FUJITSU
SSL II EXTENDED CAPABILITIES
USER'S GUIDE II
(SCIENTIFIC SUBROUTINE LIBRARY)

FUJITSU

Fifth Edition September 2001

The contents of this manual may be revised without prior notice.

Preface

This manual describes how to use Scientific Subroutine Library II (SSL II) Extended Capabilities II.

This manual is a second volume for SSL II Extended Capabilities. This manual provides additional algorithms and functions that are effective for high-speed processing of large-scale scientific computations on a supercomputer.

This manual is organized as follows:

Part I Overview

Part I describes briefly the functions provided in SSL II Extended Capabilities II and indicates precautions to take when using them.

Part II Using Subroutines

Part II describes how to use individual subroutines. Subroutines are listed and described in alphabetical order.

In order to support the latest techniques, SSL II Extended Capabilities II contains improvements and additions. Existing subroutine functions are preserved within the improved and added functions. Please note that if the new subroutines perform better than the existing ones, the existing subroutines may be eliminated some time in the future.

For a complete description of rules, standard functions, and extended capabilities, refer to the following manuals:

Fujitsu SSL II User's Guide (Scientific Subroutine Library),

Fujitsu SSL II Extended Capabilities User's Guide (Scientific Subroutine Library)

SSL II Extended Capabilities II was developed through the collaboration of the Australian National University (ANU) and Fujitsu. Development at the ANU was led by professors Mike Osborne and Richard Brent and coordinated by Dr. Bob Gingold, Head, ANU Supercomputer Facility. The following ANU staff members were involved in the design and implementation of SSL II Extended Capabilities II. Fujitsu acknowledges their cooperation.

Professor Richard Peirce Brent

Dr Andrew James Cleary

Dr Murray Leslie Dow

Dr Christopher Robert Dun

Dr Lutz Grosz

Dr David Laurence Harrar II

Dr Markus Hegland

Ms Judith Helen Jenkinson

Dr Margaret Helen Kahn

Mr Jeoffrey Keating

Dr Zbigniew Leyk

Mr Gavin John Mercer

Mr David John Miron

Mr Ole Møller Nielsen

Professor Michael Robert Osborne

Dr Peter Frederick Price

Dr Stephen Gwyn Roberts

Dr David Barry Singleton

Dr David Edward Stewart

Note

In the text, a number in brackets (e.g., [18]) refers to a reference listed at the end of the manual.

SSL II Extended Capabilities II Overview

Linear calculations

Subroutine name	Description	Page
VLSBX	System of linear equations with a symmetric positive definite banded matrix (modified Cholesky decomposition)	II-81
VBLDL	LDL^T decomposition of a symmetric positive definite banded matrix (modified Cholesky decomposition)	II-22
VBLDX	System of linear equations with an LDL^T decomposed symmetric positive definite banded matrix	II-26
VLBX	System of linear equations with a banded real matrix (Gaussian elimination)	II-73
VBLU	LU decomposition of a banded real matrix (Gaussian elimination)	II-29
VBLUX	System of linear equations with an LU decomposed banded real matrix	II-34
VLDIV	The inverse of a positive-definite symmetric matrix decomposed into the factors L,D and L^T	II-79
VLTQR	System of linear equations with real tridiagonal matrix (QR factorization)	II-85
VBCSD	System of linear equations with an unsymmetric or indefinite sparse real matrix (BICGSTAB(*l*) method, diagonal storage format)	II-14
VBCSE	System of linear equations with an unsymmetric or indefinite sparse real matrix (BICGSTAB(*l*) method, ELLPACK storage format)	II-18
VCGD	System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, diagonal storage format)	II-38
VCGE	System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, ELLPACK storage format)	II-44
VCRD	System of linear equations with an unsymmetric or indefinite sparse real matrix (MGCR method, diagonal storage format)	II-56
VCRE	System of linear equations with an unsymmetric or indefinite sparse real matrix (MGCR method, ELLPACK storage format)	II-60
VQMRD	System of linear equations with an unsymmetric or indefinite sparse real matrix (QMR method, diagonal storage format)	II-117
VQMRE	System of linear equations with an unsymmetric or indefinite sparse real matrix (QMR method, ELLPACK storage format)	II-121
VTFQD	System of linear equations with an unsymmetric or indefinite sparse real matrix (TFQMR method, diagonal storage format)	II-146

Subroutine name	Description	Page
VTFQE	System of linear equations with an unsymmetric or indefinite sparse real matrix (TFQMR method, ELLPACK storage format)	II-150
VMBV	Multiplication of a real band matrix and a real vector	II-88
VMVSD	Multiplication of a real sparse matrix and a real vector (diagonal storage format)	II-111
VMVSE	Multiplication of a real sparse matrix and a real vector (ELLPACK storage format)	II-114

Eigenvalues and eigenvectors

Subroutine name	Description	Page
VHEVP	Eigenvalues and eigenvectors of a Hermitian matrix (tridiagonalization, multisection method, and inverse iteration)	II-63
VLAND	Eigenvalues and eigenvectors of a real symmetric sparse matrix (Lanczos method, diagonal storage format)	II-68
VSEVP	Eigenvalues and eigenvectors of a real symmetric matrix (tridiagonalization, multisection method, and inverse iteration)	II-131
VTDEV	Eigenvalues and eigenvectors of real tridiagonal matrix	II-140

Transforms

Subroutine name	Description	Page
VCPF3	Three-dimensional prime factor discrete complex Fourier transform	II-50
VMCF2	Singlevariate, multiple and multivariate discrete complex Fourier transform (complex array, mixed radix)	II-90
VMCFT	Singlevariate, multiple and multivariate discrete complex Fourier transform (real and imaginary array seperated, mixed radix)	II-94
VMRF2	Singlevariate, multiple and multivariate discrete real Fourier transform (Mixed radix)	II-99
VMRFT	Multiple and multivariate discrete real Fourier transform (Mixed radices of 2, 3, and 5)	II-105
VRPF3	Three-dimensional prime factor discrete real Fourier transform	II-125
VSRFT	One-dimensional and multiple discrete real Fourier transform (Mixed radices of 2, 3, and 5)	II-136
VWFLT	Wavelet filter generation	II-153
V1DWT	One-dimensional wavelet transform	II-156
V2DWT	Two-dimensional wavelet transform	II-160

Random numbers

Subroutine name	Description	Page
DVRAN3	Generation of normal random numbers (double precision)	II-1
DVRAN4	Generation of normal random numbers (double precision, Wallace's method)	II-5
DVRAU4	Generation of uniform random numbers [0, 1) (double precision)	II-9

Contents

Preface iii

Part I Overview

Chapter 1 Description of SSL II Extended Capabilities II I-1

Chapter 2 General Rules for SSL II Extended Capabilities II I-3

2.1 Subroutine Precision I-3
2.2 Subroutine Names I-3
2.3 Parameters I-3
2.4 Condition Codes..... I-3

Chapter 3 Data Storage Methods I-5

3.1 Banded Matrices..... I-5
3.2 Sparse Matrices I-5
3.2.1 Storage methods for sparse matrices I-5
3.2.1.1 Storage method for general sparse matrices I-5
3.2.1.2 Storage methods for symmetric positive definite sparse matrices.....I-7
3.2.1.3 Storage method selection criteria.....I-9
3.2.2 Discretization of partial differential operators and storage examples for them.....I-9

Chapter 4 Iterative Linear Equation Solvers and Convergence I-17

4.1 Scaling..... I-17
4.2 Symmetry of Matrix and Iterative Solver I-18
4.3 Eigenvalues Distribution of Matrix and Convergence I-18

Part II Using Subroutines

DVRAN3 ..... **II-1**
DVRAN4 ..... **II-5**
DVRAU4 ..... **II-9**
DVBCSD **II-14**
DVBCSE **II-18**
VBLDL, DVBLDL **II-22**
VBLDX, DVBLDX **II-26**
VBLU, DVBLU **II-29**
VBLUX, DVBLUX **II-34**
VCGD, DVCGD **II-38**
VCGE, DVCGE **II-44**
VCPF3, DVCPF3 ..... **II-50**
VCRD, DVCRD **II-56**
VCRE, DVCRE **II-60**
VHEVP, DVHEVP **II-63**
VLAND, DVLAND **II-68**
VLBX, DVLBX **II-73**
VLDIV, DVLDIV **II-79**
VLSBX, DVLSBX **II-81**
VLTQR, DVLTQR **II-85**
VMBV, DVMBV **II-88**
VMCF2, DVMCF2 ..... **II-90**
VMCFT, DVMCFT **II-94**
VMRF2, DVMRF2 ..... **II-99**
VMRFT, DVMRFT **II-105**
VMVSD, DVMVSD **II-111**
VMVSE, DVMVSE **II-114**
VQMRD, DVQMRD **II-117**
VQMRE, DVQMRE **II-121**
VRPF3, DVRPF3 ..... **II-125**
VSEVP, DVSEVP **II-131**
VSRFT, DVSRFT **II-136**
VTDEV, DVTDEV **II-140**

VTFQD, DVTFQD II-146
VTFQE, DVTFQE II-150
VWFLT, DVWFLT II-153
V1DWT, DV1DWT II-156
V2DWT, DV2DWT II-160

Appendix A References A-1

Appendix B Contributors and Their Work B-1

Index IN-1

Part I
Overview

Chapter 1
Description of SSL II Extended Capabilities II

This chapter briefly describes the algorithms that are provided for use in large-scale, scientific computations as SSL II Extended Capabilities II.

(1) Double-precision random numbers (uniform/normal)

These algorithms provide random numbers with good statistical characteristics and long periods of at least 10^{52} for large-scale simulation. For normal random numbers Polar method and faster Wallace's method are provided.

(2) Sparse matrix linear equations (symmetric positive definite matrix/unsymmetric or indefinite real matrix)

These subroutines solve sparse matrix linear equations using the iterative method. These subroutines make it possible to solve large-scale problems at high speeds with reduced memory usage. For data storage methods, see Chapter 3, "Data Storage Methods."

For symmetric positive definite matrices, the conjugate gradient (CG) method is provided. Two types of preconditioners may be specified in CG method: first order approximation of the Neumann series and modified incomplete Cholesky decomposition. The preconditioner through the modified incomplete Cholesky decomposition is useful for linear equations obtained through discretization of elliptic partial differential equations.

For unsymmetric or indefinite real matrices, the robust and high-speed modified generalized conjugate residuals (MGCR) method is provided.

For unsymmetric or indefinite real matrices, the higher-speed quasi minimal residual method (QMR method) , transpose-free quasi-minimal residual method (TFQMR method) and Bi-Conjugate Gradient Stabilised (*l*) (BICGSTAB(*l*)) method are provided. About the guideline of the usage of these methods, refer to Chapter 4, "Iterative Linear Equation Solvers and Convergence".

(3) Sparse real matrix and vector multiplication

(4) System of linear equations with real tridiagonal matrix

This system supplies a method of solving a large-scale system of linear equations with real tridiagonal matrix at high speed by QR factorization.

(5) Banded matrix linear equations (symmetric positive definite/real matrix)

These subroutines use data storage methods and algorithms that optimize performance on vector computers. Although the direct method is robust, it uses memory in proportion to the size of the bandwidth. Therefore, the direct method is unsuitable for large banded matrices with a sparse structure. For large banded matrices with sparse structure, please use the previously mentioned iterative method, which uses storage methods suited to sparse structures.

(6) Eigenvalue problem

This system supplies the Lanczos method to obtain a few of the largest and/or smallest eigenvalues and corresponding eivenvectors in a large-scale real symmetric sparse matrix.

It also supplies a method of obtaining eigenvalues and eigenvectors in real tridiagonal, real symmetric or Hermitian matrices at high speed.

(7) Fourier transforms

These subroutines provide high-performance algorithms (mixed radix and complex/real), multiple Fourier transforms and multivariate Fourier transforms on vector or scalar computers. The functions are also high-speed for one-dimensional Fourier transforms. Three-dimensional, prime factor Fourier transforms (complex/real) are also provided.

(8) Wavelet transform

This system supplies high-performance one- and two-dimensional wavelet transform.

Chapter 2
General Rules for SSL II Extended Capabilities II

This chapter provides general rules that are common to all of the functions.

2.1 Subroutine Precision

Single- and double-precision routines are provided. However, random number routines are double precision only.

2.2 Subroutine Names

The names of single-precision routines start with a V. The names of double-precision routines start with a DV.

The names of slave routines start with a U (single precision) or DU (double precision).

2.3 Parameters

(1) Order of parameters

The order of parameters is the same as the order used in SSL II standard functions. As a rule, the order conforms to the following format:

(Input-output-parameter-list, Input-parameter-list, Output-parameter-list, ICON)

(2) Parameter types

Integer-type parameters start with an I, J, K, L, M or N. Complex-type parameters start with a Z.

Unless otherwise specified, parameters that start with any other letter are single-precision real type in single-precision routines, and double-precision real type in double-precision routines.

2.4 Condition Codes

The ICON parameter indicates the resultant status after execution of the subroutine.

The condition code is from 0 to 39,999. As shown in the following table, the range into which the code falls indicates how reliable the processing results are.

Table 2.1 Results of condition codes

Code	Explanation	Reliability of result	Result
0	Processing terminated normally.	Result is guaranteed.	Normal
1 to 9999	Processing terminated normally, but additional information is included.		
10000 to 19999	Due to an internal restriction imposed during processing, processing terminated.	The result is guaranteed subject to restrictions.	Warning
20000 to 29999	Due to an error that occurred during processing, processing stopped.	The result is not guaranteed.	Error
30000 to 39999	Due to an error in the input parameter(s), processing stopped.		

Chapter 3
Data Storage Methods

SSL II Extended Capabilities provides the following storage methods for solving linear equations of banded and sparse matrices.

3.1 Banded Matrices

Storage methods suitable for vector computers are used instead of the standard SSL II storage methods for banded matrices. See the descriptions of the routines used with banded matrices.

3.2 Sparse Matrices

This section describes the storage methods for sparse matrices.

3.2.1 Storage methods for sparse matrices

Each function that applies to sparse matrices is provided respectively for the ELLPACK storage format and the diagonal storage format respectively.

The ELLPACK storage format is a method of compressing and storing the non-zero elements of each row vector in a coefficient matrix.

The diagonal storage format is a method of storing diagonals containing non-zero elements.

3.2.1.1 Storage method for general sparse matrices

a. ELLPACK storage format for general sparse matrices

In the ELLPACK storage format for general sparse matrices, non-zero elements of row vectors in coefficient matrix ***A*** are compressed and stored in corresponding row vectors of array ***COEF***. (For the ELLPACK format, see [23] and [33]). In addition, the column number of non-zero elements that were stored in ***COEF*** are stored in corresponding ***ICOL*** array elements. It is not necessary to left-adjust the non-zero elements of the row vectors of coefficient matrix ***A*** when storing them in ***COEF***.

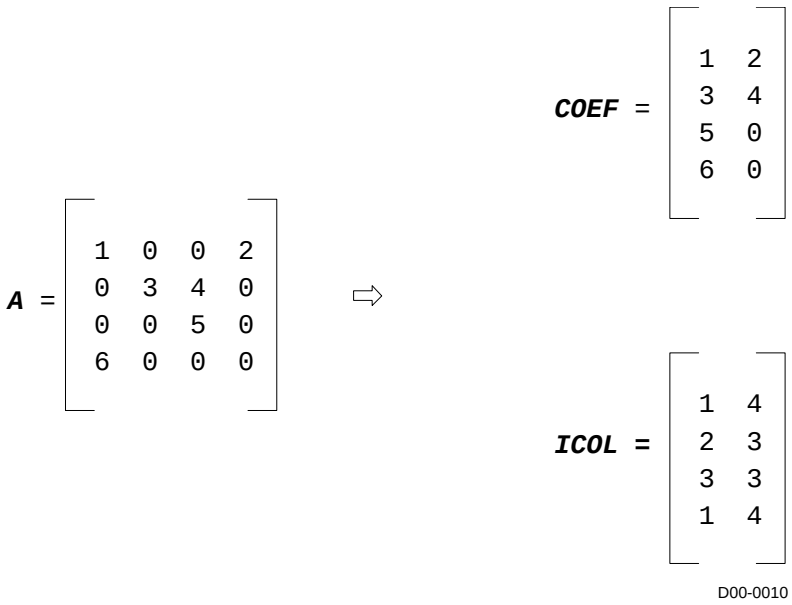
For storage, the ***COEF*** (1 : *N*, *) and ***ICOL*** (1 : *N*, *) parts of the two arrays, ***COEF*** (*K*, *IWIDT*) and ***ICOL*** (*K*, *IWIDT*), are used.

If the maximum number of non-zero elements appearing in the row vectors of matrix ***A*** is set to nz and the order of the coefficient matrix is set to n, then $IWIDT \geq nz$ and $K \geq n$.

When the number of non-zero elements in row vectors of coefficient matrix ***A*** is less than *IWIDT*, set the remaining elements in the row vectors of the array ***COEF*** to zero. Set the corresponding array elements of ***ICOL*** to the values showing the row number of the row vectors in which they are contained. (Assume ***COEF*** (*i*, *j*) = 0 and ***ICOL*** (*i*, *j*) = *i*.)

Example:

Storing coefficient matrix ***A*** using ***COEF*** and ***ICOL***



D00-0010

b. Diagonal storage format for general sparse matrices

In the diagonal storage format for sparse matrices, diagonals containing non-zero elements are stored as column vectors of the array ***DIAG***. (For the diagonal format, see [26] and [30].

In this manual, for the integer *k*, the following diagonal-direction vector is called a diagonal. The vector consisting of diagonal elements is called the main diagonal.

$(a_{1,\ 1+k},\ a_{2,\ 2+k},\ \ldots,\ a_{n,\ n+k})$

if $i + k < 1$ or $i + k > n$, $a_{i,\ i+k} = 0$.

There is no special restriction on the order in which a diagonal is stored in the array ***DIAG***.

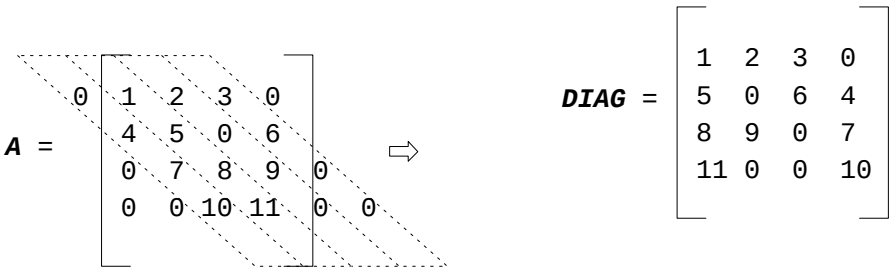
The offset between the diagonal vector stored in ***DIAG*** (*, *i*) and the main diagonal vector is stored in ***NOFST*** (*i*). *k* in the previously mentioned diagonal indicates the offset. The offset of the diagonal consisting of the main diagonal elements is zero. The offset of diagonal in the upper triangular matrix is a positive integer. The offset of diagonal in the lower triangular matrix is a negative integer.

If ***NOFST*** (*m*) = *k*, then storage is performed according to ***DIAG*** (*i*, *m*) = $a_{i,\ i+k}$ (i = 1, ..., n).

