by the congruence method, a sequence of *n* pseudo random numbers based on a stating value from a uniform distribution on the range (0, 1). $n \geq 1$.

Parameters

IX....	Input. Starting value. A non-negative integer. (must be INTEGER*4) Output. Starting value for the next call of RANU2. See comments on use.
A.....	Output. *n* pseudo random numbers. One-dimensional array of size *n*.
N.....	Input. Number of pseudo random numbers to be generated.
ICON..	Output. Condition codes. See Table RANU2-1.

Table RANU2-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	$IX < 0$ or $N \leq 0$	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL
  FORTRAN basic function ... DMOD

- Notes
  Starting value IX
  When a sequence of pseudo random integers, $\{IX_j\}$ is to be obtained by the congruence method (3.1), the user must give an starting value IX_0 (usually zero is given).

$$IX_{i+1} \equiv a \times IX_i + c \pmod{m}, i = 0,1,...,n-1 \qquad (3.1)$$

where IX_i, a, *c* and *m* are all non-negative integers. This sequence $\{IX_i\}$ is normalized into (0, 1) and a sequence of pseudo random numbers is generated in parameter A. After generation of *n* pseudo random numbers IX_n is given to parameter IX. Thus, when next *n* pseudo random numbers are to be generated successively, they will be generated with the current starting value IX_n in parameter IX, unless it is changed.
When this subroutine is repeatedly called *n* times with parameter N set to 1, *n* uniform pseudo random integers $\{IX_n\}$ are obtained.

Test for uniform random numbers
Uniform random numbers have two main properties: probability unity and randomness. It is important to understand these properties when using this subroutine.
Table RANU2-2 shows the results of testing of statistical hypothesis on pseudo random numbers generated by this subroutine with IX = 0. Generally speaking, we cannot generate pseudo random numbers suitable for all cases and expect them to pass all the tests. However, as Table RANU2-2 shows, this subroutine has been implemented with the values of "*a*" and "*c*" (refer to "method"), properly selected such that the resultant pseudo random numbers have passable properties to stand the tests of frequency and randomness.

- Example
  10,000 uniform pseudo random integers are generated and a histogram of their frequency distribution is plotted.

```
C     **EXAMPLE**
      DIMENSION A(10000),HSUM(10)
      INTEGER*4 IX
      DATA X1,X2,NINT,XINT/0.0,1.0,10,0.1/
      DATA IX,N/0,10000/
      DO 10 I=1,10
   10 HSUM(I)=0.0
      CALL RANU2(IX,A,N,ICON)
C     SUM NOS. IN HISTGRAM FORM
      ISW=1
      DO 20 I=1,N
      X=A(I)
   20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
C     PLOT DISTRIBUTION
      WRITE(6,600)
      ISW=2
      CALL HIST(X,X1,X2,NINT,XINT,HSUM,
     *          ISW)
      STOP
  600 FORMAT('1',10X,'UNIFORM RANDOM',
     *' NUMBER DISTRIBUTION'//)
      END
```

Subroutine HIST in this example computes the frequency distribution and plots the histogram. The contents are shown below.

```
      SUBROUTINE HIST(X,X1,X2,NINT,XINT,
     *                HSUM,ISW)
      DIMENSION HSUM(1)
      CHARACTER*4 IMAGE(31),II,IBLK,IAST
      DATA II,IBLK,IAST/'I   ','    ',
     *     '*   '/
      IF(ISW.NE.1) GO TO 30
C     TO SET UP HISTGRAM FOR RANDOM NOS.
      J=0
      IF(X.GT.(X1+X2)/2.0) J=NINT/2
      BK=X1+J*XINT
   10 J=J+1
      BK=BK+XINT
      IF(X.LT.X1) RETURN
      IF(X.LT.BK) GO TO 20
      IF(X.LT.X2) GO TO 10
      RETURN
   20 HSUM(J)=HSUM(J)+1.0
      RETURN
C     TO GET MAX X FOR PLOTTING
   30 Y=HSUM(1)
      DO 40 I=2,NINT
   40 Y=AMAX1(HSUM(I),Y)
      IMAGE(1)=II
      DO 50 I=2,31
   50 IMAGE(I)=IBLK
      BK=X1-XINT
      WRITE(6,600)
      DO 60 I=1,NINT
      BK=BK+XINT
      WRITE(6,610) BK
      J=30.0*HSUM(I)/Y+1.5
      IMAGE(J)=IAST
      WRITE(6,620) HSUM(I),IMAGE
      IMAGE(J)=IBLK
      IMAGE(1)=II
   60 CONTINUE
      BK=BK+XINT
      WRITE(6,610) BK
      RETURN
  600 FORMAT(2X,'BREAK PT',3X,'PT SUM',
     *11X,'GRAPH OF DISTRIBUTION')
  610 FORMAT(2X,F5.1,1X,48('-'))
  620 FORMAT(12X,F7.1,5X,31A1)
      END
```

Method

Nowadays, almost all the pseudo uniform random numbers are generated according to Lehmer's congruence method.

$$\mathrm{IX}_{i+1} \equiv a \times \mathrm{IX}_i + c \pmod{m} \qquad (4.1)$$

where IX_i, a, c and m are non-negative integers. Since the congruence method relates two numbers in a certain definite association, it is foreign to the idea of probability, but if a sequence of pseudo random numbers, properly generated from the method, stands satistical tests, we can rely the congruence method as the pseudo random number generator.

If values are given to IX_0, "a" and "c" in (4.1), $\{\mathrm{IX}_i\}$ forms a sequence or residues with modules m and all the elements of $\{\mathrm{IX}_i\}$ satisfy $\mathrm{IX}_i < m$.

Letting $r_i = \mathrm{IX}_i / m$ for $\{\mathrm{IX}_i\}$, we can obtain a pseudo random number sequence $\{r_i\}$ which distributes on the interval (0, 1).

Here, for h such that $\mathrm{IX}_h = \mathrm{IX}_0$, h is called the period of the sequence $\{\mathrm{IX}_i\}$. This follows from the fact that $\mathrm{IX}_{h+1} = \mathrm{IX}_1$, $\mathrm{IX}_{h+2} = \mathrm{IX}_2$, ...

Table RANU2-2 χ^2-tests of RANU2

Test items	Size of samples*	Number of samples**	Number of samples not rejected at the level of significance a%.			Remarks
			10%	5%	1%	
One-dimensional frequency	1000	100	94	99	99	10 equal subintervals
	2000	50	45	48	50	10 equal subintervals
	10000	10	10	10	10	10 equal subintervals
Two-dimensional	10000x2	20	18	19	20	10x10 equal subintervals
Three-dimensional	5000x3	20	19	19	20	5x5x5 equal subintervals
Serial correction	1000	100	100	100	100	lag k = 60
	10000	10	10	10	10	lag k = 600
Gap	1000	100	68	84	91	The gaps of width longer than 7 were grouped into one class.
	10000	10	6	6	10	The gaps of width longer than 8 were grouped into one class.
Run up and down	1000	100	91	94	97	The runs of length longer than 3 were grouped into one class.
	10000	10	10	10	10	The runs of length longer than 4 were grouped into one class.
Run above and below the average	1000	100	92	94	100	The runs of length longer than 6 were grouped into one class.
	10000	10	10	10	10	The runs of length longer than 9 were grouped into one class.

* Number of random numbers contained in one sample.
** Test was done on the first successive sequences of samples.

There is a theory that such a period *h* always exists in any sequence of pseudo random numbers and its largest value relates to *m*.

This simply means that a pseudo random number sequence without period cannot be obtained with the congruence method.

In practice, if we assign a sufficiently large number to *m* we can make *h* long enough to make the sequence {IX$_i$} like random numbers.

In this subroutine, *m* was assigned to

$$m = 2^{31} \tag{4.2}$$

The value for *a* is assigned according to Greenberger's formula, which shows that a value close to $m^{\frac{1}{2}}$ minimizes the 1st order serial correlation among pseudo-random numbers.

Thus, from (4.2)

$$a = m^{\frac{1}{2}} + 3 = 2^{\frac{31}{2}} + 3 = 32771 \tag{4.3}$$

The value of *c* is selected so that *c* and *m* are prime each other.

Thus, *c* = 1234567891

To tell the truth, *a* and *c* were not simply determined, they are selected as a best pair among several other combinations of *a* and *c* through repetitive testing. This shows that the selection depends largely on empirical data.

Further details should be referred to Reference [89] pp.43-57.

J11-10-0201 RANU3

Generation of shuffled uniform (0, 1) pseudo random numbers
CALL RANU3 (IX, A, N, ISW, IVW, ICON)

Function

This subroutine generates, by the congruence method with shuffling, a sequence of *n* pseudo random numbers based on a starting value from a uniform distribution on the range (0, 1). Where $n \geq 1$.

Parameters

IX.....	Input. Starting value. A non-negative integer (must be INTEGER*4) Output. Starting value for the next call of RANU3. See "Comments on use".
A.....	Output. *n* uniform random numbers. One-dimensional array of size *n*.
N.....	Input. The number of uniform random numbers to be generated.
ISW...	Input. Specify 0 for the first call. Specify 1 for the subsequent calls.
IVW...	Work area. One-dimensional array of size 128 (must be INTEGER*4)
ICON..	Output. Condition code. See Table RANU3-1.

Table RANU3-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	IX < 0, N≤0, ISW < 0 or ISW > 1.	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, RANU2
  FORTRAN basic function ... DMOD

- Notes
  This subroutine enhances randomness which is one of two properties, probability unity and randomness, of random numbers. Therefore, the property of random numbers in this subroutine is superior to that in subroutine RANU2 (particularly for multiple dimensional distribution), but the processing speed is rather slow.
  If the random number should be generated quickly, it is advisable to use subroutine RANU2.
  [Starting value IX]
  Using the congruence method in (3.1), the starting value IX_0 when requiring uniform random integer sequence $\{IX_i\}$ is specified in parameter IX.

$$IX_{i+1} = a \times IX_i + c(\text{mod } m), \quad i = 0,1,... \tag{3.1}$$

  where, IX_i, a, c and m are non-negative integers.
  After generating random numbers, final IX_i is given to parameter IX. If generating random numbers in succession, IX_i can be used as a starting value.
  [Successive generation of random numbers]
  Random numbers can be generated in succession by calling this subroutine repeatedly.
  To perform this, ISW = 1 is entered for the subsequent calls.
  In this case, the contents of parameter IX and IVW should not be altered.
  If ISW = 0 is entered for the subsequent calls, another series of random numbers may be generated using the value of parameter IX at that time value.

- Example
  10000 pseudo random numbers are generated from uniform distribution in the range (0, 1) and a histogram of their frequency distribution is plotted.

```
C      **EXAMPLE**
       DIMENSION A(10000),HSUM(10),IVW(128)
       INTEGER*4 IX,IVW
       DATA X1,X2,NINT,XINT/0.0,1.0,10,0.1/
       DATA IX,N/0,10000/,ISW /0/
       DO 10 I=1,10
    10 HSUM(I)=0.0
       CALL RANU3(IX,A,N,ISW,IVW,ICON)
C      SUM NOS. IN HISTGRAM FORM
       ISW=1
       DO 20 I=1,N
       X=A(I)
    20 CALL HIST(X,X1,X2,NINT,XINT,HSUM,
      *          ISW)
C      PLOT DISTRIBUTION
       WRITE(6,600)
       ISW=2
       CALL HIST(X,X1,X2,NINT,XINT,HSUM,
      *          ISW)
       STOP
   600 FORMAT('1',10X,'UNIFORM RANDOM',
      *' NUMBER DISTRIBUTION'//)
       END
```

  For detailed information on subroutine HIST, see the Example in subroutine RANU2.

Method

Uniform random number have two main properties: probability unity and randomness.
The pseudo random number sequence generated using the Lehmer method

$$IX_{i+1} = a \times IX_i + c\,(\text{mod } m), \quad i = 0,1,2,... \tag{4.1}$$

increases in regularity as its order becomes higher in multiple-dimensional distribution and thus the randomness of pseudo random numbers is decreased. For example, if we construct points so that each point is coupled such as P_i (IX_i, IX_{i+1}) and plot P_i on the *x*-*y* coordinate, these points will be located on several parallel lines.

This subroutine uses the method with shuffling to decrease the demerit of the Lehmer method.

The way of generating random numbers is the same as that of the Lehmer method. However it enhances the randomness by determining the order of a random number sequence by using random numbers.

Random number sequence is generated as follows:

1) Generation of basic random number table
   Using the Lehmer method, ordered uniform random integers of length 128 are generated. Let express as:

$$IX_i^B,\ i=1, 2, ...,80 \qquad (4.2)$$

2) Generation of random number sequence
   Using the Lehmer method, one subsequent random number is generated which we call IX_0. The following procedure is repeated using l = 1, 2, ..., n to generate n pseudo random numbers.

– Using l-1-th random number, one random number is chosen from a basic random number table. This number is called IX_l.

$$IX_l = IX_j^B, \quad j = IX_{l-1}\ (mod\ 80)+1 \qquad (4.3)$$

IX_l is normalized in the interval (0, 1) and stored in A(l) as the l-th pseudo uniform random number.

– Subsequently, the l + 1-th random number, IX_{l+1} is generated and stored in the j-th position in the basic random number table.

$$IX_j^B = IX_{l+1} \qquad (4.4)$$

J21-10-0101 RATF1

Frequency test of uniform pseudo random numbers (0, 1)
CALL RATF1 (A, N, L, ALP, ISW, IFLG, VW, IVW, ICON)

Function

This subroutine performs the one-dimensional frequency test for n pseudo uniform (0, 1) random number sequence $\{x_i\}$ based upon the test of statistical hypothesis.

On the null hypothesis that pseudo uniform random numbers are uniform random numbers, this subroutine divides the range (0, 1) into l intervals and performs chi-square test using the actual frequency and the expected frequency of the random numbers failing in each interval.

Then it judges whether or not to accept this hypothesis on significance level α%.

When the number of random numbers is great, they can be tested in succession by calling this subroutine repeatedly by dividing the random number sequence into parts.

Where, $n \geq l \geq 2$ and $100 > \alpha > 0$.

Parameters

A.....	Input. Pseudo uniform random number sequence $\{x_i\}$. One-dimensional array of size n.
N.....	Input. The number of uniform random numbers. See Notes.
L.....	Input. The number of intervals. See Notes.
ALP..	Input. Significance level α. See Notes.
ISW...	Input. Control information. Specifies whether or not the test is performed independently for several random number sequences by calling this subroutine repeatedly. When ISW = 0 is specified, the test is performed independently for each random number sequence. When ISW = 1 is specified, the test is performed continuously. However ISW = 0 is specified for the first call. See Notes.
IFLG...	Output. Test results. When IFLG = 0 is specified, the hypothesis is accepted. When IFLG = 1 is specified, the hypothesis is rejected.
VW.....	Work area. One-dimensional array of size 2.
IVW....	Work area. One-dimensional array of size L + 1 (must be INTEGER*4).
ICON...	Output. Condition code. See Table RATF1-1.

Table RATF1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Some expected frequency F_i is small and the approximation of chi-square distribution is poor. Increase n or decrease l.	Continued. Resultant of test is not so reliable.
30000	N < L, L < 2, ALP ≥ 100, ALP ≤ 0, ISW < 0 or ISW > 1.	Bypassed

Comments on use

- Subprograms used
  SSL II ... UX2UP, MGSSL
  FORTRAN basic functions ... SQRT, EXP, ALOG, FLOAT, ABS, ERFC, ATAN

- Notes
  [Standard for setting the number of random numbers (n) and the number of subrange (l)]
  To enhance the reliability of testing, it is desirable to make the number of random numbers (n) large enough.
  Generally, the expected frequency F_i should satisfy

$$F_i\left(=n/l\right) > 10 \qquad (3.1)$$

This subroutine sets ICON = 10000 when (3.1) is not satisfied.

When m is large enough, the value of l is generally by:

$$l = \left[1 + 3.322 \, log_{10} \, n\right] \qquad (3.2)$$

For example, when n = 10000, l = 10 is adequate and when n = 1000, l = 14 is adequate.(3.2) is empirical formula based on the sense of sight appropriateness for the frequency distribution of random numbers falling in each interval.

[Standards for setting significance level α]
Significance level can be optionally specified according to the theory of statistical hypothesis test.

The significance level, however, indicates the probability of error of the first kind which means to reject the hypothesis although it is true, the value specified should lie between 1% and 10%

Generally either 5% or 1% is used.

[Testing in successing]
Suppose uniform random number sequence $\{y_i\}$ of size m is divided into s sets of random number sequences.

Letting

$$\{y_i\} = \{y_{i_1}\} + \{y_{i_2}\} + ... + \{y_{i_s}\}$$

and letting numbers in each random number sequence be m_1, m_2, ..., m_s that is,

$m = m_1 + m_2 + ... + m_s$

One call of this subroutine enables the testing of random number sequence $\{y_i\}$. By calling repeatedly for each random number sequence it is possible to obtain the final test results for $\{y_i\}$, also.

Table RATF1-2 shows the relationship between the contents of parameters and the object of the test for repeated calling of random number sequences.

Table RATF1-2 Testing in succession

Calling sequence	A	N	ISW	Object of test
1	$\{y_{i_1}\}$	m_1	0	$\{y_{i_1}\}$
2	$\{y_{i_2}\}$	m_2	1	$\{y_{i_1}\}+\{y_{i_2}\}$
:	:	:	:	:
S	$\{y_{i_s}\}$	m_s	1	$\{y_{i_1}\}+\{y_{i_2}\}$ + ... + $\{y_{i_s}\}$

Note: The value of parameter L and ALP should be constant.

If ISW = 0 is specified each time this subroutine is called, the random number sequence is tested individually.

When calling this subroutine repeatedly, the contents of work areas VW and IVW should not be altered.

[Contents of work area]

After executing this subroutine, the following values are stored in the work area:

VW(1):	Upper probability of χ^2 distribution of freedom degree $l - 1$ at χ_0^2.
VW(2):	Value of χ_0^2 for the frequency distribution of random sequence $\{x_i\}$.
IVW(I):	The actual frequency of random numbers falling in the I-th interval. I = 1, 2, ..., l.

- Example

Pseudo uniform random number sequence $\{x_i\}$ of size 10000 is generated by subroutine RANU2 and divided into 10 sets of random number sequence of size 10000 and the frequency tests for each set are performed.

Where, $l = 10$ and $\alpha = 5\%$.

```
C      **EXAMPLE**
       DIMENSION A(10000),VW(2),IVW(11)
       INTEGER*4 IX,IVW
       DATA IX,N/0,1000/,IOK/10/
       DATA L,ALP/10,5.0/,ISW/0/
       NS=10
       LL=L-1
       WRITE(6,600) IX,N,NS,LL,ALP
       IS=1
       IE=N
       DO 20 I=1,NS
       CALL RANU2(IX,A,N,ICON)
       CALL RATF1(A,N,L,ALP,ISW,IFLG,VW,
      *           IVW,ICON)
       IF(ICON.EQ.0) GOTO 10
       WRITE(6,610) I,IS,IE
       IOK=IOK-1
       GOTO 15
    10 IOK=IOK-IFLG
       IF(IFLG.EQ.0) WRITE(6,620) I,IS,IE
       IF(IFLG.EQ.1) WRITE(6,630) I,IS,IE
    15 IS=IE+1
    20 IE=IE+N
       RATE=IOK*100.0/NS
       WRITE(6,640) IOK,RATE
       STOP
   600 FORMAT('1',60('*')/6X,
      *'FREQUENCY TEST FOR UNIFORM',
      *' RANDOM NUMBERS.'/1X,60('*')//
      *6X,'INITIAL VALUE    IX=',I5/
      *6X,'SAMPLING LENGTH   N=',I5/
      *6X,'SAMPLE NUMBER      =',I5/
      *6X,'FREE DEGREE     L-1=',I5/
      *6X,'SIGNIFICANCE LEVEL =',F5.1,'%'/
      *6X,'RESULTANTS :'//
      *9X,'NO.   SAMPLE    JUDGEMENT')
   610 FORMAT(9X,I2,I7,'-',I5,4X,'ERROR')
   620 FORMAT(9X,I2,I7,'-',I5,4X,'SAFE')
   630 FORMAT(9X,I2,I7,'-',I5,4X,'FAIL')
   640 FORMAT(//6X,'TOTAL CONCLUSION:'//
      *9X,I2,' (',F5.1,
      *' %) SAMPLES ARE SAFE.')
       END
```

Method

Divide the interval (0, 1) into l equal intervals and let the number of pseudo random numbers falling if n i-th interval to be f_i. Also suppose the expected frequency corresponding to f_i to be F_i.

The value

$$\chi_0^2 = \sum_{i=1}^{l} \frac{(f_i - F_i)^2}{F_i} \qquad (4.1)$$

forms chi-square distribution of freedom degree l - 1 for many samples of $\{x_i\}$.

This subroutine obtains value χ_0^2 from (4.1) and tests the null hypothesis that n pseudo random numbers are uniform random numbers on significance level α%.

In (4.1), the expected frequency F_i can be obtained by

$$F_i = n / l \qquad (4.2)$$

For details, refer to Reference [93].

J21-10-0201 RATR1

Runs test of up-and-down of uniform (0.1) pseudo random numbers
CALL RATR1 (A, N, L, ALP, ISW, IFLG, VW, IVW, ICON)

Function

This subroutine performs the runs test of up-and-down for n pseudo uniform (0, 1) random number sequences $\{x_i\}$ based upon the test of statistical hypothesis.

On the null hypothesis that pseudo uniform random numbers are uniform random numbers, this subroutine performs chi-square test using the actual frequency and expected frequency of run of length r ($1 \le r \le l$, run of length greater than l is assumed to be length l) and tests whether or not the hypothesis is rejected on significance level α%.

Run of length r means a sub-sequence in which continued r elements are incremented (or decreased) monotonically in a random number sequence.

When the number of random number sequences is great, they can be tested in succession by calling this subroutine repeatedly.

Where, $n \ge l + 2$, $l \ge 2$, $100 > \alpha > 0$.

Parameters

A.....	Input. Pseudo uniform random number sequence $\{x_i\}$. One-dimensional array of size n.
N.....	Input. The number of uniform random numbers.
L.....	Input. The length l of the maximum run. See Notes.
ALP...	Input. Significance level α. See Notes.
ISW...	Input. Control information. Specifies whether or not the test is performed independently for several random number sequences by calling this subroutine repeatedly. When ISW = 0 is specified, the test is performed independently for each random number sequence. When ISW = 1 is specified, the test is performed continuously. However ISW = 0 is specified for the first call. See Notes.
IFLG...	Output. Test results. When IFLG = 0 is specified, the hypothesis is accepted. When IFLG = 1 is specified, the hypothesis is rejected.
VW.......	Work area. One-dimensional array of size 3.
IVW.....	Work area. One-dimensional array of size L + 8 (must be INTEGER*4).
ICON..	Output. Condition code. See Table RATR1-1.

Table RATR1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	Some expected frequency F_i is small and the approximation of chi-square distribution is poor.	Continued. Resultant of test is not so reliable.
30000	N < L + 2, L < 2, ALP ≥ 100, ALP ≤0, ISW < 0 or ISW > 1.	Bypassed

Comments on use

- Subprograms used
  SSL II ... UX2UP, MGSSL
  FORTRAN basic functions ... SQRT, EXP, ALOG, FLOAT, ATAN, ERFC, MAX0, MIN0

- Notes
  [Standard for setting the length l of maximum run]
  The expected frequency F_r to make the length of run r is expressed by:

$$F_r = 2n\frac{r^2+3r+1}{(r+3)!} - 2\frac{r^3+3r^2-r-4}{(r+3)!} \quad , r=1,2,...,n-2 \tag{3.1}$$

That is the expected frequency F_r decreases as

$$F_{r+1} \approx F_r/(r+1)$$

For example, when n = 10000, F_1 = 4160, F_2 = 1834, ..., F_5 = 20, F_6 = 3.
Therefor, in this case, should be taken as
$l = 5$

This subroutine sets ICON = 10000 when the expected frequency F_i does not satisfy
$F_i > 10$

[Standard for setting significance level]
The significance level can be optionally specified according to the theory of statistical hypothesis test.

The significance level, however, indicates the probability of error of the first kind which means to reject the hypothesis although it is true, the value specified should lie between 1% and 10%.

Generally either 5% or 1% is used.

[Testing in succession]
Suppose uniform random number sequence $\{y_i\}$ of size m is divided into s sets of random number sequences. Letting

$$\{y_i\} = \{y_{i1}\} + \{y_{i2}\} + ... + \{y_{is}\}$$

and letting numbers in each random number sequence be $m_1, m_2, ..., m_s$, that is ,

$$m = m_1 + m_2 + ... + m_s$$

One call of this subroutine enables the testing of random number sequence $\{y_i\}$. By calling repeatedly for each random numbers sequence it is possible to obtain the final test results for $\{y_i\}$, also.

Table RATR1-2 shows the relationship between the contents of parameters and the object of the test for repeated calling of random number sequences.

Table RATR1-2 Successive testing

Calling sequence	A	N	ISW	Object of test
1	$\{y_{i_1}\}$	m_1	0	$\{y_{i_1}\}$
2	$\{y_{i_2}\}$	m_2	1	$\{y_{i_1}\}+\{y_{i_2}\}$
:	:	:	:	:
S	$\{y_{i_s}\}$	m_s	1	$\{y_{i_1}\}+\{y_{i_2}\}$ $+\ldots+\{y_{i_s}\}$

Note: The value of parameter L and ALP should be constant.

If ISW = 0 is specified each time this subroutine is called, the random number sequence is tested individually.

When calling this subroutine repeatedly, the contents of work areas VW and IVW should not be altered.

[Contents of work area]

After executing this subroutine, the following values are stored in the work area:

VW(1): Upper probability of χ^2 distribution of degree of freedom, $l-1$ at χ_0^2.

VW(2): Value of χ_0^2 for the frequency distribution of random sequence $\{x_i\}$.

IVW(I): The actual frequency of runs of length l. I = 1, 2, ..., l.

- Example

  Pseudo uniform random number sequence $\{x_i\}$ of size 10000 is generated by subroutine RANU2 and divided into 10 sets of random number sequence of size 1000 and the runs tests of up-and-down for each set are performed.

  Where, l = 4 and α = 5%.

```
C       **EXAMPLE**
        DIMENSION A(1000),VW(3),IVW(12)
        DATA IX,N/0,1000/,IOK/10/
        DATA L,ALP/4,5.0/,ISW/0/
        NS=10
        LL=L-1
        WRITE(6,600) IX,N,NS,LL,ALP
        IS=1
        IE=N
        DO 20 I=1,NS
        CALL RANU2(IX,A,N,ICON)
        CALL RATR1(A,N,L,ALP,ISW,IFLG,VW,
       *           IVW,ICON)
        IF(ICON.EQ.0) GOTO 10
        WRITE(6,610) I,IS,IE
        IOK=IOK-1
        GOTO 15
   10   IOK=IOK-IFLG
        IF(IFLG.EQ.0) WRITE(6,620) I,IS,IE
        IF(IFLG.EQ.1) WRITE(6,630) I,IS,IE
   15   IS=IE+1
   20   IE=IE+N
        RATE=IOK*100.0/NS
        WRITE(6,640) IOK,RATE
        STOP
  600   FORMAT('1',60('*')/6X,
       *'RUNS TEST FOR UNIFORM',
       *' RANDOM NUMBERS.'/1X,60('*')//
       *6X,'INITIAL VALUE    IX=',I5/
       *6X,'SAMPLING LENGTH   N=',I5/
       *6X,'SAMPLE NUMBER      =',I5/
       *6X,'FREE DEGREE     L-1=',I5/
       *6X,'SIGNIFICANCE LEVEL =',F5.1,'%'/
       *6X,'RESULTANTS :'//
       *9X,'NO.   SAMPLE    JUDGEMENT')
  610   FORMAT(9X,I2,I7,'-',I5,4X,'ERROR')
  620   FORMAT(9X,I2,I7,'-',I5,4X,'SAFE')
  630   FORMAT(9X,I2,I7,'-',I5,4X,'FAIL')
  640   FORMAT(//6X,'TOTAL CONCLUSION:'//
       *9X,I2,' (',F5.1,
       *' %) SAMPLES ARE SAFE.')
        END
```

Method

The up run of length r means a sub-sequence which consists of r element satisfies following

$$......x_{k-1} > x_k < x_{k+1} < ... < x_{k+r} > x_{k+r+1}...$$

where the first element of the sub-sequence is x_k ($1 \le k < n$). However, when k = 1 or $k + r = n$, the inequality signs should not be used at the ends.

For down run, the similar definition is used.

Now the number of runs of length r (maximum l, $1 \le r \le l$) in $\{x_i\}$ is expressed as f_r.

f_l is the total number of runs in which the length is more than l. Letting the expected frequency of length r be F_r, the value forms chi-square distribution freedom degree l - 1 for many samples of $\{x_i\}$.

$$\chi_0^2 = \sum_{i=1}^{l} \frac{(f_i - F_i)^2}{F_i} \tag{4.1}$$

This subroutine tests the null hypothesis that n pseudo random numbers are uniform random numbers on significance level α% after obtaining value χ_0^2 from (4.1). If (4.1), the expected frequency F_r is

$$F_r = 2n\frac{r^2+3r+1}{(r+3)!} - 2\frac{r^3+3r^2-r-4}{(r+3)!} \tag{4.2}$$

$$, r = 1,2,...,n-2$$

and the total sum is

$$\sum_r F_r = (2n-7)/3 \tag{4.3}$$

Although the following will be supposed in chi-square test,

$$\sum_r f_r = \sum_r F_r \tag{4.4}$$

actually (4.4) is not satisfied.

Therefore this subroutine uses $F*_r$ instead of F_r as

$$F_r^* = \sum_r f_r \frac{F_r}{\sum_r F_r} \tag{4.5}$$

in which expected frequency F_r is modified by the actual frequency f_r.

This enhances the accuracy of the testing.

For details, refer to Reference [93].

F11-31-0101 RFT, DRFT

Discrete real Fourier transform
CALL RFT (A, N, ISN, ICON)

Function

When one-variable real time series data $\{x_j\}$ of dimension n is given, this subroutine performs a discrete real Fourier transform or its inverse Fourier transform using the Fast Fourier Transform (FFT) Method.
n must be a number expressed as $n = 2^l$ (n: positive integer).

- Fourier transform
  When $\{x_j\}$ is input, this subroutine performs the transform defined in(1.1), and determines Fourier coefficients $\{na_k\}$ and $\{nb_k\}$.

$$\left.\begin{aligned} na_k &= 2\sum_{j=0}^{n-1} x_j \cos\frac{2\pi kj}{n}, k = 0,...,\frac{n}{2} \\ nb_k &= 2\sum_{j=0}^{n-1} x_j \sin\frac{2\pi kj}{n}, k = 1,...,\frac{n}{2}-1 \end{aligned}\right. \qquad (1.1)$$

- Inverse Fourier transform
  When $\{a_k\}$, $\{b_k\}$ are input, this subroutine performs the transform defined in (1.2), and determines the values $\{2x_j\}$ of the Fourier series.

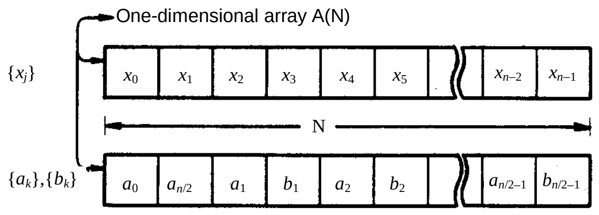
$$\begin{aligned} 2x_j &= a_0 + 2\sum_{k=1}^{n/2-1}\left(a_k \cos\frac{2\pi kj}{n} + b_k \sin\frac{2\pi kj}{n}\right) \qquad (1.2) \\ &\quad + a_{n/2}\cos\pi j \\ &\quad j=0,...,n\text{-}1 \end{aligned}$$

Parameters

A.....	Input. $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$ Output. $\{na_k\}$, $\{nb_k\}$ or $\{2x_j\}$ One-dimensional array of size N. See Fig. RFT-1 for the data storage format.
N.....	Input. Dimension n.
ISN...	Input. Specifies normal or inverse transform ($\neq 0$). For transform: ISN = + 1 For inverse transform: ISN = - 1 (See Notes).
ICON..	Output. Condition code See Table RFT-1

Table RFT-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	ISN = 1 or N $\neq 2^l$ (l: positive integer)	Bypassed



Note: $\{na_k\}$, $\{nb_k\}$ correspond to $\{a_k\}$, $\{b_k\}$.

Fig. RFT-1 Data storage method

Comments on use

- Subprograms used
  SSL II ... CFTN, PNR, URFT, and MGSSL
  FORTRAN basic functions ... ATAN, ALOG, SQRT, SIN, and IABS

- Notes
  General definition of discrete real Fourier transform:
  Discrete real Fourier transforms and inverse Fourier transforms are generally defined as:

$$\left.\begin{aligned} a_k &= \frac{2}{n}\sum_{j=0}^{n-1} x_j \cos\frac{2\pi kj}{n}, k = 0,...,\frac{n}{2} \\ b_k &= \frac{2}{n}\sum_{j=0}^{n-1} x_j \sin\frac{2\pi kj}{n}, k = 1,...,\frac{n}{2}-1 \end{aligned}\right\} (3.1)$$

$$\begin{aligned} x_j &= \frac{1}{2}a_0 + \sum_{k=1}^{n/2-1}\left(a_k \cos\frac{2\pi kj}{n} + b_k \sin\frac{2\pi kj}{n}\right) \qquad (3.2) \\ &\quad + \frac{1}{2}a_{n/2}\cos\pi j, j = 0,...,n-1 \end{aligned}$$

This routine determines $\{na_k\}$, $\{nb_k\}$, or $\{2x_j\}$ in place of $\{a_k\}$, $\{b_k\}$ of (3.1) or $\{x_j\}$ of (3.2). Scaling of the resultant values is left to the user.

Notice that a normal transform followed by an inverse transform returns the original data multiplied by the value $2n$.

Specifying ISN:
ISN is used to specify normal or inverse transform. It is also used as follows;
If $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$ are stored in an area of size N·I in intervals of I, the following specification is made.
For transform: ISN = + I
For inverse transform: ISN = -I
In this case, the results of the transform are also stored in intervals of I.

- Example
  Real time series data $\{x_j\}$ of dimension n is put, transform is performed using this routine, and the results are scaled to obtain $\{a_k\}$, $\{b_k\}$.
  In case of $n \leq 1024$ (= 2^{10}).

```
C      **EXAMPLE**
       DIMENSION A(1024)
       READ(5,500) N,(A(I),I=1,N)
       WRITE(6,600) N,(I,A(I),I=1,N)
       ISN=1
       CALL RFT(A,N,ISN,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       WRITE(6,620) A(1)
       N1=N/2-1
       IF(N1.NE.0)
      * WRITE(6,630) (I,A(2*I+1),A(2*I+2),
      *              I=1,N1)
       N1=N1+1
       WRITE(6,630) N1,A(2)
       STOP
   500 FORMAT(I5/(2E20.7))
   600 FORMAT('0',10X,'INPUT DATA N=',I5/
      * /(15X,I5,E20.7))
   610 FORMAT('0',10X,'RESULT ICON=',I5/)
   620 FORMAT('0',17X,'K',10X,'A(K)',16X,
      * 'B(K)'//19X,'0',E20.7)
   630 FORMAT(/15X,I5,2E20.7)
       END
```

Method

This subroutine performs a discrete real Fourier transform (hereafter referred to as real transform) of dimension n $(=2\cdot m)$ using the radix 8 and 2 Fast Fourier Transform (FFT) method.

By considering real data $\{x_j\}$ to be transformed as complex data with the imaginary part zero, a real transform can be done by a discrete complex Fourier transform (hereafter referred to as complex transform). However in such case, the complex transform can be done efficiently using the characteristics as described below.

Let the complex transform be defined as

$$\alpha_k = \sum_{j=0}^{n-1} x_j \exp\left(-2\pi i \frac{kj}{n}\right), k = 0,...,n-1 \tag{4.1}$$

Then, when $\{x_j\}$ is real data, the complex conjugate relationship results as shown in (4.2)

$$\alpha_{n-k} = \alpha_k^*, k = 1,...,n-1 \tag{4.2}$$

where * represents complex conjugates

Now, the relationship between the results of a real transform $\{a_k\}$, $\{b_k\}$, and the results of a complex transform $\{a_k\}$ are

$$\left.\begin{array}{l} a_0 = 2\cdot\alpha_0, \alpha_{n/2} = 2\cdot\alpha_{n/2} \\ a_k = (\alpha_k + \alpha_{n-k}), k = 1,...,n/2-1 \\ b_k = i(\alpha_k - \alpha_{n-k}), k = 1,...,n/2-1 \end{array}\right\} \tag{4.3}$$

Therefore, when a complex transform is used for real data from (4.2) and (4.3) it can be seen that $\{a_k\}$, $\{b_k\}$ can be determined by determined only a_k, $k = 0, ..., n/2$ considering (4.2) and (4.3). This means that complex transform on real data has redundancy in calculating conjugate elements.

In this routine this redundancy is avoided by use of inherent complex transform as described below:

- Real transform by the complex Fourier transform
  First, expansion of the complex transform (4.1) applying the principles of the Fast Fourier Transform (FFT) method is be considered. Since $n = 2\cdot m$, k and j can be expressed as

$$\begin{array}{ll} k = k_0 + k_1\cdot m, & 0 \le k_0 \le m-1, \quad 0 \le k_1 \le 1 \\ j = j_0 + j_1\cdot 2, & 0 \le j_0 \le 1, \quad 0 \le j_1 \le m-1 \end{array} \tag{4.4}$$

If (4.4) is substituted in (4.1) and common terms rearranged, (4.5) results.

$$\begin{aligned} \alpha_{k_0+k_1\cdot m} = & \sum_{j_0=0}^{1} \exp\left(-2\pi i \frac{j_0 k_1}{2}\right) \exp\left(-2\pi i \frac{j_0 k_0}{n}\right) \\ & \cdot \sum_{j_1=0}^{m-1} \exp\left(-2\pi i \frac{j_0 k_0}{m}\right) x_{j_0+j_1 2} \end{aligned} \tag{4.5}$$

(4.5) means that a complex transform of dimension $2m$ can be performed by two sets of elementary complex transforms of dimension m by $\sum_{j_1}$ and m sets of elementary complex transforms of dimension 2 by $\sum_{j_0}$.

(Refer to the section on CFT for principle of the Fast Fourier Transform method.) In this routine (4.5) is successively calculated for complex data with the imaginary parts zero.

– Transform by $\sum_{j_1}$

  Two sets of elementary complex transforms of dimension m are performed by $\sum_{j_1}$ respect to j_0.

  By performing complex transforms on the even-number data $\{x1_j\}$ and the odd-number data $\{x2_j\}$, $\{\alpha_k^{x1}\}$ and $\{\alpha_k^{x2}\}$ can be determined. Since the imaginary Parts of the complex data $\{x1_j\}$ and $\{x2_j\}$ are zero, just as in (4.2), the conjugate relationships shown in (4.6) exist.

$$\left.\begin{array}{l} \alpha_{m-k}^{x1} = (\alpha_k^{x1})^* \\ \alpha_{m-k}^{x2} = (\alpha_k^{x2})^* \end{array}\right\}, k = 1,...,m-1 \tag{4.6}$$

  The above two sets of elementary complex transforms can be performed as a single transform plus a few additional operations; that is, the real parts of $\{x1_j\}$ and $\{x2_j\}$ are paired as shown in (4.7) to form the real and imaginary parts of new complex data $\{z_j\}$.

$$z_j = x1_j + i\cdot x2_j, j = 0,...,m-1 \tag{4.7}$$

and complex transform of dimension m is done with respect to $\{z_j\}$ and $\{\alpha_k^z\}$ is determined. Then, using (4.8), $\{\alpha_k^{x1}\}$ and $\{\alpha_k^{x2}\}$ are obtained from

$$\left.\begin{array}{l}\left.\begin{array}{l}\alpha_k^{x1}=\frac{1}{2}\left(\alpha_k^z+\left(\alpha_{m-k}^z\right)^*\right)\\ \alpha_k^{x2}=\frac{1}{2i}\left(\alpha_k^z-\left(\alpha_{m-k}^z\right)^*\right)\end{array}\right\},k=1,...,\frac{m}{2}\\ \alpha_0^{x1}=\mathrm{Re}\left(\alpha_0^z\right),\alpha_0^{x2}=\mathrm{Im}\left(\alpha_0^z\right)\end{array}\right\} \quad (4.8)$$

Transform by $\sum_{j_0}$

m sets of complex transforms of dimension 2 are performed by $\sum_{j_0}$ with respect to j_1.

The result $\{\alpha_k\}$ is the complex transform of complex data whose imaginary parts are zero. To obtain $\{a_k\}$ and $\{b_k\}$, $\{\alpha_k\}$ needs to be determined only for k = 0, ..., $n/2$. In complex transforms by $\sum_{j_0}$, calculation conjugate terms is omitted.

- Processing in this routine

The processing for a real Fourier transform in this routine is discussed. Refer to Fig. RFT-2 for specific example of dimension 16.

(a) Let the even number real data of dimension n be $\{x1_j\}$ and odd number data be $\{x2_j\}$, then complex transform of dimension m is performed with respect to $\{z_j\}$ in (4.7) to determine $\{\alpha_k^z\}$.

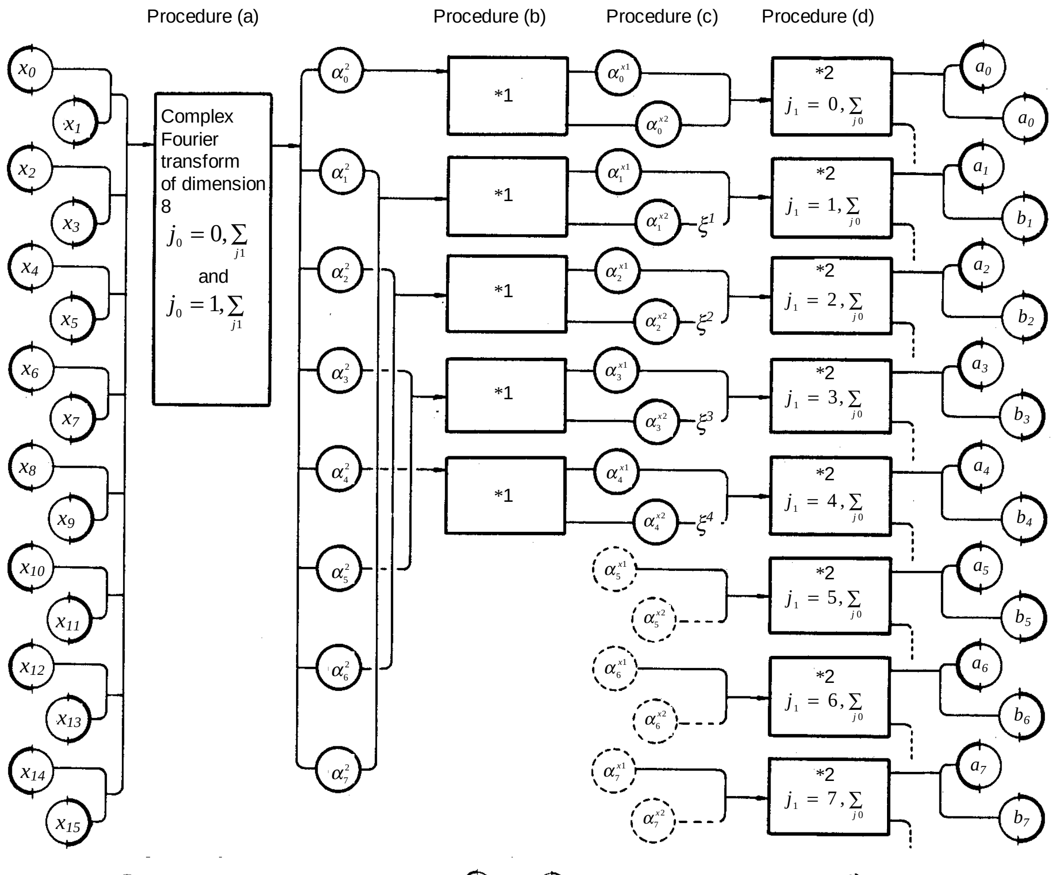
(b) Using (4.8), the transformed results $\{\alpha_k^{x1}\}$ and $\{\alpha_k^{x2}\}$ correspond to $\{x1_j\}$ and $\{x2_j\}$ are determined from $\{\alpha_k^z\}$.

(c) $\{\alpha_k^{x2}\}$ is multiplied by the rotation factor.

(d) Complex transforms of dimension 2 are performed.

In this routine, the complex transform in (a) is performed by subroutines CFTN and PNR.

For further information, refer to References [55], [56], and [57].



Procedure (a)
Procedure (b)
Procedure (c)
Procedure (d)
x_0 … x_{15}
Complex Fourier transform of dimension 8
$j_0 = 0, \sum_{j1}$
and
$j_0 = 1, \sum_{j1}$
α_0^z … α_7^z
*1
α_k^{x1}, α_k^{x2}
ξ^1 ξ^2 ξ^3 ξ^4
*2
$j_1 = 0, \sum_{j0}$ … $j_1 = 7, \sum_{j0}$
a_0 … a_7
b_1 … b_7

Note: Symbol ○ indicates complex data, symbols ○ and ○ indicate real data. Symbols ◌ and .... indicate operations that may be omitted. ξ is the rotation factor; ξ= exp (-2πi/16)

*1 Separation into two sets of 8 term complex Fourier transforms of dimension 8

*2 Complex Fourier transform of dimension 2

Fig. RTF-2 Flowchart of a real Fourier transform of dimension 16

C22-11-0111 RJETR, DRJETR

Zeros of a polynominal with real coefficients (Jenkins-Traub method)
CALL RJETR (A, N, Z, VW, ICON)

Function

This subroutine finds zeros of a polynomial with real coefficients;

$$a_0x^n + a_1x^{n-1} + ... + a_n = 0$$
$$(a : \text{real}, \quad a_0 \neq 0, n \geq 1)$$

by Jenkins-Traub's three-stage algorithm.

Parameters

A.....	Input. Coefficients of the polynomial equation. One-dimensional array of size $(n + 1)$ A(1) = a_0, A(2) = a_1, ..., A(N + 1) = a_n in order. The contents of A are altered on output.
N.....	Input. Degree of the equation. Output. The number of roots obtained. (See "Comments on use".)
Z.....	Output. n roots. Complex one-dimensional array of size n. Obtained roots are returned in Z(1), Z(2), ... So, if the number of obtained roots is N, those roots are returned in Z(1), ..., Z(N).
VW...	Work area. One-dimensional array of size 6 $(n + 1)$
ICON..	Output. Condition code. See Table RJETR-1.

Table RJETR-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	All of n roots were not obtained.	The number of obtained roots is put into parameter N. These roots themselves are put into Z(1)~Z(N).
30000	n < 1 of a_0 = 0	Bypassed

Comments on use

- Subprograms used
  SSL II ... MGSSL, AMACH, RQDR, IRADIX, AFMAX, AFMIN, and UJET
  FORTRAN basic functions ... ABS, ALOG, EXP, SQRT, CMPLX, SIN, ATAN, REAL, AIMAG, and AMAX1

- Notes
  A COMPLEX declaration for array Z must be done in the program which calls this subroutine.

  An n degree polynomial equation has n roots. All of the roots can not always be obtained. Users should be aware of this and make sure of the values of parameters ICON and N after calculation.

- Example
  Degree n and real coefficients a_i are input and roots are calculated for $1 \leq n \leq 50$.

```
C     **EXAMPLE**
      DIMENSION A(51),Z(50),VW(306)
      COMPLEX Z
      READ(5,500) N
      N1=N+1
      READ(5,510) (A(I),I=1,N1)
      DO 10 I=1,N1
      K=I-1
   10 WRITE(6,600) K,A(I)
      CALL RJETR(A,N,Z,VW,ICON)
      WRITE(6,610) N,ICON
      IF(ICON.EQ.30000) STOP
      WRITE(6,620) (I,Z(I),I=1,N)
      STOP
  500 FORMAT(I2)
  510 FORMAT(5F10.0)
  600 FORMAT(10X,'A(',I2,')=',E20.8)
  610 FORMAT(10X,'N=',I2,5X,'ICON=',I5)
  620 FORMAT(10X,'Z(',I2,')=',2E20.8)
      END
```

Method

This subroutine employs the Jenkins-Traub's three-stage algorithm. The three-stage algorithm consists of:

- Using K polynomial (described later) defined differently at each of the three stages, the roots are pursued so that the smallest root is first found. (state 1)
- To make sure if the calculation can converge. (stage 2)
- Finally to speed up the convergence and to obtain the roots. (stage 3)

Especially, if a real coefficient polynomial equation is given, since the roots are pursued as linear or quadratic factors, the discrimination if a linear or a quadratic factor is converging is made in the stage 2. Then accordingly the calculation is speeded up in two ways in the stage 3. If the second order factors are determined, the roots are obtained with the quadratic equation formula.

- Features
  a. Only real arithmetic is used. Complex conjugate roots are found as quadratic factors.
  b. Roots are calculated in roughly increasing order of modulus; this avoids the instability which occurs when the polynomial is deflated with a large root.
  c. The rate of convergence of the third stage is

faster than second order.

- K polynomials
  Before describing the algorithm two important sequences of polynomials are introduced and their characteristics are described below. In equation

$$f(x) \equiv a_0 x^n + a_1 x^{n-1} + \dots + a_n = \prod_{i=1}^{j} (x-\rho_i)^{mi} = 0, \rho_i \neq \rho_k (i \neq k) \tag{4.1}$$

Let's assume $a_0 = 1$ and $a_n \neq 0$, but there is no loss of generality. Starting from $(n-1)$-th degree arbitrary polynomial $K^{(0)}(x)$, for $\lambda = 0, 1, \dots$, $K^{(\lambda)}(x)$ is defined as

$$K^{(\lambda+1)}(x) = \frac{1}{x}\left[K^{(\lambda)}(x) - \frac{K^{(\lambda)}(0)}{f(0)} f(x)\right] \tag{4.2}$$

Obviously, every $K^{(\lambda)}(x)$ are of degree at most $n-1$. Let

$$K^{(0)}(x) = \sum_{i=1}^{j} c_i f_i(x), \qquad f_i(x) = \frac{f(x)}{x-\rho_i} \tag{4.3}$$

$K^{(\lambda)}(x)$ is expressed as follows.

$$K^{(\lambda)}(x) = \sum_{i=1}^{j} c_i \rho_i^{-\lambda} f_i(x) \tag{4.4}$$

From (4.4), if ρ_1 exists such that $|\rho_1| < |\rho_i|$ $(i \geq 2)$ and c $\neq 0$,

$$\lim_{\lambda\to\infty} f(x)/\overline{K}^{(\lambda)}(x) = x - \rho_1 \tag{4.5}$$

holds, where $\overline{K}^{(\lambda)}(x)$ is defined to be $K^{(\lambda)}(x)$ divide by its leading coefficient. The rate of convergence of (4.5) depends on the ratio of ρ_1 and ρ_2 $(i \geq 2)$. The polynomial defined by (4.2) is called a no-sift polynomial. The second polynomials are defined as follow. Starting from $(n-1)$-th degree polynomial $K^{(0)}(x)$, for $\lambda = 0, 1, \dots$, it is defined as

$$K^{(\lambda+1)}(x) = \frac{1}{\sigma(x)}\left[K^{(\lambda)}(x) + \left(A^{(\lambda)}x + B^{(\lambda)}\right) f(x)\right] \tag{4.6}$$

Here $\sigma(x)$ is a real quadratic $x^2 + ux + v$ with roots s_1 and s_2 such that $s_1 \neq s_2$, $|s_1| = |s_2| = \beta$, $\beta \leq \min|\rho_i|$ and $f(s_1)$ $f(s_2) \neq 0$. $A^{(\lambda)}$ and $B^{(\lambda)}$ are chosen so that the expression in the bracket [ ] of (4.6) can be divided by σ(x) and are

$$A^{(\lambda)} = \begin{vmatrix} f(s_1) & f(s_2) \\ K^{(\lambda)}(s_1) & K^{(\lambda)}(s_2) \end{vmatrix} \Bigg/ \begin{vmatrix} s_1 f(s_1) & s_2 f(s_2) \\ f(s_1) & f(s_2) \end{vmatrix}$$

$$B^{(\lambda)} = \begin{vmatrix} K^{(\lambda)}(s_1) & K^{(\lambda)}(s_2) \\ s_1 f(s_1) & s_2 f(s_2) \end{vmatrix} \Bigg/ \begin{vmatrix} s_1 f(s_1) & s_2 f(s_2) \\ f(s_1) & f(s_2) \end{vmatrix} \tag{4.7}$$

respectively. The every $K^{(\lambda)}(x)$ of (4.6) is of degree n - 1 at most. Employing (4.3) for $K^{(0)}(x)$ as before, the polynomial defined by (4.6) is given as follows:

$$K^{(\lambda)}(x) = \sum_{i=1}^{j} c_i \sigma_i^{-\lambda} f_i(x), \quad \sigma_i = \sigma(\rho_i) \tag{4.8}$$

The polynomial defined by (4.6) is called fixed-shift polynomial. It has the following two important properties.

- If there exists ρ_1 such that

$$|\sigma_1| < |\sigma_i|, i \geq 2 \tag{4.9}$$

then, ρ_1 is real and

$$\lim_{\lambda\to\infty} f(x)/\overline{K}^{(\lambda)}(x) = x - \rho_1 \tag{4.10}$$

holds,
where $K^{(\lambda)}(x)$ is defined to be $K^{(\lambda)}(x)$ divided by its leading coefficient.

- If there exist ρ_1 and ρ_2 such that

$$|\sigma_1| = |\sigma_2| < |\sigma_i|, i \geq 3 \tag{4.11}$$

then, by letting

$$K_0^{(\lambda)}(x) = K^{(\lambda)}(x), \quad K_{v+1}^{(\lambda)}(x) = \frac{1}{x}\left[K_v^{(\lambda)}(x) - \frac{K_v^{(\lambda)}(0)}{f(0)} f(x)\right], \quad v = 0,1 \tag{4.12}$$

and

$$\sigma^{(\lambda)}(x) = \begin{vmatrix} K_0^{(\lambda)}(s_1) & K_0^{(\lambda)}(s_2) & x^2 \\ K_1^{(\lambda)}(s_1) & K_1^{(\lambda)}(s_2) & x \\ K_2^{(\lambda)}(s_1) & K_2^{(\lambda)}(s_2) & 1 \end{vmatrix} \Bigg/ \begin{vmatrix} K_1^{(\lambda)}(s_1) & K_1^{(\lambda)}(s_2) \\ K_2^{(\lambda)}(s_1) & K_2^{(\lambda)}(s_2) \end{vmatrix} \tag{4.13}$$

holds.

Three-stage algorithm
The three-stage algorithm consists of three stages. (n – 1)-th degree polynomials $K^{(\lambda)}(x)$ which play a basic role at each step, are generated as different sequences for each of the steps. Polynomial $K^{(M)}(x)$ at the end of stage 1 is used as the beginning polynomial for stage 2, and $K^{(L)}(x)$ at the end of stage 2 is used as the beginning polynomial for stage 3.

- Stage 1 (no-sift process)
  The algorithm starts with $K^{(0)}(x) = f'(x)$ and

the process (4.2) is iterated *M*, times. As for *M*, it is described later. The purpose of this stage is to make the term which include p_i of small absolute value dominant in $K^{(M)}(x)$.

- Stage 2 (fixed-shift process)

Applying (4.6) for $\lambda = M, M-1, ..., L-1$, a sequence of fixed-shift polynomials are produced. As for *L*, see "Stopping Criterion". The purpose of producing fixed-shift polynomials is to see if (4.9) of (4.11) holds. However, since σ_k, $k = 1, ..., j$ cannot be calculated, $f(x)/\overline{K}^{(\lambda)}(x)$ and $\sigma^{(\lambda)}(x)$ must be calculated as well as finxed-polynomials, then by using both of them, whether roots converge on linear or quadratic factors is determined. Which of (4.10) of (4.13) holds depends on the choice of s_1 and s_2. If any convergence is not met, it can be considered to be due to improper choice of s_1 and s_2, and then they would be chosen again. (See "Notes on algorithm".)

When $f(x)/\overline{K}^{(\lambda)}(x)$ starts to converge in stage 2, the rate of convergence is accelerated by shifting it with current approximate. And when $\sigma^{(\lambda)}(x)$ starts to converge, the speed is accelerated by replacing $\sigma(x)$ of (4.6) with $\sigma^{(\lambda)}(x)$. This motivates going to the next variable-shift process.

- Stage 3 (variable-shift process) This step is divided into the following two procedures depending on the state of convergence (upon linear or quadratic factors) in stage 2.

  (a) Iteration for a linear factor
  Letting

$$s^{(L)} = \mathrm{Re}\left(s_1 - f(s_1)/\overline{K}^{(L)}(s_1)\right)$$

$$K^{(\lambda+1)}(x) = \frac{1}{x - s^{(\lambda)}}\left[K^{(\lambda)}(x) - \frac{K^{(\lambda)}\left(s^{(\lambda)}\right)}{f\left(s^{(\lambda)}\right)} f(x)\right] \quad (4.14)$$

$$s^{(\lambda+1)} = s^{(\lambda)} - f\left(s^{(\lambda)}\right)/\overline{K}^{(\lambda)}\left(s^{(\lambda)}\right)$$

$$\lambda = L, L+1, ... \quad (4.15)$$

  then the sequences $s^{(\lambda)}$ converges to ρ_1

  (b) For $\lambda = L + 1, ..., \sigma^{(\lambda+1)}(x)$ are calculated based on the following variable-shift polynomial $K^{(\lambda+1)}(x)$.

$$K^{(\lambda+1)}(x) = \frac{1}{\sigma^{(\lambda)}(x)}$$

$$\left[K^{(\lambda)}(x) + \frac{\begin{vmatrix} f\left(s_1^{(\lambda)}\right) & f\left(s_2^{(\lambda)}\right) \\ K^{(\lambda)}\left(s_1^{(\lambda)}\right) & K^{(\lambda)}\left(s_2^{(\lambda)}\right) \end{vmatrix} x + \begin{vmatrix} K^{(\lambda)}\left(s_1^{(\lambda)}\right) & K^{(\lambda)}\left(s_2^{(\lambda)}\right) \\ s_1^{(\lambda)} f\left(s_1^{(\lambda)}\right) & s_2^{(\lambda)} f\left(s_2^{(\lambda)}\right) \end{vmatrix}}{\begin{vmatrix} s_1^{(\lambda)} f\left(s_1^{(\lambda)}\right) & s_2^{(\lambda)} f\left(s_2^{(\lambda)}\right) \\ f\left(s_1^{(\lambda)}\right) & f\left(s_2^{(\lambda)}\right) \end{vmatrix}} f(x)\right] \quad (4.16)$$

$$K_0^{(\lambda+1)}(x) = K^{(\lambda+1)}(x)$$

$$K_{v+1}^{(\lambda+1)}(x) = \frac{1}{x}\left[K_v^{(\lambda+1)} - \frac{K_v^{(\lambda+1)}(0)}{f(0)} f(x)\right], v = 0,1 \quad (4.17)$$

$$\sigma^{(\lambda+1)}(x) = \frac{\begin{vmatrix} K_0^{(\lambda+1)}\left(s_1^{(\lambda)}\right) & K_0^{(\lambda+1)}\left(s_2^{(\lambda)}\right) & x^2 \\ K_1^{(\lambda+1)}\left(s_1^{(\lambda)}\right) & K_1^{(\lambda+1)}\left(s_2^{(\lambda)}\right) & x \\ K_2^{(\lambda+1)}\left(s_1^{(\lambda)}\right) & K_2^{(\lambda+1)}\left(s_2^{(\lambda)}\right) & 1 \end{vmatrix}}{\begin{vmatrix} K_1^{(\lambda+1)}\left(s_1^{(\lambda)}\right) & K_1^{(\lambda+1)}\left(s_2^{(\lambda)}\right) \\ K_2^{(\lambda+1)}\left(s_1^{(\lambda)}\right) & K_2^{(\lambda+1)}\left(s_2^{(\lambda)}\right) \end{vmatrix}}$$

Where $s_1^{(\lambda)}$ and $s_2^{(\lambda)}$ are two roots of $\rho^{(\lambda)}(x)$.
Then the sequence $\rho^{(\lambda)}(x)$ converges to $(x-\rho_1)(x-\rho_2)$.
As for stopping rule see "Stopping Criterion".

Notes on algorithm

- Initial values s_1, s_2
  Solving the following equation by Newton's method.

$$x^n + |a_1|x^{n-1} + ... + |a_{n-1}|x - |a_n| = 0$$

  let only one positive root be β, then s_1, and s_2 are set as:

$$s_1 = \beta(\cos\theta + i\sin\theta), s_2 = \beta(\cos\theta + i\sin\theta) \quad (4.18)$$

where theta $\theta \neq k\pi \propto k = 0, \pm 1, \pm 2, ...$

- Normalization of $K^{(\lambda)}(x)$
  To avoid overflows in operations, it is taken into account that $K^{(0)}(x) = f'(x)/n$, and $K^{(\lambda)}(x)$ are normalized by diving (4.6) with $A^{(\lambda)}$ in (4.7). (4.6) with $A^{(\lambda)}$ in (4.7).
- Calculation of $K^{(\lambda)}$(x) and $\delta^{(\lambda)}(x)$
  Let

$$f(x) = Q_f(x)\sigma(x) + b(x+u) + a,$$

$$K^{(\lambda)}(x) = Q_K^{(\lambda)}(x)\sigma(x) + d(x+u) + c.$$

  Then, using these *a*, *b*, *c*, *d*, *u*, *v*, $\boldsymbol{Q}_1(x)$ and $\boldsymbol{Q}_k^{(\lambda)}(x)$, $K^{(\lambda+1)}(x)$ is calculated as follows.

$$K^{(\lambda+1)}(x) = \left(\frac{a^2 + uab + vb^2}{bc - ad}\right) Q_k^{(\lambda)}(x) + \left(x - \frac{ac + uad + vbd}{bc - ad}\right) Q_f(x) + b$$

In addition, $\sigma^{(\lambda)}(x)$ is also calculated using *a*, *b*, *c*, *d*, *u*, and *v*, in the same way.

Stopping criterion

- Stage 1
  The purpose of this stage is to accentuate the smaller roots. In the implementain, *M* is set to 5, a number arrived at by numerical experience.

- Stage 2
  Letting $t_\lambda = -f(0)/\overline{K}^{(\lambda)}(0)$, if

$$|t_{\lambda+1} - t_\lambda| \leq \frac{1}{2}|t_\lambda| \text{ and } |t_{\lambda+2} - t_{\lambda+1}| \leq \frac{1}{2}|t_{\lambda+1}|$$

  then Stage 2 is terminated and iteration for a linear factor in Stage 3 is performed.

Letting $\sigma^{(\lambda)}(x) = x^2 + u\lambda x + v\lambda$, the sequence $\sigma^{(\lambda)}(x)$ is monitored by applying the same test to v_λ. If the condition is satisfied, Stage 2 is terminated and iteration for a quadratic factor in Stage 3 is performed. But if both t_λ and v_λ do not converge even after $20 \times I$ times iteration, s_1 and s_2 are reselected by rotationg θ through appropriate degree in (4.18). Where I is the number of times s_1 and s_2 are reselected and its maximum limit is 20. When I exceeds 20, processing is terminated with ICON = 10000.

- Stage 3
  (a) Iteration for a linear factor
  Letting $s = s^{(\lambda)}$, if
  $$\left|\sum_{i=0}^{n} a_i s^{n-i}\right| \le u \cdot \sum_{i=0}^{n} \left|a_i s^{n-i}\right|$$
  is satisfied, s is adopted as the root. Where u is the round-off unit.
  (b) Iteration for a quadratic factor
  Letting the quotient produced by dividing polynomial $f(x)$ by a quadratic factor $x^2 + u_\lambda x + v_\lambda$ be $\boldsymbol{Q}(x)$, and its remainder be

  $$B(x + u_\lambda) + A$$
  $$f(x) = (x^2 + u_\lambda x + v_\lambda)\,\boldsymbol{Q}(x) + B(x + u_\lambda) + A$$

  where B and A are calculated by synthetic division. When both B and A lose their significant digits, quadratic factor are judged to have converged. That it, letting

$$b_0 = a_0, \qquad c_0 = |a_0|$$
$$b_1 = a_1 - u_\lambda \cdot b_0, \qquad c_1 = \max\{|a_1|, |u_\lambda| \cdot c_0\}$$

and for $k = 2, ..., n-2$

$$b_k = a_k - u_\lambda \cdot b_{k-1} - v_\lambda \cdot b_{k-2},$$
$$c_k = \max\{|a_k|, |u_\lambda| \cdot c_{k-1}, |v_\lambda| \cdot c_{k-2}\}$$

then

$$B = b_{n-1} = a_{n-1} - u_\lambda \cdot b_{n-2} - v_\lambda \cdot b_{n-2} - v_\lambda \cdot b_{n-3},$$
$$A = b_n = a_n - u_\lambda \cdot b_{n-1} - v_\lambda \cdot b_{n-2}$$

and

$$D = c_{n-1} = \max\{|a_{n-1}|, |u_\lambda| \cdot c_{n-2}, |v_\lambda| \cdot c_{n-3}\},$$
$$C = c_n = \max\{|a_n|, |u_\lambda| \cdot c_{n-1}, |v_\lambda| \cdot c_{n-2}\}$$

are calculated. If

$$|B| \le D \cdot u \text{ and } |A| \le C \cdot u$$

(u is the round-off unit)

is satisfied, $x^2 + u_\lambda x + v_\lambda$ is adopted as a quadrate factor of $f(x)$. Limit number of iteration is 20 in both (a) and (b). If convergence is not met within the limit, the stopping condition met within the limit, the stopping condition in Stage 2 is made severe and the rest of Stage 2 is repeated.

For details, see References [30] and [31].

H11-20-0111 RKG, DRKG

A system of first order ordinary differential equations (Runge-Kutta-Gill method)
CALL RKG (Y, F, K, N1, H, M, SUB, VW, ICON)

Function

This subroutine solves a system of first order differential equations:

$$\left.\begin{array}{l} y_1' = f_1(x, y_1, y_2, ..., y_n), y_{10} = y_1(x_0) \\ y_2' = f_2(x, y_1, y_2, ..., y_n), y_{20} = y_2(x_0) \\ \quad : \qquad\qquad\qquad\qquad : \\ y_n' = f_n(x, y_1, y_2, ..., y_n), y_{n0} = y_n(x_0) \end{array}\right\} \quad (1.1)$$

$$n \geq 1$$

on a mesh $x_0 + h$, $x_0 + 2h$, ..., $x_0 + (m\text{-}1)h$ with initial values y_1 (x_0) y_2 (x_0) ... y_n (x_0), by the Runge-Kutta-Gill method.

Parameters

Y..... Input. Initial values x_0, y_{10}, y_{20}, ..., y_{n0} are specified as follows:
$Y(1,1) = x_0$, $Y(2,1) = y_{10}$, $Y(3,1) = y_{20}$, ..., $Y(N1, 1) = y_{n0}$
Y is two-dimensional array, Y(K,M).
Output. Values y_1, y_2, ..., y_n for $x_j = x_0 + jh$ (j=1, ..., m-1).
They are output as follows:
For J = 1, 2, ..., n–1
Y(1, J + 1): x_j
Y(2, J + 1): $y_{1j} = y_1(x_j)$
: :
Y(N1, J + 1): $y_{nj} = y_{nj}(x_j)$

F .... Output. Values of y'_1, y'_2, ..., y'_n for $x_j = x_0 + jh$ (j = 0, l, ..., m-1), i.e., the values of f_1, f_2, ..., f_n.
F is a two-dimensional array, F(K,M). They are output as follows:
Let
F(1, J + 1) :1.0
F(2, J + 1) : $y'_{1j} = y'_1(x_j)$
: :
F(N1, J + 1) : $y'_{nj}=y'_n(x_j)$

K .... Input. Adjustable dimension of arrays Y and F.

N1 ... Input. n + 1 where n is the number of equations in the system.

H.... Input. Stepsize h.

M.... Input. The number of discrete points of independent variable x at which approximations y_i (i=1, ..., n) are to be obtained. Starting point x_0 is also included in this number. $m \geq 2$.

SUB.. Input. The name of subroutine which evaluates f_i(i = 1, 2, ..., n) in (1.1).
The subroutine is provided by the user as follows:
SUBROUTINE SUB (YY,FF)
Parameters
YY: Input. One-dimensional array of size n + 1, where
YY(1) = x, YY(2) = y_1, YY(3)=y_2, ..., YY(n +1) = y_n
FF: Output. One-dimensional array of size n +1,
where FF(2) = f_1, FF(3) = f_2, FF(4) = f_3, ...,
FF(n +l) = f_n
(See the example).
Nothing must be substituted in FF(1).

VW.... Work area. One-dimensional array of size n+1.

ICON.. Output. Condition code. See Table RKG-1.

Table RKG-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N1 < 2, K < N1, M <2, or H = 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II... MGSSL
  FORTRAN basic function... None

- Notes
  SUB must be declared as EXTERNAL in the program from which this subroutine is called.
  This subroutine generates approximations at fixed intervals using a constant stepsize. As the computation proceeds, the errors tend to become larger in general; it is usually not advisable to obtain approximations at points very far from x_0 though it depends on the size of h.
  However, this subroutine has the advantage that it is a one step method, i.e., in determining the solution at $x_0 + jh$ only the solution at x_0 + (j - 1) h is necessary. This is suitable for calculating a few starting values in the multistep method, and should be exclusively used for that purpose.

- Example
  The initial value problem (3.1) of a system of differential equations is solved.

$$\left.\begin{array}{ll} y_1' = y_2 & , y_{10} = y_1(1) = 5 \\ y_2' = \dfrac{4}{x^2} y_1 + \dfrac{2}{x} y_2 & , y_{20} = y_2(1) = 3 \end{array}\right\} \quad (3.1)$$

h=0.1,m=10

```
C      **EXAMPLE**
       DIMENSION Y(3,10),F(3,10),VW(3)
       EXTERNAL SUB
       Y(1,1)=1.0
       Y(2,1)=5.0
       Y(3,1)=3.0
       H=0.1
       CALL RKG(Y,F,3,3,H,10,SUB,VW,ICON)
       IF(ICON.NE.0) STOP
       WRITE(6,600)
       WRITE(6,610) ((Y(I,J),I=1,3),
      * (F(I,J),I=2,3),J=1,10)
       STOP
   600 FORMAT('1'/' ',20X,'X',19X,'Y1',18X,
      * 'Y2',18X,'F1',18X,'F2'//)
   610 FORMAT(' ',10X,5E20.8)
       END
       SUBROUTINE SUB(YY,FF)
       DIMENSION YY(3),FF(3)
       FF(2)=YY(3)
       FF(3)=4.0*YY(2)/(YY(1)*YY(1))+2.0*
      *      YY(3)/YY(1)
       RETURN
       END
```

Method
Considering the independent variable x itself as a function;

$$y_0(x) = x, \quad y_{00} = y_0(x_0) = x_0 \tag{4.1}$$

initial value problem (1.1) of a system of first order differential equations, can be written as

$$\begin{aligned}
y_0' &= f_0(y_0, y_1, \dots, y_n), y_{00} = x_0 \\
y_1' &= f_1(y_0, y_1, \dots, y_n), y_{10} = y_1(x_0) \\
y_2' &= f_2(y_0, y_1, \dots, y_n), y_{20} = y_2(x_0) \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
y_n' &= f_n(y_0, y_1, \dots, y_n), y_{n0} = y_n(x_0)
\end{aligned}$$

To simplify the notation, the following vectors are introduced.

$$\begin{aligned}
\boldsymbol{y} &= (y_0, y_1, y_2, \dots, y_n)^{\mathrm{T}} \\
\boldsymbol{y}_0 &= (y_{00}, y_{10}, y_{20}, \dots, y_{n0})^{\mathrm{T}} \\
\boldsymbol{f}(\boldsymbol{y}) &= (f_0(\boldsymbol{y}), f_1(\boldsymbol{y}), \dots, f_n(\boldsymbol{y}))^{\mathrm{T}} \\
& \quad f_i(\boldsymbol{y}) = f_i(y_0, y_1, \dots, y_n)^{\mathrm{T}} \\
& \qquad\qquad i = 0,1,\dots,n
\end{aligned} \tag{4.3}$$

Let the vector which has elements y_i (i = 0, 1, ..., n) be represented as $\boldsymbol{y}'$. Then (4.2) can be simplified to

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}), \boldsymbol{y}_0 = \boldsymbol{y}(\boldsymbol{x}_0) \tag{4.4}$$

With respect to (4.4), the procedure of the Runge-Kutta-Gill method to approximate $\boldsymbol{y}(x_0 + h)$ is shown in (4.5) below. In the following expressions, $\boldsymbol{q}_i$ and $\boldsymbol{k}_i$ denote (n+ 1) dimensional vectors just as $\boldsymbol{y}_i$ and $\boldsymbol{f}_i$ Initially assumes a zero vector.

$$\begin{aligned}
\boldsymbol{k}_1 &= h\boldsymbol{f}(\boldsymbol{y}_0) \\
&\quad \boldsymbol{y}_1 = \boldsymbol{y}_0 + \frac{1}{2}(\boldsymbol{k}_1 - 2\boldsymbol{q}_0) \\
&\quad \boldsymbol{q}_1 = \boldsymbol{q}_0 + 3(\boldsymbol{y}_1 - \boldsymbol{y}_0) - \frac{1}{2}\boldsymbol{k}_1 \\
\boldsymbol{k}_2 &= h\boldsymbol{f}(\boldsymbol{y}_1) \\
&\quad \boldsymbol{y}_2 = \boldsymbol{y}_1 + \left(1 - \sqrt{\frac{1}{2}}\right)(\boldsymbol{k}_2 - \boldsymbol{q}_1) \\
&\quad \boldsymbol{q}_2 = \boldsymbol{q}_1 + 3(\boldsymbol{y}_2 - \boldsymbol{y}_1) - \left(1 - \sqrt{\frac{1}{2}}\right)\boldsymbol{k}_2 \\
\boldsymbol{k}_3 &= h\boldsymbol{f}(\boldsymbol{y}_2) \\
&\quad \boldsymbol{y}_3 = \boldsymbol{y}_2 + \left(1 + \sqrt{\frac{1}{2}}\right)(\boldsymbol{k}_3 - \boldsymbol{q}_2) \\
&\quad \boldsymbol{q}_3 = \boldsymbol{q}_2 + 3(\boldsymbol{y}_3 - \boldsymbol{y}_2) - \left(1 + \sqrt{\frac{1}{2}}\right)\boldsymbol{k}_3 \\
\boldsymbol{k}_4 &= h\boldsymbol{f}(\boldsymbol{y}_3) \\
&\quad \boldsymbol{y}_4 = \boldsymbol{y}_3 + \frac{1}{6}(\boldsymbol{k}_4 - 2\boldsymbol{q}_3) \\
&\quad \boldsymbol{q}_4 = \boldsymbol{q}_3 + 3(\boldsymbol{y}_4 - \boldsymbol{y}_3) - \frac{1}{2}\boldsymbol{k}_4
\end{aligned} \tag{4.5}$$

Then $\boldsymbol{y}_4$ is taken as the approximation to $\boldsymbol{y}(x_0 + h)$. When determining the solution at $x + 2h$, setting

$$\begin{aligned}
\boldsymbol{q}_0 &= \boldsymbol{q}_4 \\
\boldsymbol{y}_0 &= \boldsymbol{y}_4
\end{aligned}$$

,(4.5), is repeated. In this way, the approximations at $x_0 + 3h$,......,$x_0 + (m\text{-}1)\, h$, are determined.
For more information, see Reference [69].

C21-11-0101 RQDR, DRQDR

Zeros of a quadratic with real coefficients
CALL RQDR (A0, A1, A2, Z, ICON)

Function

This subroutine finds zeros of a quadratic with real coefficients;

$$a_0x^2 + a_1x + a_2 = 0 \quad (a_0 \neq 0, n \geq 1)$$

Parameters

A0, Al, A2...Input. Coefficients of the quadratic equation.
Z..... Output. Roots of the quadratic equation. Z is a complex one-dimensional array of size 2.
ICON.. Output. Condition code. See Table RQDR-1.

Table RQDR-l Condition codes

Code	Meaning	Processing
0	No error	
10000	$a_0 = 0.0$	$-a_2/a_1$ is stored in the real part of Z (1), and 0.0 is stored in the imaginary part. Z (2) may be incorrect.
30000	$a_0 = 0.0$ and $a_1 = 0.0$	Bypassed

Comments on use

- Subprograms used
  SSL II... AMACII and MGSSL
  FORTRAN basic functions ... CMPLX, SQRT, and ABS

- Example
  The coefficients of a quadratic equation are entered and roots Z, are determined.

```
C      **EXAMPLE**
       DIMENSION Z(2)
       COMPLEX Z
       READ(5,500) A0,A1,A2
       CALL RQDR(A0,A1,A2,Z,ICON)
       WRITE(6,600) ICON,A0,A1,A2
       IF(ICON.EQ.30000) STOP
       WRITE(6,610) (Z(I),I=1,2)
       STOP
   500 FORMAT(3F10.0)
   600 FORMAT(10X,'ICON=',I5/10X,'A=',3E15.6)
   610 FORMAT(10X,'Z=',2E15.6)
       END
```

Method

The roots of a quadratic equation ($a_0x^2 + a_1x + a_2 = 0$) can be obtained from

$$x = \frac{-P \pm \sqrt{P_1^2 - 4P_2}}{2}$$

where, $P_1 = a_1/a_0$ and $P_2 = a_2/a_0$

When $\left|P_1^2\right| >> \left|4P_2\right|$ a great loss of precision will result in one of the calculations $-P_1 + \sqrt{P_1^2 - 4P_2}$ or $-P_1 - \sqrt{P_1^2 - 4P_2}$. To avoid this problem, root formulas with rationalized numerators are used in calculations. These are shown below.

Let $D = P_1^2 - 4P_2$

For $D \leq 0$, (conjugate complex numbers, multiple roots)

$$\begin{aligned} x_1 &= -P_1/2 + i\sqrt{-D}/2 \\ x_2 &= -P_1/2 - i\sqrt{-D}/2 \end{aligned} \qquad (4.1)$$

For $D > 0$ (two real roots)
If $P_1 > 0$

$$\begin{aligned} x_1 &= -2P_2/(-P_1 + \sqrt{D}) \\ x_2 &= -(P_1 + \sqrt{D})/2 \end{aligned}$$

If $P_1 \leq 0$

$$\begin{aligned} x_1 &= (-P_1 + \sqrt{D})/2 \\ x_2 &= 2P_2/(-P_1 + \sqrt{D}) \end{aligned} \qquad (4.2)$$

In calculation of discriminant $D = P_1^2 - 4P_2$, if $|P_1|$ is very large, P_1^2 may cause an overflow. In order to avoid this situation, the condition $|P_1| > 10^{35}$ is checked. If the condition is satisfied, the above-described procedure is used. If the condition is satisfied, the roots are discriminated using $D_0 = 1 - 4P_2/P_1/P_1$ and $\sqrt{|D|}$ is calculated as $|P_1|\sqrt{|D_0|}$.

A52-31-0202 SBDL, DSBDL

LDL^T-decomposition of a positive-definite symmetric band matrix (Modified Cholesky's method)
CALL SBDL (A, N, NH, EPSZ, ICON)

Function

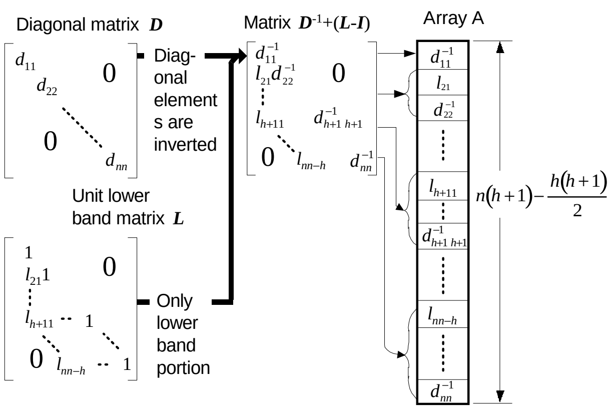
This subroutine computes LDL^T decomposition (1.1).

$$\boldsymbol{A} = \boldsymbol{LDL}^T \qquad (1.1)$$

of the $n \times n$ real positive-definite symmetric band matrix $\boldsymbol{A}$ with upper and lower band widths h by using the modified Cholesky's method, where $\boldsymbol{L}$ is a unit lower band matrix with band widths h, $\boldsymbol{D}$ is a diagonal matrix , and $n > h \geq 0$.

Parameters

A.....	Input. Matrix $\boldsymbol{A}$. Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$. Refer to Fig.SBDL-1. Matrix $\boldsymbol{A}$ is stored in a one dimensional array of size $n(h + 1)–h(h + 1)/2$ in the compressed mode for symmetric band matrices.
N.....	Input. Order n of matrix $\boldsymbol{A}$.
NH....	Input. Lower band width h .
EPSZ..	Input. Tolerance for relative zero test of pivots in decomposition process of matrix A ($\geq$ 0.0). If EPSZ is 0.0, a standard value is used.
ICON..	Output. Condition code. Refer to Table SBDL-1



Diagonal matrix $\boldsymbol{D}$
Matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$
Array A
Diag-onal elements are inverted
Unit lower band matrix $\boldsymbol{L}$
Only lower band portion
$n(h+1)-\frac{h(h+1)}{2}$

Note: On output, diagonal and lower band portions of matrix $D^{-1}+(L-I)$ are stored in one-dimensional array A in the compressed mode for symmetric band matrices.

Fig. SBDL-1 Storage of the decomposed elements

Table SBDL-l Condition codes

Code	Meaning	Processing
0	No error	
10000	The negative pivot occurred. The matrix is not positive-definite.	Continued
20000	The relatively zero pivot occurred. The matrix is possibly singular.	Discontinued
30000	NH<0, NH $\geq$ N or EPSZ < 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II... AMACH, MGSSL
  FORTRAN basic function... ABS

- Notes
  Since this subroutine omits the operations concerning the elements out of the band, the processing speed is faster than subroutine SLDL provided for positive-definite symmetric matrices.
  If EPSZ is set to 10^{-s}, this value has the following meaning: While performing the LDL^T -decomposition by modified Cholesky's method, if the loss of over s significant digits occurred for the pivot, the LDL^T-decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero.
  The standard value of EPSZ is $16 \cdot u$, where u is a unit round off, but the result is not always guaranteed.
  If the negative pivot occurred in the decomposition, the matrix is not a positive-definite.
  In this case, this subroutine is continued, with ICON = 10000. However, it should be noted that large calculation errors may occur since the pivoting is not performed.
  This subroutine performs LDL^T decomposition, but it should be noted that $\boldsymbol{D}^{-1}$ is output to the array instead of $\boldsymbol{D}$.
  The determinant of the matrix can be obtained by multiplying all the n diagonal elements of array A (the diagonal elements of $\boldsymbol{D}^{-1}$) after the subroutine has been executed and then by determining the inverse number.
  Notice that the array A is in the compressed mode for symmetric band matrices.

- Example
  The $n \times n$ matrix with the lower and upper band width h is input and LDL^T decomposition is computed.
  $n \leq 100$ and $h \leq 50$.

```
C     **EXAMPLE**
      DIMENSION A(3825)
   10 READ(5,500) N,NH
      IF(N.EQ.0) STOP
      NH1=NH+1
      NT=N*NH1-NH*NH1/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,630)
      L=0
      LS=1
      DO 20 I=1,N
      L=L+MIN0(I,NH1)
      JS=MAX0(1,I-NH1)
      WRITE(6,600) I,JS,(A(J),J=LS,L)
   20 LS=L+1
      CALL SBDL(A,N,NH,1.0E-6,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,640)
      L=0
      LS=1
      DET=1.0
      DO 30 I=1,N
      L=L+MIN0(1,NH1)
      JS=MAX0(1,I-NH1)
      WRITE(6,600) I,JS,(A(J),J=LS,L)
      DET=DET*A(L)
   30 LS=L+1
      DET=1.0/DET
      WRITE(6,620) DET
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(4E15.7)
  600 FORMAT(' ','(',I3,',',I3,')'/
     * (10X,5E17.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(//10X,
     * 'DETERMINANT OF MATRIX=',E17.8)
  630 FORMAT(/10X,'INPUT MATRIX')
  640 FORMAT(/10X,'DECOMPOSED MATRIX')
      END
```

Method

- Modified Cholesky's method

  A real positive-definite symmetric matrix ***A*** can always be decomposed in the following form.

$$\boldsymbol{A} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^{\mathrm{T}} \qquad (4.1)$$

Where $\tilde{\boldsymbol{L}}$ is a lower triangular matrix. Further, if ***L*** is defined by $\tilde{\boldsymbol{L}} = \boldsymbol{L}$ diag(T_{ii} equations (4.1) can be rewritten to

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \qquad (4.2)$$

Where ***L*** is a unit lower triangular matrix, and ***D*** is a positive-definite diagonal matrix. The modified Cholesky's method gives the following equations to decompose as shown in equations (4.2).

$$l_{ij}d_j = a_{ij} - \sum_{k=1}^{j-1} l_{ik}d_k l_{jk}, j = 1,...,i-1 \qquad (4.3)$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}d_k l_{jk} \qquad (4.4)$$

Where $i = 1, ..., n$.