Two arrays, ***DIAG*** (*K*, *NDIAG*) and ***NOFST*** (*NDIAG*), are used. The coefficient matrix is stored in ***DIAG*** (1 : *N*, *NDIAG*). When the number of diagonals to be stored is set to *nd* and the order of the coefficient matrix is set to *n*, then $NDIAG \geq nd$ and $K \geq n$.

Example:

Storing coefficient matrix ***A*** using ***DIAG*** and ***NOFST***



D00-0020

3.2.1.2 Storage methods for symmetric positive definite sparse matrices

In both the ELLPACK storage format and the diagonal storage format, the upper and lower triangular matrix parts of normalized symmetric positive definite matrices are stored in the order from the upper part then lower part.

Using a diagonal matrix that has the reciprocals of the square root of the diagonal elements of a symmetric positive definite matrix ***A***, the symmetric positive definite matrix ***A*** can be normalized into symmetric matrix $\boldsymbol{A}^*$ with diagonal elements of 1.

$$\boldsymbol{A}^* = \boldsymbol{D}^{-1/2}\,\boldsymbol{A}\,\boldsymbol{D}^{-1/2}$$

where $\boldsymbol{D}^{-1/2} = diag\,(a_{11}^{-1/2}, a_{22}^{-1/2}, ..., a_{nn}^{-1/2})$

$= diag\,(d_1^{-1/2}, d_2^{-1/2}, ..., d_n^{-1/2})$

The linear equation with order *n*

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

can be transformed to a linear equation with the normalized matrix $\boldsymbol{A}^*$.

$$(\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\,(\boldsymbol{D}^{1/2}\boldsymbol{x}) = \boldsymbol{D}^{-1/2}\boldsymbol{b}$$

$$\boldsymbol{A}^*\boldsymbol{x}^* = \boldsymbol{b}^*$$

$$a_{ij}^* = a_{ij}d_i^{-1/2}d_j^{-1/2}\ ,b_i^* = b_i d_i^{-1/2}$$

$$x_i^* = x_i d_i^{1/2}\ , i = 1,\ ...,n,\ j = 1,\ ...,n$$

a. The ELLPACK storage format for symmetric positive definite sparse matrices

The upper and lower triangular matrix parts without diagonal elements in a normalized symmetric positive definite sparse matrix with unit diagonal elements are stored respectively using the ELLPACK storage format. Then these stored matrix parts are stored in a single array ***COEF***. First, the upper triangular matrix part is stored. Then the lower triangular matrix part is stored.

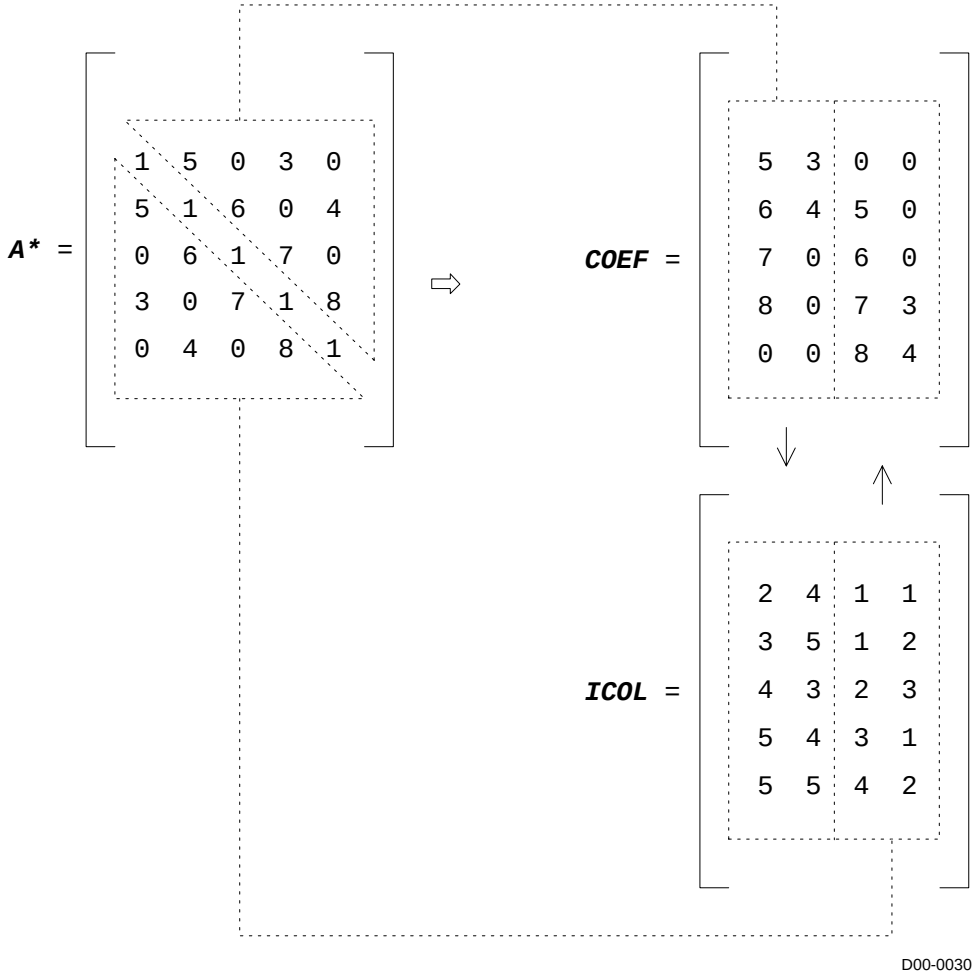
The maximum number of non-zero elements in each row vector of the upper triangular matrix part is set to *NSU*. The maximum number of non-zero elements in each row vector of the lower triangular matrix part is set to *NSL*.

When *NSH = max (NSU, NSL)*, the non-zero elements of the upper triangular matrix part are stored in ***COEF*** (**, 1: NSH*). The non-zero elements of the lower triangular matrix part are stored in ***COEF*** (**, NSH* + 1: 2 × *NSH*).

Set the remaining elements in the array ***COEF*** to zero. Set the corresponding array elements of ***ICOL*** to the row numbers of the row vectors in which they are contained. (Assume ***COEF*** (*i, j*) = 0 and ***ICOL*** (*i, j*) = *i*.)

Example:

Storing the upper and lower triangular part of a normalized coefficient matrix ***A**** using the ELLPACK storage format



b. The diagonal storage format for symmetric positive definite sparse matrices

The upper and lower triangular matrix parts without diagonal elements in a normalized symmetric positive definite matrix with unit diagonal elements are stored respectively using the diagonal storage format. Then, the stored matrix parts are stored in a single array ***DIAG***. When the number of diagonals in the upper (lower) triangular matrix part containing non-zero elements is set to NDT, the upper triangular matrix part is stored in ***DIAG*** (*1 : NDT*), while the lower triangular matrix part is stored in ***DIAG*** *(NDT + 1: NW)*.

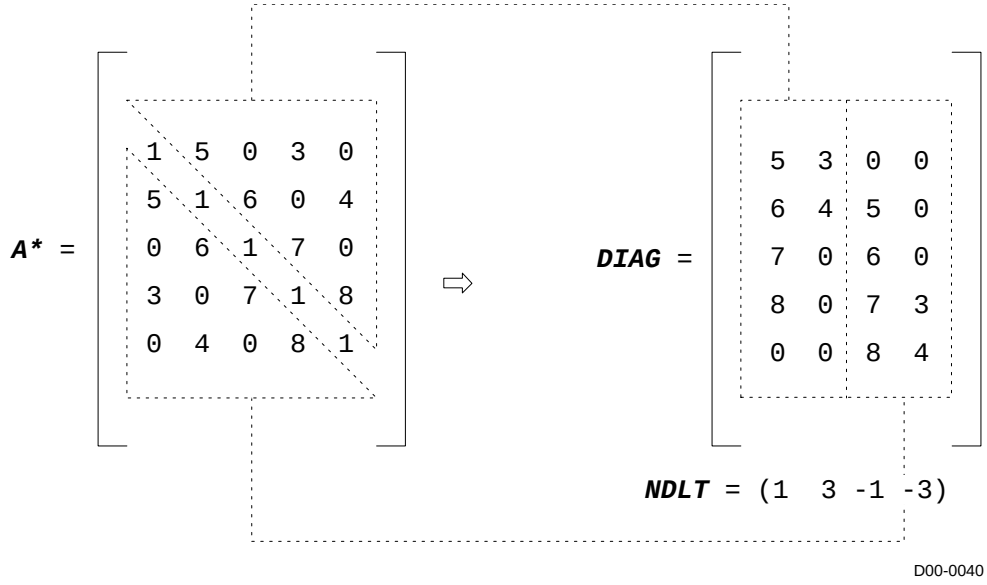
At that time, the upper triangular matrix part must be stored in ascending order with respect to distance (***NDLT*** value). The lower triangular matrix part must be stored in descending order.

This method uses arrays the ***DIAG*** (*K*, *NW*) and ***NDLT*** (*NW*).

The following equivalence applies: $NW = 2 \times NDT$.

Example:

Storing the upper and lower triangular part of a normalized coefficient matrix ***A**** using the diagonal storage format



3.2.1.3 Storage method selection criteria

When the sparse matrix is structured so that its non-zero elements are concentrated in the diagonal-direction vectors of the coefficient matrix, use the diagonal storage format.

3.2.2 Discretization of partial differential operators and storage examples for them

This section describes the representative sparse coefficient matrices which appear when solving problems through discretizing elliptic partial differential equations and constructing linear equations. When solving actual problems, these coefficient matrices must be stored using the ELLPACK storage format and diagonal storage format.

Coefficient matrices by discretization of the elliptical partial differential operators in the three-dimensional region with Dirichlet boundary conditions is shown in the following sections. As a result of discretization, the operators appear in the unsymmetric sparse matrix. Also shown are subroutines that store the generated coefficient matrices according to the sparse matrix storage methods.

Linear equations with these coefficient matrices can be solved with subroutine (D) VCRE or (D) VCRD.

a. Discretization of elliptic partial differential operators and construction of coefficient matrices

Operator $\boldsymbol{L}$

$$Lu = -\Delta u + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial u}{\partial y} + a_3 \frac{\partial u}{\partial z} + cu$$

(Δ represents Laplacian. $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$)

Region $\Omega = [0,l_x] \times [0,l_y] \times [0,l_z]$

Boundary condition Dirichlet boundary condition $u = 0$ on the boundary Ω

Here, a_1, a_2, a_3 as well as c are constants.

When each dimension of Ω is divided into $n_x + 1$, $n_y + 1$, and $n_z + 1$ in equal subintervals respectively, the $n_x \times n_y \times n_z$ grid points exist inside Ω. When the value of variables x, y, and z at the grid point is expressed as (x_i, y_j, z_k),

the value of function u at the grid point ($1 \le i \le n_x$, $1 \le j \le n_y$, $1 \le k \le n_z$)

is expressed as $u_{i,j,k} = u(x_i, y_j, z_k)$.

Using this notation, the partial differential coefficient for variable x is approximated as follows.

$$\frac{\partial u}{\partial x}(x_i, y_j, z_k) \approx (u_{i+1,j,k} - u_{i-1,j,k})(n_x+1) / (2l_x)$$

$$\frac{\partial u^2}{\partial x^2}(x_i, y_j, z_k) \approx (u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k})(n_x+1)^2 / l_x^2$$

The partial differential coefficients for variables y and z are approximated in a similar fashion.

Considering function $u = 0$ on the boundary Ω,

the approximation $\boldsymbol{Lu} \approx \boldsymbol{Av}$ is obtained through discretizing $\boldsymbol{Lu}$ into coefficient matrix $\boldsymbol{A}$.

Here, $\boldsymbol{v}$ is vector $\boldsymbol{v} = (v_1, v_2, ..., v_n)$, consisting of values at the grid point of function u,

and there is a corresponding relationship $v_m = u_{i,j,k}$, $m = (k - 1) n_x n_y + (j - 1) n_x + i$.

b. Subroutines that store coefficient matrices using the sparse matrix storage method

Examples of subroutines that store discretized operators in the ELLPACK format or the diagonal format are described in this section.

The subroutine INIT_MAT_ELL stores coefficient matrices in the ELLPACK format. The subroutine INIT_MAT_DIAG stores them in the diagonal format.

The arguments n_x, n_y, and n_z correspond to NX, NY, and NZ. l_x, l_y, and l_z correspond to XL, YL, and ZL. a_1, a_2, a_3, and c correspond to VA1, VA2, VA3, and VC.

In INIT_MAT_ELL, the coefficient matrix is stored in A_L and ICOL_L. In INIT_MAT_DIAG, it is stored in D_L and OFFSET.

When a subroutine is called with IWIDTH = 7, a coefficient matrix for three-dimensional region Ω is generated.

A subroutine is called with NDIVP = $n_x n_y n_z$.

(When using such a subroutine, the value of IWIDTH should not be greater than 7. If the value is 7 or less (for example, 5 or 3), the number of diagonal columns decreases

correspondingly. If IWIDTH = 5 or IWIDTH = 3, the problem is reduced to a two- or one-dimensional problem, respectively. It is useful to set NZ = 1 when IWIDTH = 5, and NZ = 1, NY = 1 when IWIDTH = 3. Values of IWIDTH other than 7, 5, or 3 have no special meaning and can be used in testing.)

Example 1:

Subroutine that discretizes the partial differential operators described previously and stores them according to the ELLPACK storage format

```
      SUBROUTINE INIT_MAT_ELL(VA1,VA2,VA3,VC,
     &   A_L,ICOL_L,NX,NY,NZ,XL,YL,ZL,IWIDTH,NDIVP,LD)
      IMPLICIT NONE
      INTEGER NX,NY,NZ,IWIDTH,NDIVP,LD
      DOUBLE PRECISION A_L(LD,IWIDTH)
      DOUBLE PRECISION VA1,VA2,VA3,VC,XL,YL,ZL
      INTEGER ICOL_L(LD,IWIDTH)

      DOUBLE PRECISION HX, HY, HZ
      INTEGER I,J,L,JS,IWIDTH_LOC
      INTEGER I0,J0,K0

      IF (IWIDTH .LT. 1) THEN
         WRITE (*,*) 'SUBROUTINE INIT_MAT_ELL:'
         WRITE (*,*) ' IWIDTH SHOULD BE GREATER THAN OR
     & EQUAL TO 1'
         RETURN
      ENDIF
      IWIDTH_LOC = IWIDTH
C     IWIDTH CANNOT BE GREATER THAN 7
      IF (IWIDTH .GT. 7) IWIDTH_LOC = 7

C     INITIAL SETTING
      HX = XL/(NX+1)
      HY = YL/(NY+1)
      HZ = ZL/(NZ+1)

      DO 110 J = 1,IWIDTH
         DO 100 I = 1,NDIVP
            A_L(I,J) = 0.0
            ICOL_L(I,J) = I
 100     CONTINUE
 110  CONTINUE

C     MAIN LOOP
      DO 200 J = 1,NDIVP
        JS = J
        L = 1
```

```
C       DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NX/NY+1
        IF (K0 .GT. NZ) RETURN
        J0 = (JS-1-NX*NY*(K0-1))/NX+1
        I0 = JS - NX*NY*(K0-1) - NX*(J0-1)

        IF (IWIDTH_LOC .GE. 7) THEN
           IF (K0 .GT. 1) THEN
               A_L(J,L) = -(1.0/HZ+0.5*VA3)/HZ
               ICOL_L(J,L) = JS-NX*NY
               L = L+1
           ENDIF
        ENDIF
        IF (IWIDTH_LOC .GE. 5) THEN
           IF (J0 .GT. 1) THEN
              A_L(J,L) = -(1.0/HY+0.5*VA2)/HY
              ICOL_L(J,L) = JS-NX
              L = L+1
           ENDIF
        ENDIF
        IF (IWIDTH_LOC .GE. 3) THEN
           IF (I0 .GT. 1) THEN
              A_L(J,L) = -(1.0/HX+0.5*VA1)/HX
              ICOL_L(J,L) = JS-1
              L = L+1
           ENDIF
        ENDIF
        A_L(J,L) = 2.0/HX**2+VC
        IF (IWIDTH_LOC .GE. 5) THEN
           A_L(J,L) = A_L(J,L) + 2.0/HY**2
           IF (IWIDTH_LOC .GE. 7) THEN
              A_L(J,L) = A_L(J,L) + 2.0/HZ**2
           ENDIF
        ENDIF
        ICOL_L(J,L) = JS
        L = L+1
        IF (IWIDTH_LOC .GE. 2) THEN
           IF (I0 .LT. NX) THEN
              A_L(J,L) = -(1.0/HX-0.5*VA1)/HX
              ICOL_L(J,L) = JS+1
              L = L+1
           ENDIF
        ENDIF
        IF (IWIDTH_LOC .GE. 4) THEN
           IF (J0 .LT. NY) THEN
              A_L(J,L) = -(1.0/HY-0.5*VA2)/HY
              ICOL_L(J,L) = JS+NX
              L = L+1
           ENDIF
        ENDIF
        IF (IWIDTH_LOC .GE. 6) THEN
           IF (K0 .LT. NZ) THEN
              A_L(J,L) = -(1.0/HZ-0.5*VA3)/HZ
```

```
               ICOL_L(J,L) = JS+NX*NY
             ENDIF
           ENDIF

 200   CONTINUE

       RETURN
       END
```

Example 2:

Subroutine that discretizes the partial differential operators described previously and stores them according to the diagonal storage format

```
       SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET,
      &    NX,NY,NZ,XL,YL,ZL,NDIAG,NDIVP,LD)
       IMPLICIT NONE
       INTEGER NX,NY,NZ,NDIAG,NDIVP,LD
       DOUBLE PRECISION D_L(LD,NDIAG)
       DOUBLE PRECISION VA1,VA2,VA3,VC,XL,YL,ZL
       INTEGER OFFSET(NDIAG)

       DOUBLE PRECISION HX, HY, HZ
       INTEGER I,J,L,JS,NXY,NDIAG_LOC
       INTEGER J0,I0,K0

       IF (NDIAG .LT. 1) THEN
          WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
          WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR
      & EQUAL TO 1'
          RETURN
       ENDIF
       NDIAG_LOC = NDIAG
       IF (NDIAG .GT. 7) NDIAG_LOC = 7

C      INITIAL SETTING
       HX = XL/(NX+1)
       HY = YL/(NY+1)
       HZ = ZL/(NZ+1)

       DO 110 J = 1,NDIAG
          DO 100 I = 1,NDIVP
             D_L(I,J) = 0.0
 100     CONTINUE
 110   CONTINUE
C      OFFSET SETTING
       L = 1
       NXY = NX*NY
       IF (NDIAG_LOC .GE. 7) THEN
          OFFSET(L) = -NXY
          L = L+1
       ENDIF
```

```
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
         OFFSET(L) = -1
         L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
         OFFSET(L) = 1
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
         OFFSET(L) = NX
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
         OFFSET(L) = NXY
      ENDIF

C     MAIN LOOP
      DO 200 J = 1,NDIVP
         JS = J
C        DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
         K0 = (JS-1)/NXY+1
         IF (K0 .GT. NZ) RETURN
         J0 = (JS-1-NXY*(K0-1))/NX+1
         I0 = JS - NXY*(K0-1) - NX*(J0-1)

         L = 1
         IF (NDIAG_LOC .GE. 7) THEN
            IF (K0 .GT. 1) D_L(J,L) = -(1.0/HZ+0.5*VA3)/HZ
            L = L+1
         ENDIF
         IF (NDIAG_LOC .GE. 5) THEN
            IF (J0 .GT. 1) D_L(J,L) = -(1.0/HY+0.5*VA2)/HY
            L = L+1
         ENDIF
         IF (NDIAG_LOC .GE. 3) THEN
            IF (I0 .GT. 1) D_L(J,L) = -(1.0/HX+0.5*VA1)/HX
            L = L+1
         ENDIF
         D_L(J,L) = 2.0/HX**2+VC
         IF (NDIAG_LOC .GE. 5) THEN
            D_L(J,L) = D_L(J,L) + 2.0/HY**2
            IF (NDIAG_LOC .GE. 7) THEN
               D_L(J,L) = D_L(J,L) + 2.0/HZ**2
            ENDIF
         ENDIF
         L = L+1
         IF (NDIAG_LOC .GE. 2) THEN
```

```
            IF (I0 .LT. NX) D_L(J,L) = -(1.0/HX-0.5*VA1)/HX
            L = L+1
         ENDIF
         IF (NDIAG_LOC .GE. 4) THEN
            IF (J0 .LT. NY) D_L(J,L) = -(1.0/HY-0.5*VA2)/HY
            L = L+1
         ENDIF
         IF (NDIAG_LOC .GE. 6) THEN
            IF (K0 .LT. NZ) D_L(J,L) = -(1.0/HZ-0.5*VA3)/HZ
         ENDIF
200   CONTINUE

      RETURN
      END
```

Chapter 4
Iterative Linear Equation Solvers and Convergence

4.1 Scaling

It is strictly recommended to scale the equation in order to balance the matrix entries for the efficient usage of iterative linear equation solver. This normalisation of the matrix strongly improves the numerical stability and the convergence rate of the iterative solver. The normalised coefficient matrix $\hat{\boldsymbol{A}}$ should have non--negative entries in the main diagonal and, for instance, the sum of absolute values in each row should be approximately equal to one.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

A normalised form of the linear system (1) can be constructed by multiplying the coefficient matrix $\boldsymbol{A}$ by a diagonal matrix $\boldsymbol{L}$ from the left and with a diagonal matrix $\boldsymbol{R}$ from the right. By introducing a new variable $\hat{\boldsymbol{x}} = \boldsymbol{R}^{-1}\boldsymbol{x}$ the linear system(1) is written as

$$\boldsymbol{LAR}\,\hat{\boldsymbol{x}} = \boldsymbol{Lb} \quad \Leftrightarrow \quad \hat{\boldsymbol{A}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}} \tag{2}$$

where, $\hat{\boldsymbol{A}} = \boldsymbol{LAR}$, $\hat{\boldsymbol{b}} = \boldsymbol{Lb}$.

Instead of $\boldsymbol{A}$ the normalised matrix $\hat{\boldsymbol{A}}$ is used in the iterative solver. Keep in mind that the right hand side $\boldsymbol{b}$ has to be transformed by multiplication with $\boldsymbol{L}$ before the solver is called and the returned solution approximation has to be transformed by multiplication with $\boldsymbol{R}$.

If for all i=1,...,n the $s_i = \sum_{j=1}^{n} \left| a_{ij} \right|$ value is the absolute sum of entries in the i-th row one can set

$$L_{ij} = \begin{cases} \dfrac{\operatorname{sgn}(a_{ii})}{\sqrt{s_i}} & i = j \\ 0 & i \neq j \end{cases} \quad \text{if} \tag{3}$$

$$R_{ij} = \begin{cases} \dfrac{1}{\sqrt{s_i}} & i = j \\ 0 & i \neq j \end{cases} \quad \text{if} \tag{4}$$

for all i,j=1,...,n. It is emphasized that there are other possible ways of introducing a normalisation with rather different effects on the convergence rate of the iterative solvers, see [43] for an overview.

Notice , that with selection (3) and (4) the normalised matrix $\hat{\boldsymbol{A}}$ is symmetric and positive definite if and only if the original matrix is symmetric and positive definite.

4.2 Symmetry of Matrix and Iterative solvers

a. Symmetric Matrix

If the matrix A is symmetric, ie. $a_{ij}=a_{ji}$ for all $i,j=1,...,n$, and positive definite the classical conjugate gradient method(see [21]) can be used to solve the linear system.

If the matrix is not positive definite a break down will occurred.

b. Non-symmetrical or Indefinite Matrix

In case of a non-symmetrical or indefinite coefficient matrix a set of solvers are available. The optimal solver for the given linear system depends on the properties of the coefficient matrix $\boldsymbol{A}$ (or if the normalised system $\hat{\boldsymbol{A}}$ is considered). For the different classes of matrices the following solvers are available:

4.3 Eigenvalues Distribution of Matrix and Convergence

a. MGCR method

If the eigenvalues of the coefficient matrix are close to the positive real axis (see Figure 4.3-1) can be used with a small number of search directions (eg. 5-10). If the imaginary part of any eigenvalue is large more search directions must be considered in order to get good convergence. This increases the storage requirements as well as the amount of computation per iteration step which makes MGCR (see [25]) less efficient.

For a small number of search directions MGCR is a very fast but not very robust method.

b. TFQMR method

If the eigenvalues are in the positive half plane but there are eigenvalues with large imaginary part (see Figure 4.3-2) TFQMR(see [12]) is the recommended method. Also the solvers converge best if the minimal real part of any eigenvalue is as large as possible. So, for example, the convergence will be poor if there is an eigenvalue which has a very small nonzero real part. The convergence rate of TFQMR can be worse than the convergence rate of MGCR with a large number of search directions. However, every iteration step of TFQMR is much cheaper than MGCR with a large number of search directions so that a solution is calculated within less CPU time. So TFQMR is more robust but slower than MGCR with a small number of search directions.

c. BICGSTAB(*l*) method

Similarly to TFQMR BICGSTAB(*l*)(see [38]) is suitable for matrices with eigenvalues that are in the positive half plane. Also the solvers converge best if the minimal real part of any eigenvalue is as large as possible. So, for example, the convergence will be poor if there is an eigenvalue which has a very small nonzero real part. In some applications where the eigenvalues of the coefficient matrix are close to the positive real axis BICGSTAB(*l*) has an even faster convergence rate than MGCR with a small

number of search directions. However, every iteration step of BICGSTAB(*l*) is very expensive as it requires two matrix vector multiplications. Therefore in some cases MGCR or TFQMR are faster than BICGSTAB(*l*) but BICGSTAB(*l*) is more robust.

If no information about the eigenvalues of the (normalised) coefficient matrix is available it is suggested to try the methods MGCR, TFQMR and BICGSTAB(*l*) one after the other. MGCR should be used with 5 and 10 search directions. The order in which the methods are tested is important. So the fast but less robust methods should be tested before more robust methods are used. A suitable criterion for the quality is the CPU time the solver needs to reach the accuracy 0.1.

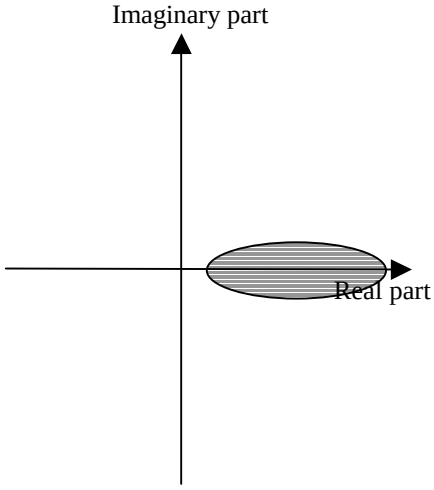


Figure 4.3-1
Eigenvalues distribution for convergent MGCR

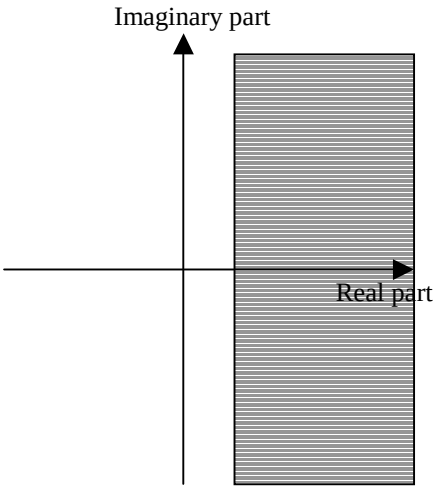


Figure 4.3-2
Eigenvalues distribution for convergent TFQMR and BICGSTAB(*l*)

Part II
Using Subroutines

J11-20-0401 DVRAN3

Generation of normal random numbers (double precision)
CALL DVRAN3 (DAM, DSD, IX, DA, N, DWORK, NWORK, ICON)

(1) Function

This subroutine generates pseudo-random numbers from a normal distribution density function (1.1) with a given mean *m* and standard deviation σ.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \qquad (1.1)$$

(2) Parameters

DAM..... Input. Mean *m* of normal distribution.
Double-precision real type.

DSD..... Input. Standard deviation σ of normal distribution.
Double-precision real type.

IX..... Input. Starting value.
On the first call, set IX to a positive value. Subsequently, call the subroutine with the return value remaining 0. A different sequence of random numbers are generated with a different starting value on the first call.
(See item (3), "Comments on use," b., 1).)
4-byte integer type (INTEGER*4).
Output. 0.

DA..... Output. N pseudo-random numbers.
Double-precision real type one-dimensional array of size N.

N..... Input. The number of normally distributed pseudo-random numbers to be returned in DA.
(See item (3), "Comments on use," b., 2).)

DWORK..... Work area. One-dimensional array of the double-precision real type and size NWORK.
Do not modify the contents of this subroutine between repeated calls.
DWORK contains information necessary for repeated calls to this subroutine.
(See item (3), "Comments on use," b., 3).)

NWORK..... Input. The size of the array DWORK. NWORK ≥ 1,156.

ICON..... Output. Condition code.
See Table DVRAN3-1, "Condition codes."

Table DVRAN3-1 Condition codes

Code	Description	Processing
0	No error	
30001	NWORK is too small.	Processing is stopped.
30002	IX<0	
30003 to 30008	DWORK was modified. Or IX was set to zero on the initial call.	

(3) Comments on use

a. Subprograms used

SSL II: DUF2G3, DUITG3, DURN3B, DURUG3, DUR2G3, DUSKG3, DUSQG3, DUVRG3, DVRAU4, MGSSL

b. Comments

1) Starting point IX

When a sequence of pseudo-random numbers is to be generated by a deterministic program, there must be random input. To do this, give a starting value IX. On the first call to this subroutine, the starting value IX must be a positive integer. (For exceptions, see item 5).) For subsequent calls, set IX to zero. This indicates that more pseudo-random numbers from the same sequence are to be generated. To simplify programming, this subroutine returns zero in IX after the initial call.

This subroutine generates normal random numbers with the Polar method, which uses uniform random numbers with a long period of at least 10^{52}. A different starting value gives a different random number sequence. That is, a random number sequence is generated from different random number subsequences that are created through the segmentation of a long-period random number sequence. These subsequences are separated by a distance of at least $2^{60} > 10^{18}$ intervals. For details, see “DVRAU4,” item (4), “Method.”

2) Parameter N

This subroutine returns the next N pseudo-random numbers from the infinite sequence defined by the starting value IX. If $N \leq 0$, no pseudo-random numbers are returned.

For efficiency, the user should make N sufficiently large (for instance, N = 100,000). This reduces the overhead of subroutine calls and allows vectorization. N may be changed on consecutive calls to this subroutine provided that the size of array DA is as large as the maximum value of N.

3) Work area DWORK

DWORK is a work area to store state information for repeated calls to this subroutine. The calling program must not change the contents of DWORK while the subroutine is being called.

4) Parameter NWORK

DWORK (1), ..., DWORK (NWORK) are used by this subroutine. NWORK should remain unchanged on each call to the subroutine. NWORK should be at least 1,156. For efficiency on vector processors, NWORK should be large (for example, NWORK = 100,000).

5) Repeated generation of the same random number

If DWORK (1), ..., DWORK (NWORK) is saved, the same sequence of random numbers can be generated again (from the point where DWORK was saved) by reusing DWORK (1), ..., DWORK (NWORK) and calling this subroutine with argument IX = 0.

c. Example of use

In this example, one million normal pseudo-random numbers are generated, and the first- and second-order moments are calculated. The starting value is 12345. The first-order moment is $\sum X_i F_i$ when the frequency of the variable X_i is F_i. The second-order moment is $\sum X_i^2 F_i$ when the frequency of the variable X_i is F_i.

```
C      **EXAMPLE**
       PARAMETER (NRAN = 1000000)
       PARAMETER (NSEED = 12345)
       PARAMETER (NWMAX = 100000)
       PARAMETER (NBUF = 120000)
       REAL*8 DA(NBUF)
       REAL*8 DWORK(NWMAX)
       REAL*8 DSUM,DSUM2
       REAL*8 DMEAN,DM2
       IA = NSEED
       PRINT *, ' Seed ', IA
       N = NBUF
       NWORK = NWMAX
       DSUM = 0.0D0
       DSUM2 = 0.0D0
C NGEN counts down to 0
       NGEN = NRAN
       PRINT *, ' Generating ', NGEN,
      $ ' numbers'
C Generate NRAN numbers ,
C maximum NBUF at a time
       KRPT = (NRAN+NBUF-1)/NBUF
       PRINT *, ' with ', KRPT,
      $ ' call to dvran3'
       DO 20 J = 1, KRPT
       N = MIN0 (NBUF, NGEN)
C First two arguments are mean
C add standard deviation
       CALL DVRAN3 (0.0D0, 1.0D0, IA,
      $ DA, N, DWORK, NWORK, ICON)
       IF (ICON .NE. 0) THEN
         PRINT *, ' Error Return ', ICON
         STOP
       ENDIF
C Accumulate sum of numbers generated
       DO 10 I = 1, N
       DSUM = DSUM + DA(I)
```

```
C Accumulate sum of squares
   10 DSUM2 = DSUM2 + DA(I)*DA(I)
   20 NGEN = NGEN - N
C Compute sample mean
      DMEAN = DSUM/DFLOAT(NRAN)
      PRINT *, ' First moment ', DMEAN
C Compute sample second moment about 0
      DM2 = DSUM2/DFLOAT(NRAN)
      PRINT *, ' Second moment ', DM2
      STOP
      END
```

(4) Method

To generate normally distributed pseudo-random numbers, DVRAN3 uses the Polar method with fast elementary function evaluation. The uniform pseudo-random numbers required in this method are generated using DVRAU4.

The Polar method is described in item [24]. For implementation details and a comparison with other methods, see [4].

J11-20-0501 DVRAN4

Generation of normal random numbers (double precision, Wallace's method)
CALL DVRAN4 (DAM, DSD, IX, DA, N, DWORK, NWORK, ICON)

(1) Function

This subroutine generates pseudo-random numbers from a normal distribution density function (1.1) with a given mean *m* and standard deviation σ.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \qquad (1.1)$$

(2) Parameters

DAM..... Input. Mean *m* of normal distribution.
Double-precision real type.

DSD..... Input. Standard deviation σ of normal distribution.
Double-precision real type.

IX..... Input. Starting value.
On the first call, set IX to a positive value. Subsequently, call the subroutine with the return value remaining 0. A different sequence of random numbers are generated with a different starting value on the first call.
(See item (3), "Comments on use," b., 1).)
4-byte integer type (INTEGER*4).
Output. 0.

DA..... Output. N pseudo-random numbers.
Double-precision real type one-dimensional array of size N.

N..... Input. The number of normally distributed pseudo-random numbers to be returned in DA.
(See item (3), "Comments on use," b., 2).)

DWORK..... Work area. One-dimensional array of the double-precision real type and size NWORK.
Do not modify the contents of this subroutine between repeated calls.
DWORK contains information necessary for repeated calls to this subroutine.
(See item (3), "Comments on use," b., 3).)

NWORK..... Input. The size of the array DWORK. NWORK ≥ 1,350.

ICON..... Output. Condition code.
See Table DVRAN4-1, "Condition codes."

Table DVRAN4-1 Condition codes

Code	Description	Processing
0	No error	
30001	NWORK is too small. IX < 0, DSD ≤ 0	Processing is stopped.
30002	Internal check failed.	
30003 to 30008	DWORK was overwritten or IX was set to zero on the initial call.	
30009	IX is too large.	
40001 to 40002	DWORK was over written or IX was set to zero on the initial call.	

(3) Comments on use

a. Subprograms used

SSL II: DUF2G3, DUITG3, DURN3B, DURUG3, DUR2G3, DUSKG3, DUSQG3, DUVRG3, DVRAU4, MGSSL

b. Comments

1) Starting point IX

When a sequence of pseudo-random numbers is to be generated by a deterministic program, there must be random input. To do this, give a starting value IX. On the first call to this subroutine, the starting value IX must be a positive integer. (For exceptions, see item 5).) For subsequent calls, set IX to zero. This indicates that more pseudo-random numbers from the same sequence are to be generated. To simplify programming, this subroutine returns zero in IX after the initial call.

2) Parameter N

This subroutine returns the next N pseudo-random numbers from the infinite sequence defined by the starting value IX. If N ≤ 0, no pseudo-random numbers are returned.

For efficiency, the user should make N sufficiently large (for instance, N = 100,000). This reduces the overhead of subroutine calls and allows vectorization. N may be changed on consecutive calls to this subroutine provided that the size of array DA is as large as the maximum value of N.

3) Work area DWORK

DWORK is a work area to store state information for repeated calls to this subroutine. The calling program must not change the contents of DWORK while the subroutine is being called.

4) Parameter NWORK

DWORK (1), ..., DWORK (NWORK) are used by this subroutine. NWORK should remain unchanged on each call to the subroutine. NWORK should be at least 1,350. For efficiency on vector processors, NWORK should be large (for example, NWORK = 500,000).

5) Repeated generation of the same random number

If DWORK (1), ..., DWORK (NWORK) is saved, the same sequence of random numbers can be generated again (from the point where DWORK was saved) by reusing DWORK (1), ..., DWORK (NWORK) and calling this subroutine with argument IX = 0.