(For further details, see the Method for subroutine SLDL). If matrix ***A*** is a positive-definite symmetric band matrix, in equations (4.2), matrix ***A*** becomes a lower band matrix, with the identical band width as ***A***. Therefore, in this subroutine, the calculation concerning the elements out of the band are omitted, and the decomposition is actually computed using the following equations (4.5) and (4.6).

$$l_{ij}d_j = a_{ij} - \sum_{k=\max(1,\, i-h)}^{j-1} l_{ik}d_k l_{jk},$$
$$j = i-h,...,n \qquad (4.5)$$

$$d_i = a_{ii} - \sum_{k=\max(1,\, i-h)}^{j-1} l_{ik}d_k l_{ik}, \qquad (4.6)$$

Where $i = 1, ..., n$.

For further details, see Reference [7].

A52-21-0202 SBMDM, DSBMDM

MDM^T decomposition of a real indefinite symmetric band matrix (block diagonal pivoting method)
CALL SBMDM (A, N, NH, MH, EPSZ, IP, IVW, ICON)

Function

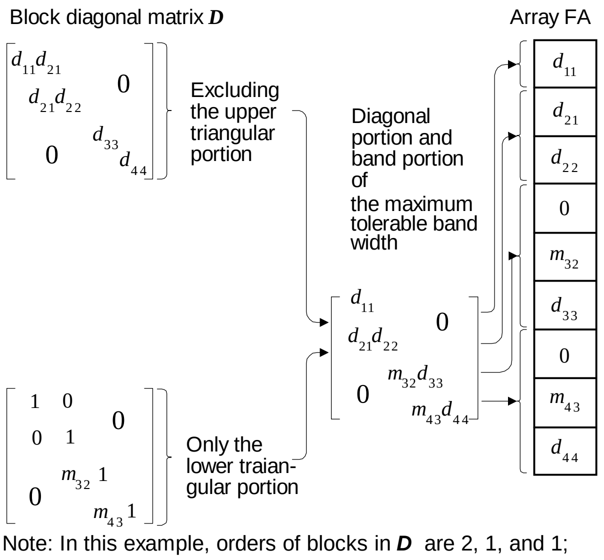
An $n \times n$ nonsingular real symmetric band matrix $\boldsymbol{A}$ having band width $\tilde{h}$ is MDM^T-decomposed using the block diagonal pivoting method. (There are similar two methods, Crout-like and Gaussian-like methods, of which the Gaussian-like method is used in this subroutine.)

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \qquad (1.1)$$

where $\boldsymbol{P}$ is a permutation matrix to be used to exchange rows of matrix $\boldsymbol{A}$ in the pivoting operation, $\boldsymbol{M}$ is a unit lower band matrix, and $\boldsymbol{D}$ is a symmetric block diagonal matrix comprising symmetric blocks each at most of order 2; further $d_{k+1,\,k} \neq 0$ if $m_{k+1,k}=0$,where $\boldsymbol{M} = (m_{ij})$ and $\boldsymbol{D} = (d_{ij})$ for $n \geq h \geq 0$.

Parameters

A..... Input. Matrix $\boldsymbol{A}$ given in compressed mode for the symmetric band matrix assuming $\boldsymbol{A}$ to have band width h_m. (See "Comments on Use.")
Output. Matrices $\boldsymbol{M}$ and $\boldsymbol{D}$. (See Figure SBMDM-l.)
One-dimensional array of size $n(h_m+1)- h_m(h_m+1)/2$


Block diagonal matrix D
Array FA
Excluding the upper triangular portion
Diagonal portion and band portion of the maximum tolerable band width
Only the lower traiangular portion
d_{11}
d_{21}
d_{22}
0
m_{32}
d_{33}
0
m_{43}
d_{44}

Note: In this example, orders of blocks in $\boldsymbol{D}$ are 2, 1, and 1; band width of $\boldsymbol{M}$ is 1, and the maximum tolerable band width is 2.

Fig.SBMDM-1 Decomposed element storing method

N..... Input. Order n of matrix $\boldsymbol{A}$.

NH.... Input. Band width h of matrix $\boldsymbol{A}$.
Output. Band width $\tilde{h}$ of matrix $\boldsymbol{M}$. (See "Comments on Use.")

MH..... Input. Maximum tolerable band width h_m(N > MH ≥ NH).(See "Comments on Use.")

EPSZ .. Input. Relative zero criterion (≥ 0.0) for pivoting operation. The default value is used if 0.0 is specified. (See, "Comments on Use.")

IP..... Output. Transposition vector indicating the history of row exchanges by pivoting operation. One-dimensional array of size n.
(See "Comments on Use.")

IVW..... Work area. One-dimensional array of size n.

ICON.. Output. Condition code. (See Table SBMDM-l.)

Table SBMDM-l Condition codes

Code	Meaning	Processing
0	No error.	
20000	The relatively zero pivot occurred. The coefficient matrix may be nonsingular.	Bypassed.
25000	The band width exceeded the maximum tolerable band width during processing.	Bypassed.
30000	NH < 0, NH > MH, MH ≥ N or EPSZ <0.0.	Bypassed.

Comments on use

- Subprograms used
  SSL II... AMACH, MGSSL
  FORTRAN basic functions... MAX0, MIN0, ABS, AMAX1

- Notes
  When 10^{-s} is set as relative zero criterion EPSZ for pivoting operation:
  If loss of significant digits exceeds decimals in the pivot (determinant of 1×1 or 2×2 pivot matrix) during the MDM^T decomposition in the block diagonal pivoting method, the pivot value is regarded as a relative zero and ICON = 20000 is set, then processing is stopped.
  The default value of EPSZ is $16 \cdot u$, where u is the unit round off.
  If the calculation must not be stopped even if the pivot value becomes small, specify a very small value in the EPSZ parameter. In this case, however, the result is not guaranteed. result is not guaranteed.

The transposition vector is considered to be analogues to permutation matrix $\boldsymbol{P}$ in

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}}$$

used in the $\mathrm{MDM}^{\mathrm{T}}$ decomposition in the block diagonal pivoting method. In this subroutine, contents of array $\boldsymbol{A}$ are actually exchanged and the related historical data is stored in IP. Note that data storing methods are different between 1×1 and 2×2 pivot matrices. At step k of the decomposition, data is stored as follows:

For 1×1 pivot matrix, no row is exchanged and k is stored in IP(k). For 2×2 pivot matrix, $-k$ is stored in IP(k)and the negative value of the row (and column) number that is exchanged by row (k + 1) (and column (k + l)) is stored in IP(k+1).

The determinant of matrix $\boldsymbol{A}$ is equal to that of calculated $\boldsymbol{D}$. The elements of matrices $\boldsymbol{M}$ and $\boldsymbol{D}$ are stored in array $\boldsymbol{A}$ (Figure SBMDM-l). (See “Example” for subroutine LSBIX.)

Generally, the matrix band width increases when rows and columns are exchanged in the pivoting operation. This means that the user must specify band width h_m greater than actual band width h . If the band width exceeds h_m , processing is stopped assuming ICON = 25000. The output value for NH indicates the necessary and sufficient condition for h_m

Simultaneous linear equations are solved by calling subroutine BMDMX after this subroutine. In an ordinary case, the solution is obtained by the calling subroutine LSBIX.

The numbers of positive and negative eigenvalues of matrix $\boldsymbol{A}$ can be obtained. (See “Example.”)

- Example

Given an $n \times n$ real symmetric band matrix having width h, the numbers of positive and negative eigenvalues are obtained under conditions $n \le 100$ and $h \le h_m \le 50$.

```
C      **EXAMPLE**
       DIMENSION A(3825),IP(100),IVW(100)
       READ(5,500) N,NH,MH
       WRITE(6,600) N,NH,MH
       MHP1=MH+1
       NT=(N+N-MH)*MHP1/2
       READ(5,510) (A(J),J=1,NT)
       EPSZ=0.0
       CALL SBMDM(A,N,NH,MH,EPSZ,IP,IVW,
      *           ICON)
       WRITE(6,610) ICON
       IF(ICON.GE.20000) STOP
       INEIG=0
       IPEIG=0
       I=1
       J=1
   10  M=IP(J)
       IF(M.EQ.J) GO TO 20
       IPEIG=IPEIG+1
       INEIG=INEIG+1
       I=MIN0(MH,J)+MIN0(MH,I+1)+2+I
       J=J+2
       GO TO 30
   20  IF(A(I).GT.0.0) IPEIG=IPEIG+1
       IF(A(I).LT.0.0) INEIG=INEIG+1
       I=MIN0(MH,J)+1+I
       J=J+1
   30  IF(J.LE.N) GO TO 10
       WRITE(6,620) IPEIG,INEIG
       STOP
  500  FORMAT(3I4)
  510  FORMAT(4E15.7)
  600  FORMAT('1'/10X,'N=',I3,5X,'NH=',I3,5X,
      *'MH=',I3)
  610  FORMAT(/10X,'ICON=',I5)
  620  FORMAT('0',5X,'POSITIVE EIGENVALUE=',
      * I4/6X,'NEGATIVE EIGENVALUE=',I4)
       END
```

Method

- Block diagonal pivoting method

A positive-definite symmetric matrix A can be decomposed as (4.1) using the modified Cholesky method:

$$\boldsymbol{A} = \boldsymbol{M}_1\boldsymbol{D}_1\boldsymbol{M}_1^{\mathrm{T}} \qquad (4.1)$$

where $\boldsymbol{M}_1$ is a unit lower triangular matrix and $\boldsymbol{D}_1$ is a diagonal matrix.

If A is not a positive-definite matrix, this decomposition is generally impossible or numerically unstable even if it is possible; however, this difficulty is resolved by rearranging (4.1) to (4.2):

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \qquad (4.2)$$

where $\boldsymbol{P}$ is the permutation matrix to be used for exchanging rows in the pivoting operation, $\boldsymbol{M}$ is a unit lower triangular matrix, and $\boldsymbol{D}$ is a symmetric matrix comprising symmetric blocks each at most of order 2. Decomposition to the form of (4.2) is referred to as the block diagonal pivoting method.

- Procedures in this subroutine

This subroutine uses Algorithm $\boldsymbol{D}$ (references [9] and [10]) to minimize increase of the band width caused by exchanging rows (and columns) in the pivoting operation. In Algorithm $\boldsymbol{D}$, elements of the coefficient matrix are updated at each decomposition step. The coefficient matrix obtained at step k completion is expressed as $\boldsymbol{A}^{(k)} = (a_{ij}^{(k)})$, where $\boldsymbol{A}^{(0)}=\boldsymbol{A}$. In this case, processing at step k (k=l,2,..., n) consists of

1) Determines

$$\rho = \left|a_{mk}^{(k-1)}\right| = \max_{k<i\leq n}\left|a_{ik}^{(k-1)}\right| \tag{4.3}$$

2) If

$$\left|a_{kk}^{(k-1)}\right| \geq \alpha\rho, (\alpha = 0.525) \tag{4.4}$$

is satisfied, proceeds to step 5), otherwise, proceeds to step 3).

3) Determines

$$\sigma = \max_{k<j\leq n}\left|a_{mj}^{(k-1)}\right| \tag{4.5}$$

4) If

$$\alpha\rho^2 \leq \left|a_{kk}^{(k-1)}\right|\sigma \tag{4.6}$$

is satisfied, proceeds to step 5), otherwise, proceeds to step 6).

5) If

$$\left|a_{kk}^{(k-1)}\right| < \varepsilon, \left(\varepsilon = \max\left|a_{ij}\right| \cdot EPSZ\right) \tag{4.7}$$

is satisfied, the matrix is considered to be singular, then processing is stopped; otherwise, $a_{kk}^{(k-1)}$ is used as 1×1 pivot matrix to calculate

$$s = a_{ik}^{(k-1)} / a_{kk}^{(k-1)}, \tag{4.8}$$

$$a_{ji}^{(k)} = a_{jk}^{(k-1)} \cdot s + a_{ji}^{(k)}, j = i,...,n$$

$$a_{ij}^{(k)} = -s \tag{4.9}$$

$$, i = k+1,...,n$$

Increments k by one, then procees to step 1).

6) Exchange row (and column) $k + 1$ and m. If

$$\left|a_{k+1,k}^{(k-1)}\right| < \varepsilon$$

is satisfied, the matrix is considered to be singular, then processing is stopped; otherwise,

$$\begin{pmatrix} a_{kk}^{(k-1)} & a_{k,k+1}^{(k-1)} \\ a_{k+1,k}^{(k-1)} & a_{k+1,k+1}^{(k-1)} \end{pmatrix}$$

is used as the 2×2 pivot matrix to calculate

$$r = a_{k,k}^{(k-1)} / a_{k+1,k}^{(k-1)} \tag{4.12}$$

$$d = r \cdot a_{k+1,k+1}^{(k-1)} / a_{k+1,k}^{(k-1)} - a_{k+1,k}^{(k-1)} \tag{4.13}$$

$$s = \left(a_{i,k+1}^{(k-1)} - a_{i,k}^{(k-1)} \cdot a_{k+1,k+1}^{(k-1)} / a_{k+1,k}^{(k-1)}\right) / d \tag{4.14}$$

$$t = \left(a_{ik}^{(k-1)} - r a_{i,k+1}^{(k-1)}\right) / d \tag{4.15}$$

$$a_{ji}^{(k+1)} = s \cdot a_{jk}^{(k-1)} + t \cdot a_{j,k+1}^{(k-1)} + a_{ji}^{(k-1)} \tag{4.16}$$

$$, j = i,...,n$$

$$a_{ik}^{(k+1)} = -s, \tag{4.17}$$

$$a_{i,k+1}^{(k+1)} = -t \tag{4.18}$$

$$i = k+2,...,n$$

Increments k by 2, then proceeds to step 1).

When decomposition is completed, the diagonal block portion of $\boldsymbol{A}^{(n)}$ is $\boldsymbol{D}$ and the other portions are $\boldsymbol{M}$.

The band structure of the coefficient matrix has been ignored for simplifying explanations above, however, the band structure is used to efficiently process calculations in the actual program.

Generally, the band width increases when a 2×2 pivot matrix is used. This subroutine reduces unnecessary calculations by exactly tracing the band width change.

(See references [9] and [10] for details.)

B21-21-0101 SEIG1, DSEIG1

Eigenvalues and corresponding eigenvectors of a real symmetric matrix (QL method)
CALL SEIG1 (A, N, E, EV, K, M, VW, ICON)

Function

All eigenvalues and corresponding eigenvectors of an *n*-order real symmetric matrix ***A*** are determined using the QL method. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$. $n \geq 1$.

Parameters

A.....	Input. Real symmetric matrix ***A***. Compressed storage mode for symmetric matrix. A is a one-dimensional array of size *n* (*n*+1)/2. The contents of A are altered on output.
N.....	Input. Order *n* of matrix ***A***.
E..	Output. Eigenvalues. E is a one-dimensional array of size *n*.
EV......	Output. Eigenvectors. Eigenvectors are stored in columns of EV. EV(K,N) is a two-dimensional array.
K.....	Input. Adjustable dimension of array EV. (≥ *n*)
M.....	Output. Number of eigenvalues/eigenvectors obtained.
VM....	Work area. VW is a one-dimensional array of size 2*n*.
ICON..	Output. Condition code See Table SEIG1-1.

Table SEIG1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	E(1) = A(1), EV (1, 1) = 1.0
15000	Some of eigenvalues and eigenvectors could not be determined.	M is set to the number of eigenvalues and eigenvectors that were determined.
20000	None of eigenvalues and eigenvectors could be determined.	M = 0
30000	N < 1 or K < N	Bypassed

Comments on use

- Subprograms used
  SSL II... TRID1, TEIG1, TRBK, AMACH, and MGSSL.
  FORTRAN basic functions... SQRT, SIGN, ABS, and DSQRT

- Notes
  All eigenvalues and corresponding eigenvectors are stored in the order that eigenvalues are determined.
  Parameter M is set to *n* when ICON = 0, when ICON = 15000, parameter M is set to the number of eigenvalues and corresponding eigenvectors that were obtained.
  This subroutine is used for a real symmetric matrix. When determining all eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix, subroutine TEIG1 should be used.
  If only the eigenvalues of a real symmetric matrix are to be determined, subroutines TRID1 and TRQL should be used.

- Example
  All eigenvalues and corresponding eigenvectors of an *n*-order real symmetric matrix ***A*** are determined. *n* ≤ 100.

```
C     **EXAMPLE**
      DIMENSION A(5050),E(100),EV(100,100),
     *          VW(200)
   10 CONTINUE
      READ(5,500) N
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1, NN)
      WRITE(6,600) N
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      CALL SEIG1(A,N,E,EV,100,M,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL SEPRT(E,EV,100,N,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',20X,'ORIGINAL MATRIX',15X,
     *       'ORDER=',I3/'0')
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

This example, subroutine SEPRT is used to print the eigenvalues and corresponding eigenvectors of a real symmetric matrix. The contents of SEPRT are:

```
      SUBROUTINE SEPRT(E,EV,K,N,M)
      DIMENSION E(M),EV(K,M)
      WRITE(6,600)
      KAI=(M-1)/5+1
      LST=0
      DO 10 KK=1,KAI
      INT=LST+1
      LST=LST+5
      IF(LST.GT.M) LST=M
      WRITE(6,610) (J,J=INT,LST)
      WRITE(6,620) (E(J),J=INT,LST)
      DO 10 I=1,N
      WRITE(6,630) I,(EV(I,J),J=INT,LST)
   10 CONTINUE
      RETURN
  600 FORMAT('1',20X,
     *      'EIGENVALUE AND EIGENVECTOR')
  610 FORMAT('0',5I20)
  620 FORMAT('0',5X,'ER',3X,5E20.8/)
  630 FORMAT(5X,I3,3X,5E20.8)
      END
```

Method

All eigenvalues and corresponding eigenvectors of an *n*-order real symmetric matrix $\boldsymbol{A}$ are determined.

Using the orthogonal similarity transformation in (4.1), real symmetric matrix $\boldsymbol{A}$ can be reduced to diagonal matrix $\boldsymbol{D}$.

$$\boldsymbol{D} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Q} \tag{4.1}$$

Where $\boldsymbol{Q}$ is an orthogonal matrix. Diagonal elements of diagonal matrix $\boldsymbol{D}$ obtained in (4.1) become all eigenvalues of real symmetric matrix $\boldsymbol{A}$; and the *i*-th column of $\boldsymbol{Q}$ is the eigenvector which corresponds to *i*-th diagonal element of $\boldsymbol{D}$. In this routine, eigenvalues and eigenvectors ($\boldsymbol{Q}$) are determined as follows.

- Using the Householder method, real symmetric matrix $\boldsymbol{A}$ is reduced to tridiagonal matrix $\boldsymbol{T}$.

$$\boldsymbol{T} = \boldsymbol{Q}_H^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Q}_H \tag{4.2}$$

where $\boldsymbol{Q}_H$ is an orthogonal matrix obtained as the product of transformation matrices in the Householder method.

$$\boldsymbol{Q}_H = \boldsymbol{P}_1\boldsymbol{P}_2 \cdots \cdot \boldsymbol{P}_{n-2} \tag{4.3}$$

$\boldsymbol{T}$ is obtained using subroutine TRID 1.

- $\boldsymbol{Q}_H$ is computed from (4.3).
- Using the QL method, tridiagonal matrix $\boldsymbol{T}$ is reduced to diagonal matrix $\boldsymbol{D}$ to determine the eigenvalues. For information on the QL method, see the section on TRQL. This transformation is

$$\boldsymbol{D} = \boldsymbol{Q}_L^{\mathrm{T}}\boldsymbol{T}\boldsymbol{Q}_L \tag{4.4}$$

$\boldsymbol{Q}_L$ is an orthogonal matrix obtained as the product of transformation matrices in the QL method.

$$\boldsymbol{Q}_L = \boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \cdot \boldsymbol{Q}_s \tag{4.5}$$

From (4.1), (4.2), and (4.4), eigenvector $\boldsymbol{Q}$ can be represented as

$$\boldsymbol{Q} = \boldsymbol{Q}_H\boldsymbol{Q}_L \tag{4.6}$$

By performing the transformation of (4.4) and the computation of (4.6) at the same time, all eigenvalues and corresponding eigenvectors can be obtained together. This is done by subroutine TEIG1.

The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$. For further information see References [12], [13] pp 191-195, [13] pp.212-248, and [16] pp.177-206.

B21-21-0201 SEIG2, DSEIG2

Selected eigenvalues and corresponding eigenvectors of a real symmetric matrix (Bisection method, inverse iteration method)
CALL SEIG2 (A, N, M, E, EV, K, VW, ICON)

Function

The m largest or m smallest eigenvalues of an n-order real symmetric matrix $\boldsymbol{A}$ are determined using the bisection method. Then the corresponding eigenvectors are determined using the inverse iteration method. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$. $1 \le m \le n$.

Parameters

A.....	Input. Real symmetric matrix $\boldsymbol{A}$ Compressed storage mode for symmetric matrix. A is a one-dimensional array of size $n(n+1)/2$. The contents of A are altered on output.
N.....	Input. Order n of real symmetric matrix $\boldsymbol{A}$.

M.....	Input.
	M = + m ... The m largest eignvalues desired
	M = – m ... The m smallest eigenvalues desired

E.....	Output. Eigenvalues. E is a one-dimensional array of size m .
EV	Output. Eigenvectors. Eigenvectors are stored in columns of EV. EV(K,M) is a two-dimensionable array
K.....	Input. Adjustable dimension of array EV. ($\ge n$)
VW.....	Work area. One-dimensional array of size $7n$.
ICON ..	Output. Condition code. See Table SEIG2-1.

Table SEIG2-l Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	E(1) = A(1), EV(1, 1) = 1.0
15000	Some of eigenvectors could not be determined although m eigenvalues were determined.	The eigenvector is treated as vector 0.
20000	The eigenvector could not be determined.	The eigenvector is treated as vector 0.
30000	M = 0, N < |M| or K < N	Processing is bypassed.

Comments on use

- Subprograms used
  SSL II... TRID1, TEIG2, TRBK, AMACH, UTEG2 and MGSSL.
  FORTRAN basic functions ... IABS, SQRT, SIGN, ABS, AMAX1 and DSQRT.

- Notes
  This subroutine is used for real symmetric matrices. When m eigenvalues/eigenvectors of a real symmetric tridiagonal matrix are to be determined, subroutine TEIG2 should be used.
  When determining m eigenvalues of a real symmetric matrix without the corresponding eigenvectors, subroutines TRID1 and BSCT1 should be used.

- Example
  The m largest or m smallest eigenvalues and corresponding eigenvectors of an n-order real symmetric matrix $\boldsymbol{A}$ are determined. $n \le 100$, $m \le 10$.

```
C     **EXAMPLE**
      DIMENSION A(5050),E(10),
     *          EV(100,10),VW(700)
   10 CONTINUE
      READ(5,500) N,M
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N,M
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
      WRITE(6,610) I,(A(J),J=NI,NE)
   20 CONTINUE
      CALL SEIG2(A,N,M,E,EV,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,100,N,MM)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',20X,'ORIGINAL MATRIX',5X,
     *       'N=',I3,5X,'M=',I3/'0')
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',20X,'ICON=',I5)
      END
```

In this example, subroutine SEPRT is used to print eigenvalues and corresponding eigenvectors of the real symmetric matrix. For detail on this subroutine, refer to the example in section SEIG1.

Method

The m largest or m smallest eigenvalues of an n order real symmetric matrix $\boldsymbol{A}$ are determined using the bisection method. Then, the corresponding eigenvectors are determined using the inverse iteration method.

First, real symmetric matrix $\boldsymbol{A}$ is reduced to tridiagonal matrix $\boldsymbol{T}$ using the Householder method:

$$\boldsymbol{T} = \boldsymbol{Q}_{\mathrm{H}}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{Q}_{\mathrm{H}} \tag{4.1}$$

where $\boldsymbol{Q}_H$ is an orthogonal matrix.

This is done by subroutine TRID1.

Next, m eigenvalues are determined using the bisection method. Then, corresponding eigenvectors of $\boldsymbol{T}$ are determined using the inverse iteration method, which determines eigenvectors by solving equation (4.2) iteratively.

$$(\boldsymbol{T} - \mu \boldsymbol{I})\boldsymbol{x}_r = \boldsymbol{x}_{r-1}, r = 1, 2, \ldots \tag{4.2}$$

Where μ is an eigenvalue determined using the bisection method, and $\boldsymbol{x}_0$ is an appropriate initial vector. The subroutine TEIG2 performs this operation.

Let eigenvectors of $\boldsymbol{T}$ be $\boldsymbol{y}$, then eigenvectors $\boldsymbol{x}$ of $\boldsymbol{A}$ are obtained using $\boldsymbol{Q}_H$ in (4.1) as

$$\boldsymbol{x} = \boldsymbol{Q}_H \boldsymbol{y} \tag{4.3}$$

which is back transformation corresponding the Householder's reduction. This is done by subroutine TRBK. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2 = 1$. For further information, see References [12] and [13] pp 418-439.

I11-51-0101 SFRI, DSFRI

Sine Fresnel integral $S(x)$
CALL SFRI (X, SF, ICON)

Function

This subroutine computes Sine Fresnel integral

$$S(x)=\frac{1}{\sqrt{2\pi}}\int_0^x \frac{\sin(t)}{\sqrt{t}}dt=\int_0^{\sqrt{\frac{2}{\pi}x}}\sin\left(\frac{\pi}{2}t^2\right)dt$$

by series and asymptotic expansions, where $x \geq 0$.

Parameters

X.....	Input. Independent variable x.
SF......	Output. Value of $S(x)$.
ICON..	Output. Condition code. See Table SFRI-1.

Table SFRI-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$X \geq t_{max}$	SF = 0.5
30000	X < 0	SF = 0.0

Comments on use

- Subprograms used
  SSL II... MGSSL, UTLIM
  FORTRAN basic functions ... SIN, COS, and SQRT

- Notes
  The valid ranges of parameter X are:
  $0 \leq X < t_{max}$
  This is provided because sin(x) and cos(x) lose their accuracy if X exceeds the above ranges.

- Example
  The following example generates a table of $S(x)$ from 0.0 to 100.0 with increment 1.0.

```
C      **EXAMPLE**
       WRITE(6,600)
       DO 10 K=1,101
       X=K-1
       CALL SFRI(X,SF,ICON)
       IF(ICON.EQ.0) WRITE(6,610) X,SF
       IF(ICON.NE.0) WRITE(6,620) X,SF,ICON
   10 CONTINUE
       STOP
  600 FORMAT('1','EXAMPLE OF FRESNEL ',
     * 'INTEGRAL',///6X,'X',9X,'S(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'S=',E17.7,5X,'CONDITION=',
     * I10)
       END
```

Methods

Two different approximation formulas are used depending on the ranges of x divided at $x = 4$.

- For $0 \leq x < 4$
  The power series expansion of $S(x)$

$$S(x)=\sqrt{\frac{2}{\pi}}x\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!\,(4n+3)}x^{2n+1} \quad (4.1)$$

is calculated with the following approximation formulas:

Single precision:

$$S(x)=\frac{x\sqrt{x}}{2}\sum_{k=0}^{6}a_k z^{2k}, z=x/4 \quad (4.2)$$

Double precision:

$$S(x)=\sqrt{x}\sum_{k=0}^{11}a_k x^{2k+1} \quad (4.3)$$

- For $x \geq 4$
  The asymptotic expansion of $S(x)$

$$S(x)=\frac{1}{2}+\sin(x)P(x)+\cos(x)Q(x) \quad (4.4)$$

is calculated through use of the following approximate expressions of $P(x)$ and $Q(x)$:

Single precision:

$$P(x)=\frac{2}{\sqrt{x}}\sum_{k=0}^{10}a_k z^{k+1}, z=4/x \quad (4.5)$$

$$Q(x)=-\frac{2}{\sqrt{x}}\sum_{k=0}^{11}a_k z^{k}, z=4/x \quad (4.6)$$

Double precision:

$$P(x)=\frac{1}{\sqrt{x}}\sum_{k=0}^{10}a_k z^{k+1}\Bigg/\sum_{k=0}^{11}b_k z^k, z=4/x \quad (4.7)$$

$$Q(x)=\frac{1}{\sqrt{x}}\sum_{k=0}^{10}c_k z^{k}\Bigg/\sum_{k=0}^{10}d_k z^k, z=4/x \quad (4.8)$$

A21-11-0201 SGGM, DSGGM

Subtraction of two matrices (real general)
CALL SGGM (A, KA, B, KB, C, KC, M, N, ICON)

Function

This subroutine performs subtraction of an $m \times n$ real general matrix $\boldsymbol{B}$ from a real general matrix $\boldsymbol{A}$.

$$\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$$

where $\boldsymbol{C}$ is an $m \times n$ real general matrix. $m, n \geq 1$.

Parameters

A.........	Input. Matrix $\boldsymbol{A}$, two-dimensional array, A(KA, N).
KA.......	Input. The adjustable dimension of array A, (≥ M).
B..........	Input. Matrix $\boldsymbol{B}$, two-dimensional array, B(KB, N).
KB........	Input. The adjustable dimension of array B, (≥ N).
C...........	Output. Matrix $\boldsymbol{C}$, two-dimensional array C(KC, N). (See Notes.)
KC.........	Input. The adjustable dimension of array C, (≥ M).
M..........	Input. The number of rows m of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.
N...........	Input. The number of columns n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.
ICON ...	Output. Condition code. See Table SGGM-1.

Table SGGM-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	M<1, N<1, KA < M, KB < M or KC < M	Bypassed

Comments on use

- Subprograms used
  SSL II... MGSSL
  FORTRAN basic function ... None

- Notes
  Saving the storage area:
  If there is no need to keep the contents on array A or B, more storage area can be saved by the following CALL statement.
  – When the contents of array A are not needed:
    CALL SGGM (A, KA, B, KB, A, KA, M, N, ICON)
  – When the contents of array B are not needed:
    CALL SGGM (A, KA, B, KB, B, KB, M, N, ICON)
    In the above two cases, matrix $\boldsymbol{C}$ is stored in array A or B.

- Example
  The following shows an example of obtaining the subtraction of a real general matrix $\boldsymbol{B}$ from $\boldsymbol{A}$. Here, $m, n \leq 50$.

```
C     **EXAMPLE**
      DIMENSION A(50,50),B(60,60),C(100,100)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
      DATA KA/50/,KB/60/,KC/100/
   10 READ(5,100) M,N
      IF(M.EQ.0) STOP
      WRITE(6,150)
      READ(5,200) ((A(I,J),I=1,M),J=1,N)
      READ(5,200) ((B(I,J),I=1,M),J=1,N)
      CALL SGGM(A,KA,B,KB,C,KC,M,N,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PGM(IA,1,A,KA,M,N)
      CALL PGM(IB,1,B,KB,M,N)
      CALL PGM(IC,1,C,KC,M,N)
      GOTO 10
  100 FORMAT(2I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX ADDITION **')
      END
```

The subroutine PGM in the example is for printing a real matrix. This program is shown in the example for subroutine MGSM.

G21-11-0101 SIMP1, DSIMP1

Integration of a tabulated function by Simpson's rule (equally spaced)
CALL SIMP1 (Y, N, H, S, ICON)

Function

Given function values $y_i = f(x_i)$ at equally spaced points $x_i = x_1 + (i\text{-}1)\ h$, i=1,..., n this subroutine obtains the integral:

$$S = \int_{x_1}^{x_n} f(x)dx \quad n \geq 3, h > 0.0$$

by Simpson's rule, where h is the increment.

Parameters

Y...........	Input. Function values y_i . One-dimensional array of size n.
N...........	Input. Number of discrete points n.
H...........	Input. Increment h of the abscissas.
S............	Output. Integral S.
ICON..	Output. Condition code. See Table SIMP1-1.

Table SIMP1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	n = 2	Calculation is based on the trapezoidal rule.
30000	n < 2 or h ≤ 0.0	S is set to 0.0.

Comments on use

- Subprograms used
  SSL II......MGSSL
  FORTRAN basic function......none

- Example
  Function values y_i and the increment h are input and the integral S is determined.

```
C     **EXAMPLE**
      DIMENSION Y(100)
      READ(5,500) N,H
      READ(5,510) (Y(I),I=1,N)
      CALL SIMP1(Y,N,H,S,ICON)
      WRITE(6,600) ICON,S
      STOP
  500 FORMAT(I3,F10.0)
  510 FORMAT(6F10.0)
  600 FORMAT(10X,'ILL CONDITION =',I5
     *      /10X,'INTEGRAL VALUE =',E15.7)
      END
```

Method

Using function values y_i at discrete points in the interval $[x_i, x_n]$, integration is performed using Simpson's rule. The first three points are approximated using a second degree interpolating polynominal and integration is performed over the interval

$$\int_{x_1}^{x_3} f(x)dx \approx \frac{h}{3}(y_1 + 4y_2 + 2y_3) \qquad (4.1)$$

Next, the same calculation is continued for the succesive three points;

$$\int_{x_1}^{x_n} f(x)dx \approx \frac{h}{3}(y_1 + 4y_2 + 2y_3 + \dots \\ \dots + 4y_{n-1} + y_n) \qquad (4.2)$$

If the number of discrete points is odd this calculation is done completely. However, if it is even the above method is used over the interval $[x_1, x_{n-3}]$, and the Newton-Cotes 3/8 rule is used over the remaining interval $[x_{n-3}, x_n]$

$$\int_{x_{n-3}}^{x_n} f(x)dx \approx \frac{3}{8}h(y_1 + 3y_2 + 3y_3 + y_4) \qquad (4.3)$$

For n = 2, since the Simpson's rule cannot be used, the trapezoidal rule is used

$$\int_{x_1}^{x_2} f(x)dx \approx \frac{h}{2}(y_1 + y_2) \qquad (4.4)$$

For more information, see Reference [46] pp.114-121.

G23-11-0101 SIMP2, DSIMP2

Integration of a function by adaptive Simpson's rule
CALL SIMP2 (A, B, FUN, EPS, S, ICON)

Function

Given a function $f(x)$ and constants a, b, and ε, this subroutine obtains an approximation S such that

$$\left| S - \int_a^b f(x)dx \right| \le \varepsilon \qquad (1.1)$$

by adaptive Simpson's rule. $f(x)$ must have finite values over the integration interval.

Parameter

A........... Input. Lower limit a of the interval.
B........... Input. Upper limit b of the interval.
FUN ... Input. The name of the function subprogram which evaluates the integrand $f(x)$. See the example.
EPS ... Input. The absolute error tolerance ε (≥ 0.0) for the integral. If EPS = 0.0 is specified, the integral will be calculated as accurately as this subroutine can.
Output. The estimated error bound of the approximation obtained (See Notes.)
S........... Output. Approximation to the integral.
ICON.. Output. Condition code.
See Table SIMP2-1.

Table SIMP2-l Condition codes

Code	Meaning	Processing
0	No error	
10000	For ε > 0.0, an S such that (1.1) is satisfied could not be obtained	The approximate value that was determined and its max. absolute error are output to parameters S and EPS.
30000	ε < 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II... MGSSL, AMACH
  FORTRAN basic function... ABS

- Notes
  FUN must be declared as EXTERNAL in the program from which this subroutine is called. This subroutine is designed to treat efficiently integrands $f(x)$ having the following properties:
  a) $f(x)$ and its first five derivatives are continuous in the integration interval, and
  b) $f(x)$ does not have high frequency oscillations.

  Even if $f(x)$ has the form $g(x)|x-x_0|^\alpha$, this subroutine will work efficiently. However, $g(x)$ must satisfy a) and b), α must be nonnegative, and x_0 must be a, b, or $(a + b)/2$.
  If it is know that $f(x)$ or any of its first five derivatives are not continuous at any point(s) other than a, b, or $(a + b)/2$, the interval of the integration should be divided into smaller intervals at that (those) point(s). This subroutine can then be used on each of the resulting intervals.
  Accuracy of S-that is output
  This subroutine determines S such that (1.1) is satisfied.
  However, sometimes S can not be determined due to the difficult form of the integrand $f(x)$ or a too small. In such cases, this subroutine calculates an approximation with as high an accuracy as possible, and estimates the error bound and then these values are returned in parameters S and EPS with ICON set to 10000.
  The parameter EPS can also be specified as EPS = 0.0. This condition corresponds to the above case, however ICON is set to 0 especially in this case.

$$\int_0^1 \frac{1}{x^2 + 10^{-6}} dx$$

is determined. EPS=0.0