6) The implementation of Wallace's method in DVRAN4 is about three times faster than the implementation of the Polar method in DVRAN3.

c. Example of use

In this example, one million normal pseudo-random numbers are generated, and the first- and second-order moments are calculated. The starting value is 12345. The first-order moment is $\sum X_i F_i$ when the frequency of the variable X_i is F_i. The second-order moment is $\sum X_i^2 F_i$ when the frequency of the variable X_i is F_i.

```
C     ** EXAMPLE **
      PARAMETER  (NRAN  = 1000000)
      PARAMETER  (NSEED = 12345)
      PARAMETER  (NWMAX = 100000)
      PARAMETER  (NBUF  = 120000)
      REAL*8     DA(NBUF)
      REAL*8     DWORK(NWMAX)
      REAL*8     DSUM, DSUM2
      REAL*8     DMEAN, DM2
      IA = NSEED
      PRINT *, ' Seed ', IA
      N = NBUF
      NWORK = NWMAX
      DSUM  = 0.0D0
      DSUM2 = 0.0D0
C NGEN counts down to 0
      NGEN = NRAN
      PRINT *, ' Generating ', NGEN,
     $           ' numbers'
C Generate NRAN numbers ,
C maximum NBUF at a time
      KRPT = (NRAN+NBUF-1)/NBUF
      PRINT *, ' with ', KRPT,
     $           ' call to dvran4'
      DO 20 J = 1, KRPT
      N = MIN0 (NBUF, NGEN)
C First two arguments are mean
C add standard deviation
      CALL DVRAN4 (0.0D0, 1.0D0, IA,
     $ DA, N, DWORK, NWORK, ICON)
      IF (ICON .NE. 0) THEN
        PRINT *, ' Error Return ', ICON
        STOP
        ENDIF
C Accumulate sum of numbers generated
      DO 10 I = 1, N
      DSUM = DSUM + DA(I)
C Accumulate sum of squares
 10   DSUM2 = DSUM2 + DA(I)*DA(I)
```

```
 20   NGEN  = NGEN - N
C Compute sample mean
      DMEAN = DSUM/DFLOAT(NRAN)
      PRINT *, ' First moment ', DMEAN
C Compute sample second moment about 0
      DM2 = DSUM2/DFLOAT(NRAN)
      PRINT *, ' Second moment ', DM2
      STOP
      END
```

(4) Method

DVRAN4 uses a variant of Wallace’s method to generate normally distributed pseudo-random numbers. This requires uniform pseudo-random numbers, which are generated using DVRAU4.

Wallace's method is described in reference [42]. Implementation details and comparisons with other methods are given in references [4] and [5]

J11-11-0301 DVRAU4

Generation of uniform [0, 1) pseudo-random numbers (double precision)
CALL DVRAU4 (IX, DA, N, DWORK, NWORK, ICON)

(1) Function

This subroutine generates a sequence of pseudo-random numbers from a uniform distribution on [0, 1).

(2) Parameters

IX..... Input. Starting value.
On the first call, set IX to a positive value. Subsequently, call the subroutine with the return value remaining 0.
(See item (3), "Comments on use," b., 1).)
A different sequence of random numbers are generated with a different IX value on the first call.
(See item (4), "Method.")
4-byte integer type (INTEGER*4).
Output. 0.

DA..... Output. N pseudo-random numbers independent and uniform in [0, 1).
Double-precision real type one-dimensional array of size N.

N..... Input. The number of uniformly distributed pseudo-random numbers to be returned in DA.
(See item (3), "Comments on use," b., 2).)

DWORK..... Work area. One-dimensional array of double-precision real type with size of at least NWORK.
Do not modify the contents between repeated calls to this subroutine.
DWORK contains all of the current state information necessary to call this subroutine again from its current point.
(See item (3), "Comments on use," b., 3).)

NWORK..... Input. The size of the array DWORK. NWORK ≥ 388.

ICON..... Output. Condition code.
See Table DVRAU4-1, "Condition codes."

Table DVRAU4-1 Condition codes

Code	Description	Processing
0	No error	
30001	NWORK is too small.	Processing is stopped.
30002	IX<0	
30003 to 30008	DWORK was modified. Or IX was set to zero on the initial call.	

(3) Comments on use

a. Subprograms used

SSL II: DUITG3, DURUG3, DUR2G3, DUF2G3, DUSKG3, DUSQG3, DUVRG3, MGSSL

b. Comments

1) Starting value IX

When a sequence of pseudo-random numbers is to be generated by a deterministic program, there must be random input. To do this, give a starting value IX. This starting value is often called the "*seed*." On the first call to this subroutine, the starting value IX must be a positive integer. (For exceptions, see item 5).) On subsequent calls, set IX to zero. This indicates that subsequent pseudo-random numbers from the same sequence are to be generated. To simplify programming, this subroutine returns zero in IX after the initial call.

2) Parameter N

This subroutine returns the next N pseudo-random numbers from the infinite sequence defined by the starting value IX. If $N \leq 0$, no pseudo-random numbers are returned.

For efficiency, make N sufficiently large (for example, N = 100,000). This reduces the overhead of subroutine calls and allows vectorization. N may be different on successive calls to this subroutine, provided that the size of array DA is as large as the maximum value of N.

3) Work area DWORK

DWORK is a work area used to store state information for repeated calls to this subroutine. The calling program must not change the contents of DWORK while the subroutine is being called.

4) Parameter NWORK

DWORK (1), ..., DWORK (NWORK) are used by this subroutine. NWORK should remain unchanged on each call to the subroutine. NWORK should be at least 388. For efficiency on vector processors, NWORK should be large (for example, NWORK = 45,000).

5) Repeated generation of the same random number

If DWORK (1), ..., DWORK (NWORK) is saved, the same sequence of random numbers can be generated again (from the point where DWORK was saved). Reusing DWORK (1), ..., DWORK (NWORK) and call this subroutine with argument IX = 0.

c. Example of use

In this example, one million uniform pseudo-random numbers are generated and their mean value is calculated. The starting value is 123.

```
C     **EXAMPLE**
      PARAMETER (NRAN = 1000000)
      PARAMETER (NSEED = 123)
      PARAMETER (NWMAX = 45000)
      PARAMETER (NBUF = 160000)
      REAL*8 DA(NBUF)
      REAL*8 DWORK(NWMAX)
      REAL*8 DSUM, DMEAN, DSIG
```

```
      IX = NSEED
      PRINT *, ' SEED ', IX
      N = NBUF
      NWORK = NWMAX
      DSUM = 0.0D0
C NGEN counts down to 0
      NGEN = NRAN
      PRINT *, ' Generating ', NGEN,
     $ ' Numbers'
C Generate NRAN numbers,
C Maximum NBUF at a time
      KRPT = (NRAN+NBUF-1)/NBUF
      PRINT *, ' with ', KRPT,
     $ ' calls to dvrau4'
      DO 20 J = 1, KRPT
      N = MIN0 (NBUF, NGEN)
      CALL DVRAU4 (IX, DA, N,
     $ DWORK, NWORK, ICON)
      IF (ICON .NE. 0) THEN
        PRINT *, ' Error return ', ICON
        STOP
      ENDIF
C Accumulate sun of numbers generated
      DO 10 I = 1, N
   10 DSUM = DSUM + DA(I)
   20 NGEN = NGEN - N
C Compute mean
      DMEAN = DSUM/DFLOAT(NRAN)
      PRINT *, ' Mean ', DMEAN
C Compute deviation from 0.5 normalized
C by expected value 1/sqrt(12*NRAN).
C This should be (approximately) normally
C distributed with mean 0, variance 1.
      DSIG = DMEAN - 0.5D0
      DSIG = DSIG*DSQRT(12.0D0*NRAN)
      PRINT *, ' Norm. deviation ', DSIG
      STOP
      END
```

(4) Method

This subroutine uses the generalized Fibonacci method. If the sequence of pseudo-random numbers is X (1), X (2), ..., then

$X(J) = \alpha * X(U - r) + \beta * X(J - s)$ (*modulo 1*)

where $J > r > s$.

Here, r and s are fixed positive integers (often called "lags"), and α and β are small odd integers.

On the first call (or any call with IX > 0), this subroutine selects a pair (r, s) defining a primitive trinomial (mod 2) and a corresponding linear recurrence. There are 14 possible pairs (r, s), and the one with the largest r is chosen, subject to the constraint that N and NWORK are large enough.

Thus, the user can choose:

- A good generator with a moderately long period, low initialization overhead, and small storage requirements by setting NWORK = 1,000, for example
- A very good generator with an extremely long period, high initialization overhead, and high storage requirements by setting NWORK = 133,000, for example
- An intermediate compromise, without having to know the precise details of how to choose (r, s).

  The pairs (r, s) used by this subroutine are given in Table DVRAU4-2. For tables of primitive trinomials, see [20].

Table DVRAU4-2 Pairs (r, s)

r	s	r	s
127	97	4423	2325
258	175	9689	5502
521	353	19937	10095
607	334	23209	13470
1279	861	44497	23463
2281	1252	110503	56784
3217	2641	132049	79500

This subroutine chooses the parameters (α, β) = (7, 9) if r ≤ 1,000, and (α, β) = (1, 15) if r > 1,000. The rationale is that performance on statistical tests is likely to be improved if α > 1. However, this improvement is only significant for smaller choices of r. For larger choices of r, the performance on statistical tests is very good, even if α = 1. This choice increases the speed of random number generation.

The period of the sequence is $W(2^r - 1)$, where r is in the range 127 (for the smallest NWORK) to 132,049 (for N ≥ 264,098 and NWORK ≥ 132,056). The factor *W* depends on the wordlength. (On the Fujitsu VPP series, $W = 2^{48}$, and the minimum period is at least 10^{52}.)

The initialization ensures that sequences of pseudo-random numbers returned for different starting value IX are separated by a distance of at least $2^{60} > 10^{18}$ in the full periodic sequence. Thus, for all practical purposes, different starting values IX ensure different sequences of pseudo-random numbers.

The method and implementation details are described in more detail in [2] and [3]. For a further explanation and comparison with other methods, see [1], [11], [22], and [27].

(5) Testing of uniform random numbers

Table DVRAU4-3 shows the result of testing statistical hypotheses for the pseudo-random numbers generated by DVRAU4 with NWORK = 44,504 (so r = 44,497 and s = 23,463).

Table DVRAU4-3 Results of χ^2 testing (uniform deviation at *n*-dimensions unit hypercube)

Dimension(*1)	**Size**(*2)	**res$_l$**(*3)	**res$_v$**(*4)	**Density**(*5)	$\sqrt{2\chi^2} - \sqrt{2f-1}$
1	10^9	5×10^7	50000000	20.00	1.21
1	0.8×10^9	1.25×10^7	12500000	64.00	-0.67
2	10^9	7071	49999041	10.00	-0.10
2	2×10^9	3535	12496225	80.02	-0.37
3	2×10^9	368	49836032	13.38	1.40
3	2×10^9	232	12487168	53.39	-0.96
4	2×10^9	84	49787136	10.04	0.76
4	2×10^9	59	12117361	41.26	-0.38

*1 Dimension: Dimension of the unit hypercube.

*2 Size: Number of pseudo-random numbers generated.

*3 res$_l$: Number of equal subintervals partitioning [0, 1) in each dimension.

*4 res$_v$: Number of equal hypercubes partitioning the unit hyper cube.

*5 Density: Average number of random points per small hypercube.

In the table, the number of degrees of freedom '*f*' of chi-squared testing is very large (1,000,000 level). In this case, the expression $\sqrt{2\chi^2} - \sqrt{2f-1}$ should be approximated extremely well as a normal deviate with unit variance.

A72-27-0101 VBCSD, DVBCSD

System of linear equations with unsymmetric or indefinite sparse matrix (BICGSTAB(*l*) method, diagonal storage format)
CALL VBCSD (A, K, NDIAG, N, NOFST, B, ITMAX, EPS, IGUSS, L, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations with an $n \times n$ unsymmetric or indefinite sparse matrix using the Bi-Conjugate Gradient Stabilized(*l*) method (BICGSTAB(*l*)).

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

The $n \times n$ coefficient is stored with the diagonal storage format. Vectors *b* and *x* are *n*-dimensional vectors.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 "Iterative linear equation solvers and Convergence," in Part I.

(2) Parameters

A..... Input. Stores non-zero elements in a coefficient matrix.
Two-dimensional array A (K, NDIAG). Stores coefficient matrix *A* in A (1:N, NDIAG) with the diagonal storage format. For the diagonal storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Methods for General Sparse Matrices," b., "Diagonal Storage Format for General Sparse Matrices."

K..... Input. Size of adjustable dimension of array A

NDIAG..... Input. The number of diagonal vectors in coefficient matrix *A* that contain non-zero elements.
Size of second-dimension of array A.

N..... Input. Order *n* of matrix *A*.

NOFST.... Input. Stores the distance from the main diagonal vector corresponding to diagonal vectors stored in array A. Superdiagonal vectors have positive values; Subdiagonal vectors have negative values.
One-dimensional array NOFST (NDIAG).

B..... Input. One-dimensional array of size *n*. Stores the constant vector of the right-hand side term of a linear equation system.

ITMAX..... Input. The upper limit of iterations in BICGSTAB(*l*) method (> 0).

EPS..... Input. A convergence criterion value in judgment of convergency.
If EPS is 0.0 or less, it is set to 10^{-6} in double-precision routines and 10^{-4} in single-precision routines.
(See item (3), "Comments on use," b., 1).)

IGUSS..... Input. Sets control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=0: Approximate value of the solution vector is not specified.
IGUSS≠0: The iterative computation starts from the approximate value of the solution vector specified in array X.

L..... Input. The order of stabiliser in the BICGSTAB(*l*) algorithm. ($1 \leq L \leq 8$). The value of L should usually be set to 1 or 2. (See item(3), "Comments on use," b., 2).)

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of the solution vector.
Output. The solution vector is stored.

ITER..... Output. Number of iteration performed using the BICGSTAB(*l*) method.

VW..... Work area. One-dimensional array $K \times (4+2 \times L) + N + NBANDL + NBANDR$. NBANDL indicates a lower bandwidth; NBANDR indicates an upper bandwidth. If the order or the bandwidth of the matrix are not constant parameters, it is enough to set the size of VW array to be $K \times (4+2 \times L) +3 \times K$.

ICON..... Output. Condition code
See Table VBCSD-1, "Condition codes."

Table VBCSD-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	Break-down occurred	Processing is stopped.
20001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate value obtained up to this point in array X is output, but their precision cannot be guaranteed.
30000	K<1, N<1, K<N, NDIAG<1, L<1, L>8, K<NDIAG, or ITMAX≤ 0	Processing is stopped.
32001	| NOFST (I) | > N-1	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, UBCRL, UBCSD, UBGRS, UQITB, URELT, URIPA,
URITI, URITT, URMVD, URSTE, USVCN, USVCP, USVN2,
MGSSL, UMGSL

b. Comments

1) Convergent criterion
In the BICGSTAB(*l*) method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of Matrix *A* and EPS.
The residual which used for convergence judgement is computed recursively and it may differ from the true residual.

2) Parameter L
The maximum value of L is set to 8. For L=1, this algorithm coincides with BiCGSTAB. Using smaller L usually results in faster speed, but in some

situations larger L brings a good convergence, although the steps of an iteration are more expensive for larger L.

3) Notes on using the diagonal format
A diagonal vector element outside coefficient matrix *A* must be set to zero.
There is no restriction in the order in which diagonal vectors are stored in array A.
The advantage of this method lies in the fact that the matrix vector multiplication can be calculated without the use of indirect indices. The disadvantage is that matrices without the diagonal structure cannot be stored efficiently with this method.

4) Diagonal scaling
Scaling the equations so that the main diagonal to be 1 may results in better convergence.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0,1] \times [0,1] \times [0,1]$, with the Dirichlet boundary condition (function value zero at the boundary).

This type of partial differential operator is described in Part I, "Overview," Section 3.2.2 "Discretization of partial differential operators and storage examples for them."

For INIT_MAT_DIAG, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them."
GET_BANDWIDTH_DIAG is a routine that estimates band width. INIT_SOL is a routine that generates solution vectors to be sought with random numbers.

```
c     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER MACH
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000)
      PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
      PARAMETER (NDIAG = 7, LEN = N+400+400)
      PARAMETER (L = 4)
      PARAMETER (NVW = (4+2*L)*K+LEN)
      DOUBLE PRECISION A(K,NDIAG),X(N),B(N),SOLEX(N)
      INTEGER NOFST(NDIAG)
      DOUBLE PRECISION VW(NVW)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUSIONS'
      PRINT*,'X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
C
      PRINT *
      PRINT *,'    BiCGstab(l) METHOD'
      PRINT *,'    DIAGONAL FORMAT'
C
      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
```

```
      ZL = 1.0
C
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,A,NOFST
     &           ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      CALL GET_BANDWIDTH_DIAG(NOFST,NDIAG,NBANDL,NBANDR)
          DO 110 I = 1,N
              VW(I+NBANDL) = SOLEX(I)
 110      CONTINUE
      CALL DVMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON)
      PRINT*,'DVMVSD ICON= ',ICON
C
      EPS = 1D-10
      IGUSS = 0
      ITMAX = 2000
      CALL DVBCSD(A,K,NDIAG,N,NOFST,B,ITMAX
     &             ,EPS,IGUSS,L,X,ITER,VW,ICON)
C
      PRINT* ,'ITER = ',ITER
      PRINT* ,'DVBCSD ICON = ',ICON
      PRINT*, 'COMPUTED VALUES'
      PRINT*, 'X(1) = ',X(1),' X(N) = ',X(N)
      STOP
      END
```

(4) Method

The BICG algorithm is described in [37] in Appendix B, “References.” The BICGSTAB(*l*) algorithm is a modification of the BICGSTAB method, see [41] and [16] in Appendix B, “References.”

A72-28-0101 VBCSE, DVBCSE

System of linear equations with unsymmetric or indefinite sparse matrix (BICGSTAB(*l*) method, ELLPACK storage format)
CALL VBCSE (A, K, IWIDT, N, ICOL, B, ITMAX, EPS, IGUSS, L, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations with an $n \times n$ unsymmetric or indefinite sparse matrix using the Bi-Conjugate Gradient Stabilized(*l*) method (BICGSTAB(*l*)) method.

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

Coefficient matrices ($n \times n$) are stored with the ELLPACK format. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 "Iterative linear equation solvers and Convergence," in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, IWIDT).
For the ELLPACK storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices."

K..... Input. Size of adjustable dimension ($\geq n$) of A and ICOL.

IWIDT..... Input. The maximum number of non-zero-elements in row vector direction on the coefficient matrix *A*.
Two dimensional size of A and ICOL.

N..... Input. Order *n* of matrix *A*.

ICOL..... Input. Store the column indices of the element stored in the array A using the ELLPACK format, indicating which column vectors the corresponding elements in the array A belong to.
Two-dimensional array ICOL (K, IWIDT)

B..... Input. One-dimensional array of size *n*. Stores a constant vector of the right-hand-side term of a linear equation system.

ITMAX..... Input. The upper limit of iterations in BICGSTAB(*l*) method (> 0).

EPS..... Input. A convergence criterion value in judgment of convergency.
If EPS is 0.0 or less, it is set to 10^{-6} in double-precision routines and 10^{-4} in single-precision routines.
(See item (3), "Comments on use," b., 1).)

IGUSS..... Input. Control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=0: Approximate value of the solution vector is not set.
IGUSS≠0: The iterative computation starts from the approximate value of the solution vector specified in array X.

L..... Input. The order of stablilser in the BICGSTAB(*l*) algorithm. ($1 \leq L \leq 8$).
The value of L should usually be set to 1 or 2. (See item(3), "Comments on use," b., 2).)

X..... Input. One-dimensional array of size *n*. An approximate value of a solution vector can be specified.
Output. Stores a solution vector.

ITER..... Output. The real number of iteration steps in BICGSTAB(*l*) method.

VW..... Work area. One-dimensional array $K \times (4+2 \times L)$.

ICON..... Output. Condition code
See Table VBCSE-1, "Condition codes."

Table VBCSE-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	Break-down occurred	Processing is stopped.
20001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
30000	K<1, N<1, K<N, IWIDT<1, L<1, L>8, K<IWIDT, or ITMAX≤ 0	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, UBCRL, UBCSE, UBGRS, UQITB, URELT, URIPA, URITI, URITT, URMVE, URSTE, USVCN, USVCP, USVN2, MGSSL, UMGSL

b. Comments

1) Convergent criterion
In the BICGSTAB(*l*) method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and obtained approximate solution is equal to the product of the condition number of matrix A and EPS.
The residual which used for convergence judgement is computed recursively and it may differ from the true residual.

2) Parameter L
The maximum value of L is set to 8. For L=1, this algorithm coincides with BiCGSTAB. Using smaller L usually results in faster speed, but in some situations larger L brings a convergence, although the steps of a iteration are more expensive for larger L.

3) Diagonal scaling
Scaling the equations so that the main diagonal to be 1 may results in better convergence.

c. Example of use

In this example, linear equations of coefficient matrices obtained by descretizing partial differential operators are solved in the region $[0,1] \times [0,1] \times [0,1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of

partial differential operator is described in Part I, “Overview,” Section 3.2.2, “Descretization of partial differential operator and storage examples for them.”

For INIT_MAT_ELL, see Part I, “Overview,” Section 3.2.2, “Discretization of partial differential operators and storage examples for them.”

INIT_SOL is the routine that generates the solution vectors to be sought in random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER MACH
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000)
      PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
      PARAMETER (IWIDT = 7, L = 4)
      PARAMETER (NVW = (4+2*L)*K)
      DOUBLE PRECISION A(K,IWIDT),X(N),B(N),VW(NVW),SOLEX(N)
      INTEGER ICOL(K,IWIDT)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUTION'
      PRINT*,'X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
C
      PRINT *
      PRINT *,'    BiCGstab(l) METHOD'
      PRINT *,'    ELLPACK FORMAT'
C
      AVI = 3D0
      AV2 = 1D0/3D0
      AV3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
C
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL,
     &          NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)
      CALL DVMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON)
      PRINT*,'DVMVSE ICON = ',ICON
C
      EPS = 1D-10
      IGUSS = 0
      ITMAX = 2000
      CALL DVBCSE(A,K,IWIDT,N,ICOL,B,ITMAX,
     &           EPS,IGUSS,L,X,ITER,VW,ICON)
C
      PRINT*,'DVBCSE ICON = ',ICON
      PRINT*,'COMPUTED VALUE'
      PRINT*,'X(1) = ',X(1),' X(N) = ',X(N)
      STOP
      END
```

(4) Method

The BICG algorithm is described in [37] in Appendix B, “References.” The BICGSTAB(*l*) algorithm is a modification of the BICGSTAB method, see [41] and [16] in Appendix B, “References.”

A53-31-0102 VBLDL, DVBLDL

LDL^T decomposition of symmetric positive definite banded matrix (modified Cholesky decomposition)
CALL VBLDL (A, N, NH, EPSZ, ICON)

(1) Function

Using modified Cholesky decomposition, this routine computes the LDL^T decomposition

$\boldsymbol{A} = \boldsymbol{LDL}^T$

of an $n \times n$ symmetric positive definite banded matrix $\boldsymbol{A}$ with an upper and lower bandwidth h, where $\boldsymbol{L}$ is a unit lower banded matrix with the lower bandwidth h, and $\boldsymbol{D}$ is a diagonal matrix.

The condition $n > h \geq 0$ must be met.

In order to exploit vector computer performance, this routine adopts the method of storage in the order of column vectors.

(2) Parameters

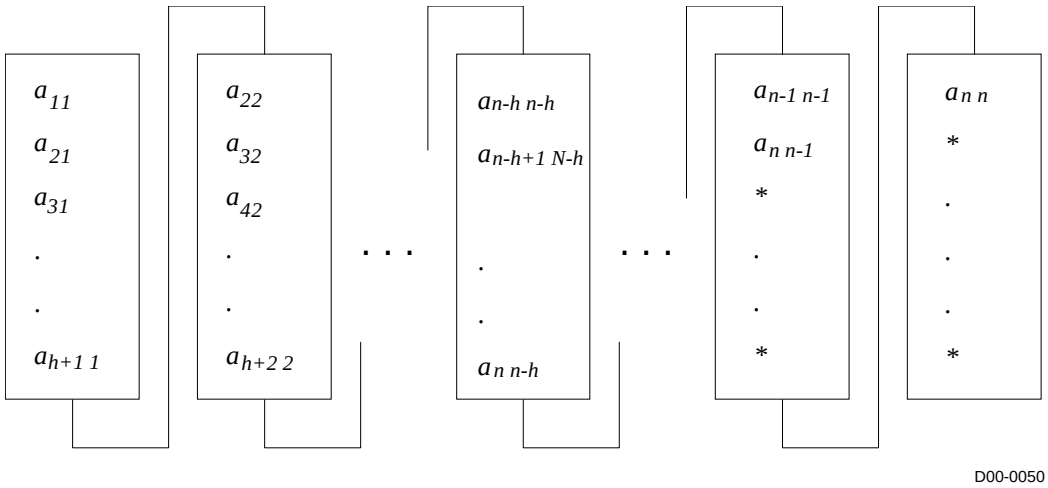
A..... Input. One-dimensional array of size $(h + 1) \times n$.
Stores diagonal elements of the coefficient matrix $\boldsymbol{A}$ and the lower banded matrix.
For the storage method for matrix $\boldsymbol{A}$, see Figure VBLDL-1.
Output. Stores LDL^T decomposed $\boldsymbol{D}$ and $\boldsymbol{L}$.
For the storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{D}$, see Figure VBLDL-2.

N..... Input. Order n of matrix $\boldsymbol{A}$.

NH..... Input. Lower bandwidth h.

EPSZ..... Input. Value of pivot judgment of relative zero (≥ 0.0).
When it is 0.0, standard values are applied.
(See item (3), “Comments on use,” b., 1).)

ICON..... Output. Condition code.
See Table VBLDL-1, “Condition codes.”



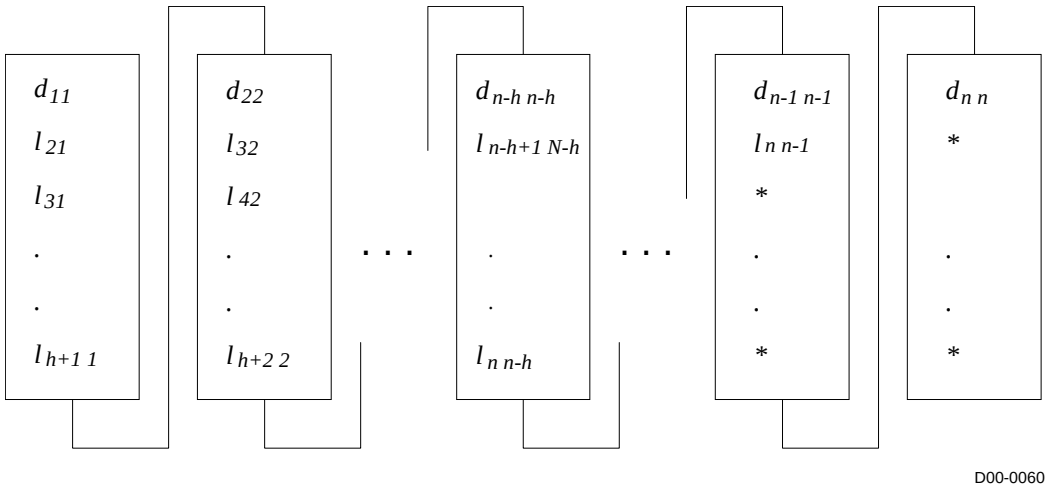
* (asterisk): Undefined value

Figure VBLDL-1 Storage method for matrix *A* in array A

i column vectors of the lower banded matrix ***A*** are stored according to

A ($(h + 1) \times (i - 1) + j - i + 1) = a_{ji}$

where $j = i, ..., i + h,\ i = 1, ... , n$



* (asterisk): Undefined value

Figure VBLDL-2 Storage method for matrices *L* and *D* in array A

d_{ii} is stored in A ($(h + 1) \times (i - 1) + 1$).

l_{ji} is stored in A ($(h + 1) \times (i - 1) + j - i + 1$).

$j = i + 1, ... , i + h,\ i = 1, ... , n$

Table VBLDL-1 Condition codes

Code	Description	Processing
0	No error	
10000	A negative pivot. Matrix ***A*** is not positive definite.	Processing continues.
20000	Pivot is relative zero. Strong possibility that matrix ***A*** is singular.	Processing is stopped.
30000	NH < 0, NH ≥ N or EPSZ < 0.0	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, MGSSL

b. Comments

1) In this subroutine, the case of the pivot value being less than EPSZ is considered relative zero, and processing is stopped with ICON = 20,000.

The standard value of EPSZ is $16 \times u$, where u is the round off unit.

2) If the pivot value becomes negative during decomposition, the coefficient matrix is not positive definite. In such a case, this subroutine continues processing, but with ICON = 10,000.

3) The elements of matrix ***L*** that result from the decomposition are stored in array A, as demonstrated in Figure VBLDL-2. Thus, the determinant is obtained from the multiplication of n diagonal elements: A ($(h + 1) \times (i - 1) + 1$), $i = 1, ..., n$.

c. Example of use

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(NH=128)
      PARAMETER(N=128*128)
      DIMENSION A((NH+1)*N),C(NH+1,N)
      EQUIVALENCE(A,C)
C
C     Zero clear
C
      DO 10 I=1,N*(NH+1)
      A(I)=0.0
   10 CONTINUE
C
C     Coefficient Matrix is built
C     b = A*y , where y=(1,1,....,1)
C
      DO 20 I=1,N
      C(1,I)=1.0
      IF(I+NH.LE.N)THEN
      C(NH+1,I)=-0.25
      ENDIF
      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(2,I)=-0.25
      ENDIF
```

```
   20 CONTINUE
C
C     LDL^T decomposition
C
      EPSZ=0.0D0
      CALL DVBLDL(A,N,NH,EPSZ,ICON)
      PRINT*,'ICON=',ICON
      IF(ICON.NE.0)STOP
C
      DET=1.0D0
      DO 30 I=1,N
      DET=DET*C(1,I)
   30 CONTINUE
C
      PRINT*,'DETERMINANT=',DET
      STOP
      END
```

(4) Method

LDL^T decomposition is performed with the modified Cholesky decomposition of the outer product type. (See [31].)

A53-31-0202 VBLDX, DVBLDX

System of linear equations with an LDL^T decomposed symmetric positive definite banded matrix
CALL VBLDX (B, FA, N, NH, ICON)

(1) Function

This routine solves the following linear equations of LDL^T decomposed symmetric positive definite banded matrix contained in the coefficient matrix:

$$\boldsymbol{LDL^Tx = b} \qquad (1.1)$$

L and ***D*** are each an $n \times n$ unit lower banded matrix with the lower bandwidth h. ***D*** is a diagonal matrix, ***b*** is an n-dimensional real constant vector, and ***x*** is an n-dimensional solution vector.

The condition $n > h \geq 0$ must be met.

(2) Parameters

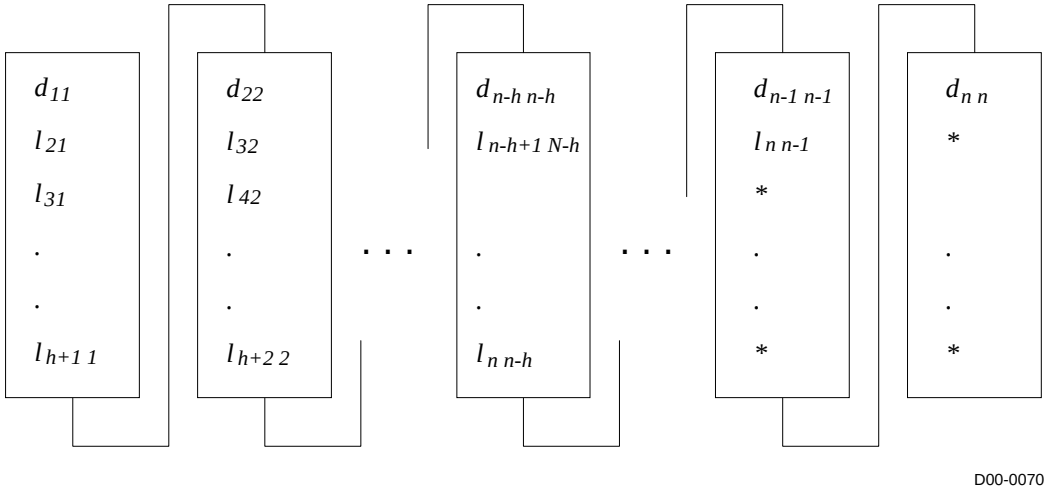
B..... Input. Constant vector ***b***.
Output. Solution vector ***x***.
One-dimensional array of size n.

FA..... Input. One-dimensional array of size $(h + 1) \times n$.
See Figure VBLDX-1, "Storage method for matrices ***L*** and ***D*** in array FA," for the storage method of LDL^T decomposed matrices L and D.

N..... Input. Order n of matrix ***A***.

NH..... Input. Lower bandwidth h.

ICON..... Output. Condition code.
See Table VBLDX-1, "Condition codes."



* (asterisk): Undefined value

Figure VBLDX-1 Storage method for matrices *L* and *D* in array FA

d_{ii} is stored in FA ($(h + 1) \times (i - 1) + 1$).

l_{ji} is stored in FA ($(h + 1) \times (i - 1) + j - i + 1$).

$j = i + 1, ..., i + h,\ i = 1, ... , n$

Table VBLDX-1 Condition codes

Code	Description	Processing
0	No error	
10000	Coefficient matrix A is not positive definite.	Processing continues.
30000	NH < 0, NH ≥ N	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: UBLTS, UBUTS, MGSSL

b. Comments

1) Linear equations can be solved by calling this routine consecutively after the subroutine VBLDL. However, one call to the subroutine VLSBX usually brings the same solution.

c. Example of use

In this example, a symmetric positive definite banded matrix, where bandwidth h = 256 and $n = 256 \times 256$, is LDL^T decomposed and $\boldsymbol{Ax} = \boldsymbol{b}$ is solved.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(NH=128)
      PARAMETER(N=128*128)
      DIMENSION A((NH+1)*N),B(N),C(NH+1,N)
      EQUIVALENCE(A,C)
C
C     Zero clear
C
      DO 10 I=1,N*(NH+1)
      A(I)=0.0
   10 CONTINUE
C
      DO 15 I=1,N
      B(I)=0.0
   15 CONTINUE
C
C     Coefficient Matrix is built
C     b = A*y , where y=(1,1,....,1)
C
      DO 20 I=1,N
      C(1,I)=4.0
      B(I)=B(I)+4.0
C
      IF(I+NH.LE.N)THEN
      C(NH+1,I)=-1.0
      B(I+NH)=B(I+NH)-1.0
      B(I)=B(I)-1.0
```