```
C      **EXAMPLE**
       EXTERNAL FUN
       A=0.0
       B=1.0
       EPS=0.0
       WRITE(6,600) A,B,EPS
       CALL SIMP2(A,B,FUN,EPS,S,ICON)
       WRITE(6,610) ICON,S,EPS
       STOP
   600 FORMAT('1'/' ',30X,'A=',E16.8,5X,
      * 'B=',E16.8,5X,'INPUT EPS=',E16.8//)
   610 FORMAT(' ',30X,'***RESULT***'/
      * ' ',30X,'ICON=',I5,5X,'S=',E16.8,5X,
      * 'EPS=',E16.8)
       END
       FUNCTION FUN(X)
       FUN=1.0/(X*X+1.0E-6)
       RETURN
       END
```

Method

This subroutine is based on adaptive Simpson's rule. In the adaptive algorithm, the choice of points at which the integrand is evaluated is based on the behaviour of the integrand, and as a result the integrand is evaluated at many points where the integrand changes rapidly or irregularly, but not so many points where the integrand changes smoothly or regularly. To simplify the explanation, $a < b$ will be assumed.

Using the strategy described below, the integration intervel [a, b] is subdivided, and Simpson's rule is apolied to each of the subintervals.

Some of the subintervals are further subdivided. Finally, by summing the integral over the subintervals, the integral over the entire interval $[a, b]$ is obtained. This subroutine is designed to obtain an approximation to a specified absolute accuracy. Hopefully, the approximation is within the absolute error tolerance ε.

- Simpson's rule and error estimation
  Let the interval $[\alpha,\beta]$ be a subinterval in $[a, b]$, and $h= \beta - \alpha$. The quadrature rule and error estimation method used for the integral:

$$\int_{\alpha}^{\beta} f(x)dx \tag{4.1}$$

are discussed below

$R[\alpha, \beta]\, f(x)$ is defined as

$$R[\alpha,\beta]f(x) \equiv \frac{h}{6}\left\{f(\alpha)+4f\left(\frac{\alpha+\beta}{2}\right)+f(\beta)\right\} \tag{4.2}$$

This is Simpson's rule based on three points. And $R^{(2)}[\alpha, \beta]f(x)$ is also defined as

$$\begin{aligned} R^{(2)}[\alpha,\beta]f(x) &\equiv R[\alpha,(\alpha+\beta)/2]f(x) \\ &+ R[(\alpha+\beta)/2,\beta]f(x) \end{aligned} \tag{4.3}$$

This is derived from splitting $[\alpha, \beta]$ into two subintervals and then applying Simpson's rule based on three points to each sub-interval. In this subroutine, (4.1) is approximated by (4.3). The error of $R^{(2)}[\alpha, \beta]f(x)$can be estimated using the Euler-MacLaurin expansion formula. If $f(x)$ and its first five derivatives are continuous in $[\alpha, \beta]$, then

$$\begin{aligned} &R^{(2)}[\alpha,\beta]f(x) - \int_{\alpha}^{\beta} f(x)dx \\ &= \frac{C_4 h^4}{2^4}\int_{\alpha}^{\beta} f^{(4)}(x)dx + O\left(h^6\right) \end{aligned} \tag{4.4}$$

where C_4 is a constant which is independent of $f(x)$. Similarly,

$$\begin{aligned} &R[\alpha,\beta]f(x) - \int_{\alpha}^{\beta} f(x)dx \\ &= C_4 h^4 \int_{\alpha}^{\beta} f^{(4)}(x)dx + O\left(h^6\right) \end{aligned} \tag{4.5}$$

is also true. From (4.4) and (4.5), if the term $\boldsymbol{O}(h^6)$ can be ignored,

$$\begin{aligned} &R^{(2)}[\alpha,\beta]f(x) - \int_{\alpha}^{\beta} f(x)dx \\ &\approx \frac{1}{15}\left\{R[\alpha,\beta]f(x) - R^{(2)}[\alpha,\beta]f(x)\right\} \end{aligned} \tag{4.6}$$

Since the righthand side of (4.6) can be evaluated during calculations, it is used for the error estimation of $R^{(2)}[\alpha,\beta]\, f(x)$. In (4.6) it is assumed that the round-off error is small.

The quantity $\varepsilon\,(\beta - \alpha)/(b - a)$ is assigned to the subinterval $[\alpha,\beta]$ as the limit such that the error of $R^{(2)}[\alpha,\beta]f(x)$ in (4.6) should not be exceeded; i.e., if

$$\frac{1}{15}\left|R[\alpha,\beta]f(x) - R^{(2)}[\alpha,\beta]f(x)\right| \le \frac{\beta-\alpha}{b-a}\varepsilon \tag{4.7}$$

$R^{(2)}[\alpha,\beta]\, f(x)$ is used as an approximation to (4.1). If (4.7) is not satisfied, $[\alpha,\beta]$ is further subdivided. In actual calculations, instead of (4.7), the equivalent expression (4.8) is used.

$$\left|D[\alpha,\beta]f(x)\right| \le \mathrm{E} \tag{4.8}$$

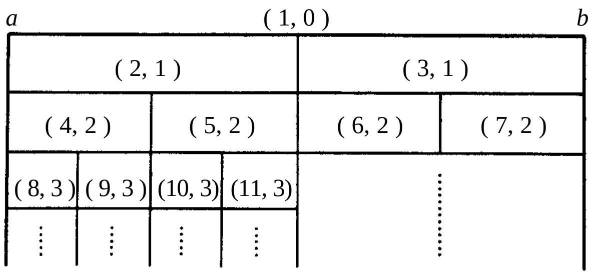
where

$$\begin{aligned} D[\alpha,\beta]f(x) = \frac{12}{\beta-\alpha}&\left\{R[\alpha,\beta]f(x)\right. \\ &\left. - R^{(2)}[\alpha,\beta]f(x)\right\} \end{aligned} \tag{4.9}$$

and

$$E = \frac{180\varepsilon}{b-a} \tag{4.10}$$

- Strategy for subdividing the integration interval$[a, b]$
  How to subdivide the interval and how to select the subinterval to which the quadrature rule (4.3) is applied, are described here. For the sake of explanation, every subinterval is assigned its number and level as follows. The integration interval [a, b] is defined as number 1, level 0 interval. The interval $[a, (a + b) / 2]$ is defined as number 2, level 1 interval, and the interval $[(a + b) / 2, b]$ is defined as number 3, level l interval. In general, if the interval $[\alpha,(\alpha + \beta)/2]$ is number 2 N, level (L+ 1) interval, then the interval $[(\alpha + \beta)/2, \beta]$ is number (2N+l), level (L+ 1) interval (See Fig. SIMP2-1).



Note: The left member in parentheses is the number and the right member is the level.

Fig. SIMP2-1 Numbers and levels

Subdivision is done as follows. First, (4.3) is applied to number 2, level 1 interval. If (4.8) is satisfied, $R^{(2)}[a,(a+b)/2]\,f(x)$ is used as an approximation over the interval $[a,(a+b)/2]$. If (4.8) is not satisfied, (4.3) is then applied to the number 4, level 2 interval.

In general, (4.3) is used on an interval of number N, level L and then the test (4.8) is applied. If (4.8) is not satisfied, the same procedure follows with the interval of number $2N$, level $(L+1)$.

However, if (4.8) is satisfied, the $R^{(2)}[\alpha,\beta]f(x)$ at that time is used as an approximation value over the interval $[\alpha,\beta]$. That value is then added to a running sum. Then the same procedure is applied to number $M(K)+1$, Level $L-K$ interval, where $M(K)$ is the first even integer of the sequence.

$$M(0)=N, M(1)=\frac{M(0)-1}{2},\ldots,$$
$$M(J+1)=\frac{M(J)-1}{2},\ldots$$

When $L-K=0$, the integration over the interval $[a, b]$ is complete. Thus, (4.11) is output as the approximation over the interval $[a, b]$.

$$\sum_{i=1}^{n} R^{(2)}[a_{i-1},a_i]f(x) \qquad (a_0=a, a_n=b) \tag{4.11}$$

Each $R^{(2)}[a_{i-1}, a_i]f(x)$ satisfies (4.8) and, consequently, (4.7).
(4.12) results from (4.6) and (4.7).

$$\left|\sum_{i=1}^{n} R^{(2)}[a_{i-1},a_i]f(x)-\int_a^b f(x)dx\right|\le\varepsilon \tag{4.12}$$

- Round-off error
  It has been explained that whether or not $R^{(2)}[\alpha,\beta]\,f(x)$ for a subinterval $[\alpha,\beta]$ is accepted, is determined by whether or not (4.8) is satisfied. If (4.8) is not satisfied, $[\alpha,\beta]$ is subdivided and integration for $[\alpha,(\beta+\beta)/2\ ]$ is considered.
  During these processes of subdivision, from a certain point on, it is probable that $D[\alpha,\beta]f(x)$ of (4.8) will have no significant digits.

This is because the significant digits of the function value of $f(x)$ are lost in the calculation of $D\,[\alpha,\beta]\,f(x)$. If this condition occurs in a particular sub-interval, the subdividing must not be continued, so the following considerations are made.

Theoretically, if in $[\alpha,\beta]$, $f(x)$ and its first four derivatives are continuous, and $f^{(4)}(x)$ is of constant sign, then

$$\left|D[\alpha,(\alpha+\beta)/2]f(x)\right|\le\left|D[\alpha,\beta]f(x)\right| \tag{4.13}$$

Therefore, if (4.13) is not satisfied during the calculations, the cause is either that zeros of $f^{(4)}(x)$ exist in $[\alpha,\beta]$, or the round-off error has completely dominated the calculation of $D[\alpha,\beta]\,f(x)$ or $D\,[\alpha,(\alpha+\beta)/2]\,f(x)$ (however, it is difficult to determine which is the actual cause). For most cases, if the subinterval is small, the cause comes from irregularities caused by the round-off error rather than the existence of zeros in $f^{(4)}(x)$ When this type of situation occurs, it is advisable to discontinue the subdivision process and substitute a different value ε' for ε in (4.7), i.e., substitute a different value E' for the E of (4.8).

In this subroutine, the following is used for the control of E'.
– If for a subinterval $[\alpha,\beta]$ in level 5 or higher,

$$\left|D[\alpha,\beta]f(x)\right|>E', \tag{4.14}$$

$$\left|D[\alpha,(\alpha+\beta)/2]f(x)\right|>E' \tag{4.15}$$

and

$$\left|D[\alpha,(\alpha+\beta)/2]f(x)\right|\ge\left|D[\alpha,\beta]f(x)\right| \tag{4.16}$$

occur, E' is changed to

$$E'=\left|D[\alpha,(\alpha+\beta)/2]f(x)\right| \tag{4.17}$$

Then $R^{(2)}[\ \alpha,(\alpha,\beta)/2]\ f(x)$ is accepted as the approximation over $[\alpha,(\alpha+\beta)/2]$.

If in subinterval $[\alpha,\beta]$

$$\left|D[\alpha,\beta]f(x)\right|\le E' \tag{4.18}$$

occurs, $R^{(2)}[\ \alpha,\beta]f(x)$ is accepted as the approximation over $[\alpha,\beta]$, and if $|D[\alpha,\beta]f(x)|\neq 0$, E' is changed as

$$E'=\max\left(E,\left|D[\alpha,\beta]f(x)\right|\right) \tag{4.19}$$

Even when E' is controlled as explained above, there are still other problems. Even if a zero of $f^{(4)}(x)$ exists in $[\alpha,\beta]$, it is judged as a round-off error, and E' in (4.18) may become a little bit larger

than necessary.
Some devices for the problems are taken to a degree (the details are omitted). For level 30 intervals, $R^{(2)}[\alpha, \beta] f(x)$ is accepted unconditionally, and E' is not changed.
Due to the control of E', the final approximation (4.11) no longer satisfies (4.12), and the error will become

$$\varepsilon_{\text{eff}} = \frac{1}{b-a}\sum_{i=1}^{n}\varepsilon'(a_i - a_{i-1}) \tag{4.20}$$

where ε' corresponds to E′ as

$$E' = \frac{180\varepsilon'}{b-a} \tag{4.21}$$

In this subroutine ε_{eff} of (4.20) is output to parameter EPS.

For further information, see Reference [61].

I11-41-0101 SINI, DSINI

Sine integral $S_i(x)$
CALL SINI (X, SI, ICON)

Function

This subroutine computes Sine integral

$$S_i(x) = \int_0^x \frac{\sin(t)}{t} dt$$

by the series and asymptotic expansions.

Parameters

X..........	Input. Independent variable x.
SI.........	Output. Function value of $S_i(x)$
ICON ..	Output. Condition codes. See Table SINI-1.

Table SINI-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	$\lvert X\rvert \geq t_{max}$	SI = sign (X)·π / 2

Comments on use

- Subprogram used
  SSL II... MGSSL, UTLIM
  FORTRAN basic functions ... ABS, SIN, and COS

- Notes
  The valid ranges of parameter X are:
  $|X| < t_{max}$
  This is provided because sin (x) and cos (x) lose their accuracy if |X| exceeds the above range.

- Example
  The following example generates a table of $S_i(x)$ from 0.0 to 10.0 with increment 0.1.

```
C     **EXAMPLE**
      WRITE (6,600)
      DO 10 K=1,101
      A=K-1
      X=A/10.0
      CALL SINI(X,SI,ICON)
      IF(ICON.EQ.0) WRITE(6,610) X,SI
      IF(ICON.NE.0) WRITE(6,620) X,SI,ICON
   10 CONTINUE
      STOP
  600 FORMAT('1','EXAMPLE OF SINE ',
     * 'INTEGRAL FUNCTION'///6X,'X',9X,
     * 'SI(X)'/)
  610 FORMAT(' ',F8.2,E17.7)
  620 FORMAT(' ','** ERROR **',5X,'X=',
     * E17.7,5X,'SI=',E17.7,5X,'CONDITION=',
     * I10)
      END
```

Methods

Two different approximation formulas are used depending on the ranges of x divided at $x = \pm 4$.

- For $0 \leq |x| < 4$
  The power series expansion of $S_i(x)$,

$$S_i(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!(2n+1)} \qquad (4.1)$$

is evaluated with the following approximation formulas:
Single precision:

$$S_i(x) = \sum_{k=0}^{6} a_k z^{2k+1}, z = x/4 \qquad (4.2)$$

Double precision:

$$S_i(x) = \sum_{k=0}^{11} a_k x^{2k+1} \qquad (4.3)$$

- For $|x| \geq 4$
  The asymptotic expansion of

$$S_i(x) = \text{sign}(x) \cdot \left[\pi/2 + \{P(x)\cos(x) - Q(x)\sin(x)\}/x\right] \qquad (4.4)$$

is calculated through use of the following approximate expressions of $P(x)$ and $Q(x)$:
Single precision:

$$P(x) = \sum_{k=0}^{11} a_k z^k, z = 4/x \qquad (4.5)$$

$$Q(x) = \sum_{k=0}^{11} b_k z^k, z = 4/x \qquad (4.6)$$

Double precision:

$$P(x) = \sum_{k=0}^{11} a_k z^k \Bigg/ \sum_{k=0}^{11} b_k z^k, z = 4/x \qquad (4.7)$$

$$Q(x) = -\sum_{k=0}^{10} c_k z^k \Bigg/ \sum_{k=0}^{11} d_k z^k, z = 4/x \qquad (4.8)$$

A22-51-0202 SLDL, DSLDL

LDL^T-deconmposition of a positive-definite symmetric matrix (Modified Cholesky's method)
CALL SLDL (A, N, EPSZ, ICON)

Function

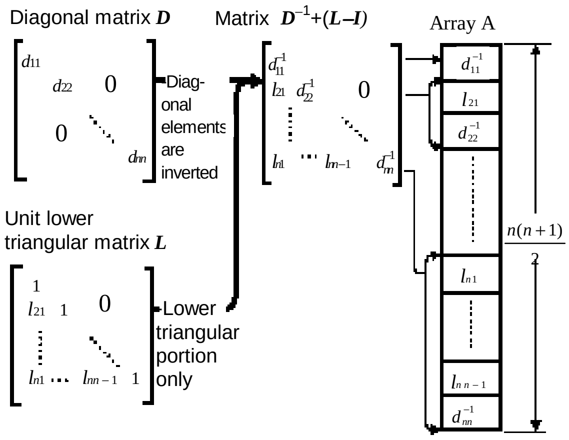
An $n \times n$ positive symmetric matrix $\boldsymbol{A}$ is LDL^T decomposed using the modified Cholesky's method.

$$\boldsymbol{A} = \boldsymbol{LDL}^T \qquad (1.1)$$

Where $\boldsymbol{L}$ is a unit lower triangular matrix, $\boldsymbol{D}$ is a diagonal matrix, and $n \geq 1$.

Parameters

A.....	Input. Matrix $\boldsymbol{A}$. Output. Matrices Land $\boldsymbol{D}^{-1}$. See Fig. SLDL-1. $\boldsymbol{A}$ is stored in a one-dimensional array of size $n(n+1)/2$ in the compressed mode for symmetric matrices.
N.....	Input. Order n of the matrix $\boldsymbol{A}$.
EPSZ..	Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ (≥ 0.0). When EPSZ = 0.0, a standard value is used. (See Notes.)
ICON..	Output. Condition code. See Table SLDL-1.



Diagonal matrix $\boldsymbol{D}$
Matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$
Array A
Diagonal elements are inverted
Unit lower triangular matrix $\boldsymbol{L}$
Lower triangular portion only
$\frac{n(n+1)}{2}$

Note: On output, the diagonal and lower triangular portions of the matrix $\boldsymbol{D}^{-1}+(\boldsymbol{L}-\boldsymbol{I})$ are stored in the one-dimensional array A in the compressed mode for symmetric matrices.

Fig. SLDL-1 Storage of the decomposed elements

Table SLDL-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	The negative pivot occurred. Matrix A is not a positive-definite.	Continued
20000	The relatively zero pivot occurred. It is highly probable that matrix $\boldsymbol{A}$ is singular.	Discontinued
30000	N <1 or EPSZ < 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II... AMACH, MGSSL
  FORTRAN basic function ... ABS

- Notes
  If EPSZ is set to 10^{-s}, this value has the following meaning: while performing the LDL^T decomposition by modified Cholesky's method,if cancellation of over s significant digits occured for the pivot, the LDL^T decomposition should be discontinued with ICON = 20000 regarding the pivot to be relatively zero.

  Let u be the unit round off, then the standard value of EPSZ is 16 u. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.

  If the negative pivot occurs in the decomposition, the coefficient matrix is not positive. In this subroutine, the condition code is set accordingly (ICON = 10000) and processing is continued. However, it should be noted that large errors may occur in such cases because pivoting was not performed.

  In this subroutine, LDL^T decomposition is performed, but is should be noted that $\boldsymbol{D}^{-1}$ is output to the array instead of $\boldsymbol{D}$.

  The determinant of the matrix can be obtained by multiplying all the n diagonal elements of the array A (the diagonal elements of $\boldsymbol{D}^{-1}$) after the subroutine has been executed and then by determining the inverse number. Note that the array A is in the compressed mode for symmetric matrices.

  For a positive-definite symmetric band matrix, the subroutine SBDL processes faster than this subroutine because the operation for the elements out of the band is omitted.

- Example
  A $n \times n$ matrix is input and LDL^T decomposition is computed. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,630)
      L=0
      LS=1
      DO 20 I=1,N
      L=L+I
      WRITE(6,600) I,(A(J),J=LS,L)
   20 LS=L+1
      CALL SLDL(A,N,1.0E-6,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GOTO 10
      WRITE(6,640)
      L=0
      LS=1
      DET=1.0
      DO 30 I=1,N
      L=L+I
      WRITE(6,600) I,(A(J),J=LS,L)
      DET=DET*A(L)
   30 LS=L+1
      DET=1.0/DET
      WRITE(6,620) DET
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(5E10.2)
  600 FORMAT(' ',I5/(10X,5E16.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(//10X,
     *'DETERMINANT OF MATRIX=',E16.8)
  630 FORMAT(/10X,'INPUT MATRIX')
  640 FORMAT(/10X,'DECOMPOSED MATRIX')
      END
```

Method

- Modified Cholesky's method

  A real positive-symmetric matrix $\boldsymbol{A}$ can always be decomposed as

$$\boldsymbol{A} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^{\mathrm{T}} \tag{4.1}$$

where, $\tilde{\boldsymbol{L}}$ is a lower triangular matrix. Decomposition is uniquely defined if all the positive diagonal elements of $\tilde{\boldsymbol{L}}$ are required. In the Cholesky's method, decomposition is performed as follows:

$$\tilde{l}_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik}\tilde{l}_{jk} \right) \Big/ \tilde{l}_{jj}\,, \quad j = 1,\ldots,i-1 \tag{4.2}$$

$$\tilde{l}_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} \tilde{l}_{ik}^{\,2} \right)^{1/2} \tag{4.3}$$

where, $\boldsymbol{A} = \left(a_{ij}\right)$, $\tilde{\boldsymbol{L}} = \left(\tilde{l}_{ij}\right)$

If $\boldsymbol{L}$ is determined such that $\tilde{\boldsymbol{L}} = \boldsymbol{L}\,\mathrm{diag}\left(\tilde{l}_{ii}\right)$, then

$$\begin{aligned} \boldsymbol{A} &= \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^{\mathrm{T}} = \boldsymbol{L}\,\mathrm{diag}\left(\tilde{l}_{ii}\right)\mathrm{diag}\left(\tilde{l}_{ii}\right)\boldsymbol{L}^{\mathrm{T}} \\ &= \boldsymbol{L}\,\mathrm{diag}\left(\tilde{l}_{ii}^{\,2}\right)\boldsymbol{L}^{\mathrm{T}} = \boldsymbol{LDL}^{\mathrm{T}} \end{aligned} \tag{4.4}$$

where, $\boldsymbol{L}$ is a unit lower triangular matrix, and $\boldsymbol{D}$ is a positive-definite diagonal matrix. While in the modified Cholesky's method, the decomposition is performed through using the following equations.

$$l_{ij}d_j = a_{ij} - \sum_{k=1}^{j-1} l_{ik}d_k l_{jk}\,, j = 1,\ldots,i-1 \tag{4.5}$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}d_k l_{ik} \tag{4.6}$$

where, $i = 1,\ldots,n$

Although the Cholesky's method needs a square root calculation in equation (4.3), the modified Cholesky's method does not need it.

For more information, see Reference [2].

A22-21-0202 SMDM, DSMDM

MDM^T - decomposition of a real indefinite symmetric matrix (Block diagonal pivoting method)
CALL SMDM (A, N, EPSZ, IP, VW, IVW, ICON)

Function

An $n \times n$ real indefinite symmetric matrix $\boldsymbol{A}$ is MDM^T-decomposed

$$\boldsymbol{PAP}^T = \boldsymbol{MDM}^T \qquad (1.1)$$

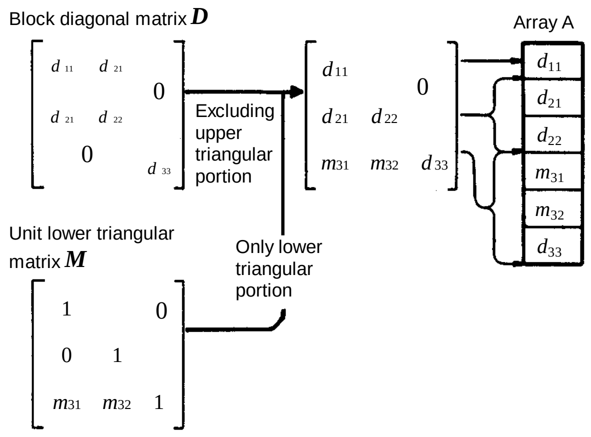
by the block diagonal pivoting method (there are two similar methods which are called Croutlike method and Gaussianlike method, respectively. This subroutine uses the format method), where $\boldsymbol{P}$ is a permutation matrix that exchanges rows of the matrix $\boldsymbol{A}$ required in its pivoting, $\boldsymbol{M}$ is a unit lower triangular matrix, and $\boldsymbol{D}$ is a symmetric block diagonal matrix that consists of only symmetric blocks, each at most of order 2. In addition, if $d_{k+1,k} \neq 0$ then $m_{k+1,k} = 0$, where $\boldsymbol{M}= (m_{ij})$ and $\boldsymbol{D} = (d_{ij})$, and $n \geq 1$.

Parameters

A	Input. Matrix $\boldsymbol{A}$ Compressed mode for a symmetric matrix Output. Matrices $\boldsymbol{M}$ and $\boldsymbol{D}$. See Fig. SMDM-1. One-dimensional array of size $n (n + 1)/2$.
N.....	Input. Order n of the matrix $\boldsymbol{A}$
EPSZ ..	Input. Tolerance for relative zero test of pivots in decomposition process of $\boldsymbol{A}$ (≥ 0.0). If EPSZ = 0.0, a standard value is used. (See Notes.)
IP......	Output. Transposition vector that indicates the history of exchanging rows of the matrix $\boldsymbol{A}$ required in pivoting. One-dimensional array of size n. (See Notes.)
VW	Work area. One-dimensional array of size $2n$.
IVW ...	Work area. One-dimensional array of size n.
ICON ..	Output. Condition code. See Table SMDM-1.

Table SMDM-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	The relatively zero pivot occurred. It is highly probable that the matrix is singular.	Discontinued
30000	N < 1 or EPSZ < 0.0	Bypassed


Block diagonal matrix $\boldsymbol{D}$
d_{11} d_{21}
d_{21} d_{22}
0
0
d_{33}
Excluding upper triangular portion
d_{11}
d_{21} d_{22}
m_{31} m_{32} d_{33}
0
Array A
d_{11}
d_{21}
d_{22}
m_{31}
m_{32}
d_{33}
Unit lower triangular matrix $\boldsymbol{M}$
1 0
0 1
m_{31} m_{32} 1
Only lower triangular portion

[Note] After computation, the diagonal portion and lower triangular portion of the matrix $\boldsymbol{D}$+($\boldsymbol{M}$–$\boldsymbol{I}$) are stored in the one-dimensional array A in compressed mode for a symmetric matrix. In this case, $\boldsymbol{D}$ consists of blocks of order 2 and 1.

Fig. SMDM-l Storing method for decomposed elements

Comments on use

- Subprograms used
  SSL II... AMACH, MGSSL, USCHA
  FORTRAN basic functions ... ABS, SQRT, IABS, ISIGN

- Notes
  If EPSZ is set to 10^{-s} this value has the following meaning: while performing the MDM^T-decomposition by the block diagonal pivoting method, if the loss of over s significant digits occurred for the pivot value (i.e., determinant of a 1×1 or 2×2 matrix of the pivot), the MDM^T- decomposition should be discontinued with ICON = 20000 regarding the pivot value as relatively zero.
  Let u be the unit round off, then the standard value of EPSZ is $16 \cdot u$

If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.

The transposition vector corresponds to the permutation matrix $\boldsymbol{P}$ in the MDM^T-decomposition with pivoting,

$$\boldsymbol{PAP}^T = \boldsymbol{MDM}^T$$

which is done by the block diagonal pivoting method.' This subroutine exchanges the elements of array A in its pivoting, and records its history in the parameter IP. Note that, for a 1×1 or 2×2 pivot the way of storing in the IP is a little different. The storing method at the k-th step of the decomposition is as follows: for 1×1 pivot, the row (and column) number $r(\geq k)$ that is exchanged by the k-th row (and column) is stored in IP(k), and for 2×2 pivot, the negative value of the row (and column) number $s(\geq k+1)$ that is exchanged by the (k+ l)st row (and column) is also stored in IP (k + 1), i.e., -*s* is stored in IP (*k*+1).

The determinant of matrix ***A*** is equal to the determinant of matrix ***D*** created by the computation, and elements of matrix ***D*** are stored in the array ***A*** as shown in Fig. SMDM-1. Refer to the example for the subroutine LSIX.

This subroutine makes use of symmetric matrix characteristics also while decomposing in order to save the data storage area. One way to solve a system of linear equations is to call this subroutine followed by the subroutine MDMX. However, instead of these subroutines, subroutine LSIX can be normally called to solve such equations in one step.

The number of positive and negative eigenvalues for the matrix ***A*** can be obtained. Refer to the example below.

- Example

  Using the subroutine SMDM, this example obtains the numbers of both positive and negative eigenvalues, by which the characteristics of a matrix can be investigated. Here, an $n \times n$ real symmetric matrix is used, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),VW(200),
     *          IP(100),IVW(100)
      CHARACTER*4 IA
      DATA IA/'A   '/
      READ(5,500) N
      NT=(N*(N+1))/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N
      CALL PSM(IA,1,A,N)
      EPSZ=0.0
      CALL SMDM(A,N,EPSZ,IP,VW,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      INEIG=0
      IPEIG=0
      I=1
      J=1
   10 IF(IP(J+1).GT.0) GO TO 20
      IPEIG=IPEIG+1
      INEIG=INEIG+1
      J=J+2
      I=I+J-1+J
      GO TO 30
   20 IF(A(I).GT.0.0) IPEIG=IPEIG+1
      IF(A(I).LT.0.0) INEIG=INEIG+1
      J=J+1
      I=I+J
   30 IF(J.LT.N) GO TO 10
      IF(J.NE.N) GO TO 40
      IF(A(I).GT.0.0) IPEIG=IPEIG+1
      IF(A(I).LT.0.0) INEIG=INEIG+1
   40 WRITE(6,620) IPEIG,INEIG
      STOP
  500 FORMAT(I3)
  510 FORMAT(4E15.7)
  600 FORMAT('1'
     *  /6X,'CLASSIFICATION OF EIGENVALUE'
     *  /6X,'ORDER OF MATRIX=',I4)
  610 FORMAT(' ',5X,'ICON OF SMDM=',I6)
  620 FORMAT(' ',5X,'POSITIVE EIGENVALUE=',
     *  I4/6X,'NEGATIVE EIGENVALUE=',I4)
      END
```

The subroutine PSM that is used in this example is used only to print a real symmetric matrix. Its program is described in the example for the subroutine MGSM.

Method

- Block diagonal pivoting method

  A positive-definite symmetric matrix ***A*** can be decomposed as shown in Eq. (4.1) using the modified Cholesky method,

$$\boldsymbol{A} = \boldsymbol{M}_1 \boldsymbol{D}_1 \boldsymbol{M}_1^{\mathrm{T}} \tag{4.1}$$

  where $\boldsymbol{M}_1$ is a unit lower triangular matrix and $\boldsymbol{D}_1$ is a diagonal matrix. A real symmetric matrix ***A*** is not always decomposed as above. It may be unstable in the sense that there is no bound on the element growth in decomposition process. Rewrite Eq. (4.1) into the form of Eq. (4.2) to overcome this complexity.

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MDM}^{\mathrm{T}} \tag{4.2}$$

The method to decompose the matrix into Eq. (4.2) is called the block diagonal pivoting method, where ***P*** is a permutation matrix that exchanges row of the matrix based on the pivoting, ***M*** is a unit lower triangular matrix, and ***D*** is a symmetric block diagonal matrix consisting of symmetric blocks at most of order 2.
This subroutine, when the real symmetric matrix ***A*** is given, obtains the matrices ***P***, ***D*** and ***M***, all of which satisfy Eqs. (4.3) and (4.4) by using the block diagonal pivoting method.

$$\boldsymbol{PAP}^{\mathrm{T}} = \boldsymbol{MS} \tag{4.3}$$

$$\boldsymbol{S} = \boldsymbol{DM}^{\mathrm{T}} \tag{4.4}$$

- Procedure performed in this subroutine

  At the *k*-th step ($k = 1, ..., n$) of the decomposition in this subroutine, the *k*-th column of the matrices ***M***, ***S*** and ***D*** are each obtained in the following computations (in which the elements not defined explicitly are all zeros). If a 2×2 pivot is chosen, the ($k + 1$)-th column is also obtained. For better understanding of the following explanation, *n*-dimensional vectors ***Q*** and ***R*** are introduced. The elements of the matrices and vectors are $\boldsymbol{A} = (a_{ij})$, $\boldsymbol{M} = (m_{ij})$, $\boldsymbol{D}= (d_{ij})$, $\boldsymbol{S}=(s_{ij})$, $\boldsymbol{Q}= (q_i)$, $\boldsymbol{R} =(r_i)$.

(a) $s_{ik} = d_{i,i-1}m_{k,i-1} + d_{ii}m_{ki} + d_{i,i+1}m_{k,i+1}$

$$i = 1,\ldots,k-1$$

$$q_i = a_{ik} - \sum_{l=1}^{k-1} m_{il}s_{lk} \quad ,i = k,\ldots,n$$

(b) $\lambda = |q_j| = \max_{k<i\le n} |q_i|$

If $|q_k| \ge \alpha\lambda, q_k$ is chosen as a 1 × 1 pivot.

$d_{kk} = q_k$

$m_{ik} = q_i / q_k \; ,i = k+1,\ldots,n$

Go to step (g). Where a good value of α is $(1+\sqrt{17}\,)/8$.

(See the reference items.)

(c) The (k + 1)st rows (and columns) of matrices ***M*** and ***A*** are exchanged with the j -th rows (and columns) and also q_{k+1} is exchanged with q_j.

$s_{i,k+1} = d_{i,i-1}m_{k+1,i-1} + d_{ii}m_{k+1,i} + d_{i,i+1}m_{k+1,i+1}$

$$i = 1,\ldots, k-1$$

$$r_i = a_{i,k+1} - \sum_{l=1}^{k-1} m_{il}s_{l,k+1}, \quad i = k+1,\ldots, n$$

(d) If $\sigma \; |q_k| \ge \alpha\lambda^2, q_k$ is chosen as a 1 × 1 pivot

$d_{kk} = q_k$

$m_{ik} = q_i / q_k \; ,i = k+1,\ldots,n$

Go to step (g).

(e) $\sigma = \max\,(\lambda, \max_{k+1_i\le n} |r_i|)$

If $|r_{k+1}|$ 2>α_σ, r_{k+1} is chosen as a 1 × 1 pivot.
The k-th row (and columns) of matrices ***M*** and ***A*** are exchanged with the (k + 1)-th row (and columns), respectively.

$d_{kk} = r_{k+1}$

$m_{k+1,k} = q_{k+1} / r_{k+1}$

$m_{ik} = r_i / r_{k+1} \; ,i = k+2,\ldots,n$

Go to step (g).

(f) det $\begin{pmatrix} q_k & q_{k+1} \\ q_{k+1} & r_{k+1} \end{pmatrix}$ $(= q_k r_{k+1} - q_{k+1}q_{k+1})$ is chosen as a 2 × 2 pivot.

As a 2 × 2 pivot

$d_{kk} = q_k$

$d_{k,k+1} = d_{k+1,k} = q_{k+1}$

$d_{k+1,k+1} = r_{k+1}$

$$\left(\begin{array}{l} m_{ik} = (q_i A_{k+1} - r_i q_{k+1}) / (q_k r_{k+1} - q_{k+1}q_{k+1}) \\ m_{i,k+1} = (r_i q_k - q_i q_{k+1}) / (q_k r_{k+1} - q_{k+1}q_{k+1}) \end{array}\right.$$

$$i = k+2,\ldots, n$$

(g) The next computational step is defined as the (k + l)-th step if the k-th step pivot is 1 × 1 and as the (k +2)-th step if 2 × 2. Go to step (a).
This algorithm takes into consideration whether or not the elements of the matrix D are zeros when calculating s_{ik} and/or $s_{i,k+1}$. If the k-th step execution has terminated at either step (***d***) or (***e***), the values of $s_{i,k+1}$, (i =1, ..., k) and q_i (i = k + l, ..., n) at the (k + 1)-th step have already been calculated except for one more multiplication and addition yet to be performed.

Precision of the inner products in this subroutine has been raised to minimize the effect of rounding errors. For further information, see References [9] and [10].

E31-11-0101 SMLE1, DSMLE1

Data smoothing by local least squares polynomials (equally spaced data points)
CALL SMLE1 (Y, N, M, L, F, ICON)

Function

Given a set of observed data at equally spaced points, this subroutine produces the smoothed values based on polynomial local least squares fit.

Each of the data is smoothed by fitting least squares polynomial of specified degree, not over all the data, but over a subrange of specified data points centered at the point to be smoothed. This process is applied to all the observed values. A limitation exists concerning m and l.

Table SMLE1-1 Limitation of m and l

Degree (m)	Number of observed values (l)
1	3
	5
3	5
	7

Parameters

Y.....	Input. Observed data y_i One-dimensional array of size n.
N....	Input. Number (n) of observed data.
M....	Input. Degree (m) of local least squares polynomials.
L.....	Input. Number (l) of observed data to which a least squares polynomials is fit.
F....	Output. Smoothed values. One-dimensional array of size n.
ICON.	Output. Condition code. See Table SMLE1-2.

Table SMLE 1-2 Condition codes

Code	Meaning	processing
0	No error	
30000	(1) $m \neq 1$ and $m \neq 3$ (2) When $m = 1$, $l \neq 3$ and $l \neq 5$ When $m = 3$, $l \neq 5$ and $l \neq 7$ (3) $n < l$	Aborted

Comments on use

- Called subprograms
  SSL II ..... MGSSL
  FORTRAN basic function ..... none

- Notes
  This subroutine presupposes that the original function cannot be approximated by single polynomial, but can be approximated locally by a certain degree of polynomial.
  The choice of m and l should be done carefully after considering the scientific information of the observed data and the experience of the user.
  Note that the extent of smoothing increases as l increases, but decreases as m increases.
  It is possible to repeat calling this subroutine, that is, to apply them m-th degree least squares polynomial relevant to l points to smoothed values. But if it is repeated too many time, its result tends to approach to the one which is produced by applying the m-th degree least squares polynomial to overall observed data. So, when it is repeated, the user decides when to stop it.
  If the user wants to apply smoothing formulas with m and l other than those prepared here, subroutine SMLE2 is recommended to use.

- Example
  At equally spaced data points, the number (n) of observed data, observed data y_i, the degree (m) of local least squares approximation and the number (l) of observed data which is used in the polynomial are input and each observed data is smoothed. ($n \leq 20$)

```
C     **EXAMPLE**
      DIMENSION Y(20),F(20)
      READ(5,500) N,M,L
      READ(5,510) (Y(I),I=1,N)
      CALL SMLE1(Y,N,M,L,F,ICON)
      WRITE(6,600)
      WRITE(6,610) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,620) M,L
      WRITE(6,630) (I,Y(I),F(I),I=1,N)
      STOP
  500 FORMAT(3I5)
  510 FORMAT(5F10.0)
  600 FORMAT('1'////26X,
     *'**** SMOOTHING BY SMLE1 ****')
  610 FORMAT('0',37X,'ICON=',I5)
  620 FORMAT('0',20X,'DEGREE OF ',
     *'POLYNOMIAL =',I2/
     *' ',23X,'POINT OF SUBRANGE =',I2/
     *'0',17X,'NO.',10X,'OBSERVED VALUES',
     *10X,'SMOOTHED VALUES')
  630 FORMAT(' ',16X,I4,10X,F15.7,10X,F15.7)
      END
```

Method

This subroutine smoothes *n* given observed data by fitting *m*-th degree local least squares polynomials relevant to *l* data points instead of fitting a single least squares polynomial over all the data.

Namely, an observed data y_k is smoothed by fitting *m*-th degree least squares polynomial relevant to *l* (= 2*r* +1) observed data $y_{k-r}, \ldots, y_{k-1}, y_k, y_{k+1}, \ldots, y_{k+r}$ and evaluating it at *k*.

Suppose,

$$\eta_s = y_i, \quad s = i - k, \quad k - r \le i \le k + r$$

(See Fig. SMLE1-1)

$$\begin{tikzcd}[column sep=small]
y_{k-r}, \arrow[d] & \ldots, & y_{k-1}, \arrow[d] & y_k, \arrow[d] & y_{k+1}, \arrow[d] & \ldots, & y_{k+r} \arrow[d] \\
\eta_{-r}, & \ldots, & \eta_{-1}, & \eta_0, & \eta_1, & \ldots, & \eta_r
\end{tikzcd}$$

Fig. SMLEI-1 Correspondence between omeω_k and f$_\sigma$

The *m*-th degree least squares polynomial relevant to these η_s is expressed as follows:

$$\bar{y}_m(s) = \sum_{t=-r}^{r}\sum_{j=0}^{m}\left\{\frac{(2j+1)[(2r)!]^2}{(2r+j+1)!(2r-j)!}\cdot P_j(t,2r)P_j(s,2r)\right\}\eta_t \qquad (4.1)$$

where

$$P_j(\xi,2r) = \sum_{k=0}^{j}(-1)^{j+k}\frac{(j+k)^{(2k)}}{(k!)^2}\frac{(r+\xi)^{(k)}}{(2r)^{(k)}} \qquad (4.2)$$

$$(\xi)^{(k)} = \xi(\xi-1)(\xi-2)\cdots(\xi-k+1)$$

(4.1) is called a *m*-th degree smoothing formula relevant to *l* points. This formula can be derived from the least squares polynomial which was described at the "Method" of subroutine LESQ1 in case $w(x_i)$ an abscissas are equally spaced.

y_k is smoothed by (4.3).

$$\bar{y}_m(0) = \sum_{t=-r}^{r}\sum_{j=0}^{m}\left\{\frac{(2j+1)[(2r)!]^2}{(2r+j+1)!(2r-j)!}\cdot P_j(t,2r)P_j(0,2r)\right\}\eta_t \qquad (4.3)$$

When y_k is either among $y_1, \ldots, y_r$ or among $y_{n-r+1}, \ldots, y_n$, y_k does not have *r* observed data equally on both sides and can not be smoothed by $\bar{y}_m$ (0). In case of $y_1, \ldots, y_r$, the smoothing is done by $\bar{y}_m(-r), \ldots, \bar{y}_m(-1)$ and in case of $y_{n-r+1}, \ldots, y_n$, by $\bar{y}_m(1), \ldots, \bar{y}_m(r)$.

And in (4.1) the coefficient of *m* in *l* is the same as that of *m*= l in *l* = 3:(*r* = 1).

Smoothing formulas with *m* and *l* used in this subroutine are show below.

The smoothing formula concerning *m* = 1 and *l* = 3: (*r*= 1)

$$\bar{y}_1(-1) = \frac{1}{6}(5\eta_{-1} + 2\eta_0 - \eta_1)$$

$$\bar{y}_1(0) = \frac{1}{3}(\eta_{-1} + \eta_0 + \eta_1)$$

....

The smoothing formula concerning *m* = 2 and *l* = 5; (*r* = 2)

$$\bar{y}_1(-2) = \frac{1}{5}(3\eta_{-2} + 2\eta_{-1} + \eta_0 - \eta_2)$$

$$\bar{y}_1(-1) = \frac{1}{10}(4\eta_{-2} + 3\eta_{-1} + 2\eta_0 + \eta_1)$$

$$\bar{y}_1(0) = \frac{1}{5}(\eta_{-2} + \eta_{-1} + \eta_0 + \eta_1 + \eta_2)$$

....

The smoothing formula concerning *m* = 3 and *l* = 5; (*r*= 2)

$$\bar{y}_3(-2) = \frac{1}{70}(69\eta_{-2} + 4\eta_{-1} - 6\eta_0 + 4\eta_1 - \eta_2)$$

$$\bar{y}_3(-1) = \frac{1}{35}(2\eta_{-2} + 27\eta_{-1} + 12\eta_0 - 8\eta_1 + 2\eta_2)$$

$$\bar{y}_3(0) = \frac{1}{35}(-3\eta_{-2} + 12\eta_{-1} + 17\eta_0 + 12\eta_1 - 3\eta_2)$$

....

The smoothing formula concerning *m* = 3 and *l* = 7; (*r*= 3)

$$\bar{y}_3(-3) = \frac{1}{42}(39\eta_{-3} + 8\eta_{-2} - 4\eta_{-1} - 4\eta_0 + \eta_1 + 4\eta_2 - 2\eta_3)$$

$$\bar{y}_3(-2) = \frac{1}{42}(8\eta_{-3} + 19\eta_{-2} + 16\eta_{-1} + 6\eta_0 - 4\eta_1 - 7\eta_2 + 4\eta_3)$$

$$\bar{y}_3(-1) = \frac{1}{42}(-4\eta_{-3} + 16\eta_{-2} + 19\eta_{-1} + 12\eta_0 + 2\eta_1 - 4\eta_2 + \eta_3)$$

$$\bar{y}_3(0) = \frac{1}{21}(-2\eta_{-3} + 3\eta_{-2} + 6\eta_{-1} + 7\eta_0 + 6\eta_1 + 3\eta_2 - 2\eta_3)$$

....

For details, see Reference [46] pp.228 to 254, [51] pp.314 to 363.

E31-21-0101 SMLE2, DSMLE2

Data smoothing by local least squares polynomials (unequally spaced data points)
CALL SMLE2 (X, Y, N, M, L, W, F, VW, ICON)

Function

Given a set of observed data at $x_1, x_2, ..., x_n$ ($x_1 < x_2 < ... < x_n$) and corresponding weights $w(x_i)$, $i = 1, 2, ..., n$, this subroutine produces the smoothed values based on polynomial local least squares fit.

Each of the data is smoothed by fitting least squares polynomial of specified degree m, not over all the data, but over a subrange of specified l data points centered at the point to be smoothed.

Where $n \geq l$, $w(x_i) \geq 0$ (i= 1, ..., n), $l \geq m$+ 2, $m \geq 1$ and l must be an odd integer.

Parameters

X....	Input. Discrete points x_i. One-dimensional array of size n.
Y....	Input. Observed values y_i. One-dimensional array of size n.
N....	Input. Number (n) of observed values.
M....	Input. Degree (m) of local least squares a polynomials.
L.....	Input. Number (l) of observed data to which least squares polynomial is fit.
W.....	Input. Weight functions $w(x_i)$. Normally $w(x_i)$ = 1. One-dimensional array of size n.
F	Output. Smoothed values. One-dimensional array of size n.
VW...	Working area. One-dimensional array of size 2l.
ICON..	Output. Condition code. See Table SMLE2-1.

Table SMLE2-2 Condition codes

Code	Meaning	processing
0	No error	
30000	One of the following happened: (1) $x_1 < x_2 < ... < x_{n-1} < x_n$ is not satisfied. (2) n < 1 (3) m < 1 or m+2 > 1 (4) Some of m (x_i) are negative. (5) 1 is even.	Bypassed

Comments on use

- subprograms used
  SSL II... MGSSL
  FORTRAN basic function ... None

- Notes
  This subroutine presupposes that the original function cannot be approximated by single polynomial. but can be approximated locally by a certain degree of polynomial.
  The choice of m and l should be done carefully after considering the scientific information of the observed data and the experience of the user.
  Note that the extent of smoothing increases as l increases, but decreases as m increases.
  It is possible to repeat calling this subroutine, that is, to apply the m-th degree least squares polynomial relevant to l points to smoothed values. But if it is repeated too many times, its result tends to approach to the one which is produced by applying the m-th degree least squares polynomial to overall observed data. So, when it is repeated, the user decides when to stop it.
  This subroutine can be used in the case data points are not equally spaced. But it takes more processing time than the subroutine SMLE1.

- Example
  Number (n) of discrete points, discrete points x_i, observed values y_i, degree m of local least squares polynomial and number of observed values l are given to smooth the observed values.

$$n \leq 20, m \leq 17, w(x_i) = 1, (i = 1,...,n)$$