```
      ENDIF
C
      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(2,I)=-1.0
      B(I+1)=B(I+1)-1.0
      B(I)=B(I)-1.0
      ENDIF
   20 CONTINUE
C
C     Solve Symmetric Positive Definite linear equation
C
      EPSZ=0.0D0
      CALL DVBLDL(A,N,NH,EPSZ,ICON)
      PRINT*,'VBLDL ICON=',ICON
      IF(ICON.NE.0)STOP
      CALL DVBLDX(B,A,N,NH,ICON)
      PRINT*,'VBLDX ICON=',ICON
      IF(ICON.NE.0)STOP
C
      PRINT*,'B(1)= ',B(1)
      PRINT*,'B(N)= ',B(N)
      STOP
      END
```

(4) Method

The solution is obtained through forward-substitution and back-substitution.

A53-11-0102 VBLU, DVBLU

LU decomposition of banded real matrix (Gaussian elimination)
CALL VBLU (A, N, NH1, NH2, EPSZ, IS, IP, VW, ICON)

(1) Function

This routine LU decomposes an $n \times n$ banded matrix with a lower bandwidth h_1 and an upper bandwidth h_2 using Gaussian elimination:

$\boldsymbol{P}\,\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U}$

where $\boldsymbol{P}$ is a permutation matrix on which the exchange of rows is performed through partial pivoting. $\boldsymbol{L}$ is a unit lower banded matrix, and $\boldsymbol{U}$ is an upper banded matrix.

The condition $n > h_1 \geq 0$, $n > h_2 \geq 0$ must be met.

In order to exploit vector computer performance, this routine adopts an appropriate banded matrix storage method.

(2) Parameters

A..... Input. One-dimensional array of size $(2 \times h_1 + h_2 + 1) \times n$ that stores the banded coefficient matrix $\boldsymbol{A}$.
For the storage method for matrix $\boldsymbol{A}$, see Figure VBLU-1, "Storage method for banded matrix in array A."
Output. Stores the LU decomposed $\boldsymbol{L}$ and $\boldsymbol{U}$.
For the storage method for matrices L and U, see Figure VBLU-2, "Storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ in array A."

N..... Input. Order of matrix $\boldsymbol{A}$.

NH1..... Input. Lower bandwidth h_1 of matrix $\boldsymbol{A}$.

NH2..... Input. Upper bandwidth h_2 of matrix $\boldsymbol{A}$.

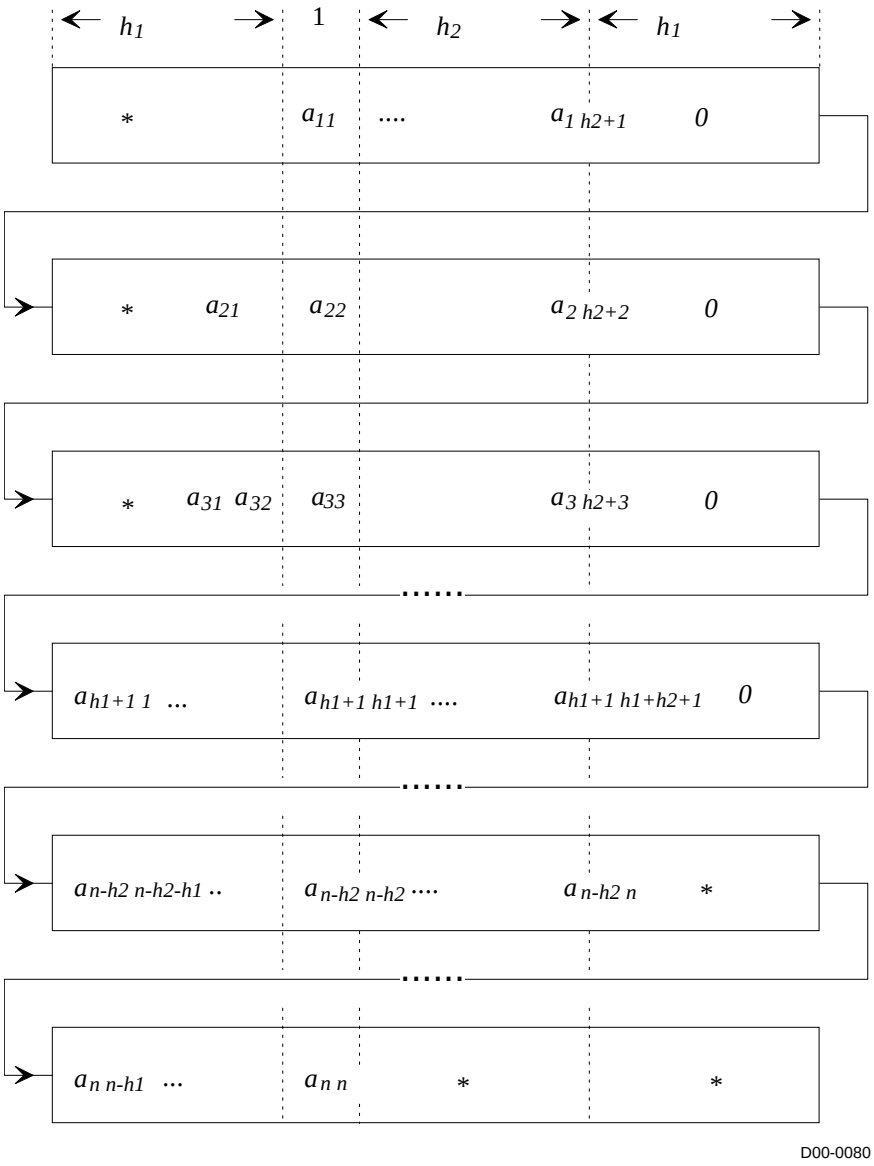
EPSZ..... Input. Value of pivot judgment of relative zero (≥ 0.0). If it is 0.0, the standard value is set.

IS..... Output. Information used when seeking the determinant of matrix $\boldsymbol{A}$.
(See item (3), "Comments on use," b., 2).)

IP..... Output. Transposition vector that shows the history of the exchange of rows performed through partial pivoting. One-dimensional array of size n.

VW..... Work area. One-dimensional array of size n.

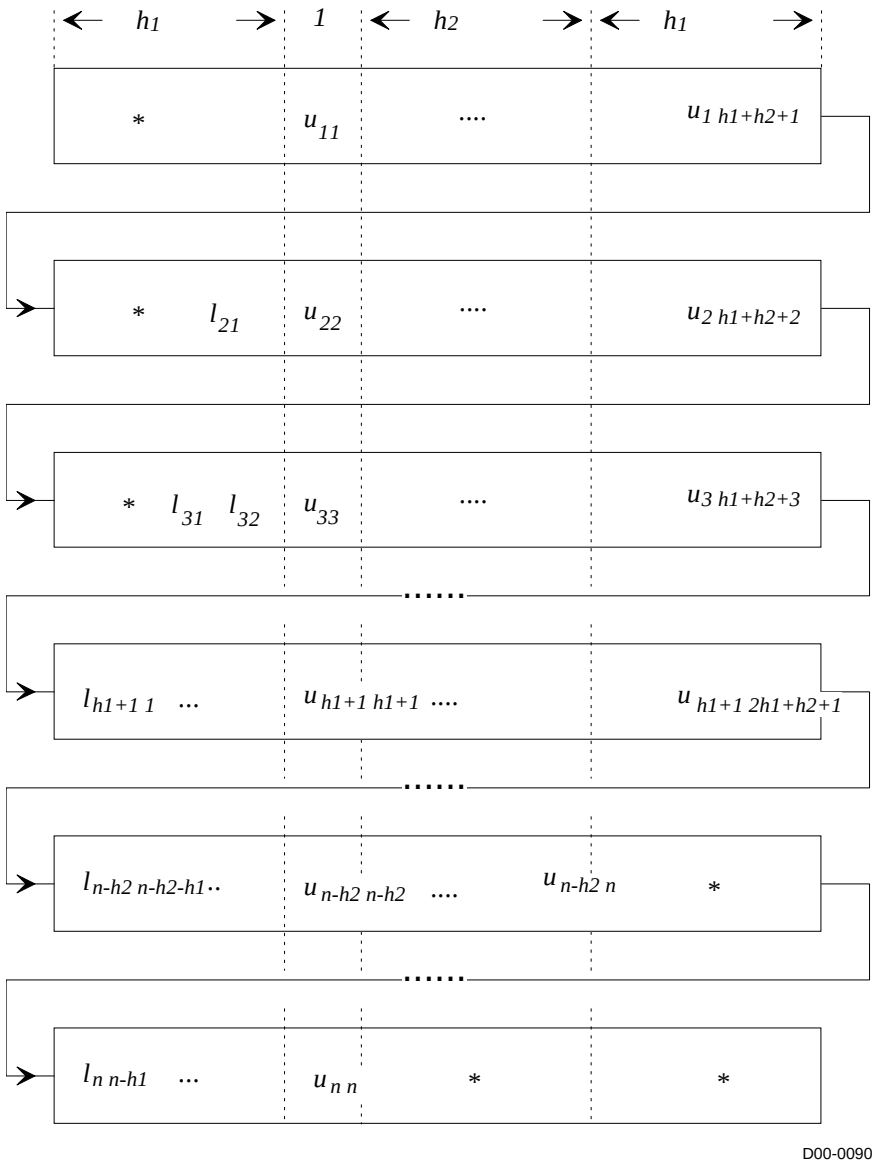
ICON..... Output. Condition code.
See Table VBLU-1, "Condition codes."



* (asterisk): Undefined value

Figure VBLU-1 Storage method for banded matrix in array A

The i-th row vector of the coefficient matrix ***A*** is stored consecutively in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + 1$: $(2 \times h_1 + h_2 + 1) \times i$). Diagonal elements a_{ii} are stored in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$). The elements outside coefficient matrix of the banded part are set to zero when being stored.



* (asterisk): Undefined value

Figure VBLU-2 Storage method for matrices *L* and *U* in array A

The *i*-th row vector without diagonal elements of matrix ***L*** is stored in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + 1$: $(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1$). The i-th row vector of matrix ***U*** is stored in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$: $(2 \times h_1 + h_2 + 1) \times i$), consecutively from the diagonal elements.

Table VBLU-1 Condition codes

Code	Description	Processing
0	No error	
20000	All the elements of a row of matrix ***A*** were zero, or the pivot is relative zero. There is a strong possibility that matrix ***A*** is singular.	Processing is stopped.
30000	N ≤ NH1, N ≤ NH2, NH1 < 0, NH2 < 0 or EPSZ < 0.0	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, MGSSL

b. Comments

1) In this subroutine, the case of the pivot value being less than EPSZ is considered relative zero. Processing is stopped with ICON = 20,000.

The standard value of EPSZ is $16 \times u$, where u is the round off unit.

2) Elements of matrix U are stored in array A, as demonstrated in Figure VBLU-2. Therefore, the determinant is obtained by multiplying the IS value by n diagonal elements, that is, the multiplication A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$), i = 1, ... , n.

3) In partial pivoting, this subroutine performs an actual exchange of the rows of array A. If at the J-th step of decomposition (J = 1, 2, ..., *n-1*) the I-th row (I ≥ J) is selected as the pivot row, the contents of the I-th and J-th row of array A are interchanged. In order to show the history, I is then stored in IP (J).

4) In order to save space in the data storage area, this subroutine stores banded matrices by taking advantage of their characteristics. However, depending on bandwidth size, a data storage area that is larger than VALU may be required. In such cases, space in the data storage area can be saved by using VALU.

Characteristics of this subroutine can be exploited when $n > 2 \times h_1 + h_2 + 1$.

c. Example of use

In this example, the determinant of an unsymmetric banded matrix with $h_1 = h_2 = 160$, $n = 160 \times 160$, is computed.

```
C       **EXAMPLE**
        IMPLICIT REAL*8 (A-H,O-Z)
        PARAMETER(NH=80)
        PARAMETER(NH1=NH)
        PARAMETER(NH2=NH)
        PARAMETER(N=NH*NH)
        PARAMETER(ALPHA=0.5/(NH1+1)/4,BETA=-ALPHA)
        DIMENSION A((2*NH1+NH2+1)*2*N)
        DIMENSION C(2*NH1+NH2+1,N),IP(N),VW(N)
        EQUIVALENCE(A,C)
C
C       Zero clear
C
        DO 10 I=1,N*(3*NH+1)
```

```
      A(I)=0.0
   10 CONTINUE
C
C     Coefficient Matrix is built
C
      DO 20 I=1,N
      C(NH1+1,I)=1.0
      IF(I.GT.NH)THEN
      C(1,I)=-0.25+ALPHA
      ENDIF
      IF(I+NH.LE.N)THEN
      C(1+NH1+NH2,I)=-0.25+BETA
      ENDIF
      IF(I.GT.1.AND.MOD(I-1,NH).NE.0)THEN
      C(NH1,I)=-0.25+ALPHA
      ENDIF
      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(NH1+2,I)=-0.25+BETA
      ENDIF
   20 CONTINUE
C
C     LU decomposition
C
      EPSZ=0.0D0
      ICON=0
      CALL DVBLU(A,N,NH1,NH2,EPSZ,IS,IP,VW,ICON)
      PRINT*,'ICON= ',ICON
      IF(ICON.NE.0)STOP
C
      DET=IS
      DO 30 I=1,N
      DET=DET*C(NH1+1,I)
   30 CONTINUE
C
      PRINT*,'DETERMINANT=',DET
      STOP
      END
```

(4) Method

LU decomposition is performed through LU decomposition of the outer product type. (See [14]).

A53-11-0202 VBLUX, DVBLUX

System of linear equations with an LU decomposed banded real matrix
CALL VBLUX (B, FA, N, NH1, NH2, IP, ICON)

(1) Function

This routine solves linear equations

$\boldsymbol{Ax = b}$

through forward-substitution and back-substitution, based on the result

$\boldsymbol{P\,A = L\,U}$

obtained by LU decomposing an $n \times n$ banded matrix with a lower bandwidth h_1 and an upper bandwidth h_2 using Gaussian elimination.

$\boldsymbol{P}$ is a permutation matrix on which the exchange of rows is performed through partial pivoting. $\boldsymbol{L}$ is a unit lower banded matrix, and $\boldsymbol{U}$ is an upper banded matrix.

The condition $n > h_1 \geq 0$, $n > h_2 \geq 0$ must be met.

(2) Parameters

B..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

FA..... Input. Stores LU decomposed $\boldsymbol{L}$ and $\boldsymbol{U}$.
One-dimensional array of size $(2 \times h_1 + h_2 + 1) \times n$.
For the storage method of matrices $\boldsymbol{L}$ and $\boldsymbol{U}$, see Figure VBLUX-1, “Storage method for $\boldsymbol{L}$ and $\boldsymbol{U}$ in array A.”

N..... Input. Order of matrix $\boldsymbol{A}$.

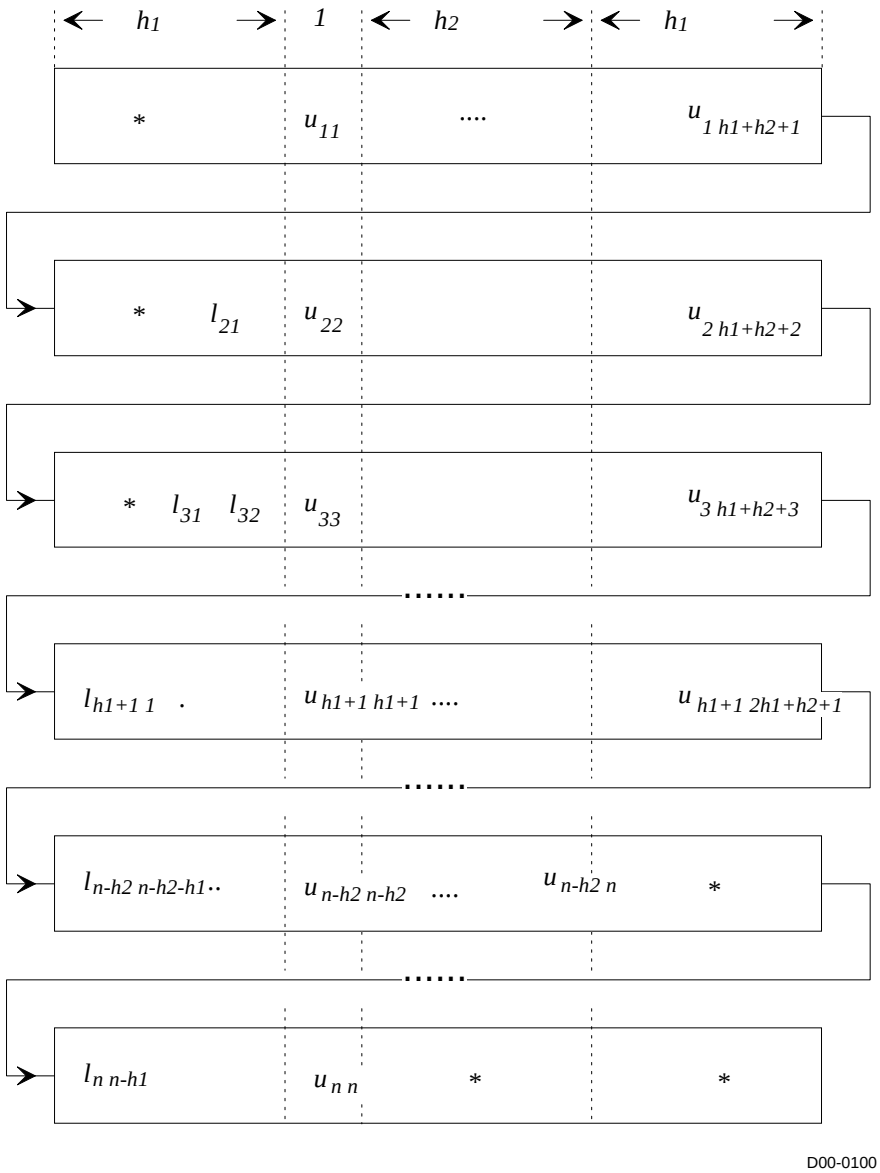
NH1..... Input. Lower bandwidth h_1 of matrix $\boldsymbol{A}$.

NH2..... Input. Upper bandwidth h_2 of matrix $\boldsymbol{A}$.

IP..... Input. Transposition vector that shows the history of the exchange of rows performed through partial pivoting. One-dimensional array of size n.

ICON..... Output. Condition code.

See Table VBLUX-1, “Condition codes.”



* (asterisk): Undefined value

Figure VBLUX-1 Storage method for *L* and *U* in array FA

The i-th row vector without diagonal elements of matrix ***L*** is stored in FA ($(2 \times h_1 + h_2 + 1) \times (i - 1) + 1$: $(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1$). The *i*-th row vector of matrix ***U*** is stored consecutively from the diagonal elements in FA ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$: $(2 \times h_1 + h_2 + 1) \times i$).

Table VBLUX-1 Condition codes

Code	Description	Processing
0	No error	
20000	Coefficient matrix is singular.	Processing is stopped.
30000	N ≤ NH1, N ≤ NH2, NH1 < 0, NH2 < 0, or error occurred in IP.	Processing is stopped.

(3) Comments on use

a. Subprogram used

SSL II: MGSSL

b. Comments

1) A linear equation can be solved by calling this subroutine after calling the subroutine VBLU. At that time, set the input parameters of this subroutine (with the exception of constant vectors) to the output parameters of VBLU.

c. Example of use

In this example, a linear equation $\boldsymbol{Ax} = \boldsymbol{b}$, which takes the unsymmetric matrix $\boldsymbol{A}$ with $h_1 = h_2 = 160$, n = 160 × 160 as a coefficient matrix, is solved.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(NH=80)
      PARAMETER(NH1=NH)
      PARAMETER(NH2=NH)
      PARAMETER(N=NH*NH)
      PARAMETER(ALPHA=0.5/(NH1+1),BETA=-ALPHA)
      DIMENSION A((2*NH1+NH2+1)*2*N),B(N)
      DIMENSION C(2*NH1+NH2+1,N),IP(N),VW(N)
      EQUIVALENCE(A,C)
C
C     Zero clear
C
      DO 10 I=1,N*(3*NH+1)
      A(I)=0.0
   10 CONTINUE
C
      DO 15 I=1,N
      B(I)=0.0
      IP(I)=0
   15 CONTINUE
C
C     Coefficient Matrix is built
C
      DO 20 I=1,N
      C(NH1+1,I)=4.0
      B(I)=B(I)+4.0
      IF(I.GT.NH)THEN
      C(1,I)=-1.0+ALPHA
      B(I)=B(I)-1.0+ALPHA
      ENDIF
```

```
      IF(I+NH.LE.N)THEN
      C(1+NH1+NH2,I)=-1.0+BETA
      B(I)=B(I)-1.0+BETA
      ENDIF
      IF(I.GT.1.AND.MOD(I-1,NH).NE.0)THEN
      C(NH1,I)=-1.0+ALPHA
      B(I)=B(I)-1.0+ALPHA
      ENDIF
      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(NH1+2,I)=-1.0+BETA
      B(I)=B(I)-1.0+BETA
      ENDIF
   20 CONTINUE
C
C     Solve Banded linear equation
C
      EPSZ=0.0D0
      ICON=0
      CALL DVBLU(A,N,NH1,NH2,EPSZ,IS,IP,VW,ICON)
      PRINT*,'VBLU ICON= ',ICON
      IF(ICON.NE.0)STOP
      CALL DVBLUX(B,A,N,NH1,NH2,IP,ICON)
      PRINT*,'VBLUX ICON= ',ICON
      IF(ICON.NE.0)STOP
      PRINT*,'B(1)= ',B(1)
      PRINT*,'B(N)= ',B(N)
      STOP
      END
```

(4) Method

The following expression is solved through forward-substitution and back-substitution:

$\boldsymbol{L}\,\boldsymbol{U}\,\boldsymbol{x} = \boldsymbol{P}\,\boldsymbol{b}$

A72-11-0101 VCGD, DVCGD

System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, diagonal storage format)
CALL VCGD (A, K, NW, N, NDLT, B, IPC, ITMAX, ISW, OMEGA, EPS, IGUSS, X, ITER, RZ, VW, IVW, ICON)

(1) Function

This routine solves linear equations

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

using the preconditioned conjugate gradient (CG) method, where an $n \times n$ normalized symmetric positive definite sparse matrix $\boldsymbol{A}$ must be used as a coefficient matrix.

The $n \times n$ coefficient matrix is normalized in such a way that the diagonal elements are 1. The non-zero elements other than the diagonal elements are stored using the diagonal storage format.

For normalization of linear equations that use symmetric positive definite sparse matrices as coefficient matrices, and for the diagonal storage format, see Part I, "Overview," Section 3.2.1.2, "Storage methods for symmetric positive definite sparse matrices." The diagonal storage format assumes that the non-zero elements of the coefficient matrix A lie on a limited number of diagonal vectors, parallel to the main diagonal vector.

This structure applies to linear equations arising from discretizing partial differential equations, particularly at lattices parallel to the defined boundaries of the region. This storage format is particularly efficient because the column vector number for each entry element in the coefficient array does not need to be stored. Only the distance from the main diagonal vector needs to be stored.