```
C      **EXAMPLE**
       DIMENSION X(20),Y(20),W(20),
      *          F(20),VW(40)
       READ(5,500) N,M,L
       READ(5,510) (X(I),I=1,N)
       READ(5,510) (Y(I),I=1,N)
       DO 10 I=1,N
    10 W(I)=1.0
       CALL SMLE2 (X,Y,N,M,L,W,F,VW,ICON)
       WRITE(6,600)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
       WRITE(6,620) M,L
       WRITE(6,630) (X(I),Y(I),F(I),I=1,N)
       STOP
   500 FORMAT(3I5)
   510 FORMAT(5F10.0)
   600 FORMAT('1'////26X,
      *'**** SMOOTHING BY SMLE2 ****')
   610 FORMAT('0',37X,'ICON=',I5)
   620 FORMAT('0',20X,'DEGREE OF ',
      *'POLYNOMIAL =',I2/
      *' ',23X,'POINT OF SUBRANGE =',I2/
      *'0',11X,'ABSCISSA',11X,'OBSERVED ',
      *'VALUES',10X,'SMOOTHED VALUES')
   630 FORMAT(' ',10X,F10.0,10X,F15.7,10X,
      *F15.7)
       END
```

Method

This subroutine smoothes n given observed data at unequally spaced points $x_1, x_2, ..., x_n$ by fitting m-th degree local least squares polynomials relevant to l

points, instead of fitting a single least squares polynomial over all the data.

Namely and observed data y_k is smoothed by fitting m-th degree least squares polynomial relevant to $l(=2r+1)$ observed data $y_{k-r}, ..., y_{k-1}, y_k, y_{k+1}, ..., y_{k+r}$ and by evaluating it at $x = x_k$.

Suppose,

$$\eta_s = y_i, \quad \xi_s = x_i, \quad s = i - k$$
$$k - r \le i \le k + r$$

(See Fig. SMLE2-l)

The m-th degree least squares polynomial relevant to these η_s is expressed as follows:

$$\bar{y}_m(\xi_s) = \sum_{j=0}^{m} \left\{ \frac{\sum_{t=-r}^{r} w(\xi_t)\eta_t p_j(\xi_t)}{\sum_{t=-r}^{r} w(\xi_t)[p_j(\xi_t)]^2} P_j(\xi_s) \right\} \qquad (4.1)$$

where

$$P_{j+1}(\xi_s) = (\xi_s - \alpha_{j+1})P_j(\xi_s) - \beta_j P_{j-1}(\xi_s)$$
$$P_0(\xi_s) = 1, \quad P_{-1}(\xi_s) = 0$$
$$j = 0,1,2,...$$

$$\alpha_{j+1} = \sum_{t=-r}^{r} w(\xi_t)\xi_t [P_j(\xi_t)]^2 / \sum_{t=-r}^{r} w(\xi_t)[P_j(\xi_t)]^2$$
$$j = 0,1,2,...$$

$$\beta_j = \sum_{t=-r}^{r} w(\xi_t)[P_j(\xi_t)]^2 \sum_{t=-r}^{r} w(\xi_t)[P_{j-1}(\xi_t)]^2$$
$$j = 1,2,... \qquad (4.2)$$

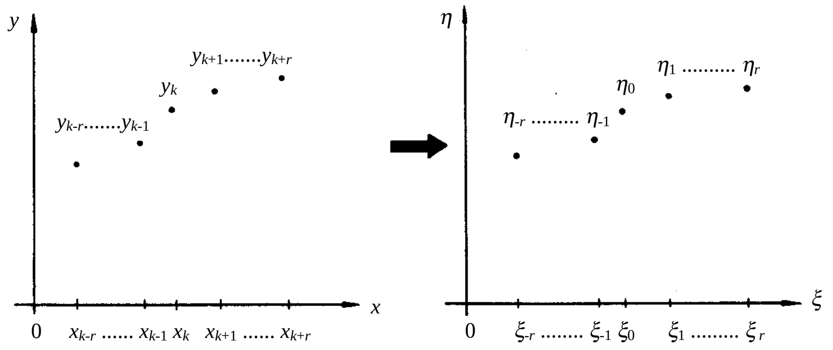
As for (4.1) refer to “Method” of the subroutine LESQ1.

Then y_k is smoothed by (4.3).

$$\bar{y}_m(\xi_0) = \sum_{j=0}^{m} \left\{ \frac{\sum_{t=-r}^{r} w(\xi_t)\eta_t P_j(\xi_t)}{\sum_{t=-r}^{r} w(\xi_t)[P_j(\xi_t)]^2} P_j(\xi_0) \right\} \qquad (4.3)$$

When y_k is either among $y_1, ..., y_r$ among $y_{n-r+1}, ..., y_n$, y_k does not have r observed data equally on both sides and can not be smoothed by $\bar{y}_m(\xi_0)$. In case of $y_1, ..., y_r$ the smoothing is done by $\bar{y}_m(\xi_1), ..., \bar{y}_m(\xi_r)$ respectively and in case of $y_{n-r+1}, ..., y_n$ at $\bar{y}_m(\xi_{-r}), ..., \bar{y}_m(\xi_{-r})$ respectively.

For details, see reference [46] pp.228 to 254 and [51] pp.314 to 363.



y
y_{k+1}.......y_{k+r}
y_k
y_{k-r}.......y_{k-1}
x
0
x_{k-r} ...... x_{k-1} x_k x_{k+1} ...... x_{k+r}
η
η_1 .... η_r
η_0
η_{-r} .... η_{-1}
ξ
0
ξ_{-r} .... ξ_{-1} ξ_0 ξ_1 .... ξ_r

Fig. SMLE2-1 (x_i, y_i) and (η_s, ξ_s)

E11-21-0101 SPLV, DSPLV

Cubic spline interpolation, differentiation
CALL SPLV (X, Y, N, ISW, DY, V, M, DV, K, VW, ICON)

Function

Given discrete points $x_1, x_2, ..., x_n$ ($x_1 < x_2 < ... < x_n$) and function values $y_i = f(x_i)$, $i = 1, 2, ..., n$ this subroutine obtains interpolated values, and 1st and 2nd order derivatives at $x = v_i$, $i = 1, 2, ..., m$ using a cubic spline interpolating function.

$$n \ge 3, x_1 \le v_1, v_2, ..., v_m \le x_n, m \ge 1$$

The boundary conditions for derivatives at both ends of discrete points ($x = x_1$, $x = x_n$) may be specified.

Parameters

X..... Input. Discrete points x_i.
X is a one-dimensional array of size n.

Y..... Input. Function values y_i.
Y is a one-dimensional array of size n.

N..... Input. Number of discrete points n.

ISW... Input Control information.
ISW is a one-dimensional array of size 2, and denotes the type of boundary conditions. Both ISW (1) and ISW (2) must be any one of 1, 2, 3, or 4. Accordingly, derivatives must be input to DY (1) and DY (2). Details are given below.
How to specify boundary conditions:
When ISW(1)=1, DY(1)=$f''(x_1)$
=2, DY(1) = $f'(x_1)$
=3, DY(1)=$f''(x_1) / f''(x_2)$
=4, DY(1) need not be input.
When ISW(2)=1, DY(2)=$f''(x_n)$
=2, DY(2) = $f'(x_n)$
=3, DY(2) =$f''(x_n) / f''(x_{n-1})$
=4, DY(2) need not be input.
When ISW(1) = 4 (or ISW(2)=4), $f'(x_1)$ (or $f'(x_n)$) is approximated using a cubic (or quadratic, when n=3) interpolating polynomial and the resultant value taken as a boundary condition.

DY.... Input. Derivatives at both end points. DY is a one-dimensional array of size 2. (See parameter ISW.)

V..... Input. Points at which interpolated values are to be obtained, v_i, $i = 1,..., m$
V is a one-dimensional array of size m.

M..... Input. Number of points m at which interpolated values are to be obtained.

DV.... Output. Interpolated value, and derivatives of order 1 and 2 at v_i.
Two-dimensional array of DV (K,3)
For I = 1,2, ..., M, interpolated value, and derivatives of order 1 and 2 at V(I) are returned respectively in DV (I,1), DV (I,2) and DV (I,3).

K..... Input. Adjustable dimension of array DV (≥ M).

VW...... Work area. VW is a one-dimensional array of size $2n$.

ICON.. Output. Condition code. See Table SPLV-1.

Table SPLV-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	1 N<3 2 $x_i \ge x_{i+1}$ 3 ISW (1) or ISW (2) is not equal to 1.2, 3 or 4 4 M < 1 5 $v_i < x_1$ or $x_1 < v_i$ 6 K < M	Aborted

Comments on use

- Subprograms used
  SSL II ... MGSSL, USPL
  FORTRAN basic function... None

- Notes
  When the derivatives at both ends are unknown, "4" may be specified for both ISW(1) and ISW(2).

- Example
  Discrete point x_i, function value y_i, $i = 1, 2, ..., n$, and boundary conditions ISW(1), DY(1), ISW(2), and DY(2) are input, and the interpolated values, first-order derivatives and second-order derivatives at points:

$$v_{2i-1} = x_i \qquad , i = 1,..., n$$
$$v_{2i} = (x_i + x_{i+1})/2 \quad , i = 1,..., n-1$$

are determined $n \le 10$.

```
C     **EXAMPLE**
      DIMENSION X(10),Y(10),ISW(2),DY(2),
     *V(50),DV(50,3),VW(20)
      READ(5,500) N
      READ(5,510) (X(I),Y(I),I=1,N)
      WRITE(6,600) (I,X(I),Y(I),I=1,N)
      READ(5,520) (ISW(I),DY(I),I=1,2)
      WRITE(6,610) (ISW(I),DY(I),I=1,2)
      N1=N-1
      DO 10 I=1,N1
      V(2*I-1)=X(I)
   10 V(2*I)=0.5*(X(I)+X(I+1))
      M=2*N-1
      V(M)=X(N)
      CALL SPLV(X,Y,N,ISW,DY,V,M,DV,50,
     *VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) STOP
      WRITE(6,630) (I,V(I),(DV(I,J),J=1,3),
     *I=1,M)
      STOP
  500 FORMAT(I5)
  510 FORMAT(2F10.0)
  520 FORMAT(I5,F10.0)
  600 FORMAT('1'//10X,'INPUT DATA'//
     *20X,'NO.',10X,'X',17X,'Y'//
     *(20X,I3,3X,E15.7,3X,E15.7))
  610 FORMAT(/10X,'BOUNDARY COND.'/
     *20X,'ISW(1)=',I3,',DY(1)=',E15.7/
     *20X,'ISW(2)=',I3,',DY(2)=',E15.7/)
  620 FORMAT(10X,'RESULTS'/20X,'ICON=',I5/)
  630 FORMAT(20X,'NO.',10X,'V',17X,'Y(V)',
     *'Y''(V)',13X,'Y''''(V)'//
     *(20X,I3,4(3X,E15.7)))
      END
```

Method

Given discrete points x_1, x_2, ..., x_n ($x_1 < x_2 < ... < x_n$) and function values $y_i = f(x_i)$, i = 1, 2, ..., n , the interpolated values and the 1st and 2nd order derivatives at any point v in $[x_1, x_n]$ are determined using a cubic interpolating spline.

Here, a cubic interpolating spline is an interpolating function $S(x)$ defined in interval $[x_1, x_n]$ and it satisfies the following conditions:

- $S(x)$ is an polynomial of degree three at most in each interval $[x_i, x_{i+1}]$, i= 1,..., n - 1
  $S(x)$ and its derivatives of up to order 2 are continuous on the interval $[x_1, x_n]$
  That is, $S(x) \in C^2 [x_1, x_n]$
- $S(x_i) = y_i$, i = 1, 2, ..., n
  Here, $S(x)$ is determined. For the convenience of explanation, $S(x)$ is sectionally expressed by (4.1).

$$\begin{aligned} S(x) &= S_i(x) \\ &= y_i + c_i(x-x_i) + d_i(x-x_i)^2 + e_i(x-x_i)^3 \\ & \qquad x_i \le x \le x_{i+1}, \quad i = 1,...,n-1 \end{aligned} \tag{4.1}$$

The above three conditions are represented using $S_i(x)$ as follows:

$$\left.\begin{aligned} S_i(x_i) &= y_i & ,i &= 1,...,n-1 \\ S_{i-1}(x_i) &= y_i & ,i &= 2,...,n \\ S'_{i-1}(x_i) &= S'_i(x_i) & ,i &= 2,...,n-1 \\ S''_{i-1}(x_i) &= S''_i(x_i) & ,i &= 2,...,n-1 \end{aligned}\right\} \tag{4.2}$$

Accordingly, coefficients c_i, d_i and e_i in (4.1) are determined by the conditions of (4.2). These coefficients are given in (4.3).

When

$$\left.\begin{aligned} &\text{For } i = 1,...,n-1 \\ &\quad c_i = \frac{y_{i+1}-y_i}{h_i} - \frac{h_i}{6}\left(y''_{i+1} + 2y''_i\right) \\ &\quad d_i = y''_i/2 \\ &\quad e_i = \frac{y''_{i+1}-y''_i}{6h_i} \\ &\text{Where} \quad h_i = x_{i+1} - x_i \\ &\qquad\quad y''_i = S''(x_i) \end{aligned}\right\} \tag{4.3}$$

y''_i satisfies the three-term relation of (4.4) according to the third condition in (4.2).

$$\begin{aligned} & h_{i-1}y''_{i-1} + 2(h_{i-1}+h_i)y''_i + h_i y''_{i+1} \\ & \quad = 6\left(\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right) \\ & \quad ,i = 2,...,n-1 \end{aligned} \tag{4.4}$$

Therefore, c_i, d_i and e_i in (4.3) may be determined by solving y''_i from (4.4). In (4.4), only (n - 2) equations are available for n unknows y''_i, so that two more equations for y''_i are required to solve (4.4) uniquely. This subroutine allows the user to assign such equations as boundary conditions.

How to specify boundary conditions

Assumption is made that equations in the following forms (4.5) and (4.6) are given as boundary conditions for solving (4.4).

$$\lambda_1 y''_1 + \mu_1 y''_2 = d_1 \tag{4.5}$$

$$\lambda_n y''_{n-1} + \mu_n y''_n = d_n \tag{4.6}$$

These constants λ_1, μ_1, d_1, λ_n, μ_n and d_n can be specified arbitrarily. In this subroutine, however, the following assumptions are made:

(a) (ISW(1) = 1, DY(1) =$f''(x_1)$ specifying $f''(x_1)$).
  Then, (4.5) is reduced to $y_1'' = f''(x_1)$.

(b) (ISW(2) =2, DY(1) = $f'(x_1)$ specifying $f''(x_1)$). The equation corresponding to (4.5) is generated in the following manner: Since the derivative S'_1 (x_1) as determined by (4.1) and (4.3) is expressed by:

$$S'_1(x_1) = \frac{y_2 - y_1}{h_1} - h_1\left(\frac{2y''_1 + y''_2}{6}\right)$$

so, by setting S'_1 $(x_1) = f'(x_1)$,

$$2y''_1 + y''_2 = \frac{6}{h_1}\left(\frac{y_2-y_1}{h_1} - f'(x_1)\right)$$ is oftained.

(c) (ISW(1) =3,DY (1)=, $f''(x_1)/f''(x_2)$) specifying $f''(x_1)/f''(x_2)$
By letting $y''_1/y''_2 = f''(x_1)/f''(x_2)$ (4.5) can be written as

$$y''_1 - \left(f''(x_1)/f''(x_2)\right)y''_2 = 0$$

(d) $f'(x_1)$ is approximated using an interpolating polynomial (ISW(1) =4, there is no need to input DY(1)). $f'(x_n)$ is approximated by an interpolating polynomial using four points of x_1, x_2, x_3, and x_4, then the case (b) is applied. (However, when only three discrete points are available, $f(x_1)$ is approximated using the three points.)
The abovedescribed four steps describe the procedure for assigning an equation corresponding to (4.5). The procedure for assigning an equation corresponding to (4.6) is based on a similar idea.

(e) (ISW(2) = 1, DY(2) = $f''(x_n)$) specifying $f''(x_n)$

(f) (ISW(2) = 2, DY(2) = $f'(x_n)$) specifying $f'(x_n)$ In this situation, (4.6) can be written as

$$y''_{n-1} + 2y''_n = \frac{6}{h_{n-1}}\left(f'(x_n) - \frac{y_n - y_{n-1}}{h_{n-1}}\right)$$

(g) (ISW(2) = 3, DY (2) = $f'(x_n)/f'(x_{n-1})$) specifying $f'(x_n)$

(h) (f) is fitted by approximating $f'(x_n)$
(There is no need to input ISW(2) = 4, DY(2).)

(a)through (d) and through (h) can be specified combined with each other.
For example, (a) and (e), (a) and (g) or (b) and (h) might be specified.

Interpolated value calculation
The interpolated value, the derivative of order 1 and 2 at any point v in interval $[x_1, x_n]$ are determined by evaluating $S_i(x)$ and $S'_i(x)$, $S'_i(x)$ which are defined on interval $[x_i, x_{i+1}]$ satisfying the condition $x_i \le v < x_{i+1}$ For further information, see Reference [48].

A21-12-0201 SSSM, DSSSM

Subtraction of two matrices (real symmetric matrix)
CALL SSSM (A, B, C, N, ICON)

Function

These subroutines perform subtraction of $n \times n$ real symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$

$$\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$$

where $\boldsymbol{C}$ is an $n \times n$ real symmetric matrix. $n \geq 1$.

Parameters

A.....	Input. Matrix $\boldsymbol{A}$, in the compressed mode, one-dimensional array of size $n\,(n+1)/2$.
B.....	Input. Matrix $\boldsymbol{B}$, in the compressed mode, one-dimensional array of size $n\,(n+1)/2$. (See Notes.)
C.....	Output. Matrix $\boldsymbol{C}$, in the compressed mode, one-dimensional array of size $n\,(n+1)/2$. (See Notes.)
N.....	Input. The order n of matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$.
ICON ..	Output. Condition codes. See Table SSSM-1.

Table SSSM-1 Condition codes

Code	Meaning	processing
0	No error	
30000	n < 1	Bypassed.

Comments on use

- Subprograms used
  SSL II... MGSSL
  FORTRAN basic function... None

- Notes
  Saving the storage area:
  When the contents of array A or B are not required.
  Save the area as follows:
  – When the contents of array A are not needed.
  CALL SSSM (A, B, A, N, ICON)
  – When the contents of array B is not needed.
  CALL SSSM (A, B, B, N, ICON)
  In the above two cases, matrix $\boldsymbol{C}$ is stored in array A or B.

- Example
  The following shows an example of obtaining the subtraction f matrix $\boldsymbol{B}$ from matrix $\boldsymbol{A}$. Here, $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),C(5050)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
   10 READ(5,100) N
      IF(N.EQ.0) STOP
      WRITE(6,150)
      NT=N*(N+1)/2
      READ(5,200) (A(I),I=1,NT)
      READ(5,200) (B(I),I=1,NT)
      CALL SSSM(A,B,C,N,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PSM(IA,1,A,N)
      CALL PSM(IB,1,B,N)
      CALL PSM(IC,1,C,N)
      GOTO 10
  100 FORMAT(I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX SUBTRACTION **')
      END
```

Subroutine PSM in the example is for printing the real symmetric matrix. This program is shown in the example for subroutine MGSM.

B21-21-0602 TEIG1, DTEIG1

Eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix (QL method)
CALL TEIG1 (D, SD, N, E, EV, K, M, ICON)

Function

All eigenvalues and corresponding eigenvectors of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are determined using the QL method. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2=1$. $n\geq1$.

Parameters

D.....	Input. Diagonal elements of real symmetric tridiagonal matrix $\boldsymbol{T}$. D is a one-dimensional array of size n. The contents of D are altered on output.
SD....	Input. Subdiagonal elements of tridiagonal matrix $\boldsymbol{T}$. The subdiagonal elements are stored in SD (2) to SD (N). The contents of SD are altered on output.
N.....	Input. Order n of tridiagonal matrix $\boldsymbol{T}$.
E.....	Output. Eigenvalues. E is a one-dimensional array of size n.
EV....	Output. Eigenvectors. Eigenvectors are stored in columns of EV. EV(K,N) is a two-dimensional array.
K.....	Input. Adjustable dimension of array EV.
M.....	Output. Number of eigenvalues/eigenvectors that were determined.
ICON ..	Output. Condition code. See Table TEIG1-1.

Comments on use

- Subprogram used

  SSL II ..... AMACH and MGSSL

  FORTRAN basic functions .. ABS, SIGN, and SQRT

- Notes

  Eigenvalues and corresponding eigenvectors are stored in the order that eigenvalues are determined.

Table TEIG1-1 Condition code

Code	Meaning	processing
10000	N=1	E(1) = D(1), EV(1,1)=1.0
15000	Some eigenvalues/eingenvectors could not be determined.	M is set to the number of eigenvalues/eigen-vector that were determined.
20000	None of the eigenvalues/eigenvectors could be determined.	M = 0
30000	N < 1 or K < n	Bypassed

Parameter M is set to n when ICON=0: parameter M is set to the number of eigenvalues/eigenvectors that were determined when ICON = 15000.

This routine is used for real symmetric tridiagonal matrices. For determining all eigenvalues and corresponding eigenvectors of a real symmetric matrix, subroutine SEIG1 should be used.

For determining all eigenvalues of a real symmetric tridiagonal matrix, subroutine TRQL should be used:

- Example

  All eigenvalues and corresponding eigenvectors of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are determined. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION D(100),SD(100),
     *          EV(100,100),E(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) (D(I),SD(I),I=1,N)
      WRITE(6,600) N,(I,D(I),SD(I),I=1,N)
      CALL TEIG1(D,SD,N,E,EV,100,M,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL SEPRT(E,EV,100,N,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(2E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',5X,
     * 'ORDER=',I3/'0',20X,'***DIAGONAL***',
     * 5X,'**SUBDIAGONAL*'//(13X,I3,5X,
     * 2(E14.7,5X)))
  610 FORMAT('0',20X,'ICON=',I5)
      END
```

In this example, subroutine SEPRT is used to print eigenvalues and corresponding eigenvectors of real symmetric matrices. For details see the example in section SEIG1.

Method

All eigenvalues and corresponding eigenvectors of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are determined using the QL method.

The QL method used to determine eigenvalues is explained in the SEIG1 section, the QL method of determining eigenvectors will be explained. In the orthogonal similarity transformation of (4.1), orthogonal matrix $\boldsymbol{Q}$ reduces $\boldsymbol{T}$ to diagonal matrix $\boldsymbol{D}$. The column vectors of $\boldsymbol{Q}$ are the eigenvectors of $\boldsymbol{T}$.

$$\boldsymbol{D}=\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{T}\boldsymbol{Q} \tag{4.1}$$

By repeatedly executing the orthogonal similarity transformation in (4.2), the QL method reduces $\boldsymbol{T}$ to $\boldsymbol{D}$, and obtains all eigenvalues as the diagonal elements of $\boldsymbol{D}$.

$$\boldsymbol{T}_{s+1} = \boldsymbol{Q}_s{}^{\mathrm{T}}\boldsymbol{T}_s\boldsymbol{Q}_s, \quad s = 1,2,3,... \tag{4.2}$$

$\boldsymbol{Q}_s$ is the orthogonal matrix obtained by QL decomposition of

$$(\boldsymbol{T}_s - k_s\boldsymbol{I}) = \boldsymbol{Q}_s\boldsymbol{L}_s \tag{4.3}$$

Where k_s is the origin shift and $\boldsymbol{L}_s$ is a lower triangular matrix. If $\boldsymbol{T}$ converges to a diagonal matrix on the m-th iteration from (4.2).

$$\boldsymbol{D} = \boldsymbol{Q}_m^{\mathrm{T}}\boldsymbol{Q}_{m-1}^{\mathrm{T}} \cdots \boldsymbol{Q}_2^{\mathrm{T}}\boldsymbol{Q}_1^{\mathrm{T}}\boldsymbol{T}_1\boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q}_{m-1}\boldsymbol{Q}_m \tag{4.4}$$

Where $\boldsymbol{T}_1 = \boldsymbol{T}$

From (4.1) and (4.4), eigenvectors are obtained as the column vectors of

$$\boldsymbol{Q} = \boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q}_{m-1}\boldsymbol{Q}_m \tag{4.5}$$

If $\boldsymbol{Q}^{(s-1)}$ is defined as

$$\boldsymbol{Q}^{(s-1)} = \boldsymbol{Q}_1\boldsymbol{Q}_2 \cdots \boldsymbol{Q}_{s-1} \tag{4.6}$$

The product $\boldsymbol{Q}^{(s)}$ of transformation matrices up to the s-th iteration is

$$\boldsymbol{Q}^{(s)} = \boldsymbol{Q}^{(s-1)}\boldsymbol{Q}_s \tag{4.7}$$

$\boldsymbol{Q}_s$ of (4.7) is calculated in the QL method as

$$\boldsymbol{Q}_s = \boldsymbol{P}_{n-1}^{(s)}\boldsymbol{P}_{n-2}^{(s)} \cdots \boldsymbol{P}_2^{(s)}\boldsymbol{P}_1^{(s)} \tag{4.8}$$

If (4.7) is substituted in (4.8).

$$\boldsymbol{Q}^{(s)} = \boldsymbol{Q}^{(s-1)}\boldsymbol{P}_{n-1}^{(s)}\boldsymbol{P}_{n-2}^{(s)} \cdots \boldsymbol{P}_2^{(s)}\boldsymbol{P}_1^{(s)} \tag{4.9}$$

After all Q can be determined by repeating (4.9) for s = 1, 2, ..., m.

In the QL method, if an eigenvalue does not converge after 30 iterations, processing is terminated. However, the M eigenvalues and eigenvectors that were determined till then can still be used.

Since matrix $\boldsymbol{Q}$ is orthogonal, the eigenvectors obtained in the above method are normalized such that $\|\boldsymbol{x}\|_2=1$.

For further information see References [12], [13] pp. 227-248 and [16] pp. 177-206.

B21-21-0702 TEIG2, DTEIG2

Selected eigenvalues and corresponding eigenvectors of a real symmetric tridiagonal matrix (bisection, method, inverse iteration method)
CALL TEIG2 (D, SD, N, M, E, EV, K, VW, ICON)

Function

The *m* largest or *m* smallest eigenvalues and corresponding eigenvectors are determined from *n*-order real symmetric tridiagonal matrix ***T***. The eigenvalues are determined using the bisection method, and the corresponding eigenvectors are determined using the inverse iteration method. The eigenvectors are normalized such that $\|\boldsymbol{x}\|_2=1$. $1\le m\le n$.

Parameters

D.....	Input. Diagonal elements of real symmetric tridiagonal matrix ***T***. D is a one-dimensional array of size *n*.
SD....	Input. Subdiagonal elements of real symmetric tridiagonal matrix ***T***. SD is a one-dimensional array of size *n*. The subdiagonal elements are stored in SD(2) to SD(N).
N....	Input. Order *n* of tridiagonal matrix ***T***.
M....	Input. M = + *m* ... The *m* largest eigenvalues desired. M = - *m* .... The *m* smallest eigenvalues desired.
E.....	Output. Eigenvalues. E is a one- dimensional array of size *m*.
EV.....	Output. Eigenvectors. Eigenvectors are stored in columns of EV. EV (K,M) is a two-dimensional array.
K.....	Input. Adjustable dimension of array EV.
VW......	Work area. VW is a one-dimensional array of size 5*n*.
ICON ..	Output. Condition code. See Table TEIG2-1.

Table TEIG2-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	E(1)=D(1), EV (1, 1) = 1.0
15000	After determining *m* eigenvalues, all of their corresponding eigenvectors could not be determined.	The eigenvectors that could not be obtained become zero vectors.
20000	None of the eigenvectors could be determined.	All of the eigenvectors become zero vectors.
30000	M = 0, N < |M| or K < N	Bypassed

Comments

- Subprograms used
  SSL II ... AMACH, MGSSL, and UTEG2
  FORTRAN basic functions ... ABS, AMAX1 and IABS

- Notes
  This subroutine is used for real symmetric tridiagonal matrices. When determining *m* eigenvalues/eigenvectors of a real symmetric matrix, subroutine SEIG2 should be used instead.
  When determining *m* eigenvalues of a tridiagonal matrix without their corresponding eigenvectors, subroutine BSCT1 should be used.

- Example
  The *m* largest or *m* smallest eigenvalues of an *n*-order real symmetric tridiagonal matrix are determined, then the corresponding eigenvectors are determined. $n \le 100$, $m \le 10$.

```
C     **EXAMPLE**
      DIMENSION D(100),SD(100),E(10),
     *          EV(100,10),VW(500)
   10 CONTINUE
      READ(5,500) N,M
      IF(N.EQ.0) STOP
      READ(5,510) (D(I),SD(I),I=1,N)
      WRITE(6,600) N,M,(I,D(I),SD(I),I=1,N)
      CALL TEIG2(D,SD,N,M,E,EV,100,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) GO TO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,100,N,MM)
      GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(2E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',5X,
     *'N=',I3,'M=',I3/'0',20X,
     *'***DIAGONAL***','**SUBDIAGONAL*'//
     *(13X,I3,5X,2(E14.7,5X)))
  610 FORMAT('0',20X,'ICON=',I5)
      END
```

In this example, subroutine SEPRT is used to print eigenvalues and corresponding eigenvectors of real symmetric matrices. For details, see the example in section SEIG1.

Method

The *m* largest or *m* smallest eigenvalues and corresponding eigenvectors of an *n*-order real symmetric tridiagonal matrix are determined.

The *m* eigenvalues are determined using the bisection method and corresponding eigenvectors are determined using the inverse iteration method.

Refer to the section BSCT1 for a description of the bisection method. The inverse iteration method is discussed below. Let the eigenvalues of tridiagonal matrix $\boldsymbol{T}$ are

$$\lambda_1 > \lambda_2 > \lambda_3 \cdots\cdot > \lambda_n \tag{4.1}$$

When μ_j is obtained as an approximation for λ_j using the bisection method, consider the determination of corresponding eigenvectors using the inverse iteration method.

In the inverse iteration method (4.2) is iteratively solved. When convergence condition has been satisfied, $\boldsymbol{x}_r$ is considered to be an eigenvector.

$$(\boldsymbol{T} - \mu_j \boldsymbol{I})\boldsymbol{x}_r = \boldsymbol{x}_{r-1},\ r = 1,2,... \tag{4.2}$$

(x_0 is an appropriate initial vector)

Let the eigenvectors which correspond to the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ be $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_n$ and the appropriate initial vector $\boldsymbol{x}_0$ can be represented as a linear combination:

$$\boldsymbol{x}_0 = \sum_{i=1}^{n} \alpha_i \boldsymbol{u}_i \tag{4.3}$$

From (4.2) and (4.3), $\boldsymbol{x}_r$ is written as

$$\boldsymbol{x}_r = \frac{1}{(\lambda_j - \mu_j)^r}\left[\alpha_j \boldsymbol{u}_j + \sum_{\substack{i=1\\ i\neq j}}^{n} \alpha_i \boldsymbol{u}_i (\lambda_j - \mu_j)^r / (\lambda_i - \mu_j)^r\right] \tag{4.4}$$

Since in general $\left|(\lambda_j - \mu_j)/(\lambda_i - \mu_j)\right| << 1.0$, if $\alpha_j \neq 0$ in (4.4), as r grows greater, $\boldsymbol{x}_r$ tends rapidly to $\alpha_j \boldsymbol{u}_j$.

The system of linear equations of (4.2) are solved using (4.5) after decomposition of ($\boldsymbol{T} - \mu_j\boldsymbol{I}$) lower triangular matrix $\boldsymbol{L}$ and upper triangular matrix $\boldsymbol{U}$.

$$\boldsymbol{LUx}_r = \boldsymbol{Px}_{r-1} \tag{4.5}$$

($\boldsymbol{P}$ is a permutation matrix used for pivoting.) (4.5) can be solved as follows

$$\boldsymbol{Ly}_{r-1} = \boldsymbol{Px}_{r-1} \quad \text{(forward substitution)} \tag{4.6}$$

$$\boldsymbol{Ux}_r = \boldsymbol{y}_{r-1} \quad \text{(backward substitution)} \tag{4.7}$$

Since any vector can be used for initial vector $\boldsymbol{x}_0$. $\boldsymbol{x}_0$ may be given such that $\boldsymbol{y}_0$ of (4.8) has a form such as $\boldsymbol{y}_0 = (1, 1, ...\ 1)^T$.

$$\boldsymbol{y}_0 = \boldsymbol{L}^{-1}\boldsymbol{Px}_0 \tag{4.8}$$

Therefore, in the first iteration the forward substitution of (4.6) can be omitted, and by repeating forward substitution and backward substitution for the second and following iterations, eigenvectors can be determined.

- The initial vector and convergence criterion used in the inverse iteration method. In this routine, the following is used for the initial vector;

$$\boldsymbol{y}_0 = \left(\sqrt{n}\cdot u\|\boldsymbol{T}\|, \sqrt{n}\cdot u\|\boldsymbol{T}\|, \cdots \sqrt{n}\cdot u\|\boldsymbol{T}\|\right)^T \tag{4.9}$$

where u is the unit round-off

$$\|\boldsymbol{T}\| = \sum_{i=1}^{n}\left(|t_{ii}| + |t_{i-1i}|\right)$$

$$\boldsymbol{T} = (t_{ij}), \quad t_{01} = 0$$

At each iteration, $\boldsymbol{x}_{r-1}$ is normalized such that

$$\|\boldsymbol{x}_{r-1}\|_1 = \|\boldsymbol{y}_0\|_2 \tag{4.10}$$

When $\boldsymbol{x}_r$ satisfies

$$\|\boldsymbol{x}_r\|_1 \geq 1 \tag{4.11}$$

$\boldsymbol{x}_r$ is accepted as an eigenvector. (4.11) can be discussed as follows. From (4.2), by normalization of $\boldsymbol{x}_r$ (4.12) is obtained

$$(\boldsymbol{T} - \mu_j\boldsymbol{I})\boldsymbol{x}_r / \|\boldsymbol{x}_r\|_1 = \boldsymbol{x}_{r-1} / \|\boldsymbol{x}_r\|_1 \tag{4.12}$$

The right hand side of (4.12); $\boldsymbol{x}_{r-1}/\|\boldsymbol{x}_r\|_1$ corresponds to the residual vector. When the norm of this residual vector satisfies.

$$\|\boldsymbol{x}_{r-1}\|_1 / \|\boldsymbol{x}_r\|_1 \leq \|\boldsymbol{y}_0\|_2 \tag{4.13}$$

In other words, when this residual vector is considered zero vector, $\boldsymbol{x}_r$ can be considered to have converged to an eigenvector. From (4.10), (4.13) becomes (4.11). In this routine, when five iterations are performed and $\boldsymbol{x}_5$ does not satisfy (4.11) ICON is set to 15000 to indicate that the eigenvector was not determined, and the elements of the corresponding column in EV is set to zero.

- Orthogonalization of eigenvectors

  The eigenvectors of a real symmetric tridiagonal matrix should be orthogonal. However, a disadvantage of the inverse iteration method is that eigenvectors corresponding to close eigenvalues may not be satisfactorily orthogonal. Therefore to insure orthogonal eigenvectors,this routine performs the following processing.

  When the eigenvalue μ_i is being determined, μ_i and the previously computed eigenvalue μ_{i-1} are tested to see whether they satisfy.

$$\mu_i - \mu_{i-1} \leq 10^{-3}\|T\| \tag{4.15}$$

If (4.15) is satisfied, the approximate eigenvector $\boldsymbol{x}_i$ obtained from μ_i is made orthogonal to the eigenvector $\boldsymbol{x}_{i-1}$ obtained from μ_{i-1} so as to satisfy

$$(\boldsymbol{x}_i, \boldsymbol{x}_{i-1}) = 0 \tag{4.16}$$

Similarly, if several consecutive eigenvalues satisfy (4.15), they are handled as a group, and their corresponding eigenvectors orthogoalized.

- Direct sum of submatrices
  If $\boldsymbol{T}$ splits a sum of m submatrices $\boldsymbol{T}_1$, $\boldsymbol{T}_2$, ..., $\boldsymbol{T}_m$, the eigenvalues and corresponding eigenvectors of $\boldsymbol{T}$ are respectively, the diagonal elements of $\boldsymbol{D}$ and the column vectors of $\boldsymbol{Q}$ shown in (4.17).

$$\left.\begin{aligned} \boldsymbol{D} &= \begin{bmatrix} \boldsymbol{D}_1 & & \boldsymbol{0} \\ & \boldsymbol{D}_2 & \\ & & \ddots \\ \boldsymbol{0} & & & \boldsymbol{D}_m \end{bmatrix} \\ \boldsymbol{Q} &= \begin{bmatrix} \boldsymbol{Q}_1 & & \boldsymbol{0} \\ & \boldsymbol{Q}_2 & \\ & & \ddots \\ \boldsymbol{0} & & & \boldsymbol{Q}_m \end{bmatrix} \end{aligned}\right\} \tag{4.17}$$

Where $\boldsymbol{D}$ and $\boldsymbol{Q}$ satisfy

$$\boldsymbol{D} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{T}\boldsymbol{Q} \tag{4.18}$$

From (4.17) and (4.18) the following are obtained;

$$\left.\begin{aligned} \boldsymbol{D}_1 &= \boldsymbol{Q}_1^{\mathrm{T}}\boldsymbol{T}_1\boldsymbol{Q}_1 \\ \boldsymbol{D}_2 &= \boldsymbol{Q}_2^{\mathrm{T}}\boldsymbol{T}_2\boldsymbol{Q}_2 \\ &: \\ &: \\ \boldsymbol{D}_m &= \boldsymbol{Q}_m^{\mathrm{T}}\boldsymbol{T}_m\boldsymbol{Q}_m \end{aligned}\right\} \tag{4.19}$$

Thus, the eigenvalues and corresponding eigenvectors of $\boldsymbol{T}$ can be obtained by determining the eigenvalues and corresponding eigenvectors of each submatrix. For more information, see references [12] and [13] pp. 4l8-439.

G21-21-0101 TRAP, DTRAP

Integration of a tabulated function by trapezoidal rule (unequally spaced)
CALL TRAP (X, Y, N, S, ICON)

Function

Given unequally spaced at points $x_1, x_2, ..., x_n$ ($x_1 < x_2 < ... < x_n$) and corresponding function values $y_i = f(x_i)$, i=1, 2, ..., n, this subroutine obtains

$$S = \int_{x_1}^{x_n} f(x)dx\ , \quad n \geq 2$$

using the trapezoidal rule.

Parameters

X	Input. Discrete points x_i One-dimensional array of size n.
Y.....	Input. Function values y_i One-dimensional array of size n.
N.......	Input. Number of discrete points n.
S.....	Output. Integral S.
ICON...	Output. Condition code. See Table TRAP-1.

Table TRAP-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	$n < 2$ or $x_i \geq x_{i+1}$	S is set to 0.0.

Comments on use

- Subprograms used
  SSL II ..... MGSSL
  FORTRAN basic function ... none

- Notes
  Even when discrete points x_i are equally spaced, this subroutine can be used, however, it is preferable to use Simpson's rule (subroutine SIMP1).

- Example
  Discrete points x_i and function values y_i are input, and the integral S is determined.

```
C      **EXAMPLE**
       DIMENSION X(20),Y(20)
       READ(5,500) N
       READ(5,510) (X(I),Y(I),I=1,N)
       CALL TRAP(X,Y,N,S,ICON)
       WRITE(6,600) ICON,S
       STOP
  500 FORMAT(I2)
  510 FORMAT(2F10.0)
  600 FORMAT(10X,'ILL CONDITION =',I5
     * /10X,'INTEGRAL VALUE =',E15.7)
       END
```

Method

The integral is approximated, in this subroutine, by trapezoidal rule:

$$\int_{x_1}^{x_n} f(x)dx \approx \frac{1}{2}\{\ (x_2 - x_1)(f(x_1) + f(x_2)) + (x_3 - x_2)(f(x_2) + f(x_3)) + \cdots + (x_n - x_{n-1}) (f(x_n) + f(x_{n-1}))\ \} = \frac{1}{2}\{\ (x_2 - x_1)f(x_1) + (x_3 - x_1)f(x_2) + \cdots + (x_n - x_{n-2})f(x_{n-1}) + (x_n - x_{n-1})f(x_n)\} \qquad (4.1)$$

For further information, refer to [46] pp. 114-121.

B21-21-0802 TRBK, DTRBK

Back transformation of the eigenvectors or a tridiagonal matrix to the eigenvectors of a real symmetric matrix
CALL TRBK (EV, K, N, M, P, ICON)

Function

Back transformation is applied to m eigenvectors of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ to form eigenvectors of real symmetric matrix $\boldsymbol{A}$. $\boldsymbol{T}$ must have been obtained by the Householder reduction of $\boldsymbol{A}$.
$1 \le m \le n$.

Parameters

EV.....	Input. m eigenvectors of real symmetric tridiagonal matrix $\boldsymbol{T}$. EV(K,M) is a two dimensional array. Output. Eigenvectors of real symmetric matrix $\boldsymbol{A}$.
K.....	Input. Adjustable dimension of array EV ($\ge n$)
N....	Input. Order n of the real symmetric tridiagonal matrix.
M.....	Input. Number m of eigenvectors.
P.....	Input. Transformation matrix obtained by the Householder's reduction (See "Comments on use".) P is a one-dimensional array of size $n(n+1)/2$.
ICON...	Output. Condition code. See Table TRBK-1.

Table TRBK-l Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	EV (1, 1) = 1.0
30000	N < |M|, K < N, or M = 0	Bypassed

Comments on use

- Subprograms used
  SSL II... MGSSL
  FORTRAN basic function... IABS

- Notes
  This subroutine is called usually after subroutine TRID1.
  Parameter A provided by TRID1 can be used as input parameter P of this subroutine. For detailed information about array P refer to subroutine TRID1.
  The eigenvectors are normalized when $\|\boldsymbol{x}\|_2 = 1$, and if parameter M is negative, the absolute value is used.

- Example
  TRID1 is first used to reduce an n-order real symmetric matrix to a tridiagonal matrix, then TEIG2 is used to obtain the eigenvalues and corresponding eigenvectors from the tridiagonal matrix, and finally this subroutine is used to back transform the resultant eigenvectors to form the eigenvectors of the real symmetric matrix. $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),D(100),SD(100),
     *          E(100),EV(100,100),VW(500)
   10 READ(5,500) N,M
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N,M
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      CALL TRID1(A,N,D,SD,ICON)
      IF(ICON.EQ.30000) GOTO 10
      CALL TEIG2(D,SD,N,M,E,EV,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GOTO 10
      CALL TRBK(EV,100,N,M,A,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GOTO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,100,N,MM)
      GOTO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'
     * //11X,'** ORDER =',I5,10X,'** M =',
     * I3/)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT(/11X,'** CONDITION CODE =',I5/)
      END
```

In this example, subroutine SEPRT is used to print the eigenvalues and corresponding eigenvectors of real symmetric matrices. For details see the example in section SEIG1.

Method

m eigenvectors of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are back transformed to obtain the eigenvectors of real symmetrical matrix $\boldsymbol{A}$. The reduction of $\boldsymbol{A}$ to $\boldsymbol{T}$ is performed using the Householder method which performs n-2 orthogonal similarity transformations shown in (4.1).

$$\boldsymbol{T} = \boldsymbol{P}_{n-2}^{\mathrm{T}} \cdots \boldsymbol{P}_2^{\mathrm{T}} \boldsymbol{P}_1^{\mathrm{T}} \boldsymbol{A} \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \tag{4.1}$$

For more details, refer to "Method" for subroutine TRID1.

Let the eigenvalues and corresponding eigenvectors of $\boldsymbol{T}$ be λ and $\boldsymbol{y}$ respectively, then

$$\boldsymbol{T}\boldsymbol{y} = \lambda \boldsymbol{y} \tag{4.2}$$

If (4.1) is substituted in (4,2), then

$$\boldsymbol{P}_{n-2}^{\mathrm{T}} \cdots \boldsymbol{P}_2^{\mathrm{T}} \boldsymbol{P}_1^{\mathrm{T}} \boldsymbol{A} \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{y} = \lambda \boldsymbol{y} \qquad (4.3)$$

If both sides of (4.3) are pre-multiplied by $\boldsymbol{P}_1\boldsymbol{P}_2 ... \boldsymbol{P}_{n-2}$.

$$\boldsymbol{A}\boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{y} = \lambda \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{y} \qquad (4.4)$$

The eigenvectors $\boldsymbol{x}$ of $\boldsymbol{A}$ are then obtained as

$$\boldsymbol{x} = \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{y} \qquad (4.5)$$

(4.5) is calculated as shown in (4.6) by setting $\boldsymbol{y} = \boldsymbol{x}_{n-1}$ (where $\boldsymbol{x} = \boldsymbol{x}_1$).

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{P}_k \boldsymbol{x}_{k+1} \\ &= \boldsymbol{x}_{k+1} - \left(\boldsymbol{u}_k \boldsymbol{u}_k^{\mathrm{T}} / h_k\right) \boldsymbol{x}_{k+1} \quad , k = n-2, \cdots, 2, 1 \end{aligned} \qquad (4.6)$$

Since the eigenvectors $\boldsymbol{x}$ are obtained from (4.5), $\|\boldsymbol{x}\|_2=1$. For further information reference [13] pp. 212-226.

B21-25-0402 TRBKH, DTRBKH

Back transformation of the eigenvectors of a tridiagonal matrix to the eigenvectors of a Hermitian matrix.
CALL TRBKH (EVR, EVI, K, N, M, P, PV, ICON)

Functions

m number of eigenvectors ***y*** of *n* order real symmetric tridiagonal matrix ***T*** is back transformed to the eigenvectors ***x*** of Hermitian matrix ***A***.

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{V}^{*}\boldsymbol{y}$$

where ***P*** and ***V*** are transformation matrices when transformed from ***A*** to ***T*** by the Householder's reduction and diagonal unitary transformation, respectively, and $1 \le m \le n$.

Parameters

EVR...	Input. *m* eigenvectors ***y*** of *n* order real symmetric tridiagonal matrix ***T***. Output. Real part of eigenvector ***x*** of *n* order Hermitian matrix ***A***. Two dimensional array, EVR (K, M) (See "Comments on use").
EVI.....	Output. Imaginary part of eigenvector ***x*** of *n* order Hermitian matrix ***A***. Two dimensional array, EVI (K,M) (See "Comments on use").
K.....	Input. Adjustable dimension of arrays EVR and EVI. ($\ge n$)
N.....	Input. Order *n* of the real symmetric tridiagonal matrix.
M....	Input. The number ***P*** of eigenvectors. (See "Comments on use").
P.....	Input. Transformation vector ***T*** obtained by the Householder's reduction from ***A*** to ***T***. In the compressed storage mode for a Hermitian matrix. Two dimensional array, P (K, N). (See "Comments on use").
PV...	Input. Transformation matrix ***V*** obtained by diagonal unitary transformation from ***A*** to ***T***. One dimensional array of size 2 *n*. (See "Comments on use").
ICON....	Output. Condition code. See Table TRBKH-1.

Table TRBKH-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N = 1	EV(1, 1) = 1.0
30000	N < |M|, K < N or M = 0	Bypassed

Comments on use

- subprograms used
  SSL II ... MGSSL
  FORTRAN basic function.... IABS

- Notes
  This subroutine is for a Hermitian matrix and not to be applied to a general complex matrix.
  Note that array P does not directly represent transformation matrix ***P*** for reduction of ***A*** to ***T***.
  This subroutine, TRBKH, is normally used together with subroutine TRIDH. Consequently the contents of A and PV output by subroutine TRIDH can be used as the contents of parameters P and PV in this subroutine. The contents of array P and PV are explained on "Method" for subroutine TRIDH.
  If input eigenvector ***y*** is normalized such as $\|\boldsymbol{y}\|_2 = 1$, then output eigenvector ***x*** is normalized as $\|\boldsymbol{x}\|_2 = 1$.
  The *l*-th element of the eigenvector that corresponds to the *j*-th eigenvalue is represented by EVR (L, J) + $i \cdot$ EVI (L, J), where $i = \sqrt{-1}$
  When parameter M is negative, its absolute value is taken and used.

- Example
  An n order Hermitian matrix is reduced to a real symmetric tridiagonal matrix using subroutine TRIDH, and *m* number of eigenvalues and eigenvectors are obtained using TEIG2.
  Lastly by this subroutine, TRBKH, the eigenvectors of the real symmetric tridiagonal matrix are back transformed to the eigenvectors of the Hermitian matrix.
  $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),D(100),SD(100),
     * V(200),EVR(100,100),EVI(100,100),
     * VW(500),E(100)
   10 CONTINUE
      READ(5,500) N,M
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,M
      DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
      CALL TRIDH(A,100,N,D,SD,V,ICON)
      WRITE(6,620) ICON
      IF(ICON.EQ.30000) GO TO 10
      CALL TEIG2(D,SD,N,M,E,EVR,100,VW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      CALL TRBKH(EVR,EVI,100,N,M,A,V,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      MM=IABS(M)
      CALL HEPRT(E,EVR,EVI,100,N,MM)
      GO TO 10
```

```
500 FORMAT(2I5)
510 FORMAT(4E15.7)
600 FORMAT('1'///40X,'**ORIGINAL ',
   * 'MATRIX**',5X,'N=',I3,5X,'M=',I3//)
610 FORMAT(/4(5X,'A(',I3,',',I3,')=',
   * E14.7))
620 FORMAT('0',20X,'ICON=',I5)
    END
```

Subroutine HEPRT in this example is used to print eigenvalue and eigenvectors of a Hermitian matrix. For its details, refer to the example for subroutine HEIG2.

Method

The m number of eigenvectors of n order real symmetric tridiagonal matrix $\boldsymbol{T}$ are back transformed to the eigenvectors of Hermitian matrix $\boldsymbol{A}$. Reduction of $\boldsymbol{A}$ to $\boldsymbol{T}$ is done by the Householder method and diagonal unitary transformation as shown in the eq. (4.1).

$$\boldsymbol{T} = \boldsymbol{V}^* \left(\boldsymbol{P}_{n-2}^* \cdots \boldsymbol{P}_2^* \boldsymbol{P}_1^* \boldsymbol{A} \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2}\right) \boldsymbol{V} \tag{4.1}$$

where $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^* / h_k$ is a diagonal unitary matrix. For detailed information, refer to subroutine TRIDH.

Letting λ and $\boldsymbol{y}$ be eigenvalue and eigenvector respectively of $\boldsymbol{T}$, then eq. (4.2) is established.

$$\boldsymbol{Ty} = \lambda \boldsymbol{y} \tag{4.2}$$

Substituting eq. (4.1) into (4.2),

$$\boldsymbol{V}^* \boldsymbol{P}_{n-2}^* \cdots \boldsymbol{P}_2^* \boldsymbol{P}_1^* \boldsymbol{A} \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{Vy} = \lambda \boldsymbol{y} \tag{4.3}$$

Premultiplying $P_1\ P_2\ \dots\ P_{n\text{-}2}\ V$ to both sides of eq. (4.3)

$$\boldsymbol{A} \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{Vy} = \lambda \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{Vy} \tag{4.4}$$

Therefore eigenvector $\boldsymbol{x}$ of $\boldsymbol{A}$ is obtained such that

$$\boldsymbol{x} = \boldsymbol{P}_1 \boldsymbol{P}_2 \cdots \boldsymbol{P}_{n-2} \boldsymbol{Vy} \tag{4.5}$$

The calculation of eq. (4.5) is carried out in the sequence shown in eqs. (4.6) and (4.7)

$$z = \boldsymbol{Vy} \tag{4.6}$$

$$z_k = z_{k+1} - \boldsymbol{u}_k \boldsymbol{u}_k^* z_{k+1} / h_k \qquad k = n-2, \dots, 2, 1 \tag{4.7}$$

where $z = z_{n\text{-}1}$ and $\boldsymbol{x} = z_1$.
Thus obtained eigenvector $\boldsymbol{x}$ ($= z_1$) is normalized as $\|\boldsymbol{x}\|_2 = 1$ if $\|\boldsymbol{y}\|_2 = 1$ in eq. (4.5). Refer to subroutine TRIDH for the Householder method and diagonal unitary transformation.
For details, see Reference [17].

B21-25-0302 TRIDH, DTRIDH

Reduction of a Hermitian matrix to a real symmetric tridiagonal matrix (Householder method, and diagonal unitary transformation).
CALL TRIDH (A, K, N, D, SD, PV, ICON)

Function

An n-order Hermitian matrix $\boldsymbol{A}$ is reduced first to a Hermitian tridiagonal matrix $\boldsymbol{H}$,

$$\boldsymbol{H} = \boldsymbol{P}^{*}\boldsymbol{AP} \qquad (1.1)$$

by the Householder method, then it is further reduced to a real symmetric tridiagonal matrix $\boldsymbol{T}$ by diagonal unitary transformation.

$$\boldsymbol{T} = \boldsymbol{V}^{*}\boldsymbol{HV} \qquad (1.2)$$

where $\boldsymbol{P}$ and $\boldsymbol{V}$ are transformation matrices, and n≥1.

Parameters

A.....	Input. Hermitian matrix $\boldsymbol{A}$. Output. Transformation matrix $\boldsymbol{P}$. (Refer to Fig. TRIDH-1). In the compressed storage mode for Hermitian matrices. (See "Comments on use"). Two dimensional arrays of size A (K, N).
K.....	Input. Adjustable dimension of the array A (≥ n)
N.....	Input. Order n of the Hermitian matrix.
D.....	Output. Main diagonal elements of real symmetric tridiagonal matrix $\boldsymbol{T}$. (See Fig. TRIDH-2). One dimensional array of size n.
SD....	Output. Subdiagonal elements of real symmetric tridiagonal matrix $\boldsymbol{T}$. (Refer to Fig. TRIDH-2). One dimensional array of size n. The outputs are stored into SD (2) to SD (N) and SD (1) is set to zero.
PV.....	Output. Transformation vector $\boldsymbol{V}$ (Refer to Fig. TRIDH-3). One dimensional array of size 2 n.
ICON..	Output. Condition code. See Table TRIDH-1.

Table TRIDH-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	Reduction is not performed.
30000	K < N or N < 1	Bypassed

×	×	$\mathrm{Im}(a_{31}^{(3)})$	$\mathrm{Im}(a_{41}^{(2)})$	$\mathrm{Im}(a_{51}^{(1)})$
×	×	$\mathrm{Im}(a_{32}^{(2)})$ $\bullet(1+\sigma_3/\tau_3)$	$\mathrm{Im}(a_{42}^{(2)})$	$\mathrm{Im}(a_{52}^{(1)})$
$\mathrm{Re}(a_{31}^{(3)})$	$\mathrm{Re}(a_{32}^{(3)})$ $\bullet(1+\sigma_3/\tau_3)$	$h_3^{\frac{1}{2}}$	$\mathrm{Im}(a_{43}^{(2)})$ $\bullet(1+\sigma_2/\tau_2)$	$\mathrm{Im}(a_{53}^{(1)})$
$\mathrm{Re}(a_{41}^{(2)})$	$\mathrm{Re}(a_{42}^{(2)})$	$\mathrm{Re}(a_{43}^{(2)})$ $\bullet(1+\sigma_2/\tau_2)$	$h_2^{\frac{1}{2}}$	$\mathrm{Im}(a_{54}^{(1)})$ $\bullet(1+\sigma_1/\tau_1)$
$\mathrm{Re}(a_{51}^{(1)})$	$\mathrm{Re}(a_{52}^{(1)})$	$\mathrm{Re}(a_{53}^{(1)})$	$\mathrm{Re}(a_{54}^{(1)})$ $\bullet(1+\sigma_1/\tau_1)$	$h_1^{\frac{1}{2}}$

Note: The symbol "×" indicates a work area. This is for n=5 Refer to eq. (4.11) or (4.13) to h_k.

Fig. TRIDH-l Elements of the array A after reduction

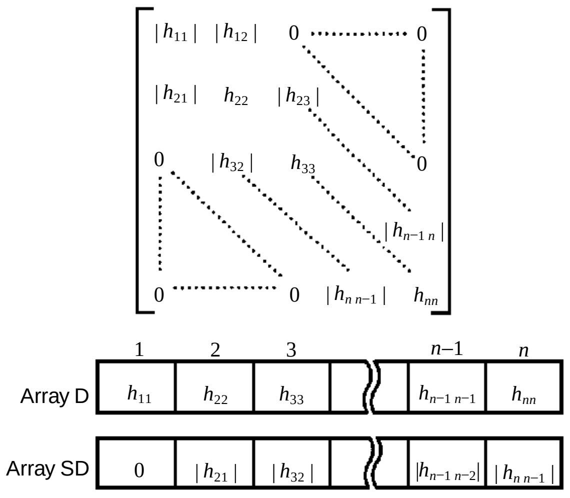

$|h_{11}|$ $|h_{12}|$ 0 0
$|h_{21}|$ h_{22} $|h_{23}|$
0 $|h_{32}|$ h_{33} 0
$|h_{n-1\,n}|$
0 0 $|h_{n\,n-1}|$ h_{nn}
1 2 3 n−1 n
Array D h_{11} h_{22} h_{33} $h_{n-1\,n-1}$ h_{nn}
Array SD 0 $|h_{21}|$ $|h_{32}|$ $|h_{n-1\,n-2}|$ $|h_{n\,n-1}|$

Fig. TRIDH-2 Correspondence between real symmetric tridiagonal matrix $\boldsymbol{T}$ and arrays D and SD

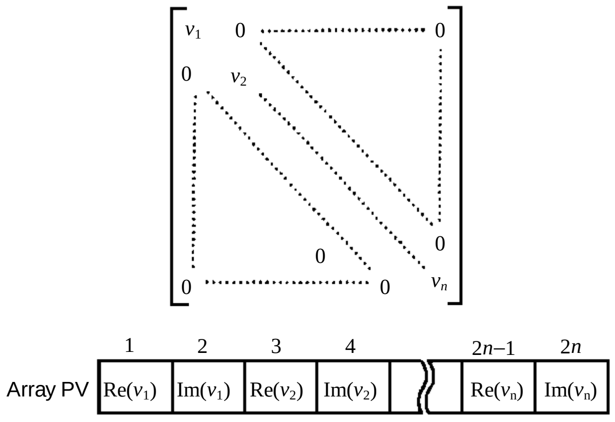

v_1 0 0
0 v_2
0 0
0 0 v_n
1 2 3 4 2n−1 2n
Array PV Re(v_1) Im(v_1) Re(v_2) Im(v_2) Re(v_n) Im(v_n)

Fig. TRIDH-3 Correspondence between transformation matrix $\boldsymbol{V}$ and Array PV

Comments on use

- Subprograms used
  SSL II ... AMACH, BSUM, and MGSSL
  FORTRAN basic functions... ABS and SQRT

- Notes
  This subroutine is used for a Hermitian matrix, and not for a general complex matrix.
  Output arrays A and PV will be needed to obtain eigenvectors of the Hermitian matrix ***A***. These A and PV correspond respectively to P and PV in subroutine TRBKH which is used to obtain eigenvectors of the Hermitian matrix.
  The accuracy of the eigenvalues is determined when tridiagonalization is made. Consequently, the tridiagonal matrix must be obtained as accurately as possible, and this subroutine takes this into account. However, when there are both very large and small eigenvalues, the smaller eigenvalues tend to be more affected by the transformation compared to the larger ones. Therefore, in this situation, it may be difficult to obtain the small eigenvalues with good accuracy.

- Example
  An *n*-order Hermitian matrix is reduced to a real symmetric tridiagonal matrix, then by subroutine TRQL, its eigenvalues are computed for $n \le 100$.

```
C      **EXAMPLE**
       DIMENSION A(100,100),D(100),SD(100),
      *V(200),E(100)
   10 CONTINUE
       READ(5,500) N,M
       IF(N.EQ.0) STOP
       READ(5,510) ((A(I,J),I=1,N),J=1,N)
       WRITE(6,600) N,M
       DO 20 I=1,N
   20 WRITE(6,610) (I,J,A(I,J),J=1,N)
       CALL TRIDH(A,100,N,D,SD,V,ICON)
       WRITE(6,620)
       WRITE(6,630) ICON
       IF(ICON.EQ.30000) GO TO 10
       WRITE(6,640)
       WRITE(6,650) (I,D(I),SD(1),I=1,N)
       CALL TRQL(D,SD,N,E,M,ICON)
       WRITE(6,630) ICON
       IF(ICON.GE.20000) GO TO 10
       WRITE(6,660)
       WRITE(6,670) (I,E(I),I=1,M)
       GO TO 10
  500 FORMAT(2I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1'///,40X,'**ORIGINAL ',
      *'MATRIX**',5X,'N=',I3,5X,'M=',I3//)
  610 FORMAT(4(5X,'A(',I3,',',I3,')=',
      *E14.7))
  620 FORMAT('0'//,11X,'**TRIDIAGONAL ',
      *'MATRIX**')
  630 FORMAT('0',20X,'ICON=',I5)
  640 FORMAT('0',//1X,3('NO.',5X,'DIAGONAL',
      *7X,'SUBDIAGONAL',2X),//)
  650 FORMAT(1X,3(I3,2E17.7))
  660 FORMAT('1'///,5X,'**EIGENVALUES**'/)
  670 FORMAT(1X,4('E(',I3,')=',E17.7))
       END
```

Method

The reduction of an *n*-order Hermitian matrix ***A*** to a real symmetric tridiagonal matrix ***T*** is done in two steps shown in eqs. (4.1) and (4.2).

- The *n*-order Hermitian matrix ***A*** is reduced to an *n*-order Hermitian tridiagonal matrix ***H*** by the Householder method. This is performed through *n*-2 iteration of the unitary similarity transformation.

$$\boldsymbol{A}_{k+1} = \boldsymbol{P}_k^* \boldsymbol{A}_k \boldsymbol{P}_k \quad ,k = 1,2,...,n-2 \tag{4.1}$$

  where $\boldsymbol{A}_1 = \boldsymbol{A}$

  This is the same equation as that of the orthogonal similarity transformation for a real symmetric matrix except that $\boldsymbol{P}_k^T$ is replaced by $\boldsymbol{P}_k^*$ (Refer to TRID1 for the Householder method of real symmetric matrix). The resultant matrix $\boldsymbol{A}_{n-1}$ in those transformations becomes a Hermitian tridiagonal matrix ***H***.

- The Hermitian tridiagonal matrix ***H*** obtained in eq. (4.1) is further reduced to an *n*-order real symmetric tridiagonal matrix ***T*** by the diagonal unitary similarity transformation (4.2).

$$\boldsymbol{T} = \boldsymbol{V}^* \boldsymbol{H}\boldsymbol{V} \qquad \text{where } \boldsymbol{H} = \boldsymbol{A}_{n-1} \tag{4.2}$$

$$\boldsymbol{V} = \mathrm{diag}(v_j) \tag{4.3}$$

  Let $\boldsymbol{H} = (h_{ij})$ and

$$v_1 = 1$$

$$v_j = h_{jj-1} v_{j-1} / \left| h_{jj-1} \right|, \quad j = 2,...,n \tag{4.4}$$

  Then from transformation (4.2), each element t_{ij} of ***T*** is given as a real number as follows:

$$t_{jj} = v_j^* h_{jj} v_j = h \tag{4.5}$$

$$\begin{aligned} t_{jj-1} &= v_j^* h_{jj-1} v_{j-1} \\ &= \left(h_{jj-1}^* v_{j-1}^* / \left| h_{jj-1} \right| \right) h_{jj-1} v_{j-1} \\ &= \left| h_{jj-1} \right| \end{aligned} \tag{4.6}$$

  That is, if the absolute value of h_{jj-1} is taken in the course of obtaining subdiagonal elements of ***H***, it will directly become subdiagonal element of ***T***.

After above two transformations, takes the form shown in Fig. TRIDH-2. Now, the transformation process for the eqs. (4.1) and (4.2) is explained. The *k*-th step transformation, for $\boldsymbol{A}_k = (a_i^{(k)})$, is represented by

$$\sigma_k = \left| a_{l1}^{(k)} \right|^2 + \left| a_{l2}^{(k)} \right|^2 + \cdots + \left| a_{ll-1}^{(k)} \right|^2 \tag{4.7}$$

where, $l = n - k + 1$

$$t_{ll-1} = \sigma_k^{\frac{1}{2}} \tag{4.8}$$

Here, if $\left|a_{ll-1}^{(k)}\right| \neq 0$

$$\tau_k = \left(\left|a_{ll-1}^{(k)}\right|^2 \sigma_k\right)^{\frac{1}{2}} \tag{4.9}$$

$$\boldsymbol{u}_k^* = \left(a_{l1}^{(k)*},...,a_{ll-2}^{(k)*},a_{ll-1}^{(k)*}\left(1+\sigma_k/\tau_k\right),0,...,0\right) \tag{4.10}$$

$$h_k = \sigma_k + \tau_k \tag{4.11}$$

or if $\left|a_{ll-1}^{(k)}\right| = 0$. eqs. (4.10) and (4.11) are represented by

$$\boldsymbol{u}_k^* = \left(a_{l1}^{(k)*},..., a_{ll-2}^{(k)*}\sigma_k^{\frac{1}{2}},0,...,0\right) \tag{4.12}$$

$$h_k = \sigma_k \tag{4.13}$$

and transformation matrix $\boldsymbol{P}_k$ is constituted as follows in eq.(4.14).

$$\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^* / h_k \tag{4.14}$$

$$\boldsymbol{A}_{k+1} = \boldsymbol{P}_k^* \boldsymbol{A}_k \boldsymbol{P}_k \tag{4.15}$$

$$t_{ll} = a_{ll}^{(k-1)} \tag{4.16}$$

By eq.(4.15), elements $a_{l1}^{(k)},..., a_{ll-2}^{(k)}$ of the Hermitian matrix $\boldsymbol{A}$ are eliminated. In the actual transformation, the following considerations are taken.

– To avoid possible underflows and/or overflows in the computations shown in eqs. (4.7) to (4.14), each element on the right hand side of eq. (4.7) is scaled by

$$\sum_{j=1}^{l-1}\left(\left|\mathrm{Re}\left(a_{lj}^{(k)}\right)\right| + \left|\mathrm{Im}\left(a_{lj}^{(k)}\right)\right|\right) \tag{4.17}$$

– Instead of using the transformation (4.15) after obtaining $\boldsymbol{P}_k$ $\boldsymbol{A}_{k+1}$ is obtained as follows:

$$\boldsymbol{p}_k = \boldsymbol{A}_k \boldsymbol{u}_k / h_k, \quad K_k = \boldsymbol{u}_k^* \boldsymbol{p}_k / 2h_k \tag{4.18}$$

$$\boldsymbol{q}_k = \boldsymbol{p}_k - K_k \boldsymbol{u}_k \tag{4.19}$$

$$\begin{aligned}\boldsymbol{A}_{k+1} &= \left(\boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^* / h_k\right)\boldsymbol{A}_k\left(\boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^* / h_k\right) \\ &= \boldsymbol{A}_k - \boldsymbol{u}_k \boldsymbol{q}_k^* - \boldsymbol{q}_k \boldsymbol{u}_k^* \end{aligned} \tag{4.20}$$

– Transformation matrices $\boldsymbol{P}_k$ and $\boldsymbol{V}$ are needed when obtaining eigenvectors of the Hermitian matrix. For that reason, $\boldsymbol{u}_k$ in eq. (4.10) or (4.12) and square root of h_k in eq. (4.11) or (4.13) are stored in the form in Fig. TRIDH-1. Diagonal elements v_j of diagonal unitary matrix $\boldsymbol{V}$ are stored into one-dimensional array PV as shown in Fig. TRIDH-3.

When input parameter N is 1, the diagonal elements of $\boldsymbol{A}$ are directly put into array D. For details, see References [13] pp.212-226 and [17].

B21-21-0302 TRID1, DTRID1

Reduction of a real symmetric matrix to a real symmetric tridiagonal matrix (Householder method)
CALL TRID1 (A, N, D, SD, ICON)

Function

An n-order real symmetric matrix $\boldsymbol{A}$ is reduced to a real symmetric tridiagonal matrix $\boldsymbol{T}$ using the Householder method (orthogonal similarity transformation).

$$\boldsymbol{T} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{AP} \qquad (1.1)$$

where $\boldsymbol{P}$ is the transformation matrix. $n \geq 1$.

Parameters

A..... Input. Real symmetric matrix $\boldsymbol{A}$.
Output. Transformation matrix $\boldsymbol{P}$.
Compressed storage mode for symmetric matrix.
A is a one-dimensional array of size $n(n+1)/2$.

N..... Input. Order n of real symmetric matrix $\boldsymbol{A}$.

D..... Output. Diagonal elements of tridiagonal matrix $\boldsymbol{T}$. D is a one-dimensional array of size n.

SD..... Output. Subdiagonal elements of tridiagonal matrix $\boldsymbol{T}$. SD is a one-dimensional array of size n. SD(2) – SD(N) is used; SD (1) is set to 0.

ICON .. Output. Condition code. See Table TRID1-1.

Table TRID1-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1 or N=2	Reduction is not performed.
30000	N < 1	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH, and MGSSL
  FORTRAN basic functions ... ABS, DSQRT, and SIGN

- Notes
  Array A that is output is needed for determining the eigenvectors of real symmetric matrix $\boldsymbol{A}$.
  The precision of eigenvalues is determined in the tridiagonal matrix reduction process. For that reason this subroutine has been implemented to calculate tridiagonal matrices with as high a degree of precision as possible. However, when very large and very small eigenvalues exist, the smaller eigenvalues tend to be affected more by the reduction process. In some case, it is very difficult to obtain small eigenvalues precisely.

- Example
  After an n-order real symmetric matrix is reduced to a tridiagonal matrix, subroutine TRQL is used to compute the eigenvalues. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),D(100),
     *          SD(100),E(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N
      NE=0
      DO 20 I=1,N
      NI=NE+1
      NE=NE+I
   20 WRITE(6,610) I,(A(J),J=NI,NE)
      CALL TRID1(A,N,D,SD,ICON)
      WRITE(6,620)
      WRITE(6,630) ICON
      IF(ICON.EQ.30000) GO TO 10
      WRITE(6,640) (I,D(I),SD(I),I=1,N)
      CALL TRQL(D,SD,N,E,M,ICON)
      WRITE(6,650)
      WRITE(6,630) ICON
      IF(ICON.EQ.20000) GO TO 10
      WRITE(6,660) (I,E(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',10X,'** ORIGINAL MATRIX'/
     *       11X,'** ORDER =',I5)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0'/11X,'** TRIDIAGONAL ',
     *       'MATRIX')
  630 FORMAT(11X,'** CONDITION CODE =',I5/)
  640 FORMAT(5X,I5,2E16.7)
  650 FORMAT('0'/11X,'** EIGENVALUES')
  660 FORMAT(5X,'E(',I3,')=',E15.7)
      END
```

Method

Reduction of an n-order real symmetric matrix $\boldsymbol{A}$ to a tridiagonal matrix is performed through n–2 iterations of the orthogonal similarity transformation

$$\boldsymbol{A}_{k+1} = \boldsymbol{P}_k^{\mathrm{T}}\boldsymbol{A}_k\boldsymbol{P}_k, \quad k = 1,2,\ldots,n-2 \qquad (4.1)$$

where $\boldsymbol{A}_1 = \boldsymbol{A}$, $\boldsymbol{P}_k$ is an orthogonal transformation matrix.
The resultant matrix $\boldsymbol{A}_{n-1}$ in above transformations becomes a real symmetric tridiagonal matrix.
The k-th transformation is performed according to the following procedure. Let $\boldsymbol{A}_k = \left(a_{ij}^{(k)}\right)$

$$\sigma_k = \left(a_{l1}^{(k)}\right)^2 + \left(a_{l2}^{(k)}\right)^2 + \cdots + \left(a_{ll-1}^{(k)}\right)^2 \qquad (4.2)$$

Where $l = n - k + 1$

$$\boldsymbol{u}_k^{\mathrm{T}} = \left(a_{l1}^{(k)}, \ldots, a_{ll-2}^{(k)}, a_{ll-1}^{(k)} \pm \sigma_k^{\frac{1}{2}}, 0, \ldots, 0 \right) \qquad (4.3)$$

$$h_k = \sigma_k \pm a_{ll-1}^{(k)} \sigma_k^{\frac{1}{2}} \qquad (4.4)$$

$$\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^{\mathrm{T}} / h_k \qquad (4.5)$$

By applying in $\boldsymbol{P}_k$ in (4.5) to (4.l), $a_{l1}^{(k)}$ to $a_{ll-2}^{(k)}$ of $\boldsymbol{A}_k$ can be eliminated. The following considerations are applied during the actual transformation.

- To avoid possible underflow and overflow in the computations of (4.2) to (4.4), the elements on the right side are scaled by $\sum_{j=1}^{l-1} \left| a_{ij}^{(k)} \right|$.
- When determining $\boldsymbol{u}_k^{\mathrm{T}}$ of (4.3), to prevent cancellation in the calculation of $a_{k+1k}^{(k)} \pm \sigma_k^{\frac{1}{2}}$ the sign of $\pm \sigma_k^{\frac{1}{2}}$ is taken to that of a $a_{k-1k}^{(k)}$.
- Instead of determining $\boldsymbol{P}_k$ of transformation of (4.1), $\boldsymbol{A}_{k+1}$ is determined as follows

$$\boldsymbol{p}_k = \boldsymbol{A}_k \boldsymbol{u}_k / h_k, \quad K_k = \boldsymbol{u}_k^{\mathrm{T}} \boldsymbol{p}_k / 2h_k \qquad (4.7)$$

$$\boldsymbol{q}_k = \boldsymbol{p}_k - K_k \boldsymbol{u}_k \qquad (4.8)$$

$$\begin{aligned} \boldsymbol{A}_{k+1} &= \left(\boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^{\mathrm{T}} / h_k\right) \boldsymbol{A}_k \left(\boldsymbol{I} - \boldsymbol{u}_k \boldsymbol{u}_k^{\mathrm{T}} / h_k\right) \\ &= \boldsymbol{A}_k - \boldsymbol{u}_k \boldsymbol{q}_k^{\mathrm{T}} - \boldsymbol{q}_k \boldsymbol{u}_k^{\mathrm{T}} \qquad (4.9) \end{aligned}$$

Transformation matrix $\boldsymbol{P}_k$ is required to determine the eigenvectors of the real symmetric matrix $\boldsymbol{A}$, so the elements of $\boldsymbol{u}_k$ of (4.3) and h_k of (4.4) are stored as shown in Fig. TRID1-l.

$$\begin{bmatrix} \times & & & & \\ \times & \times & & & \\ a_{31}^{(3)} & a_{32}^{(3)} \pm \sigma_3^{\frac{1}{2}} & h_3 & & \\ a_{41}^{(2)} & a_{42}^{(2)} & a_{43}^{(2)} \pm \sigma_2^{\frac{1}{2}} & h_2 & \\ a_{51}^{(1)} & a_{52}^{(1)} & a_{53}^{(1)} & a_{54}^{(1)} \pm \sigma_1^{\frac{1}{2}} & h_1 \end{bmatrix}$$

(Note): $\times$ represents work area, n=5

Fig. TRID1-1 Array A after Householder's reduction (A is in the compressed storage mode)

When n= 2 or n=1 the diagonal elements and the subdiagonal elements are entered as is in arrays D and SD. For further information see Reference [13] pp.212-226.

B21-21-0402 TRQL, DTRQL

Eigenvalues of a real symmetric tridiagonal matrix (QL method)
CALL TRQL (D, SD, N, E, M, ICON)

Function

All eigenvalues of an n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are determined using the QL method. $n \geq 1$

Parameters

D.....	Input. Diagonal elements of tridiagonal matrix $\boldsymbol{T}$. D is a one-dimensional array of size n. The contents of D are altered on output.
SD....	Input. Subdiagonal elements of tridiagonal matrix $\boldsymbol{T}$. The elements must be be stored in SD(2) to SD (N). The contents of SD are altered on output.
N......	Input. Order n of tridiagonal matrix.
E.....	Output. Eigenvalues E is a one-dimensional array of size n.
M.....	Output. Number of eigenvalues that were obtained.
ICON..	Output. Condition code. See Table TRQL-1.

Table TRQL-1 Condition codes

Code	Meaning	Processing
0	No error	
10000	N=1	E(1)=D(1)
15000	Some of the eigenvalues could not be determined.	M is set to the number of eigenvalues that were determined. 1 ≤ M < n.
20000	Eigenvalues could not be determined.	M=0
30000	N < 1	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH and MGSSL
  FORTRAN basic functions ... ABS, SQRT, and SIGN

- Notes
  Parameter M is set to n when ICON =0 or to the number of eigenvalues that were obtained when ICON = 15000.
  This routine uses the QL method which is best suited for tridiagonal matrices in which the magnitude of the element is graded increasingly.
  This routine is used for tridiagonal matrices. When determining eigenvalues of a real symmetric matrix, first reduce that matrix to a tridiagonal matrix using subroutine TRID1, and then call this routine.
  When determining approximately n/4 or less eigenvalues of a tridiagonal matrix, it is faster to use subroutine BSCT1.
  When eigenvectors of a tridiagonal matrix are also to be determined, TEIG1 should be used.

- Example
  An n-order real symmetric matrix is reduced to a tridiagonal matrix using subroutine TRID1, then this routine is used to obtain the eigenvalues. n≤100.

```
C     **EXAMPLE**
      DIMENSION A(5050),D(100),
     *          SD(100),E(100)
   10 CONTINUE
      READ(5,500) N
      IF(N.EQ.0) STOP
      NN=N*(N+1)/2
      READ(5,510) (A(I),I=1,NN)
      WRITE(6,600) N
      LST=0
      DO 20 I=1,N
      INT=LST+1
      LST=LST+I
      WRITE(6,610) I,(A(J),J=INT,LST)
   20 CONTINUE
      CALL TRID1(A,N,D,SD,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) GO TO 10
      CALL TRQL(D,SD,N,E,M,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,630) (I,E(I),I=1,M)
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT('1',5X,'ORIGINAL MATRIX',10X,
     *       'N=',I3)
  610 FORMAT('0',7X,I3,5E20.7/(11X,5E20.7))
  620 FORMAT('0',20X,'ICON=',I5)
  630 FORMAT('0',5X,'EIGENVALUE'//
     *       (8X,4('E(',I3,')=',E15.7,3X)))
      END
```

Method

All eigenvalues of n-order real symmetric tridiagonal matrix $\boldsymbol{T}$ are determined using the QL method. Before explaining the QL method, the representation of the tridiagonal matrix and its eigenvalues will be defined.

The tridiagonal matrix is represented as $\boldsymbol{T}$ and its eigenvalues as $(\lambda_1, \lambda_2, ..., \lambda_n)$. All eigenvalues are assumed to be different. The diagonal elements of tridiagonal matrix $\boldsymbol{T}$ are represented as $d_1, d_2,..., d_n$ and the subdiagonal elements as $e_1, e_2, ..., e_n$.That is.

$$
\boldsymbol{T}=\begin{bmatrix} d_1 & e_1 & & & 0 \\ e_1 & d_2 & e_2 & & \\ & e_2 & \ddots & \ddots & \\ & & \ddots & \ddots & e_{n-1} \\ 0 & & & e_{n-1} & d_n \end{bmatrix} \qquad (4.1)
$$

To determine eigenvalues, the QL method uses the following three theorems.

- If a tridiagonal matrix $\boldsymbol{T}$ is a real symmetric matrix, such an orthogonal matrix $\boldsymbol{X}$ that satisfies (4.2) exists.

$$\boldsymbol{D} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{T}\boldsymbol{X} \qquad (4.2)$$

where $\boldsymbol{D}$ is a diagonal matrix and diagonal elements of $\boldsymbol{D}$ are all eigenvalues of $\boldsymbol{T}$.

- If $\boldsymbol{T}$ is non-singular, it can be uniquely decomposed into an orthogonal matrix $\boldsymbol{Q}$ and a lower triangular matrix $\boldsymbol{L}$ whose diagonal elements are all positive,

$$\boldsymbol{T} = \boldsymbol{Q}\boldsymbol{L} \qquad (4.3)$$

- Let real symmetric tridiagonal matrix $\boldsymbol{T}$ be $\boldsymbol{T}_1$, and the matrix obtained after the s-th iteration be $\boldsymbol{T}_{s+1}$
  The following iterative calculation can be done:

$$\boldsymbol{T}_s = \boldsymbol{Q}_s\boldsymbol{L}_s \quad , s = 1,2,3,... \qquad (4.4)$$

$$\boldsymbol{T}_{s+1} = \boldsymbol{Q}_s^{\mathrm{T}}\boldsymbol{T}_s\boldsymbol{Q}_s \qquad (4.5)$$

From the orthogonal similarity transformation of (4.5), $\boldsymbol{T}_{s+1}$ becomes a real symmetric tridiagonal matrix. Furthermore, the $\boldsymbol{T}_{s+1}$ is determined only the n-th column of $\boldsymbol{Q}_s$ and when $\boldsymbol{S}_{\infty}$, $\boldsymbol{T}_{s+1}$ converges to a diagonal matrix.

By using the above three theorems, the QL method determines all eigenvalues. However, in the actual calculations of the QL method, (4.4) and (4.5) are calculated in parallel instead of first obtaining $\boldsymbol{Q}_s$ in (4.4) and then determining $\boldsymbol{T}_{s+1}$ of (4.5).
Since from (4.4)

$$\boldsymbol{Q}_s^{\mathrm{T}}\boldsymbol{T}_s = \boldsymbol{L}_s \qquad (4.6)$$

$\boldsymbol{T}_s$ is transformed into a lower triangular matrix $\boldsymbol{L}_s$ when premultiplied by $\boldsymbol{Q}_s\boldsymbol{T}$. Therefore, if $\boldsymbol{P}_i^{(s)\mathrm{T}}\boldsymbol{T}$ is the transformation matrix which eliminates the i-th element of the (i+l)th column of $\boldsymbol{T}_s$, $\boldsymbol{Q}_s^{\mathrm{T}}$ can be represented as

$$\boldsymbol{Q}_s^{\mathrm{T}} = \boldsymbol{P}_1^{(s)\mathrm{T}}\boldsymbol{P}_2^{(s)\mathrm{T}} \cdots \boldsymbol{P}_{n-1}^{(s)\mathrm{T}} \qquad (4.7)$$

Thus, (4.5) can be represented as

$$\boldsymbol{T}_{s+1} = \boldsymbol{P}_1^{(s)\mathrm{T}}\boldsymbol{P}_2^{(s)\mathrm{T}} \cdots \boldsymbol{P}_{n-1}^{(s)\mathrm{T}}\boldsymbol{T}_s\boldsymbol{P}_{n-1}^{(s)} \cdots \boldsymbol{P}_2^{(s)}\boldsymbol{P}_1^{(s)} \qquad (4.8)$$

By successively calculating (4.8) from the inside to outside (4.4) and (4.5) can be executed in parallel. Normally, in order to improve the rate of convergence in the QL method. ($\boldsymbol{T}_s - k\boldsymbol{I}$) origin-shifted by an appropriate constant k is used instead of $\boldsymbol{T}_s$.
Thus (4.4) becomes

$$\left(\boldsymbol{T}_s - k\boldsymbol{I}\right) = \boldsymbol{Q}_s\boldsymbol{L}_s \qquad (4.9)$$

and (4.5) becomes

$$\left(\boldsymbol{T}_{s+1} - k\boldsymbol{I}\right) = \boldsymbol{Q}_s^{\mathrm{T}}\left(\boldsymbol{T}_s - k\boldsymbol{I}\right)\boldsymbol{Q}_s \qquad (4.10)$$

Since (4.10) can also be written as

$$\boldsymbol{T}_{s+1} = \boldsymbol{Q}_s^{\mathrm{T}}\left(\boldsymbol{T}_s - k\boldsymbol{I}\right)\boldsymbol{Q}_s + k\boldsymbol{I} \qquad (4.11)$$

then,

$$\boldsymbol{T}_{s+1} = \boldsymbol{Q}_s^{\mathrm{T}}\boldsymbol{T}_s\boldsymbol{Q}_s \qquad (4.12)$$

Note that this $\boldsymbol{Q}_s$ differs from the $\boldsymbol{Q}_s$ of (4.5). That is, based on (4.9) if $\boldsymbol{Q}_s$ is calculated as

$$\boldsymbol{Q}_s^{\mathrm{T}}\left(\boldsymbol{T}_s - k\boldsymbol{I}\right) = \boldsymbol{L}_s \qquad (4.13)$$

$\boldsymbol{T}_{s+1}$ can be obtained from (4.12). The origin shift k is determined as follows: Let an eigenvalue obtained on d_i in (4.1) be λ_i and rate of the convergence of λ_i, depends on the ratio:

$$\left|\lambda_i - k\right| / \left|\lambda_{i+1} - k\right| \qquad (4.14)$$

k is determined by obtaining the eigenvalues of 2×2 submatrix $\boldsymbol{B}_s$ showing (4.15) using the Jacobian method, and using the one obtained on $d_i^{(s)}$.

$$\boldsymbol{B}_s = \begin{bmatrix} d_i^{(s)} & e_i^{(s)} \\ e_i^{(s)} & d_{i+1}^{(s)} \end{bmatrix} \qquad (4.15)$$

By this way, $e_i^{(s)}$ converges to 0 rapidly.