(2) Parameters

A..... Input. Two-dimensional array A (K, NW). Uses diagonal format to store non-zero elements of the coefficient matrix, which is a normalized symmetric positive definite sparse matrix.
For information about the diagonal storage format for normalized symmetric positive definite sparse matrices, see Part I, "Overview," Section 3.2.1.2, "Storage methods for symmetric positive definite sparse matrices," b., "The diagonal storage format for symmetric positive definite sparse matrices."

K..... Input. Size of adjustable dimension (≥ N) of array A.

NW..... Input. The size of the second dimension of array A. The number of diagonals that store coefficient matrix A using the diagonal storage format. Even number.

N..... Input. Order n of matrix $\boldsymbol{A}$.

NDLT..... Input. One-dimensional array NDLT (NW) indicating the offset from the main diagonal.

B..... Input. One-dimensional array of size n. Stores the constant vector specified in the right-hand-side term of the linear equations.

IPC..... Input. Preconditioner control information.
IPC = 1 No preconditioner.
IPC = 2 Neumann preconditioner.
IPC = 3 Preconditioner with incomplete Cholesky decomposition.

In this case, the user must specify OMEGA.
(See item (3), "Comments on use," b., 3).)

ITMAX..... Input. The upper limit of iterations (> 0).

ISW..... Input. Control information.
ISW = 1 Initial call.
ISW = 2 Subsequent calls. Must not be changed because the values of A, NDLT, VW, and IVW are used, which were set during the initial call.
(See item (3), "Comments on use," b., 1).)

OMEGA..... Input. Modification factor for incomplete Cholesky decomposition.
$0 \leq \text{OMEGA} \leq 1$
Used for IPC = 3.
(See item (3), "Comments on use," b., 3).)

EPS..... Input. Criterion value in judgment of convergency.
Judged as convergent when RZ < EPS.
If EPS ≤ 0, EPS is set to $\varepsilon \times |\boldsymbol{b}|$. $\varepsilon = 10^{-6}$ is used in double-precision routines, and $\varepsilon = 10^{-4}$ is used in single-precision routines.
(See item (3), "Comments on use," b., 2).)

IGUSS..... Input. Information about whether to start iterations from the approximate value of the solution vector specified in the array X.
IGUSS = 0 Approximation of the solution vector is not set.
IGUSS ≠ 0 Iteration computation starts from the approximate value of the solution vector specified in the array X.

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of the solution vector of linear equations.
Output. One-dimensional array of size n. Stores solution vector of linear equations.

ITER..... Output. Number of actually performed iterations.

RZ..... Output. Value of the square root of residuals *rz* after judgment of convergency.
(See item (3), "Comments on use," b., 2).)

VW..... Work area.

1) When IPC = 3
   One-dimensional array of size $K \times (NW + 6) + 2 \times NBAND$. *NBAND* is the size of the lower bandwidth or the upper bandwidth.
2) When IPC ≠ 3
   One-dimensional array of size $K \times 5 + 2 \times NBAND$. *NBAND* is the size of the lower bandwidth or the upper bandwidth.

IVW..... Work area. One-dimensional array of size $(K + 1) \times 4$.

ICON..... Output. Condition code.
See Table VCGD-1, "Condition codes."

Table VCGD-1 Condition codes

Code	Description	Processing
0	No error	
20001	Reached the maximum number of iterations.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
20003	Break down occurred.	
30003	ITMAX ≤ 0	Processing is stopped.
30005	K < N	
30006	Could not perform incomplete LL^T decomposition.	
30007	Pivot is negative.	
30089	NW is not an even number.	
30091	*NBAND* = 0	
30092	NW ≤ 0, *n* ≤ 0	
30093	K ≤ 0	
30096	OMEGA < 0, OMEGA > 1	
30097	IPC < 1, IPC > 3	
30102	Upper triangular part is not correctly stored.	
30103	Lower triangular part is not correctly stored.	
30104	The number of super-diagonals in the upper triangular part is not equal to the number of sub-diagonals in the lower triangular part.	
30105	ISW≠1, 2	
30200	| NDLT (*i*) | > *n* - 1 or NDLT (*i*) = 0	

(3) Comments on use

a. Subprograms used

SSL II: AMACH, UGCRI, UGULD, UGECD, UGFCD, UGINP, UGIPD, UGITB, UGITI, UGITN, UGITT, UGMVD, UGCGP, USSCP, USSPS, UGSD2, UGSD3, UGSTE, UGSWD, USVAD, USVCN, USVCP, USVSC, USVSU, USVUP, USVN1, USVN2, USVNM, UGWVD, URELT, MGSSL

b. Comments

1) When multiple sets of linear equations with the same coefficient matrix but different constant vectors are solved for IPC = 3, the solution on the first call is with ISW = 1, and solutions on subsequent calls are with ISW = 2. In subsequent calls, the result of the incomplete Cholesky decomposition obtained on the initial call is reused.

2) Judgment of convergency

The convergence of the solution obtained in the *n*-th iteration is assumed when

RZ = $\sqrt{(\boldsymbol{r}z)}$ < EPS

where .$\boldsymbol{r}$ is the residual vector defined by

$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_n$

$\boldsymbol{M}$ is the preconditioner matrix, and

$\boldsymbol{r}z = \boldsymbol{r}^T \boldsymbol{M}^{-1} \boldsymbol{r}$

3) Preconditioners

Two types of preconditioners and a no-preconditioner function are provided.

When elliptic partial differential equations are solved by discretization, it is effective to use a preconditioner based on an incomplete Cholesky decomposition.

If $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{N}$, the preconditioner $\boldsymbol{M}$ of the linear equation $(\boldsymbol{I} - \boldsymbol{N})\,\boldsymbol{x} = \boldsymbol{b}$ is as follows:

IPC = 1 No preconditioner $\boldsymbol{M} = \boldsymbol{I}$

IPC = 2 Neumann $\boldsymbol{M}^{-1} = (\boldsymbol{I} + \boldsymbol{N})$

IPC = 3 Incomplete Cholesky decomposition $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$

When IPC = 3, the user must specify a value for OMEGA ($0 \leq \mathrm{OMEGA} \leq 1$).

When OMEGA = 0, this is incomplete Cholesky decomposition. When OMEGA = 1, this is modified incomplete Cholesky decomposition.

For linear equations derived from discretizing partial differential equations, an optimal OMEGA value was found empirically to be in the range of 0.92 to 1.00.

c. Example of use

In this example, a linear equation is solved for a symmetric positive definite sparse matrix with *n* = 51,200 and the distance of the diagonal vector +5, -5.

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (N=51200,K=N+1)
       PARAMETER (NW=2,IWKS=4,N2=K+1)
       PARAMETER (NVW=K*(NW+6)+10)
       REAL*8 B(N),EPS,OMEGA,RZ,VW(NVW),X(N)
       INTEGER NDLT(NW)
       REAL*8 A(K,NW)
       INTEGER IVW(N2,IWKS)
C
C INITIALISE A
C
```

```
      CALL SET(A,NDLT,K,NW,N)
      ISHIFT=0
      DO 10 J=1,NW
      ISHIFT=MAX(ISHIFT,ABS(NDLT(J)))
   10 CONTINUE
C COMPUTE RHS SO AX=B SO WE KNOW SOLUTION X (X(I)=I)
      DO 30 I=1,N
   30 VW(I+ISHIFT)=I
C
C B=(A-E)*X+X
C
      CALL DVMVSD(A,K,NW,N,NDLT,ISHIFT,VW,B,ICON)
      DO 70 I=1,N
      B(I)=B(I)+VW(I+ISHIFT)
   70 CONTINUE
C
      ITMAX=8*SQRT(N+0.1)
      EPS=1D-10
      OMEGA=0D0
      ISW=1
      IGUSS=0
      DO 100 IPC=1,3
      IF(IPC.EQ.3) OMEGA=0.98
      CALL DVCGD(A,K,NW,N,NDLT,B,IPC,ITMAX,ISW,OMEGA,
     &           EPS,IGUSS,X,ITER,RZ,VW,IVW,ICON)
      IF(ICON.NE.0) WRITE(6,*)'ICON=',ICON
      IF(RZ.LE.EPS) WRITE(6,41)'CONVERGED. ACCURACY=',RZ
      IF(RZ.GT.EPS) WRITE(6,41)'FAILED. ACCURACY=',RZ
      WRITE(6,*)'X'
      DO 60 I=1,MIN(N,16),4
   60 WRITE(6,42) I,(X(M),M=I,I+3)
  100 CONTINUE
   42 FORMAT(1X,I3,4(1X,F20.10))
   41 FORMAT(A,2X,E10.3)
      STOP
      END

      SUBROUTINE SET(A,NDLT,K,NW,N)
      REAL*8 A(K,NW)
      INTEGER NDLT(NW)
      DO 10 J=1,NW
      DO 10 I=1,K
   10 A(I,J)=0D0
      N3=5
      NDLT(1)=N3
      NDLT(2)=-N3
      DO 20 I=1,N
      L=I
      IF(L.LE.N-N3) THEN
      A(I,1)=-0.25D0
      ENDIF
   20 CONTINUE
      DO 30 I=1,N
      L=I
      IF(L.GE.N3+1.AND.L.LE.N) THEN
      A(I,2)=-0.25D0
```

```
      ENDIF
   30 CONTINUE
      RETURN
      END
```

(4) Method

The standard conjugate gradient algorithm is used. (See [14].)

For the preconditioner method based on the incomplete Cholesky decomposition, see [29]. For the vectorization based on wavefront ordering, see [23].

(5) Acknowledgment

The author wishes to express thanks to the authors of ITPACK and NSPCG for permission to use the modified incomplete Cholesky decomposition and the wavefront ordering routine.

A72-12-0101 VCGE, DVCGE

System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, ELLPACK storage format)
CALL VCGE (A, K, NW, N, ICOL, B, IPC, ITMAX, ISW, OMEGA, EPS, IGUSS, X, ITER, RZ, VW, IVW, ICON)

(1) Function

This routine solves linear equations

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

using the preconditioned conjugate gradient (CG) method, where an $n \times n$ normalized symmetric positive definite sparse matrix $\boldsymbol{A}$ must be used as a coefficient matrix.

The $n \times n$ coefficient matrix is normalized in such a way that the diagonal elements are 1. The ELLPACK storage format is used to store the non-zero elements other than the diagonal elements.

For information about normalization of linear equations that use symmetric positive definite sparse matrices as coefficient matrices, see Part I, "Overview," Section 3.2.1.2, "Storage methods for symmetric positive definite sparse matrices."

(2) Parameters

A..... Input. Two-dimensional array A (K, NW). Stores non-zero elements of the coefficient matrix in the A (1:N, NW) part.
The reordering of the elements in the array A is performed for IPC = 3, when the upper triangular matrix part is not stored in A (*, 1:NW/2) and the lower triangular matrix part is not stored in A (*, NW/2 + 1:NW).
For information about the ELLPACK storage format for normalized symmetric positive definite sparse matrices, see Part I, "Overview," Section 3.2.1.2, "Storage methods for symmetric positive definite sparse matrices," a., "ELLPACK storage format for symmetric positive definite sparse matrices." (See item (3), "Comments on use," b., 1).)

K..... Input. Size of adjustable dimension (≥ N) of arrays A and ICOL.

NW..... Input. The size of the second dimension of array A.
When the maximum number of non-zero elements on the row vectors of the upper triangular matrix is *NSU* and the maximum number of non-zero elements on the row vectors of the lower triangular matrix is *NSL*,
then NW = 2 × max (*NSU*, *NSL*).
For details, see Part I, "Overview," Section 3.2.1.2, "Storage methods for symmetric positive definite sparse matrices," a., "ELLPACK storage format for symmetric positive definite sparse matrices."

N..... Input. Order n of matrix $\boldsymbol{A}$.

ICOL..... Input. A two-dimensional array ICOL (K, NW). The information about the column vectors to which the non-zero elements belong is stored in ICOL (1 : N, NW).

B..... Input. One-dimensional array of size n. Stores the constant vector specified in the right-hand-side term of the linear equation.

IPC..... Input. Preconditioner control information.
IPC = 1 No preconditioner.

IPC = 2 Neumann preconditioner.
IPC = 3 Preconditioner with incomplete Cholesky decomposition.
In this case, the user must specify OMEGA.
(See item (3), "Comments on use," b., 4).)

ITMAX..... Input. The upper limit of iterations (> 0).

ISW..... Input. Control information.
ISW = 1 Initial call.
ISW = 2 Subsequent calls. Must not be changed because the values of A, ICOL, VW and IVW, which were set during the initial call, are used.
(See item (3), "Comments on use," b., 2).)

OMEGA..... Input. Modification factor for incomplete Cholesky decomposition.
$0 \leq \text{OMEGA} \leq 1$

EPS..... Input. Criterion value in judgment of convergency.
Judged as convergent when RZ < EPS.
If EPS ≤ 0, $\varepsilon \times |\boldsymbol{b}|$ is set to EPS. $\varepsilon = 10^{-6}$ is used in double-precision routines, and
$\varepsilon = 10^{-4}$ is used in single-precision routines.
(See item (3), "Comments on use," b., 3).)

IGUSS..... Input. Sets control information about whether to start the iteration computation from the approximate value of the solution vector specified in array X.
IGUSS = 0 Approximation of the solution vector is not set.
IGUSS ≠ 0 Iteration starts from the approximate value of the solution vector specified in array X.

X..... Input. One-dimensional array of size *n*. Can specify the approximation vector of the solution for the linear equation.
Output. Stores the solution vector for the linear equation.

ITER..... Output. Number of iterations actually performed.

RZ..... Output. Value of the square root of residuals $\boldsymbol{r}z$ after judgment of convergency.
(See item (3), "Comments on use," b., 2).)

VW..... Work area.
1) If IPC = 3
One-dimensional array of size $K \times NW + 4 \times N$.
2) If IPC ≠ 3
One-dimensional array of size $N \times 3$.

IVW..... Work area.
1) If IPC = 3
One-dimensional array of size $K \times NW + N \times 4$.
2) If IPC ≠ 3
One-dimensional array of size $N \times 4$.

ICON..... Output. Condition code.
See Table VCGE-1, "Condition codes."

Table VCGE-1 Condition codes

Code	Description	Processing
0	No error.	
10000	A, ICOL elements are permuted to ***U***/***L*** format.	Processing continues.
20001	Reached the upper limit of iterations.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
20003	Break down occurred.	
30003	ITMAX ≤ 0	Processing is stopped.
30005	K < N	
30006	Could not perform incomplete LL^T decomposition.	
30007	Pivot is negative.	
30092	NW ≤ 0	
30093	K ≤ 0, N ≤ 0	
30096	OMEGA < 0, OMEGA > 1	
30097	IPC < 1, IPC > 3	
30098	ISW ≠ 1, 2	
30100	NW ≠ 2 × max (*NSU*, *NSL*)	
30104	Either the upper triangular part or the lower triangular part is not stored correctly.	
negative number	(-ICON)-th row has a non-zero diagonal element.	

(3) Comments on use

a. Subprograms used

SSL II: AMACH, UGECP, UGEUL, UGEFA, UGEMV, UGEPD, UGECG, UGEPM, UGEPV, UGESV, UGEWV, URELT, MGSSL

b. Comments

1) Sparse matrix is stored using the ELLPACK format storage method. (See [23] and [33]).

The upper triangular part is stored in A (*, 1:NW/2). The lower triangular part is stored in A (*, NW/2 + 1:NW), where NW = 2 × max (*NSU*, *NSL*).

If IPC ≠ 3 (when a preconditioner other than an incomplete Cholesky decomposition preconditioner is specified), a storage method is acceptable with conditions less stringent than those described in Part I, “Overview, “ Section 3.2.1.2, “Storage methods for symmetric positive definite sparse matrices,” a.,

"ELLPACK storage format for symmetric positive definite sparse matrices." A normalized symmetric positive definite sparse matrix without diagonal elements that is stored with the general sparse matrix ELLPACK storage method is also acceptable as input. In this case, it is not required that NW = 2 × max (*NSU*, *NSL*).

2) When multiple sets of linear equations with the same coefficient matrix but different constant vectors are solved for IPC = 3, the solution on the first call is with ISW = 1. Solutions on subsequent calls are with ISW = 2. In subsequent calls, the result of the incomplete Cholesky decomposition obtained on the initial call is reused.

3) Judgment of convergency

The convergence of the solution obtained in the *n*-th iteration is assumed when

$RZ = \sqrt{(\boldsymbol{r}z)} < EPS$

where $\boldsymbol{r}$ is the residual vector defined by

$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_n$

$\boldsymbol{M}$ is the preconditioner matrix, and

$\boldsymbol{r}z = \boldsymbol{r}^T \boldsymbol{M}^{-1} \boldsymbol{r}$

4) Preconditioners

Two types of preconditioner and the no-preconditioner functions are provided.

If $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{N}$, the preconditioner $\boldsymbol{M}$ of the linear equation $(\boldsymbol{I} - \boldsymbol{N})\ \boldsymbol{x} = \boldsymbol{b}$ is as follows:

IPC = 1 No preconditioner; $\boldsymbol{M} = \boldsymbol{I}$.

IPC = 2 Neumann; $\boldsymbol{M}^{-1} = (\boldsymbol{I} + \boldsymbol{N})$.

IPC = 3 Incomplete Cholesky decomposition; $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$.

The user must specify a value for OMEGA ($0 \leq OMEGA \leq 1$). When OMEGA = 0, this is incomplete Cholesky decomposition. When OMEGA = 1, this is modified incomplete Cholesky decomposition.

For linear equations derived from discretizing partial differential equations, an optimal OMEGA value was found empirically to be in the range of 0.92 to 1.00.

For IPC = 3, in order to optimize the preconditioners, the equations are permuted in the wavefront order.

c. Example of use

In this example, a linear equation is solved for a symmetric positive definite sparse matrix containing non-zero elements, where *n* = 51,200 and the distance from the diagonal elements is ± int (sqrt (*n* + 0.001)).

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (NW=2,N=51200,K=N+1)
       REAL*8 B(N),X(N),EPS,OMEGA,RZ,
      &       A(K,NW),VW(K*NW+4*N)
       INTEGER ICOL(K,NW),IVW(K*NW+4*N)
       WRITE(6,*) ' EXAMPLE DVCGE '
C INITIALISE A,ICOL
```

```
      CALL SET(A,ICOL,K,NW,N)
C GENERATE RHS B
      DO 10 I=1,N
   10 VW(I)=I
C COMPUTE RHS SO AX=B SO WE KNOW SOLUTION X (X(I)=I)
C
C B = (A-E)*X + E*X
      CALL DVMVSE(A,K,NW,N,ICOL,VW,B,ICON)
      PRINT*,'ERROR CODE =',ICON
      DO 20 I=1,N
      B(I)=B(I)+VW(I)
   20 CONTINUE
C
      ITMAX=4000
      EPS=1D-10
      ISW=1
      IGUSS=0
      DO 30 IPC=1,3
      IF(IPC.EQ.3)OMEGA=0.98
      CALL DVCGE(A,K,NW,N,ICOL,B,IPC,ITMAX,ISW,OMEGA
     &          ,EPS,IGUSS,X,ITER,RZ,VW,IVW,ICON)
C
      PRINT*,'ERROR CODE= ',ICON
      IF(RZ.LE.EPS) WRITE(6,41)'CONVERGED. ACCURACY=',RZ
      IF(RZ.GT.EPS) WRITE(6,41)'FAILED. ACCURACY=',RZ
      WRITE(6,*)'X'
      DO 60 I=1,MIN(N,16),4
   60 WRITE(6,42) I,(X(M),M=I,I+3)
   30 CONTINUE
   42 FORMAT(I3,4(F12.4))
   41 FORMAT(A,2X,E10.3)
      STOP
      END

      SUBROUTINE SET(A,ICOL,K,NW,N)
      INTEGER ICOL(K,NW)
      REAL*8 A(K,NW)
      N3=SQRT(N+0.001)
      DO 10 I=1,NW
      DO 10 J=1,N
      A(J,I)=0.0D0
      ICOL(J,I)=J
   10 CONTINUE
      DO 20 I=1,N-N3
      A(I,1)=-0.49D0
      ICOL(I,1)=I+N3
   20 CONTINUE
      DO 30 I=N3+1,N
      A(I,2)=-0.49D0
      ICOL(I,2)=I-N3
   30 CONTINUE
      RETURN
      END
```

(4) Method

The standard conjugate gradient algorithm is used. (See [14]). For information about the preconditioner based on the incomplete Cholesky decomposition, see [29]. For information about vectorization based on wavefront ordering, see [23].