If k is recalculated for each iteration of (4.12), the rate of convergence is further improved. Therefore, if (4.12) is calculated after recalculating k at every times, the accelerating of convergence can be performed. The k_s is k calculated for the s-th iteration. (4.12) is actually calculated in the form shown in (4.8), so the calculation of $\boldsymbol{P}_i^{(s)\mathrm{T}}$ (i = n–1, ..., 2, 1) is described next. Since $\boldsymbol{P}_i^{(s)\mathrm{T}}$ makes elements (n–1, n) of ($\boldsymbol{T}_s$ – $k_s\boldsymbol{I}$) zero when post-multiplied by ($\boldsymbol{T}_s$ – $k_s\boldsymbol{I}$), $\boldsymbol{P}_{n\text{-}1}$ can be thought of as the following matrix:

$$P_{n-1}^{(s)\mathrm{T}} = \begin{bmatrix} 1 & & & 0 & \\ & \ddots & & & \\ & & 1 & & \\ 0 & & & C_{n-1} & -S_{n-1} \\ & & & S_{n-1} & C_{n-1} \end{bmatrix}$$

where,

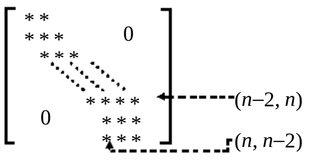
$$C_{n-1} = (d_n - k_s)/\left\{(d_n - k_s)^2 + e_{n-1}^2\right\}^{\frac{1}{2}}$$

$$S_{n-1} = e_{n-1}/\left\{(d_n - k_s)^2 + e_{n-1}^2\right\}^{\frac{1}{2}}$$

If above $P_{n-1}^{(s)\mathrm{T}}$ is used in the calculation of

$$\tilde{T}_s = P_{n-1}^{(s)\mathrm{T}} T_s P_{n-1}^{(k)} \tag{4.16}$$

$\tilde{T}_s$ will have the following form:

$\tilde{T}_s =$ 

In this form, element ($n-2$, n) and element (n ,$n-2$) are appended to a tridiagonal matrix.

Since above T_s is a symmetric matrix, it can be reduced to a tridiagonal matrix using the Givens method. Let the transformation matrices used in the Givens method for reduction of T_s to a tridiagonal matrix be

$$\tilde{Q}_s = \tilde{P}_{n-2}^{(s)} \cdots \tilde{P}_2^{(s)} \tilde{P}_1^{(s)} \tag{4.17}$$

and with respect to the T_s to T_{s+1} reduction, we obtain

$$T_{s+1} = \tilde{Q}_s^{\mathrm{T}} \tilde{T}_s \tilde{Q}_s \tag{4.18}$$

Therefore, from (4.19), (4.21) and (4.22), T_{s+1} can be calculated as

$$T_{s+1} = \tilde{P}_1^{(s)\mathrm{T}} \tilde{P}_2^{(s)\mathrm{T}} \cdots \tilde{P}_{n-2}^{(s)\mathrm{T}} \left(P_{n-1}^{(s)\mathrm{T}} T_s P_{n-1}^{(s)}\right) \tilde{P}_{n-2}^{(s)} \cdots \cdots \tilde{P}_2^{(s)} \tilde{P}_1^{(s)} \tag{4.19}$$

The computation process for this routine is shown next. Steps 1) to 6) are performed for i=1, 2, ..., n–1.

1) Shift k_s of origin is calculated according to:

$$f = \left(d_{i+1}^{(s)} - d_i^{(s)}\right)/ 2e_i^{(s)}$$

$$k_s = d_i^{(s)} - e_i^{(s)}/\left(f \pm \sqrt{f^2+1}\right) \tag{4.24}$$

(Signs of $\pm$Lf^2 +1 and f are made equal)

2) C_{n-1} and S_{n-1} are determined $\tilde{T}_s$ is obtained. Then $\tilde{T}_s$ is calculated from (4.16).
3) $\tilde{T}_s$ is reduced to a tridiagonal matrix by Givens method.
4) Steps 1) to 3) are repeated as $s = s + 1$ until one of the following conditions is satisfied.
   (a) $\left|e_j\right| \le u\left(\left|d_j\right| + \left|d_{j+1}\right|\right) \quad (j = i, i+1, ..., n-1)$ (4.20)
   where u is the unit round-off.
   (b) $s \ge 30$
   where s is number of iterations.
5) When condition (a) is satisfied with $j = i$, this means that eigenvalue λ_i is determined. When λ_{n-1} is determined, then λ_n is determined automatically.
6) When condition (b) is satisfied, λ_i is considered to be undertermined and ICON is set to 15000. The process is terminated.

The following considerations have been made in order to diminish the execution time of this subroutine.

1) If matrix T is a direct sum of submatrixes. steps 1) to 6) above are applied to each submatrix in order to reduce operations. When condition (4.20) is satisfied with $j \ne i$, matrix T is split into submatrices.
2) The above steps 2) to 3) are put together as (4.18) and are calculated as shown in (4.21).

$$\left.\begin{array}{l} e_n^{(s)} = 0.0, C_n = 1.0, S_n = 1.0 \\ g_j = C_{j+1} d_{j+1}^{(s)} - S_{j+1} e_{j+1}^{(s)} \\ w_j = \left(g_j^2 + e_j^{(s)2}\right)^{\frac{1}{2}} \\ C_j = g_j/w_j,\ S_j = e_j^{(s)2}/w_j,\ x_j = 2C_j S_j e_j^{(s)} \\ d_{j+1}^{(s+1)} = C_j^2 d_{j+1}^{(s)} + S_j^2 d_j^{(s)} + x_j \\ e_j^{(s+1)} = C_j S_j d_j^{(s)} - S_j^2 e_j^{(s)} \\ (j = n-1, n-2, ..., i+1, i) \end{array}\right\} \tag{4.21}$$

$$d_i^{(s+1)} = d_{i+1}^{(s+1)} - 2x_i$$

For further information see references [12], [13] pp. 227-248 and [16] pp.177-206.

C23-11-0111 TSDM, DTSDM

Zero of a real function (Muller's method)
CALL TSDM(X, FUN, ISW, EPS, ETA, M, ICON)

Function

This subroutine finds a zero of a real function

$f(x) = 0$

An initial approximation to the zero must be given.

Parameters

X.....	Input. Initial value of the root to be obtained. Output. Approximate root.
FUN ...	Input. Function subprogram name which computes the function $f(x)$.
ISW...	Input. Control information. One of the convergence criteria for the root obtained is specified. ISW must be either one of 1, 2, or 3. When ISW = 1, x_i becomes the root if it satisfies the condition $\lvert f(x_i)\rvert \le \mathrm{EPS}$: Convergence criteria I. When ISW = 2, x_i becomes the root if it satisfies the condition $\lvert x_i - x_{i-1}\rvert \le \mathrm{ETA}\cdot\lvert x_i\rvert$: Convergence criteria II. When ISW = 3, x_i becomes the root if either of the criteria described above are satisfied. See "Comments on use".
EPS ...	Input. The tolerance (≥ 0.0) used in the convergence criteria I. (See the parameter ISW.)
ETA......	Input. The tolerance (≥ 0.0) used in the convergence criteria II. (See the parameter ISW.)
M	Input. Upper limit of number of iterations to obtain the solution. See "Comments on use". Output. Number of iterations actually tried.
ICON..	Output. Condition code. See Table TSDM-1.

Comments on use

- Subprograms used
  (a) SSL II..... MGSSL, AMACH, AFMAX and AFMIN
  (b) FORTRAN basic functions .. ABS, EXP, SQRT and ALOG

- Notes
  The function subprogram specified by the parameter FUN mut be defined as having only one argument for the variable x, and the program which calls this TSDM subroutine must have

Table TSDM-1 Condition codes

Code	Meaning	Processing
1	The obtained approximate root satisfied the convergence criteria I. (See parameter ISW.)	Normal end
2	The obtained approximate root satisfied the convergence criteria II. (See parameter ISW.)	Normal end
10	Iterations were tried unconditionally m times. (M=–m)	Normal end
11	Although m iterations were specified (M=–m), before finishing all iterations, $f(x_j) = 0.0$ was satisfied, thereby the iteration was stopped and x_j was set as the root.	Normal end
12	Although m iterations were specified (M=–m), before finishing all iterations, $\lvert x_i - x_{i-1}\rvert \le u\cdot\lvert x_i\rvert$ was satisfied, thereby the iteration was stopped and x_j was set as the root.	Normal end
10000	The specified convergence criterion was not satisfied within the limit of iterations given.	The last iteration is value X_i stored in X.
20000	A difficulty occurred during computation. so Muller's method could not be continued. See. (4) Method.	Terminated
30000	The input parameter had error(s). when M>0, either of the followings was found: 1 ISW=1 and EPS < 0, or 2 ISW=2 and ETA<0, or 3 ISW=3, EPS<0 and ETA<0, or in other case, M=0 or ISW ≠ 1, 2, 3 was found.	Bypassed

EXTERNAL statement for that function subprogram. (See the example below.)

This subroutine, even if ISW = 1 is given, stops the iteration with ICON = 2 whenever

$$\lvert x_i - x_{i-1}\rvert \le u\cdot\lvert x_i\rvert \quad (u \text{ is the unit round off})$$

is satisfied, and also stops the iteration with ICON = 1 whenever $f(x_i) = 0.0$ is satisfied even if ISW = 2 is given.

Iterations are repeated m times unconditionally, when M is set as M = $-m$ ($m > 0$). However, if $f(x_i) = 0.0$ or $\lvert x_i - x_{i-1}\rvert \le u\cdot\lvert x_i\rvert$ is satisfied before finishing m iterations, the iteration is stopped and the result is put out with ICON = 11 or ICON = 12.

- Example

  A root of $f(x) = e^x - 1$ is obtained with the initial value $x_0 = 1$ given

```
C      **EXAMPLE**
       EXTERNAL FEXP
       X=0.0
       ISW=3
       EPS=0.0
       ETA=1.0E-6
       M=100
       CALL TSDM(X,FEXP,ISW,EPS,ETA,M,
      *ICON)
       WRITE(6,600) ICON,M,X
       STOP
  600 FORMAT(10X,'ICON=',I5/13X,'M=',I5/
      * 13X,'X=',E15.7)
       END
       FUNCTION FEXP(X)
       FEXP=EXP(X)-1.0
       RETURN
       END
```

Method

The subroutine uses Muller's method. This uses an interpolating polynomial $\boldsymbol{P}(x)$ of degree two, by using three approximate values for a root and approximates $f(x)$ near the root to be obtained. One of the roots for $\boldsymbol{P}(x) = 0$ is taken as the next approximate root of $f(x)$. In this way iteration is continued. This algorithm has the following features:

a. Derivatives of $f(x)$ are not required
b. The function is evaluated only once at each iteration
c. The order of convergence is 1.84 (for a single root).

- Muller's method

  Let α be a root of $f(x)$ and let three values x_{i-2}, x_{i-1} and x_i be approximations to the root (See later explanation for initial values x_1, x_2 and x_3). According to Newton's interpolation formula of degree two, $f(x)$ is approximated by using the three values described above as follows:

$$P(x) = f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1}) \qquad (4.1)$$

where $f_i = f(x_i)$, and $f[x_i, x_{i-1}]$ and $f[x_i, x_{i-1}, x_{i-2}]$ are the first and the second order divided differences of $f(x)$, respectively, and are defined as follows:

$$f[x_i, x_{i-1}] = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$
$$f[x_i, x_{i-1}, x_{i-2}] = \frac{f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}]}{x_i - x_{i-2}} \qquad (4.2)$$

$\boldsymbol{P}(x) = 0$ is then solved and the two roots are written as

$$x = x_i - \frac{2f_i}{\omega \pm \left\{\omega^2 - 4f_i f[x_i, x_{i-1}, x_{i-2}]\right\}^{1/2}}$$
$$\omega = f[x_i, x_{i-1}] + (x_i - x_{i-1}) f[x_i, x_{i-1}, x_{i-2}] \qquad (4.3)$$

Of these two roots for $P(x) = 0$, the root corresponding to the larger absolute value of the denominator in the second term of Eq. (4.3) is chosen as the next iteration value x_{i+1}. This means that x_{i+1} is a root closer to x_i. In Eq. (4.1), if the term of x^2 is null, i.e., if $f[x_i, x_{i-1}, x_{i-2}] = 0$, the following equation is used in place of using Eq. (4.3):

$$x = x_i - \frac{f_i}{f[x_i, x_{i-1}]}$$
$$= x_i - \frac{x_i - x_{i-1}}{f_i - f_{i-1}} \cdot f_i \qquad (4.4)$$

This is the secant method.
In Eq. (4.1) also, if both terms x and x^2 are null, $P(x)$ reduces to a constant and the algorithm fails. (See later explanation.)

- Considerations of Algorithm
  - Initial values x_1, x_2 and x_3

    The three initial values are set as follows: Let x be an initial value set by the user in the input parameter X.
    When $x \neq 0$
    $x_1 = 0.9x$
    $x_2 = 1.1x$
    $x_3 = x$
    When $x = 0$,
    $x_1 = -1.0$
    $x_2 = 1.0$
    $x_3 = 0.0$
  - When $f(x_{i-2}) = f(x_{i-1}) = f(x_i)$

    This corresponds to the case in which both terms x and x^2 in Eq. (4.1) are null, so Muller's method cannot be continued.
    The subroutine changes x_{i-2}, x_{i-1}, and x_i and tries to get out of this situation by setting

$$x'_{i-2} = (1+p^n)x_{i-2}$$
$$x'_{i-1} = (1+p^n)x_{i-1}$$
$$x'_i = (1+p^n)x_i$$

where $p = -u^{-1/10}$, u is the unit round off and n is the count of changes. Muller's method is continued by using x'_{i-2}, x'_{i-1}, and x'_i. When more than five changes are performed the subroutine terminate unsuccessfully with ICON = 20000.

- Convergence criteria
  The following two criteria are used.
  Criteria I.
  When the approximate root x_i satisfies $f(x_i) \leq$ EPS, the x_i is taken as the root.
  Criteria II. When the approximate root x_i satisfies $|x_i - x_{i-1}| \leq \text{ETA} \cdot |x_i|$ the x_i is taken as the root. When the root is a multiple root or very close to another root, ETA must be set sufficiently large. If $0 \leq \text{ETA} < u$, the subroutine resets ETA = u. For further details, see Reference [32].

C23-11-0101 TSD1, DTSD1

Zero of a real function which changes sign in a given interval (derivative not required)
CALL TSD1 (AI, BI, FUN, EPST, X, ICON)

Function

Given two points a, b such that $f(a)\,f(b) \le 0$, a zero in $[a, b]$ of the real transcendental equation

$$f(x)=0$$

is obtained. The derivatives of $f(x)$ are not required when determining the zero. The bisection method, linear interpolation method, and inverse quadratic interpolation method are used depending on the behavior of $f(x)$ during the calculations.

Parameters

AI...	Input. Lower limit a of the interval.
BI.....	Input. Upper limit b of the interval.
FUN....	Input. The name of the function subprogram which calculates $f(x)$. In the program which calls this subroutine, the user must declare EXTERNAL and prepare the function subprogram.
EPST...	Input. The tolerance of absolute error (≥ 0.0) of the approximated root to be determined (See the notes)
X.....	Output. The approximated zero
ICON...	Output. Condition code. See Table TSD1-1.

Table TSD1-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	F(a) f(b) > 0 or EPST < 0.0	Bypassed

Comments on use

- Subprograms used
  SSL II ... AMACH and MGSSL
  FORTRAN basic function... ABS

- Notes
  FUN must be declared as EXTERNAL in the program from which this subroutine is called.
  If there are several zeros in the interval $[a, b]$ as shown in Fig. TSD1-l, it is uncertain which zero will be obtained.

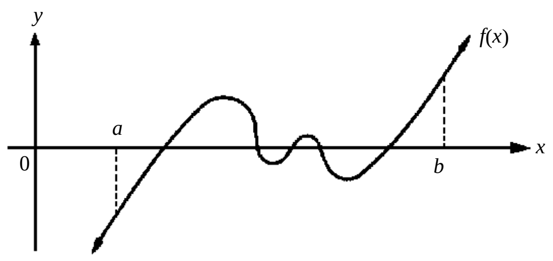


Fig. TSD1-1

The required accuracy of the root being determined is defined by parameter EPST. If the interval $[a, b]$ does not include the origin, EPST=0.0 can be specified, and the subroutine will calculate as precise root as possible.

- Example
  One root of $f(x) = \sin^2(x) - 0.5 = 0$ is calculated in the interval [0.0,1.5]. Notice that $f(x)$ has different signs at both ends of the interval, and only one root exists in this interval.

```
C      **EXAMPLE**
       EXTERNAL FSIN
       AI=0.0
       BI=1.5
       EPST=1.0E-6
       CALL TSD1(AI,BI,FSIN,EPST,X,ICON)
       WRITE(6,600) X,ICON
       STOP
   600 FORMAT(10X,E15.7,5X,I10)
       END
       FUNCTION FSIN(X)
       FSIN=SIN(X)**2-0.5
       RETURN
       END
```

Method

With some modifications, this subroutine uses what is widely known as the Dekker algorithm. The iteration formula to be used at each iteration (bisection method, linear interpolation method, inverse quadrantic interpolation method) are determined by examining the behavior of $f(x)$. $f(x)$ is a real function that is continuous in the interval $[a, b]$, and $f(a)f(b) < 0$. At the beginning of the algorithm, c is taken as a; the general procedures used thereafter are as follows.

- Procedure 1 If $f(b)\,f(c) > 0$, c is assumed to be the same as a.
  If $f(b)f(c)<0$ and $|f(b)| > |f(c)|$, b and c are interchanged and then a is assumed equal to c. In this manner, band c are determined such that

$$f(b)f(c)<0 \text{ and } |f(b)| \le |f(c)|$$

Variables *a*, *b*, and *c* have the following meanings:

a... Current approximation to a root

b... Previous *b*

c... The opposing variable of *b*; a root lies between *b* and *c*.

- Procedure 2 Let $m = (c - b)/2$, as long as $|m| < \delta$ is not satisfied, one of the following (a), (b), or (c) is used to calculate the next approximation *i* and that value is assigned to *b* (as explained in detail later, δ is a function of *b*).

  (a) When $|f(a)| \le |f(b)|$
  The bisection method, $i = (b + c)/2$ is applied

  (b) When $a = c$
  The linear interpolation (4.1) is applied

$$\begin{aligned} i &= b - \frac{(a-b)f(b)/f(a)}{1 - f(b)/f(a)} \\ &= b + p/q \end{aligned} \qquad (4.1)$$

  (c) When $|f(b)| < |f(a)| \le |f(c)|$
  The inverse quadratic interpolation method is applied
  Let $f_a = f(a)$, $f_b = f(b)$, $f_c = f(c)$,
  also $r_1 = f_a/f_c$, $r_2 = f_b/f_c$, $r_3 = f_b/f_a$,
  then

$$\begin{aligned} i &= b - f_b\left[\frac{(c-b)f_a(f_a - f_b) - (b-a)f_c(f_b - f_c)}{(f_a - f_c)(f_b - f_c)(f_b - f_a)}\right] \\ &= b - r_3\left[\frac{(c-b)r_1(r_1 - r_2) - (b-a)(r_2 - 1)}{(r_1 - 1)(r_2 - 1)(r_3 - 1)}\right] \\ &= b + p/q \end{aligned} \qquad (4.2)$$

Procedure 1 and procedure 2 are then repeated.

- Preventing overflow
  When using (4.1) and (4.2), the following is necessary to prevent overflow. That is, when

$$|p/q| > \frac{3}{4}|c-b| = \frac{3}{4}|2m| = \frac{3}{2}|m|$$

which can be rewritten

$$2|p| > 3|mq|$$

is satisfied, the division *p*/*q* is not performed, instead the bisection method (described in (a)) is used. When $|p/q| < \delta$, *i* is calculated as

$$i = b + \delta \mathrm{sign}(c - b)$$

to keep convergence from slowing down.

- Convergence criterion
  When either (a) or (b) below is satisfied, convergence is judged to have occurred. Calculation is terminated, and current *b* is taken as the root.

  (a) $f(b) = 0$(This will happen when complete underflow occurs)

  (b) $|m| < \delta = 2u|b| + \varepsilon/2 \qquad \varepsilon > 0 \qquad (4.3)$

Where *u* is the round-off unit and ε is the tolerance to the root (that corresponds to the parameter EPST of this routine). The righthand side of (4.3) has been derived experimentally. For more details see reference [3].

Let the exact root be *x*, the upper limit of the absolute error $|b-x|$ somewhat larger than the righthand side of (4.3) because of the rounding error. The result can be expressed as

$$|b-x| \le 6u|x| + \varepsilon$$

Care should be taken when specifying the value of parameter EPST. If the interval [*a*, *b*] includes the origin, it is unwise to set EPST=0.0 since there is a possibility that the exact root is the origin. Otherwise, EPST can be set to 0.0 without problems.

For further information, see Reference [28].

APPENDICES

APPENDIX A
AUXILIARY SUBROUTINES

A.1 OUTLINE

Auxiliary subroutines internally supplement the functions of SSL II subroutines and slave subroutines. In Section 2.9, for example, SSL II subroutines frequently call for the unit round off "u". Then, the auxiliary subroutine for "u" facilitates the use of "u" for various subroutines that require "u" in execution.

Thus, auxiliary subroutines are usually called from general or slave subroutines in SSL II. and have features as follows:

Features of auxiliary subroutines

- Some auxiliary subroutines are computer-dependent, (while SSL II subroutines and slave subroutines are computer-independent).
- Some auxiliary subroutines have single and double precision parameters at the same time, (whereas these two are not combined in the SSL II general subroutines and slave subroutines).
- There is no naming rule to apply to auxiliary subroutines, as with SSL II general subroutines and slave subroutines.(See Section 2.3).

Table A.1 Auxiliary subroutines

Item	Subroutine name		Feature*	Reference section
	Single precision	Double precision		
Unit round off	AMACH	DMACH	M,F	See A.2.
Printing of condition message	MGSSL		S	See A.3.
Control of message printing	MGSET**		S	See A.4.
Product sum (Real vector)	ASUM,BSUM	DSUM,DBSUM	S	See A.5.
Product sum (Complex vector)	CSUM	DCSUM	S	See A.6.
Radix of the floating-point number system	IRADIX		M,F	See A.7.
Positive max. value of the floating -point number system	AFMAX	DFMAX	M,F	See A.8.
Positive min. value of the floating -point number system	AFMIN	DFMIN	M,F	

* M: Computer dependent subprograms.
F: Function subprograms.
S: Subroutine subprograms.
** MGSET is the sub-entry of MGSSL.

A.2 AMACH, DMACH

Unit round off
(Function subprogram) AMACH(EPS)

Function

The function value defines the unit round off u in normalized floating-point arithmetic.

$u = M^{1-L}/2$(for (correctly) rounded arithmetic)
$u = M^{1-L}$(for chopped arithmetic)

where M is the radix of the number system, and L the number of digits contained in the mantissa.
Table AMACH-1 lists the values for single and double precision arithmetic.

Parameter

EPS.. Input. EPS is specified as follows depending on the working precision.

- Single precision:Single precision real variable or constant.
- Double precision:Double precision real variable or constant.

Example

The unit round off for single and double precision arithmetic are obtained.

```
C     **EXAMPLE**
      REAL*4 AEPS
      REAL*8 DEPS,DMACH
      AEPS=AMACH(AEPS)
      DEPS=DMACH(DEPS)
      WRITE(6,600) AEPS,DEPS
      STOP
  600 FORMAT(10X,'AEPS=',E16.7,
     *        5X,'DEPS=',D25.17)
      END
```

Table AMACH-1 Unit round off

Arithmetic method		AMACH	DMACH	Application
		Single precision	Double precision	
Binary:M=2	Rounded arithmetic	L=26 $u=^1/_2 \cdot 2^{-25}$	L=61 $u=^1/_2 \cdot 2^{-60}$	
Hexadecimal:M=16	Chopped arithmetic	L=6 $u=16^{-5}$	L=14 $u=16^{-13}$	FACOM M series FACOM S series SX/G 200 series
Binary:M=2	Rounded arithmetic	L=23 $u=^1/_2 \cdot 2^{-22}$	L=52 $u=^1/_2 \cdot 2^{-51}$	FM series SX/G series

A.3 MGSSL

Printing of condition messages
CALL MGSSL(ICON,MCODE)

Function

This subroutine prints the condition messages for SSL II subroutines.

Parameters

ICOM.. Input. Condition code.

MCODE..Input. Subroutine classification code.
One dimensional array of size 6.
All 11-digit classification code is specified in MCODE.

Example:For classification code, A52-22-0101:
DATA MCODE/2HA5,2H2-,2H22,2H-0,2H10,2H1/

Comments on use

- Notes
  This subroutine is called by all SSL II general subroutines upon completion of the execution except by special function subroutines when normally ending with ICON=0.

This subroutine does not usually print messages by itself. The printing message is controlled by subroutine MGSET, see MGSET.

- Example
  The following example, taken on subroutine LAX, shows how this subroutine is called internally by SSL II general subroutines.

```
C     **EXAMPLE**
      SUBROUTINE LAX(A,K,N,B,EPSZ,ISW,IS,
     *VW,IP,ICON)
      DIMENSION A(K,N),B(N),VW(N),IP(N)
      CHARACTER  MCODE(6)*4
      DATA MCODE/'A2  ','2-  ','11  ',
     *'-0  ','10  ','1   '/
      IF(ISW.EQ.1) GOTO 1000
      IF(ISW.EQ.2) GOTO 1100
      ICON=30000
      GOTO 8000
 1000 CALL ALU(A,K,N,EPSZ,IP,IS,VW,ICON)
      IF(ICON.NE.0) GOTO 8000
 1100 CALL LUX(B,A,K,N,1,IP,ICON)
 8000 CALL MGSSL(ICON,MCODE)
      RETURN
      END
```

A.4 MGSET

Control of message printing
CALL MGSET(ISET,IFLE)

Function

This subroutine controls message printing for SSL II general subroutines.

Parameters

ISET..	Input. Output level.(See Table MGSET-1.)
IFLE..	Input. File reference No. for output.(Data set identification No.)

Table MGSET-1 Output level

ISET	Control function
0	Output when the condition code is among 0~30000.
1	Output when the condition code is among 10000~30000.
2	Output when the condition code is among 20000~30000.
3	Output when the condition code is 30000.
-1	No condition message is output.

Comments on use

- Notes

  Printing of the condition messages:
  SSL II general subroutines call the auxiliary subroutine MGSSL to print condition message upon completion of the processing. Usually, auxiliary subroutine does not print messages, but messages can be printed by calling this subroutine in advance in the user's program.
  Extent of output control:
  Output control by this subroutine is retained until it is called again from the user's program. Note that, when an SSL II subroutine called by user's program calls other SSL II subroutines, they are also under the output control.
  Termination of printing:
  The message printing previously called for can be terminated by calling this subroutine again with ISET= −1.
  File reference number:
  IFLE=6 is usually specified as the standard. This subroutine is the sub-entry of the auxiliary subroutine MGSSL.

- Example

  The example shows how MGSET is used in subroutine LSX to solve a system of linear equations with a positive symmetric matrix.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100)
      CALL MGSET(0,6)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (B(I),I=1,N)
      WRITE(6,600) N
      CALL LSX(A,N,B,0.0,1,ICON)
      IF(ICON.GE.20000) GOTO 20
      WRITE(6,610) (B(I),I=1,N)
   20 CALL MGSET(-1,6)
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(5E16.8)
  600 FORMAT('0',4X,'ORDER=',I5)
  610 FORMAT(' ',4X,'SOLUTION VECTOR'
     * /(5X,E15.7))
      END
```

Several sets of systems are solved successively in this example. Subroutine MGSET is called two times in the program:first it is called after the array statement to produce the message when the first system is solved then it is called at statement number "20" to terminate the message printing in the successive operations.Since subroutine LSX internally uses component routines (SLDL, LDLX), they are also under the output control.

```
    ORDER=3
****SSL2(A22-51-0202) CONDITION 0****
****SSL2(A22-51-0302) CONDITION 0****
****SSL2(A22-51-0101) CONDITION 0****
    SOLUTION VECTOR
    0.1352613E+01
    0.1111623E+00
    0.4999998E+01

    ORDER=4
    SOLUTION VECTOR
    0.5000001E+01
    0.3333330E-01
    0.1111110E+00
    0.2499998E+00
```

A.5 ASUM(BSUM), DSUM(DBSUM)

Product sum (real vector)
CALL ASUM (A, B, N, IA, IB, SUM)

Function

Given n-dimensional real vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, this subroutine computes the product sum μ.

$$\mu = \sum_{i=1}^{n} a_i b_i$$

where, $\boldsymbol{a}^T=(a_1, a_2, ... , a_n)$, $\boldsymbol{b}^T=(b_1, b_2, ... , b_n)$

Parameters

A....	Input. Vector $\boldsymbol{a}$. One-dimensional array of size $\lvert IA\rvert \cdot N$.
B....	Input. Vector $\boldsymbol{b}$. One-dimensional array of size $\lvert IB\rvert \cdot N$.
N....	Input. Dimension n of vectors.
IA...	Input. An interval between consecutive elements of vector $\boldsymbol{a}$ (=0). Generally, it is set to 1. Refer to Notes.
IB...	Input. An interval between consecutive elements of vector $\boldsymbol{b}$ (=0). Generally, it is est to 1. See Notes.
SUM..	Output. Inner product μ . Refer to Notes.

Comments on use

- Notes

  The purpose of this subroutine:
  From the theory of error analysis on floatingpoint arithmetic, the calculation of product sum frequently used in numerical analyses requires high accuracy to maintain the needed number of significant digits. To do that, this subroutine enjoys an optimum method, which depends on the computer to be used.
  Data spacing in arrays A and B:
  Set IA=p when elements of vector $\boldsymbol{a}$ are to be stored in array A with spacing p .
  Likewise, set IB=q when elements of vector $\boldsymbol{b}$ are to be stored in array B with spacing q .
  If p, q<0 are must be taken in assigning arrays A and B(see Example).
  About BSUM:
  The functions of BSUM are equivalent to those of ASUM but BSUM computes the product sum μ in double precision. The comparison among ASUM, BSUM, DSUM and DBSUM is tabulated as follows:

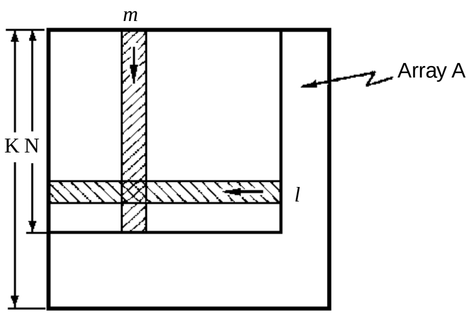
Subroutine name	Use	Difference in parameters
ASUM	Single precision routines	A,B and SUM:Single precision
BSUM		A,B: Single precision SUM: Double precision
DSUM	Double precision routine	A,B,SUM: Double precision
DBSUM		A,B: Double precision SUM: Quadruple precision

Note:
DBSUM cannot be used if the FORTRAN system does not support the quadruple-precision operation function.

- Example

  When n-dimensional real matrix $\boldsymbol{A}$ (=(a_{ij})) is defined as a two-dimensional array A (K, N), the product sum μ in the m-th column and l-th row is calculated as shown below:

$$\mu = \sum_{i-1}^{n} a_{im} a_{l\,n+1-i}$$



Where, $n \le 100$

```
C     **EXAMPLE**
      DIMENSION A(100,100)
      K=100
      READ(5,100) N,((A(I,J),I=1,N),J=1,N)
      READ(5,200) M,L
      CALL ASUM(A(1,M),A(L,N),N,1,-K,SUM)
      WRITE(6,150) M,L,SUM
      STOP
  100 FORMAT(I5/(4E15.7))
  200 FORMAT(2I5)
  150 FORMAT('1'//10X,'M=',I5,5X,'L=',I5,
     *5X,'SUM=',E16.7)
      END
```

A.6 CSUM, DCSUM

Product sum (Complex vector)
CALL CSUM (ZA, ZB, N, IA, IB, ZSUM)

Function

Given n-dimensional complex vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ this subroutine computes the product sum μ.

$$\mu = \sum_{i-1}^{n} a_i b_i$$

Where $\boldsymbol{a}^T=(a_1, a_2, ..., a_n)$, $\boldsymbol{b}^T=(b_1, b_2, ...,b_n)$

Parameters

ZA...	Input. Vector $\boldsymbol{a}$. One-dimensional complex type array of size $\lvert\mathrm{IA}\rvert \cdot \mathrm{N}$.
ZB...	Input. Vector $\boldsymbol{b}$. One-dimensional complex type array of size $\lvert\mathrm{IB}\rvert \cdot \mathrm{N}$.
N....	Input. Dimension n of vectors.
IA...	Input. An interval between consecutive elements of vector $\boldsymbol{a}$ ($\neq 0$). Generally, it is set to 1. (See Notes.)
IB...	Input. An interval between consecutive elements of vector $\boldsymbol{b}$ ($\neq 0$). Generally, it is set to 1. (See Notes.)
ZSUM.	Output. Inner product μ. Complex type variable.

Comments on use

- Notes

  The purpose of this subroutine:

  From the theory of error analysis on floating-point arithmetic, the calculation of the product sum frequently used in numerical analysis requires high precision. This subroutine uses the most appropriate method depending on the computer used.

  Data interval of array ZA, ZB:

  Specify IA=p when elements of vector are to be stored in array ZA with the interval p.

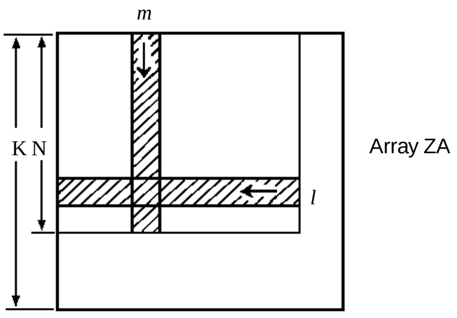
  Similarly, specify IB=q when elements of vector $\boldsymbol{b}$ are to be stored in array ZB with the interval q.

  If p, q<0, assigning of arrays ZA and ZB should be done with sufficient care. (See the example.)

- Example

  When n-dimensional complex matrix $\boldsymbol{A}(=(a_{ij}))$ is given as a two-dimensional array ZA (K, N), the following example computes the product sum u in the m-th column and the l-th row.

$$\mu = \sum_{i-1}^{n} a_{im} a_{ln+1-i}$$



```
C      **EXAMPLE**
       DIMENSION ZA(100,100)
       COMPLEX ZA,ZSUM
       K=100
       READ(5,100) N,((ZA(I,J),I=1,N),J=1,N)
       READ(5,200) M,L
       CALL CSUM(ZA(1,M),ZA(L,N),N,1,-K,ZSUM)
       WRITE(6,150) M,L,ZSUM
       STOP
   100 FORMAT(I5/(4E15.7))
   200 FORMAT(2I5)
   150 FORMAT('1'//10X,'M=',I5,5X,'L=',I5,
      *5X,'CSUM=',2E16.7)
       END
```

A.7 IRADIX

Radix of the floating-point number system
(Function subprogram) IRADIX(RDX)

Function

The radix of the floating-point number system is set as an integer-type function value.
See Table IRADIX-1.

Table IRADIX-1 Radix for floating-point

Arithmetic	IRADIX	Application
Binary	IRADIX=2	FM series SX/G 100 series
Hexadecimal	IRADIX=16	FACOM M series FACOM S series SX/G 200 series

Parameter

RDX... Input. Real variable, or a constant. Any real number can be specified as RDX.

Example

The radix of the number system can be obtained as follows.

```
C     **EXAMPLE**
      REAL*4 RDX
      RDX=IRADIX(RDX)
      WRITE(6,600) RDX
  600 FORMAT(//10X,'RDX=',E15.7)
      STOP
      END
```

A.8 AFMAX, DFMAX, AFMIN, DFMIN

Positive max. and min. values of the floating-point number system.
Maximum value AFMAX(X) Minimum value AFMIN(X)

Function

The positive maximum value fl_{max} or minimum value fl_{min} of the floating-point number system is set.
See Table AFMAX-1.

Parameter

X.. Input. This specified according to the precision as:

- Single precision:Single precision real variable or constant.
- Double precison:Double precision real variable or constant.

Example

The maximum and minimum values for single and double precisions are obtained as follows:

```
C     **EXAMPLE**
      REAL*4 X0,X1,X2
      REAL*8 Y0,Y1,Y2,DFMAX,DFMIN
      X1=AFMAX(X0)
      X2=AFMIN(X0)
      Y1=DFMAX(Y0)
      Y2=DFMIN(Y0)
      WRITE(6,600) X1,X2,Y1,Y2
      STOP
  600 FORMAT(10X,'AFMAX=',E16.7/
     *10X,'AFMIN=',E16.7/
     *10X,'DFMAX=',D25.17/
     *10X,'DFMIN=',D25.17)
      END
```

Table AFMAX-1 Max, and min. values for floating-point unmber system

Arithmetic	Maximum values		Minimum values		Application
	Single precision AFMAX	Double precision DFMAX	Single precision AFMIN	Double precision DFMIN	
Binary	$(1-2^{-26})\cdot 2^{255}$	$(1-2^{-61})\cdot 2^{255}$	$2^{-1}\cdot 2^{-256}$	$2^{-1}\cdot 2^{-256}$	
Haxadecimal	$(1-16^{-6})\cdot 16^{63}$	$(1-16^{-14})\cdot 16^{63}$	$16^{-1}\cdot 16^{-64}$	$16^{-1}\cdot 16^{-64}$	FACOM M series FACOM S series SX/G 200 series
Binary	$(1-2^{-24})\cdot 2^{128}$	$(1-2^{-53})\cdot 2^{1024}$	$2^{-1}\cdot 2^{-125}$	$2^{-1}\cdot 2^{-1021}$	FM series
		$(1-2^{-56})\cdot 2^{252}$		$2^{-1}\cdot 2^{-259}$	SX/G 100 series

APPENDIX B
ALPHABETIC GUIDE FOR SUBROUTINES

In this appendix, subroutine name of SSL II are listed in alphabetical order.
The subroutines are divided into three lists based on their uses. Only single precision subroutine names are included.

B.1 GENERAL SUBROUTINES

Table B.1 shows general subroutines.

Table B.1 General subroutines

Subroutine name	Classification code	Subprogram used
AGGM	A21-11-0101	
AKHER	E11-11-0201	AFMAX
AKLAG	E11-11-0101	AFMAX
AKMID	E11-42-0101	
AKMIN	E12-21-0201	
ALU	A22-11-0202	AMACH
AQC8	G23-11-0301	AMACH
AQE	G23-11-0401	AMACH, AFMIN
AQEH	G23-21-0101	AMACH, AFMAX
AQEI	G23-31-0101	AMACH, AFMAX
AQMC8	G24-13-0101	AMACH
AQME	G24-13-0201	AFMAX, AFMIN, UAQE1, UAQE2, UAQE3, UFNIO, UFN2O, UFN3O, UFN4O, AMACH
AQN9	G23-11-0201	AMACH
ASSM	A21-12-0101	
ASVD1	A25-31-0201	AMACH
BDLX	A52-31-0302	
BICD1	E12-32-1102	UBAS1, UCIO1,ULUI1, UMIO1
BICD3	E12-32-3302	UBAS1, UCIO3, ULUI3, UMIO3
BIC1	E12-31-0102	UBAS1, UCIO1,ULUI1, UMIO1
BIC2	E12-31-0202	UBAS1, UCIO2, ULUI1, UMIO2
BIC3	E12-31-0302	UBAS1, UCIO3, ULUI3, UMIO3
BIC4	E12-31-0402	UBAS4, UCIO4, ULUI4, UMIO4, UPEP4
BIFD1	E11-32-1101	UBAS1, UCAD1
BIFD3	E11-32-3301	UBAS1, UCAD1
BIF1	E11-31-0101	UBAS1, UCAR1
BIF2	E11-31-0201	UBAS1, UCAR1
BIF3	E11-31-0301	UBAS1, UCAR1
BIF4	E11-31-0401	UBAS4, UCAR4, UPEP4
BIN	I11-81-1201	AMACH,BI0, BI1, AFMIN, AFMAX, ULMAX
BIR	I11-83-0301	AFMIN, AMACH, ULMAX
BI0	I11-81-0601	AFMAX, ULMAX
BI1	I11-81-0701	AFMAX, ULMAX
BJN	I11-81-1001	AMACH, BJ0, BJ1, AFMIN, UTLIM
BJR	I11-83-0101	AMACH, AFMIN, UTLIM
BJ0	I11-81-0201	UTLIM
BJ1	I11-81-0301	UTLIM
BKN	I11-81-1301	BK0, BK1, ULMAX
BKR	I11-83-0401	AMACH, AFMAX, ULMAX
BK0	I11-81-0801	ULMAX
BK1	I11-81-0901	ULMAX
BLNC	B21-11-0202	IRADIX
BLUX1	A52-11-0302	ASUM
BLU1	A52-11-0202	AMACH
BMDMX	A52-21-0302	
BSCD2	E32-32-0202	UBAS0, UCDB2, UPOB2, UPCA2 UREO1
BSCT1	B21-21-0502	AMACH
BSC1	E32-31-0102	UBAR1, UCAO1, UCDB1, UNCA1, UREO1
BSC2	E32-31-0202	UBRS0, UCDB2, UPOB1, UPCA1, UREO1, AFMAX
BSEG	B51-21-0201	AMACH,BSCT1, BSVEC, BTRID
BSEGJ	B51-21-1001	AMACH, BDLX, MSBV, SBDL, TEIG1, TRBK, TRID1, UCHLS, UESRT
BSFD1	E31-32-0101	UBAS1, UCAR2
BSF1	E31-31-0101	UBAS1, UCAR1
BSVEC	B51-21-0402	AMACH
BTRID	B51-21-0302	AMACH
BYN	I11-81-1101	BY0, BY1, UBJ0, UBJ1, UTLIM
BYR	I11-83-0202	AMACH, AFMAX, UTLIM, ULMAX
BY0	I11-81-0401	UBJ0, UTLIM
BY1	I11-81-0501	UBJ1, UTLIM
CBIN	I11-82-1101	AMACH, ULMAX
CBJN	I11-82-1301	AMACH, ULMAX
CBJR	I11-84-0101	AMACH, ULMAX
CBKN	I11-82-1201	CBIN, AMACH, ULMAX, UTLIM
CBLNC	B21-15-0202	IRADIX
CBYN	I11-82-1401	CBIN, CBKN, AMACH, ULMAX, UTLIM
CEIG2	B21-15-0101	CBLNC, CHES2, CNRML, AMACH, CSUM, IRADIX
CELI1	I11-11-0101	
CELI2	I11-11-0201	
CFRI	I11-51-0201	UTLIM
CFT	F12-15-0101	CFTN, PNR
CFTM	F12-11-0101	UCFTM
CFTN	F12-15-0202	
CFTR	F12-15-0302	
CGSBM	A11-40-0101	
CGSM	A11-10-0101	
CHBK2	B21-15-0602	
CHES2	B21-15-0302	AMACH, CSUM
CHSQR	B21-15-0402	AMACH
CHVEC	B21-15-0502	AMACH, CSUM
CJART	C22-15-0101	AMACH, CQDR, UCJAR
CLU	A22-15-0202	AMACH, CSUM
CLUIV	A22-15-0602	CSUM
CLUX	A22-15-0302	CSUM
CNRML	B21-15-0702	
COSI	I11-41-0201	UTLIM
CQDR	C21-15-0101	
CSBGM	A11-40-0201	
CSBSM	A11-50-0201	
CSGM	A11-10-0201	
CSSBM	A11-50-0101	
CTSDM	C23-15-0101	AMACH, AFMAX, AFMIN
ECHEB	E51-30-0201	
ECOSP	E51-10-0201	
EIGI	B21-11-0101	AMACH, BLNC, HES1, IRADIX
ESINP	E51-20-0201	
EXPI	I11-31-0101	ULMAX, AFMAX
FCHEB	E51-30-0101	AMACH, UTABT, UCOSM
FCOSF	E51-10-0101	AMACH, UTABT, UCOSM, UNIFC
FCOSM	F11-11-0201	UCOSM, UPNR2, UTABT
FCOST	F11-11-0101	UCOSM, UPNR2, UTABT
FSINF	E51-20-0101	AMACH, UTABT, USINM, UNIFC
FSINM	F11-21-0201	USINM, UPNR2, UTABT
FSINT	F11-21-0101	USINM, UPNR2, UTABT
GBSEG	B52-11-0101	AMACH, MSBV, TRID1, TEIG1, TRBK, UCHLS, UBCHL, UBCLX, UESRT
GCHEB	E51-30-0301	
GINV	A25-31-0101	ADVD1, AMACH

Table B.1 -continued

TableB.1-conteimued

Subroutine name	Classification code	Subprogram used
GSBK	B22-10-0402	
GSCHL	B22-10-0302	AMACH, UCHLS
GSEG2	B22-10-0201	AMACH, GSBK, GSCHL, TRBK, TRID1, UCHLS, UTEG2
HAMNG	H11-20-0121	AMACH, RKG
HBK1	B21-11-0602	NRML
HEIG2	B21-25-0201	AMACH, TEIG2, TRBKH, TRIDH, UTEG2
HES1	B21-11-0302	AMACH
HRWIZ	F20-02-0201	AMACH
HSQR	B21-11-0402	AMACH
HVEC	B21-11-0502	AMACH
ICHEB	E51-30-0401	
IERF	I11-71-0301	IERFC
IERFC	I11-71-0401	IERF
IGAM1	I11-61-0101	AMACH, IGAM2, AFMAX, EXPI, ULMAX
IGAM2	I11-61-0201	AMACH, EXPI, ULMAX, AFMAX
INDF	I11-91-0301	IERF, IERFC
INDFC	I11-91-0401	IERF, IERFC
INSPL	E12-21-0101	
LAPS1	F20-01-0101	AFMAX
LAPS2	F20-02-0101	LAPS1, HRWIZ, AFMAX, AMACH
LAPS3	F20-03-0101	
LAX	A22-11-0101	ALU, AMACH, LUX
LAXL	A25-11-0101	AMACH, ULALB, ULACH
LAXLM	A25-21-0101	ADVD1, AMACH
LAXLR	A25-11-0401	AMACH, MAV, ULALB
LAXR	A22-11-0401	AMACH, LUX, MAV
LBX1	A52-11-0101	AMACH, BLUX1, BLU1
LBX1R	A52-11-0401	AMACH, BLUX1, MVB
LCX	A22-15-0101	AMACH, CLUX, CLUX, CSUM
LCXR	A22-15-0401	AMACH, CLUX, CSUM, MCV
LDIV	A22-51-0702	
LDLX	A22-51-0302	
LESQ1	E21-20-0101	AMACH
LMINF	D11-30-0101	AMACH
LMING	D11-40-0101	AMACH
LOWP	C21-41-0101	MAMCH, RQDR, UREDE, U3DEG
LPRS1	D21-10-0101	ALU, LUIV, AMACH
LSBIX	A52-21-0101	SBMDM, BMDMX, AMACH
LSBX	A52-31-0101	AMACH, BDLX, SBDL
LSBXR	A52-31-0401	AMACH, BDLX, MSBV
LSIX	A22-21-0101	AMACH, MDMX, SMDM, USCHA
LSIXR	A22-21-0401	MDMX, MSV, AMACH
LSTX	A52-31-0501	AMACH
LSX	A22-51-0101	AMACH, LDLX, SLDL
LSXR	A22-51-0401	AMACH, LDLX, MSV
LTX	A52-11-0501	AMACH
LUIV	A22-11-0602	
LUX	A22-11-0302	
MAV	A21-13-0101	
MBV	A51-11-0101	
MCV	A21-15-0101	
MDMX	A22-21-0302	
MGGM	A21-11-0301	
MGSM	A21-11-0401	
MINF1	D11-10-0101	AMACH, LDLX, UMLDL

TableB.1-continued

Subroutine name	Classification Code	Subprogram used
MING1	D11-20-0101	AMACH, AFMAX, MSV
MSBV	A51-14-0101	
MSGM	A21-12-0401	CSGM, MGGM
MSSM	A21-12-0301	CSGM, MGSM
MSV	A21-14-0101	
NDF	I11-91-0101	
NDFC	I11-91-0201	
NLPG1	D31-20-0101	AMACH, UQP, UNLPG
NOLBR	C24-11-0101	AMACH
NOLF1	D15-10-0101	AMACH
NOLG1	D15-20-0101	AMACH
NRML	B21-11-0702	
ODAM	H11-20-0141	AMACH, UDE, USTE1, UINT1
ODGE	H11-20-0151	AMACH, USDE,UINT2, USTE2, USETC, USETP, USOL, UADJU, UDEC
ODRK1	H11-20-0131	AMACH, URKV, UVER
PNR	F12-15-0402	
RANB2	J12-20-0101	
RANE2	J11-30-0101	RANU2
RANN1	J11-20-0301	
RANN2	J11-20-0101	RANU2
RANP2	J12-10-0101	ULMAX
RANU2	J11-10-0101	
RANU3	J11-10-0201	RANU2
RATF1	J21-10-0101	UX2UP
RATR1	J21-10-0201	UX2UP
RFT	F11-31-0101	CFTN, PNR, URFT
RJETR	C22-11-0101	AFMAX, AFMIN, AMACH, IRADIX, RQDR, UJET
RKG	H11-20-0111	
RQDR	C21-11-0101	AMACH
SBDL	A52-31-0202	AMACH
SBMDM	A52-21-0202	AMACH
SEIG1	B21-21-0101	AMACH,TEIG1, TRID1, TRBK
SEIG2	B21-21-0201	AMACH, TEIG2, TRBK, TRID1, UTEG2
SFRI	I11-51-0101	UTLIM
SGGM	A21-11-0201	
SIMP1	G21-11-0101	
SIMP2	G23-11-0101	AMACH
SINI	I11-41-0101	UTLIM
SLDL	A22-51-0202	AMACH
SMDM	A22-21-0202	AMACH, USCHA
SMLE1	E31-11-0101	
SMLE2	E31-21-0101	
SPLV	E11-21-0101	USPL
SSSM	A21-12-0201	
TEIG1	B21-21-0602	AMACH
TEIG2	B21-21-0702	AMACH, UTEG2
TRAP	G21-21-0101	
TRBK	B21-21-0802	
TRBKH	B21-25-0402	
TRIDH	B21-25-0302	AMACH
TRID1	B21-21-0302	AMACH
TRQL	B21-21-0402	AMACH
TSDM	C23-11-0111	AFMAX, AFMIN, AMACH
TSD1	C23-11-0101	AMACH

Note:
In the "Subprograms used"column, any subroutine names (except MGSSL) which are called in each subroutine are listed.

B.2 SLAVE SUBROUTINES

Table B.2 shows slave subroutines.

TableB.2 Slave subroutines

Subroutine name	Calling subroutines	Subroutine nama	Calling subroutines
UADJU	ODGE	ULU14	BIC4
UAQE1	AQME	UMIO1	BICD1, BIC1
UAQE2	AQME	UMIO2	BIC2
UAQE3	AQME	UMIO3	BICD3, BIC3
UBAR1	BSC1	UMIO4	BIC4
UBAS0	BSCD2, BSC2	UMLDL	MINF1
UBAS1	BICD1, BICD3, BIC1, BIC2, BIC3, BIFD1, BIFD3, BIF1, BIF2, BIF3, BSF1,BSFD1	UNCA1	BSC1
		UNIFC	FSINF, FCOSF
UBAS4	BIC4, BIF4	UNLPG	NLPG1
UBCHL	GBSEG	UPCA1	BSC2
UBCLX	GBSEG	UPCA2	BSCD2
UBJ0	BY0	UPEP4	BIF4
UBJ1	BY1	UPNR2	FCOSM, FCOST, FSINM, FSINT
UCAD1	BIFD1, BIFD3	UPOB1	BSC2
UCAO1	BSC1	UPOB2	BSCD2
UCAR1	BIF1, BIF2, BIF3, BSF1	UQP	NLPG1
UCAR2	BSFD1	UREO1	BSC1, BSC2, BSCD2
UCAR4	BIF4	UREDR	LOWP
UCDB1	BSC1	URFT	RFT
UCDB2	BSCD2, BSC2	URKV	ODRK1
UCFTM	CFTM	USCHA	SMDM
UCHLS	BSEGJ, GBSEG, GSCHL, GSEG2	USINM	FSINM, FSINT
UCIO1	BICD1, BIC1	USOL	ODGE
UCIO2	BIC2	USPL	SPLV
UCIO3	BICD3, BIC3	USTE1	ODAM
UCIO4	BIC4	USTE2	ODGE
UCJAR	CJART	UTABT	FCOSM, FCOST, FSINM, FSINT
UCOSM	FCOSM, FCOST	UTEG2	GSEG2, TEIG2
UDE	ODAM	UTLIM	BJR, BJ0, BJ1, BYR,BY0, BY1 ,CBKN, CFRI, COSI, SFRI, SINI
UDEC	ODGE		
UESRT	BSEGJ, GBSEG	UVER	ODRK1
UFN10	AQME	UX2UP	RATF1, RATR1
UFN20	AQME	U3DEG	LOWP
UFN30	AQME		
UFN40	AQME		
UNT1	ODAM		
UNIT2	ODGE		
UJET	RJETR		
ULALB	LAXL, LAXLR		
ULALH	LAXL		
ULUI1	BICD1, BIC1, BIC2		
ULMAX	BIR, BI0, BI1, BKR, BK0, BK1, BYR, CBIN, CBJN, CBJR, CBKN, EXPI, IGAM2, RANP2		
ULU13	BICD3, BIC3		

Note:
Each slave subroutine listed is called directly by the subroutine listed in its"Calling subroutines" colmn.

B.3 AUXILIARY SUBROUTINES

Table B.3 shows auxiliary subroutines.

Tabale B.3 Auxiliary subroutines

Auxiliary subroutine name	Calling subroutines
AMACH	(Called by many general and slave subroutines.)
MGSET	(Called by the user of SSL II.)
MGSSL	(Called by all general subroutines.)
IRADIX	BLNC, CBLNC, RJETR
ASUM	
BSUM	
CSUM	GEIG2, CHES2, CHVEC, CLU, CLUX, CLUIV
AFMAX	AKHER, AKLAG, AQME, BI0, BI1, BKR, BYR, CTSDM, EXPI, IGAM2, MING1, RJETR, TSDU, UPOB1
AFMIN	AQE, AQME, BIN, BIR, BJN, BJR, CTSDM, RJETR, TSDM

Note:
Refer to Appendix A.
MGSET is the sub-entry of MGSSL.
Each auxiliary subroutine listed is called directly by the subroutine listed in its"Calling subroutines"column.

APPENDIX C CLASSIFICATION CODES AND SUBROUTINES

Liner algebra

Code	Subroutine
A11-10-0101	CGSM
A11-10-0201	CSGM
A11-40-0101	CGSBM
A11-40-0201	CSBGM
A11-50-0101	CSSBM
A11-50-0201	CSBSM
A21-11-0101	AGGM
A21-11-0201	SGGM
A21-11-0301	MGGM
A21-11-0401	MGSM
A21-12-0101	ASSM
A21-12-0201	SSSM
A21-12-0301	MSSM
A21-12-0401	MSGM
A21-13-0101	MAV
A21-14-0101	MSV
A21-15-0101	MCV
A22-11-0101	LAX
A22-11-0202	ALU
A22-11-0302	LUX
A22-11-0401	LAXR
A22-11-0602	LUIV
A22-15-0101	LCX
A22-15-0202	CLU
A22-15-0302	CLUX
A22-15-0401	LCXR
A22-15-0602	CLUIV
A22-21-0101	LSIX
A22-21-0202	SMDM
A22-21-0302	MDMX
A22-21-0401	LSIXR
A22-51-0101	LSX
A22-51-0202	SLDL
A22-51-0302	LDLX
A22-51-0401	LSXR
A22-51-0702	LDIV
A25-11-0101	LAXL
A25-11-0401	LAXLR
A25-21-0101	LAXLM
A25-31-0110	GINV
A25-31-0201	ASVD1
A51-11-0101	MBV
A51-14-0101	MSBV
A52-11-0101	LBX1
A52-11-0202	BLU1
A52-11-0302	BLUX1
A52-11-0401	LBX1R
A52-11-0501	LTX
A52-21-0101	LSBIX

Linear algebra-continued

Code	Subroutine
A52-21-0202	SBMDM
A52-21-0302	BMDMX
A52-31-0101	LSBX
A52-31-0202	SBDL
A52-31-0302	BDLX
A52-31-0401	LSBXR
A52-31-0501	LSTX

Eigenvalues and eigenvectors

Code	Subroutine
B21-11-0101	EIGI
B21-11-0202	BLNC
B21-11-0302	HES1
B21-11-0402	HSQR
B21-11-0502	HVEC
B21-11-0602	HBK1
B21-11-0702	NRML
B21-15-0101	CEIG2
B21-15-0202	CBLNC
B21-15-0302	CHES2
B21-15-0402	CHSQR
B21-15-0502	CHVEC
B21-15-0602	CHBK2
B21-15-0702	CNRML
B21-21-0101	SEIG1
B21-21-0201	SEIG2
B21-21-0302	TRID1
B21-21-0402	TRQL
B21-21-0502	BSCT1
B21-21-0602	TEIG1
B21-21-0702	TEIG2
B21-21-0802	TRBK
B21-25-0201	HEIG2
B21-25-0302	TRIDH
B21-25-0402	TRBKH
B22-21-0201	GSEG2
B22-21-0302	GSCHL
B22-21-0402	GSBK
B51-21-0201	BSEG
B51-21-0302	BTRID
B51-21-0402	BSVEC
B51-21-0001	BSEGJ
B52-11-0101	GBSEG

Non-linear equations

Code	Subroutine
C21-11-0101	RQDR
C21-15-0101	CQDR
C21-11-0101	LOWP
C22-11-0111	RJETR
C22-15-0101	CJART
C23-11-0101	TSD1
C23-11-0111	TSDM
C23-15-0101	CTSDM
C24-11-0101	NOLBR

Extrema

Code	Subroutine
D11-10-0101	MINF1
D11-20-0101	MING1
D11-30-0101	LMINF
D11-40-0101	LMING
D15-10-0101	NOLF1
D15-20-0101	NOLG1
D21-10-0101	LPRS1
D31-20-0101	NLPG1

Interpolation and approximation

Code	Subroutine
E11-11-0101	AKLAG
E11-11-0201	AKHER
E11-21-0101	SPLV
E11-31-0101	BIF1
E11-31-0201	BIF2
E11-31-0301	BIF3
E11-31-0401	BIF4
E11-32-1101	BIFD1
E11-32-3301	BIFD3
E11-42-0101	AKMID
E12-21-0101	INSPL
E12-21-0201	AKMIN
E12-31-0102	BIC1
E12-31-0202	BIC2
E12-31-0302	BIC3
E12-31-0402	BIC4
E12-32-1102	BICD1
E12-32-3302	BICD3
E21-20-0101	LESQ1
E31-11-0101	SMLE1
E31-21-0101	SMLE2
E31-31-0101	BSF1
E32-31-0102	BSC1
E32-31-0202	BSC2
E31-32-0101	BSFD1
E32-32-0202	BSCD2
E51-10-0101	FCOSF
E51-10-0201	ECOSP
E51-20-0101	FSINF
E51-20-0201	ESINP
E51-30-0101	FCHEB
E51-30-0201	ECHEB
E51-30-0301	GCHEB
E51-30-0401	ICHEB

Transforms

Code	Subroutine
F11-11-0101	FCOST
F11-11-0201	FCOSM
F11-21-0101	FSINT
F11-21-0201	FSINM
F11-31-0101	RFT
F12-11-0101	CFTM
F12-15-0101	CFT
F12-15-0202	CFTN
F12-15-0302	CFTR
F12-15-0402	PNR
F20-01-0101	LAPS1
F20-02-0101	LAPS2
F20-02-0201	HRWIZ
F20-03-0101	LAPS3

Numerical differentiation and quadrature

Code	Subroutine
G21-11-0101	SIMP1
G21-21-0101	TRAP
G23-11-0101	SIMP2
G23-11-0201	AQN9
G23-11-0301	AQC8
G23-11-0401	AQE
G23-21-0101	AQEH
G23-31-0101	AQEI
G24-13-0101	AQMC8
G24-13-0201	AQME

Differential equations

Code	Subroutine
H11-20-0111	RKG
H11-20-0121	HAMNG
H11-20-0131	ODRK1
H11-20-0141	ODAM
H11-20-0151	ODGE

Special functions

Code	Subroutine
I11-11-0101	CELI1
I11-11-0201	CELI2
I11-31-0101	EXPI
I11-41-0101	SINI
I11-41-0201	COSI
I11-51-0101	SFRI
I11-51-0201	CFRI
I11-61-0101	IGAM1
I11-61-0201	IGAN2
I11-71-0301	IFRF
I11-71-0401	IERFC
I11-81-0201	BJ0
I11-81-0301	BJ1
I11-81-0401	BY0
I11-81-0501	BY1
I11-81-0601	BI0
I11-81-0701	BI1
I11-81-0801	BK0
I11-81-0901	BK1
I11-81-1001	BJN
I11-81-1101	BYN
I11-81-1201	BIN
I11-81-1301	BKN
I11-82-1101	CBIN
I11-82-1201	CBKN
I11-82-1301	CBJN
I11-82-1401	CBYN
I11-83-0101	BJR
I11-83-0201	BYR
I11-83-0301	BJR
I11-83-0401	BKR
I11-84-0101	CBJR
I11-91-0101	NDF
I11-91-0201	NDFC
I11-91-0301	INDF
I11-91-0401	INDFC

Pseudo random numbers

Code	Subroutine
J11-10-0101	RANU2
J11-10-0201	BANU3
J11-20-0101	RANN2
J11-20-0301	RANN1
J11-30-0101	RANE2
J12-10-0101	RANP2
J12-20-0101	RANB2
J21-10-0101	RATF1
J21-10-0201	RATR1

APPENDIX D REFERENCES

[1] Forsythe. G.E and Moler, C.B.Computer Solution of Linear Algebraic Systems,Prentice-Hall, Inc., 1967

[2] Martin R.S.,Peters, G. and Wilkinson, J.H. Symmetric Decomposition of A Positive Definite Matrix. Linear Algebra, Handbook for Automatic Computation, Vol.2, pp.9-30, Springer-Verlag, Berlin-Heidelberg-New York,1971

[3] Bowdler,H.J., Martin, R.S. and Wilkinson, J.H. Solution of Real and Complex Systems of Linear Equations, Linear Algebra, Handbook for Automatic Computation, Vol.2,pp.93-110,Springer-Verlag, Berlin-Heidelberg-New York,1971

[4] Parlett,B.N. and Wang,Y.The Influence of The Compiler on The Cost of Mathematical Software-in Particular on The Cost of Triangular Factorization, ACM Transactions on Mathematical Software,Vol.1,No.1,pp.35-46, March, 1975

[5] Wilkinson, J.H. Rounding Errors in Algebraic Process, Her Britannic Majesty’s Stationary Office,London, 1963

[6] Peter Businger and Gene H. Golub. Linear Least Squares Solutions by Householder Transformatinos,Linear Algebra,Handbook for Automatic Computation,Vol.2,pp.111-118, Springer-Verlag, Berlin-Heidelberg-New York,1971

[7] Martin,R.S and Wilkinson,J.H. Symmetric Decomposition of Posive Definite. Band Matrices,Linear Algebra,Handbook for Automatic Computation,Vol.2,pp.50-56, Springer-Verlag, Berlin-Heidelberg-New York ,1971

[8] Martin,R.S. and Wilkinson,J.H. Solution of Symmetric and Unsymmetric Band Equations and the Calculations of Eigenvectors of Band Matrices,Linear Algebra,Handbook for Automatic Computation,Vol.2,pp.70-92, Springer-Verlag, Berlin Heidelberg New York,1971

[9] Bunch,J.R and Kaufman,L. Some Stable Methods for Calculation Inertia and Solving Symmetric.Linear Systems, Mathematics of Computation, Vol.13,No.137,January 1977,pp.163-179

[10] Bunch,J.R. and Kaufman, L. Some Stable Methods for Calculation Inertia and Solving Symmertric Linear Systems,Univ. of Colorado Tech. Report 63, CU:CS:06375

[11] Golub,G.H. and Reinsch,C. Singular Value Decomposition and Least Squares Solutions, Linear Algebre,Handbook for Automatic Computation,Vol.2,pp.134-151,Springer-Verlag,Berlin Heidelberg-New York, 1971

[12] Wilkinson,J.H. The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965

[13] Wilkinson,J.H.and Reinsch, C.Linear Algebra, Handbook for Automatic Computation,Vol.2,Springer-Verlag, Berlin-Heidelberg-New York,1971

[14] Smith,B.T., Boyle, J.M.,Garbow,B.S.,Ikebe,Y.,Kleme,V.C and Moler,C.B. Matrix Eigensystem Routine-EISPACK Guide 2nd edit. Lecture Note in Computer Science 6, Springer-Verlag,Berlin-Heidelberg-New York,1976.

[15] Ortega,J. The Givens-Householder Method for Symmetric Matrix, Mathematical Methods for Digital Computors,Vol.2, John wiley&Sons, New York-London-Sydney,pp.94-115,1967

[16] Togawa,H.
Numerical calculations of matrices, chap.3, pp.146-210, Ohmsha Ltd., Tokyo, 1971 (Japanese only)

[17] Mueller,D.J Householder’s Method for Complex Matrices and Eigensystems of Hermitian Matrices, Numer.Math.8,pp.72-92,1996

[18] Jennings,A. Matrix Computation for Engineers and Scientists,J.Wiley, pp.301-310,1977

[19] Togawa,H.
Vibration analysis by using FEM, chap.4, pp.156-162, Saiensu-sha Co., 1970. (Japanese only)

[20] Brezinski, C.
Acceleration de la Convergence en Analyse Numerique, Lecture Notes in Mathematics 584, Springer, pp. 136-159, 1977.

[21] Wynn, P.
Acceleration Techniques for Iterated Vectol and Matrix Problems, Mathematics of Computation, Vel. 16, pp. 301-322, 1962.

[22] Yamashita,S.
On the Error Estimation in Floating-point Arithmetic, Journal of Information Processing Society of Japan, Vol.15, No.12, pp.935-939, 1974 (Japanese only)

[23] Yamashita,S. and Satake,S.
On the calculation limit of roots of the algebraic equation,
Journal of Information Processing Society of Japan, Vol.7, No.4, pp.197-201, 1966 (Japanese only)

[24] Hirano,S.
Error of the algebraic equation
- from "Error in the Numerical Calculation",
bit, extra number 12, pp.1364-1378, 1975 (Japanese only)

[25] Kiyono,T. and Matsui,T.
Solution of a low polynomial,
"Information", Data Processing Center, Kyoto University, Vol.4, No.9,
pp.9-32, 1971 (Japanese only)

[26] Garside, G.R.,Jarratt,P. and Mack, C. A New Method for Solving Polynomial Equations,Computer Journal, Vol. 11,pp.87-90,1968

[27] Ninomiya,I.
GJM solution of a polynomial,
IPSJ National Conference, p.33, 1969 (Japanese only)

[28] Brent, Richard P.
Algorithms for Minimization without Derivatives, Prentice-Hall, London-Sydney-ToKyo,
pp. 47-60, 1973

[29] Peters, G. and Wilkinson, J.H. Eigenvalues of $Ax=\lambda Bx$ with Band Symmetric A and B, Comp. J. 12,pp.398-404,1969

[30] Jenkins, M. A. and Traub, J.F. A three-stage algorithm for real polynomials using quadratic iteration, SIAM J. Number. Anal., Vol.17,pp.545-566, 1970

[31] Jenkins, M. A.Algorithm 493 Zeros of a real polynomial, ACM Transaction on Mathematical Software, Vol.1,No.2,pp.178-187,1975

[32] Traub, J. F. The Solution of Transcendental Equations, Mathematical Methods for Digital Computers, Vol. 2, 1967, pp. 171-184

[33] Cosnard, M. Y. A Comparison of Four Methods for Solving Systems of Nonlinear Equations, Cornell University Computer Science Technical Report TR75-248, 1975

[34] Fletcher, R. Fortran subroutines for minimization by quasi-Newton methods, Report R7125 AERE, Harwell, England, 1972

[35] Shanno, D. F. and Phua, K. H. Numerical comparison of several varible metric algorithms J. of Optimization Theory and Applications, Vol. 25, No. 4, 1978

[36] Marquardt, D. W. An algorithm for least squares estimatron of nonlinear parameters, Siam J. Appl. Math., 11, pp. 431-441, 1963

[37] Osborne, M. R. Nonlinear least squares-The Levenberg algorithm revisited, J. of the Australian Mathematical Vociety, 19, pp. 343-357, 1976

[38] Moriguchi,S.
Primer of Linear Programming.
JUSE Press Ltd., 1957, remodeled,1973. (Japanese only)

[39] Ono,K.
Linear Programming on the calculation basis.
JUSE Press Ltd., 1967, remodeled,1976. (Japanese only)

[40] Tone,K.
Mathematical Programming.
Asakura Publishing, 1978. (Japanese only)

[41] Kobayashi,T.
Primer of Linear Programming.
Sangyo Tosho, 1980. (Japanese only)

[42] Kobayashi,T.
Theory of Network.
JUSE Press Ltd., 1976. (Japanese only)

[43] Dantzig, G.
Linear Programming and Extensions, Princeton University Press, 1963.

[44] Simonnard, M. (translated by W. S. Jewell) Linear Programming, Prentice-Hall,1966.

[45] Vande Panne, C. Linear Programming and Related Techniqes, North-Holland, 1971,

[46] Ralston, A. A First Conrse In Numerical Analysis, Mc Graw-Hill, New York-Tronto-London, 1965.

[47] Gershinsky, M. and Levine, D. A. Aitken-Hermite Interpolation, Journal of the ACM, Vol. 11, No. 3, 1964, pp. 352-356.

[48] Greville, T. N. E. Spline Functon, Interpolation and Numerical Quadrature, Mathematical Methods for Digital Computers, Vol. 2, Johe Wiley & Sons, New York-London-Sydney, 1967, pp. 156-168.

[49] Ahlberg, J. H., Nilson, E:N. and Walsh, J. L. The Theory of Splines and Their Applications. Academic Press, New York and London, 1967.

[50] Hamming R. W. Numerical Methods for Scientists and Engineers, McGraw-Hill, New York-Tronto-London, 1973

[51] Hildebrand, F. B. Introduction to Numerical Analysis, sec, ed. McGraw-Hill, 1974.

[52] Akima, H. A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedure Journal of the ACM, Vol. 17, No. 4, 1970. pp. 589-602.

[53] Carl de Boor.
On Calculating with B-splines,
J.Approx. Theory, Vol.6, 1972, pp.50-62.

[54] Akima, H.
Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures, Comm. Of the ACM., Vol.17, No.1, 1974, pp.26-31

[55] Singlton, R.C.
On Computing the Fast Fourier Transform, Communications of The ACM,
Vol.10, No.10, pp. 647-654, October, 1967

[56] Singlton, R.C.
An ALGOL Convolution Procedure Based On The Fast Fouier Tronsform, Communications of The ACM, Vol.12, No.3, pp.179-184, March, 1969

[57] Singlton, R.C.
An Algorithm for Computing The Mixed Radix Fast Fourier Transform,
IEEE Transactions on Audio and Electroacoustics, Vol.AU-17, No.2, pp.93-103, June, 1969

[58] Torii,T.
Fast sine/cosine transforms and the application to the numerical quadrature.
Journal of Information Processing Society of Japan, Vol.15, No.9, pp.670-679,1974 (Japanese only)

[59] Torii,T.
The subroutine package of Fourier transforms (Part 1,2).
Nagoya University Computation Center News,
Vol.10, No.2, 1979 (Japanese only)

[60] Fox, L. and Parker, I.B.
Chebyshev Polynomials in Numerical Analysis, Oxford University Press, 1972

[61] Lyness, J.N.
Notes on the Adaptive Simpson Quadrature Routine, Journal of the ACM, Vol.16, No.3, 1969, PP.483-495

[62] Davis, P.J. and Rabinowitz,P.
Methods of Numerical Intergration, Academic Press, 1975

[63] Kahaner, D.K.
Comparison of numerical quadrature formulas.
In "Mathematical Software" (Ed. J.R. Rice), Academic Press, 1971, pp.229-259

[64] De Boor, C.
CADRE:An algorithm for numerical quadrature.
In "Mathematical Software" (Ed. J.R. Rice), Academic Press, 1971, pp.417-449

[65] Clenshaw, C.W. and Curtis, A.R.
A method for numerical integration on an automatic computer.
Numer.Math.2, 1960, pp.197-205

[66] Torii,T., Hasegawa,T., Ninomiya,I.
The automatic integral method with the interpolation increasing sample points in arithmetic progression.
Journal of Information Processing Society of Japan, Vol.19, No.3, 1978, pp.248-255 (Japanese only)

[67] Takahashi, H. and Mori, M.
Double Exponential Formulas for Numerical Integration. Publications of R.I.M.S, Kyoto Univ. 9, 1974, pp.721-741

[68] Mori,M.
Curves and curved surfaces.
Kyoiku Shuppan, 1974 (Japanese only)

[69] Romanelli, M.J.
Runge-Kutta Methods for Solution of Ordinary Differential Equations, Mathematical Methods for Digital Computer, Vol.2, John Wiley & Sons,
New York-London-Sydney, 1967, PP.110-120

[70] Hamming, R.W.
Stable Predictor Corrector Methods for Ordinary Differential Equations, Jouranl of the ACM, Vol.6, 1956, PP.37-47

[71] Shampine, L.F. and Gordon, M.K.
Computer Solution of Ordinary Differential Equations, Freeman, 1975

[72] Verner, J.H.
Explicit Runge-Kutta methods with estimate of the local truncation error, SIAM J. Numer. Anal Vol.15, No.4, 1978 pp.772-790

[73] Jackson, K.R., Enright, W.H. and Hull, T.E.
A theoretical criterion for comparing Runge-Kutta formulas,SIAM J.Numer, Anal., Vol.15, No.3, 1978, pp.618-641

[74] Gear, C.W.
Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, 1971

[75] Lambert, J.D.
Computational Methods in Ordinary Differential Equations, Wiley, 1973

[76] Hindmarsh, A.C. and Byrne, G.D.
EPISODE:An Effective Package for the Integration of Systems of Ordinary Differential Equations, UCID-30112, Rev.1, Lawrence Livermore Laboratory, April 1977

[77] Byrne, G.D. and Hindmarsh, A.C.
A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations, ACM Transaction of Math. Soft., Vol.1, No.1, pp.71-96, March 1975.

[78] Hart, J.F.
Complete Elliptic Integrals, John Wiley & Sons, Computer Approximations, 1968.

[79] Cody, W.J. and Thacher Jr, C.H.
Rational Chebyshev Approximations for the Exponential Integral E_i, (x),
Mathematics of Computation, Vol.22, pp.641-649, July 1968.

[80] Cody, W.J. and Thacher Jr, C.H.
Chebyshev Approximations for the Expontial Integral E_i (x), Mathematics of Computation, Vol.23, pp.289-303, Apr. 1969.

[81] Ninomiya,I.
The calculation of Bessel Function by using the recurrence formula.
Baifukan Co., Numerical calculation for the computer II,
pp.103-120, 1967. (Japanese only)

[82] Uno,T.
Bessel Function.
Baifukan Co., Numerical calculation for the computer III,
pp.166-186, 1972. (Japanese only)

[83] Yoshida,T., Asano,M., Umeno,M. and Miki,S.
Recurrence Techniques for the Calculation of Bessel Function $I_n(z)$
with Complex Argument.
Journal of Information Processing Society of Japan, Vol.14, No.1, pp.23-29, Jan.1973. (Japanese only)

[84] Yoshida,T., Ninomiya,I.
Computation of Bessel Function $K_n(n)$ with Complex Argument by Using the τ-method.
Journal of Information Processing Society of Japan, Vol.14, No.8, pp.569-575, Aug.1973. (Japanese only)

[85] Moriguchi, Utagawa and Hitotsumatsu
Iwanami Formulae of Mathematics III.
Iwanami Shoten, Publishers, 1967. (Japanese only)

[86] Forsythe,G.E. (Mori,M.'s translation)
Numerical Calculation for Computers
- the computer science researches book series No.41.
Kagaku Gijutsu Shuppan, 1978. (Japanese only)

[87] Streck, A.J.
On the Calculation of the Inverse of Error Function, Mathematics of Computation, Vol.22, 1968.

[88] Yoshida,T. and Ninomiya,I.
Computation of Modified Bessel Function $K_v(x)$ with Small Argument x.
Transactions of Information Processing Society of Japan,
Vol.21, No.3, pp.238-245, May.1980 (Japanese only)

[89] Nayler, T.H.
Computer Simulation Techiques, John Wiley & Sons, Inc., 1966, pp.43-67.

[90] Algorithm 334, Normal Random Deviates, Comm. ACM, 11 (July 1968) , p.498.

[91] Pike, M.C.
Algorithm 267, Random Normal Deviate, Comm. ACM, 8 (Oct. 1965), p.606.

[92] Wakimoto,K.
Knowledge of random numbers. Morikita Shuppan, 1970. (Japanese only)

[93] Miyatake,O. and Wakimoto,K.
Random numbers and Monte Carlo method. Morikita Shuppan, 1978. (Japanese only)

[94] Powell, M.J.D.
A Fast Algorithm for Nonlinearly constranied Optimization Calculations.
Proceedings of the 1977 Dundee Conference on Numerical Analysis, Lecture Notes in Mathematics, Springer-Verlag, 1978

[95] Powell, M.J.D., et al.
The Watchdog Technique for forcing Convergence in Algorithms for Constrained Optimization, Presented at the Tenth International Symposium on Mathematical Programming, Montreal, 1979

[96] Ninomiya,I.
Improvement of Adaptive Newton-Cotes Quadrature Methods.
Journal of Information Processing Society of Japan, Vol.21, No.5, 1980, pp.504-512. (Japanese only)

[97] Ninomiya, I.
Improvements of Adaptive Newton-Cotes Quadrature Methods, Joumal of Information Processing, Vol.3, No.3, 1980, pp.162-170.

[98] Hosono,T.
Numerical Laplace transform. Transactions A of IEEJ, Vol.99-A, No.10, Oct.,1979 (Japanese only)

[99] Hosono,T.
Basis of the linear black box. CORONA Publishing Co., 1980.
(Japanese only)

[100] Bromwich, T.J.I'A.
Introduction to the Theory of Infinite Series. Macmillan.1926

[101] Hosono, T.
Numerical inversion of Laplace transform and some applications to wave optics.
International U.R.S.I-Symposium 1980 on Electromagnetic Waves, Munich, 1980.

CONTRIBUTORS AND THEIR WORKS

Author	Subroutine	Item
M. Tanaka	ODRK1	A system of first order ordinary differential equations (Runge-Kutta-Verner method)
I. Ninomiya	CGSBM	Storage mode conversion of matrices (real general to real symmetric band)
	CSBGM	Storage mode conversion of matrices (real symmetric band to real general)
	CSSBM	Storage mode conversion of matrices (real symmetric to real symmetric band)
	CSBSM	Storage mode conversion of matrics (real symmetric band to real symmetric)
	LSBIX	A system of linear equations with a real indefinite symmetric band matrix (Block diagonal pivoting method)
	SBMDM	MDM^T decomposition of a real indefinite symmetric band matrix (Block diagonal pivoting method)
	BMDMX	A system of linear equations with real indefinite symmetirc band matrix decomposed into the factors M, D and M^T
	LAXLM	Least squares minimal norm solution with a real matrix (Singular value decomposition method)
	ASVD1	Singular value decomposition of a real matrix (Householder method, QR method)
	GINV	Moore-Penrose generalized inverse of a real matrix (Singular value decomposition method)
	CEIG2	Eigenvalues and corresponding eigenvectors of a complex matrix (QR method)
	CBLNC	Balancing of a complex matrix
	CHES2	Reduction of a complex matrix to a complex Hessenberg matrix (Stabilized elementary similarity transformation method)
	CHSQR	Eigenvaluse of a complex Hessenberg matrix (QR method)
	CHVEC	Eigenvectors of a complex Hessenberg matrix (Inverse iteration method)
	CHBK2	Back transformation of the eigenvectors of a complex Hessenberg matrix to those of a complex matrix
	CNRML	Normalization of eigenvectors of a complex matrix
	BSEG	Eigenvalues and corresponding eigenvectors of a real symmetric band matrix (Rutishauser-Shwarz method, bisection method and inverse iteration method)
	BTRID	Reduction of a rael symmetric band matrix to a tridiagonal matrix (Rutishauser-Schwarz method)
	BSVEC	Eigenvectors of a rael symmetric band matrix (Inverse iteration method)
	BSEGJ	Eigenvalues and corresponding eigenvectors of a real symmetric band matrix (Jennings method)
	GBSEG	Eigenvalues and corresponding eigenvectors of a real symmetric band generalized eigenproblem $Ax=\lambda BX$ (Jennings method)
	AQN9	Integration of a function by adaptive Newton-Cotes 9-point rule
	IERF	Inverse error function $erf^{-1}(x)$
	IERFC	Inverse complementary error function $erfc^{-1}(x)$
	NDF	Normal distribution function $\phi(x)$
	INDF	Inverse normal distribution function $\phi^{-1}(x)$
	NDFC	Complementary normal distribution function $\varphi(x)$
	INDFC	Inverse complementary normal distribution function $\varphi^{-1}(x)$
	RANNI	Fast normal pseudo random numbers
T. Torii	FCOSF	Fourier cosine series expansion of an even function (Function input, fast cosine transformation)
	ECOSP	Evaluation of a cosine series
	FSINF	Fourier sine series expansion of an odd functon (Function input, fast sine transformation)
	ESINP	Evaluation of a sine series
	FCHEB	Chebyshev series expansion of a real function

Author	Subroutine	Item
T. Torii	ECHEB	Evaluation of a Chebyshev series
	GCHEB	Differentiation of a Chebyshev series
	ICHEB	Indefinite integral for Chebyshev series
	FCOST	Discrete consine transform (Trapezoidal rule, radix 2 FFT)
	FCOSM	Discrete cosine transform (midpoint rule, radix 2 FFT)
	FSINT	Discrete sine transform (Trapezoidal rule, radix 2 FFT)
	FSINM	Discrete sine transform (midpoint rule, radix 2 FFT)
T. Hasegawa	AQC8	Integration of a function by a modified Clenshaw-Curtis rule
	AQMC8	Multiple integration of a function by a modified Clenshaw-Curtis rule
K.Hatano	BICD1	B-spline two-dimensional interpolation coefficient calculation (I -I)
	BICD3	B-spline two-dimensional interpolation coefficient calculation (III-III)
	BIC1	B-spline Interpolation coefficient calculation (I)
	BIC2	B-spline interpolation coefficient calculation (II)
	BIC3	B-spline interpolation coefficient calculation (III)
	BIC4	B-spline interpolation coefficient calculation (IV)
	BIFD1	B-spline two-dimensional interpolation, differentiation, and integration (I-I)
	BIFD3	B-spline two-dimensional interpolation, differentiation, and integration (III-III)
	BIF1	B-spline interpolation differentiation, and integration (I)
	BIF2	B-spline interpolation differentiation, and integration (II)
	BIF3	B-spline interpolation differentiation, and integration (III)
	BIF4	B-spline interpolation differentiation, and integration (IV)
	BSCD2	B-spline two-dimensional smoothing coefficient calculation variable knots
	BSC1	B-spline smoothing coefficient calculation with fixed knots
	BSC2	B-spline smoothing coefficient calculation variable knots
	BSFD1	B-spline two-dimensional smoothing
	BSF1	B-spline smoothing differentiation, and integration
Y.Hatano	AKMID	Two-dimensional quasi-Hermite interpolation
	AQE	Integration of a function by double exponential formula
	AQEH	Integration of a function over the semi-infinite interval by double exponential formula
	AQEI	Integration of a function over the infinite interval by double exponential formula
	AQME	Multiple integration of a function by double exponential formula
T.Yoshida	CBIN	Integer order modified Bessel function of the first kind with a complex variable, In (z)
	CBKN	Integer order modified Bessel function of the second kind with a complex variable,Kn (z)
	CBJN	Integer order Bessel function of the first kind with a complex variable, Jn (z)
	CBYN	Integer order Bessel function of the second kind with a complex variable, Yn (z)
	BJR	Real order Bessel function of the first kind, J*v* (x)
	BYR	Real-order Bessel function of the second kind, Y*v* (x)
	BIR	Real order modified Bessel function of the first kind, I*v* (x)
	BKR	Real order modified Bessel function of the second kind, K*v* (x)
	CBJR	Real order Bessel function of the first kind with a complex variable, J*v* (z)

K.Tone	LMINF	Minimization of function with a variable (Quadratic interpolation using function values only)
	LMING	Minimization of function with a variable (Qubic interpolation using function values and its derivatives)
	MING1	Minimization of a function with several variables (quasi-Newton method, using function values and its derivatives)
	NOLF1	Minimization of the sum of squares of functions with several variables (Revised Marquardt method, using function values only)
	NOLG1	Minimization of the sum of squares of functions (Revised Marquardt method using function values and its derivatives)
	NLPG1	Nonlinear programming problem (Powell’ s method using function values only)
T. Kobayashi	LPRS1	Solution of linear programming problem (Revised simplex menthod)
T. Hosono	LAPS1	Inversion of laplace transform of a rational function (analytic in the right half plane)
	LAPS2	Inversion of laplace transform of a rational function
	LAPS3	Inversion of laplace transform of a general function
	HRWIZ	Judgement on Hurwiz polynomials
L.F. Shampine	ODAM*	A system of first order ordinary differntial equations (Adams method)
A.C. Hindmarsh	ODGE*	A stiff system of first order ordinary differential equations (Gear’ s method)

* This program is based on that registered in ANL-NESC in U.S.A.
The original code is available direcly form the Center.

NESC: National Energy Software Center
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IIlinois 60439
U.S.A.