(5) Acknowledgment

The author wishes to express thanks to the authors of ITPACK and NSPCG for permission to use the modified incomplete Cholesky decomposition preconditioner and wavefront ordering routine.

F16-15-0401 VCPF3, DVCPF3

Three-dimensional prime factor discrete complex Fourier transform
CALL VCPF3 (A, B, L, M, N, ISN, VW1, VW2, ICON)

(1) Function

This subroutine performs a discrete complex Fourier transform or its inverse transform using the prime factor fast Fourier transform (prime factor FFT). This subroutine is for when three-dimensional (where the size of each dimension is $N1$, $N2$, $N3$) complex time series data $\{x_{J1,J2,J3}\}$ is given. The size of each dimension must satisfy the following condition.

The size must be expressed by a product of a mutual prime factor p, selected from the following numbers:

factor p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)

Calling this subroutine with 1 entered for parameter N specifies a two-dimensional complex prime factor fast Fourier transform. Calling this subroutine with 1 entered for parameter N and 1 entered for parameter M specifies a one-dimensional complex prime factor fast Fourier transform.

1) Three-dimensional complex Fourier transform

By inputting $\{x_{J1,J2,J3}\}$ and performing the transform defined in (1.1), a three-dimensional Fourier transform seeks $\{N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}\}$.

$$N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}$$

$$= \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} \sum_{J3=0}^{N3-1} x_{J1,J2,J3} \omega_1^{-J1.K1} \omega_2^{-J2.K2} \omega_3^{-J3.K3} \qquad (1.1)$$

, $K1 = 0,1, ..., N1 - 1$

, $K2 = 0,1, ..., N2 - 1$

, $K3 = 0,1, ..., N3 - 1$

, $\omega_j = \exp(2\pi i/Nj)$, $j = 1, 2, 3$

2) Three-dimensional complex Fourier inverse transform

By inputting $\{\alpha_{K1,K2,K3}\}$ and performing the transform defined in (1.2), a three-dimensional Fourier inverse transform seeks $\{x_{J1,J2,J3}\}$.

$$x_{J1,J2,J3}$$

$$= \sum_{K1=0}^{N1-1} \sum_{K2=0}^{N2-1} \sum_{K3=0}^{N3-1} \alpha_{K1,K2,K3} \omega_1^{J1.K1} \omega_2^{J2.K2} \omega_3^{J3.K3} \qquad (1.2)$$

, $J1 = 0, 1, ..., N1 - 1$

, $J2 = 0, 1, ..., N2 - 1$

, $J3 = 0, 1, ..., N3 - 1$

, $\omega_j = \exp(2\pi i/Nj)$, $j = 1, 2, 3$

(2) Parameters

A..... Input. Real part of $\{x_{J1,J2,J3}\}$ or Fourier transformed $\{\alpha_{K1,K2,K3}\}$.
Output. Real part of Fourier transformed $\{\alpha_{K1,K2,K3}\}$ or inverse transformed $\{x_{J1,J2,J3}\}$.
A (L, M, N) is a three-dimensional array.
L, M, and N are the number of data items of the first, second, and third dimensions, respectively.

B..... Input. Imaginary part of $\{x_{J1,J2,J3}\}$ or Fourier transformed $\{\alpha_{K1,K2,K3}\}$.
Output. Imaginary part of Fourier transformed $\{\alpha_{K1,K2,K3}\}$ or inverse transformed $\{x_{J1,J2,J3}\}$.
B (L, M, N) is a three-dimensional array.
L, M, and N are the number of data items of the first, second, and third dimensions, respectively.

L..... Input. Number of data items in the first dimension.
$L \leq 5{,}040$.

M..... Input. Number of data items in the second dimension.
$M \leq 5{,}040$.

N..... Input. Number of data items in the third dimension.
$N \leq 5{,}040$.

ISN..... Input. Specify either transform or inverse transform.
If $ISN \geq 0$ (non-negative integer), then transform.
If $ISN < 0$ (negative integer), then inverse transform.

VW1..... Work area. Three-dimensional array with the same size as A or B.

VW2..... Work area. Three-dimensional array with the same size as A or B.

ICON..... Output. Condition code.
See Table VCPF3-1, "Condition codes."

Table VCPF3-1 Condition codes

Code	Description	Processing
0	No error	
20000	L, M, or N exceeded 5,040. Or the product of the mutual prime factor in {2, 3, 4, 5, 7, 8, 9, 16} could not be factored.	Processing is stopped.
30000	L, M, or N is zero or a negative value.	

(3) Comments on use

a. Subprograms used

SSL II: UTRSP, UPFT1, UPFT2, MGSSL

b. Comments

1) General definition of three-dimensional Fourier transform

The three-dimensional Fourier transform and its inverse transform are generally defined in (3.1) and (3.2).

$$\alpha_{K1,K2,K3} = \frac{1}{N1 \times N2 \times N3} \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} \sum_{J3=0}^{N3-1} x_{J1,J2,J3} \omega_1^{-J1.K1} \omega_2^{-J2.K2} \omega_3^{-J3.K3} \quad (3.1)$$

$$x_{J1,J2,J3} = \sum_{K1=0}^{N1=1} \sum_{K2=0}^{N2=1} \sum_{K3=0}^{N3=1} x_{K1,K2,K3} \omega_1^{J1,K1} \omega_2^{-J2,K2} \omega_3^{-J3,K3} \quad (3.2)$$

The subroutine looks for $\{N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}\}$ or $\{x_{J1,J2,J3}\}$ corresponding to the left-hand-side terms of (3.1) and (3.2), respectively. The user must normalize the results, if necessary. If the transform and inverse transform are executed by calling the subroutine consecutively without being normalized, each element of the input data is multiplied by $N1 \times N2 \times N3$, and then output.

2) Number of data items in each dimension

The number of data items is expressed as a product of a mutual prime factor *p*, selected from the numbers that follow.

The maximum number is $5 \times 7 \times 9 \times 16 = 5{,}040$.

factor *p* ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)

3) Data storage method

The real parts of complex data $\{x_{J1,J2,J3}\}$ and $\{N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}\}$ are stored in array A. The imaginary parts are stored in array B.

c. Example of use

In this example, complex time series data $\{x_{J1,J2,J3}\}$ of *N1*, *N2*, and *N3* terms are input, and a Fourier transform is performed. The results are used to perform a Fourier inverse transform to look for $\{x_{J1,J2,J3}\}$.

Here *N1* = 12, *N2* = 12, and *N3* = 12.

```
C      **EXAMPLE**
       DIMENSION A(12,12,12),B(12,12,12),NI(3)
       DIMENSION VW1(12,12,12),VW2(12,12,12)
       DATA NI/12,12,12/,L,M,N/12,12,12/
       READ(5,500) (((A(I,J,K),B(I,J,K),I=1,NI(1)),
      *             J=1,NI(2)),K=1,NI(3))
       WRITE(6,600) (NI(I),I=1,3),
      *             (((I,J,K,A(I,J,K),B(I,J,K),I=1,NI(1)),
      *             J=1,NI(2)),K=1,NI(3))
C      NORMAL TRANSFORM
       CALL VCPF3(A,B,L,M,N,1,VW1,VW2,ICON)
       WRITE(6,610) ICON
       IF(ICON.NE.0) STOP
C      INVERSE TRANSFORM
       CALL VCPF3(A,B,L,M,N,-1,VW1,VW2,ICON)
       NT=NI(1)*NI(2)*NI(3)
       DO 10 K=1,NI(3)
       DO 10 J=1,NI(2)
```

```
      DO 10 I=1,NI(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,NI(1)),
     *             J=1,NI(2)),K=1,NI(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
```

(4) Method

The three-dimensional real Fourier transform is performed by using the prime factor fast Fourier transform with the factorized mutually prime factor as the radix (prime factor FFT).

1) Three-dimensional transform

The three-dimensional transform defined in (1.1) can be performed in the order shown in (4.1) by simplifying common terms. The order for obtaining the sum of *J1*, *J2*, and *J3* can also be replaced.

$$N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}$$

$$= \sum_{J1=0}^{N1-1} \omega_1^{-J1.K1} \sum_{J2=0}^{N2-1} \omega_2^{-J2.K2} \sum_{J3=0}^{N3-1} x_{J1,J2,J3} \omega_3^{-J3.K3} \qquad (4.1)$$

In (4.1), $\sum_{J3}$ takes $N1 \times N2$ sets of one-dimensional transforms of *N3* data. $\sum_{J2}$ takes $N1 \times N3$ sets of one-dimensional transforms of *N2* data items. $\sum_{J1}$ takes $N2 \times N3$ sets of one-dimensional transforms of *N1* data items.

This routine applies the fast Fourier transform with the factorized mutually prime factor as the radix to perform a one-dimensional transform for each dimension.

2) Prime factor fast Fourier transform

The three-dimensional real Fourier transform can be calculated by performing a multi-set of one-dimensional Fourier transforms three times. The one-dimensional Fourier transforms are performed using the prime factor fast Fourier transform (prime factor FFT).

The following explains the one-dimensional prime factor fast Fourier transform.

$$C_K = \sum_{J=0}^{N-1} x_J \omega_N^{JK} \qquad \omega_N = \exp(-2\pi i/N)$$

K=0, ..., N - 1 (4.2)

When N is factored into two mutually prime factors, N_1 and N_2, the one-dimensional fast Fourier transform can be regarded as a two-dimensional fast Fourier transform.

$<>_N$ expresses the remainder of N. (,) expresses the greatest common divisor.

The appropriate K_i is present, and mappings (4.3) and (4.4) are determined.

N=N_1 N_2

$$j = <K_1 j_1 + K_2 j_2 >_N \quad (4.3)$$

$$k = <K_3 k_1 + K_4 k_2 >_N \quad (4.4)$$

$j, k = 0, ..., N-1$

$j_1, k_1 = 0, ..., N_1 - 1$

$j_2, k_2 = 0, ..., N_2 - 1$

The presence of this mapping is known from the Chinese remainder theorem as follows.

Assuming $N_1, N_2, ..., N_k$ are mutual primes, and $n_1, n_2, ..., n_k$ are random integers, and the solution of the simultaneous modulus expression

$$x \equiv n_i \pmod{N_i} \qquad i = 1, ..., k \quad (4.5)$$

$N = N_1 \times N_2 \times ... \times N_k$,

N is present as a unique modulus as follows:

$$x \equiv \sum_{i=0}^{k} n_i M_i q_i \pmod{N} \quad (4.6)$$

$$M_i = N/N_i, \qquad M_i q_i \equiv 1 \pmod{N_i} \quad i = 1, ..., k \quad (4.7)$$

By using this mapping, the one-dimensional prime factor fast Fourier transform in (4.2) is expanded.

$X_{j1j2} = x_{k1j1 + K2j2}$

$C_{k1,k2} = c_{K3k1 + K4k2}$

$$C_{k1k2} = \sum_{j1=0}^{N1-1} \sum_{j2=0}^{N2-1} X_{j1,j2} \omega_N^{K1K3j1k1} \omega_N^{K1K4j1k2} \omega_N^{K2K3j2k1} \omega_N^{K2K4j2k2} \quad (4.8)$$

By selecting K_i as follows, (4.2) becomes a two-dimensional fast Fourier transform.

$<K_1 K_3>_N = N_2$, $<K_2 K_4>_N = N_1$

$<K_1 K_4>_N = 0$, $<K_2 K_3>_N = 0$

$$C_{k1k2} = \sum_{j1=0}^{N1-1} \sum_{j2=0}^{N2-1} X_{j1,j2} \omega_{N1}^{j1k1} \omega_{N2}^{j2k2} \quad (4.9)$$

In addition, by factorizing the mutual prime factors, a multi-dimensional Fourier transform that has dimensions up to the number of factors can be obtained. The Fourier transform performs an in-place operation for the factorized factor to obtain an in-place algorithm.

The following is an example of a two-dimensional transform to illustrate permutation. This can easily be expanded into a multi-dimensional transform.

(j_1, j_2) and (k_1, k_2) are viewed as two-dimensional indices, but the mappings (4.3) and (4.4) transform them into one-dimensional indices.

$$j \rightarrow (j_1, j_2): \ (k_1, k_2) \rightarrow k \quad (4.10)$$

The generalized Chinese remainder theorem, below, is obtained. N is factorized to a mutual prime factor, and the following expression is obtained.

$N = N_1 \times N_2 \times ... \times N_m$

$n_i = <a_i n>_{Ni}$ and $(a_i, N_i) = 1$ (4.11)

$K_i = < N / N_i < (N / N_i) >_{N_i}^{-1} a_i >_N^{-1}$ (4.12)

Here, n can be expressed as unique, shown as follows.

$n = \sum_i K_i n_i$ (4.13)

$0 \le n \le N-1$

$0 \le n_i \le N_i-1 \qquad i = 1, ..., m$

By performing the mappings defined in (4.3) and (4.4), the actual one-dimensional positions of the two-dimensional indices (k_1, k_2) are determined from the relationship that follows:

$j = <K_1 k_1 + K_2 k_2>_N$

$k = <K_3 k_1 + K_4 k_2>_N$

By applying the generalized Chinese remainder theorem, it becomes clear that the two-dimensional (k_1, k_2) has the relationship in (4.14) because of mappings (4.3) and (4.4). The computed results after the two-dimensional transform can be permuted.

$J = <K_1 <a_1 k>_{N1} + K_2 <a_2 k>_{N2}>_N$

$K_1 = \alpha N_2$ $K_2 = \beta N_1$ adding the condition

$j = << K_1 a_1 k>_N + <K_2 a_2 k>_N >_N$

$= <(K_1 a_1 + K_2 a_2) k>_N$ (4.14)

For details on permutation and fast Fourier transform using each factor as a radix, see [6] and [45].

A72-21-0101 VCRD, DVCRD

System of linear equations with an unsymmetric or indefinite sparse matrix (MGCR method, diagonal storage format)
CALL VCRD (A, K, NDIAG, N, NOFST, B, ITMAX, EPS, IGUSS, NDIRV, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations

$\boldsymbol{Ax = b}$

using the modified generalized conjugate residuals (MGCR) method, where an $n \times n$ unsymmetric or indefinite sparse matrix is treated as a coefficient matrix.

The $n \times n$ coefficient matrix is stored with the diagonal storage format, using two arrays.

$\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 “Iterative linear equation solvers and Convergence,” in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, NDIAG). Stores coefficient matrix A is stored in A (1 : N, NDIAG).
For the diagonal storage format, see Part I, “Overview,” Section 3.2.1.1, “Storage methods for general sparse matrices,” b., “Diagonal storage format for general sparse matrices.”

K..... Input. Size of adjustable dimension of array A.

NDIAG..... Input. The number of diagonals in the coefficient matrix $\boldsymbol{A}$ that contain non-zero elements.
Size of second dimension of array A.

N..... Input. Order n of matrix $\boldsymbol{A}$.

NOFST..... Input. One-dimensional array NOFST (NDIAG). Stores the offset from the main diagonal corresponding to diagonals stored in array A. Superdiagonals have positive values. Subdiagonals have negative values.

B..... Input. One-dimensional array of size n. Stores the constant vector specified in the right-hand-side term of the linear equation.

ITMAX..... Input. The upper limit of iterations in the MGCR method (> 0).

EPS..... Input. Criterion value in judgment of convergency.
If EPS = 0 or less, EPS is set to 10^{-6} in double-precision routines. EPS is set to 10^{-4} in single-precision routines.
(See item (3), “Comments on use,” b., 1).)

IGUSS..... Input. Sets control information about whether to start the iteration computation from the approximate value of the solution vector specified in the array X.
IGUSS = 0 Approximate value of the solution vector is not set.

IGUSS ≠ 0 Iterative computation starts from the approximate value of the solution vector specified in the array X.

NDIRV..... Input. The number of search direction vectors used in the MGCR method (≥ 1).
Generally, a small number between 10 and 100.

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of the solution vector.
Output. The solution vector is stored.

ITER..... Output. Number of iterations actually performed using the MGCR method.

VW..... Work area. One-dimensional array of the size N × (NDIRV + 5) + NDIRV × (NDIRV + 1).

ICON..... Output. Condition code.
See Table VCRD-1, "Condition codes."

Table VCRD-1 Condition codes

Code	Description	Processing
0	No error	
20001	Reached the upper limit of iterations.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
30000	N < 1, K < 1, N > K or NDIAG < 1, ITMAX ≤ 0.	Processing is stopped.
30004	NDIRV<1	
32001	| NOFST (I) | > N-1	

(3) Comments on use

a. Subprograms used

SSL II: AMACH, URGWD, URIPA, URITI, URITT, URMDG, URMVD, URMGD, URRCI, URRAN, USSCP, URSTE, URSTI, USVAD, USVCN, USVCP, USVSC, USVSU, USVUP, USVN1, USVN2, USVNM, URELT, MGSSL

b. Comments

1) In the MGCR method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of matrix ***A*** and EPS.

2) Comments on use of the diagonal format

The elements of diagonals out of the coefficient matrix ***A*** must be set to zero.

There is no special restriction on the order of storing the diagonal columns in array A.

The advantage of this method lies in the fact that the matrix vector multiplication can be calculated without the use of indirect indices. The disadvantage is that

matrices without the diagonal structure cannot be stored efficiently with this method.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0, 1] \times [0, 1] \times [0, 1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of partial differential operator is described in Part I, "Overview," Section 3.2.2 "Discretization of partial differential operators and storage examples for them."

For INIT_MAT_DIAG, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." GET_BANDWIDTH_DIAG is a routine that estimates bandwidth. INIT_SOL is a routine that generates solution vectors to be sought with random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER MACH
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000,NDIRV = 50)
      PARAMETER (NX=20,NY=20,NZ=20,N=NX*NY*NZ)
      PARAMETER (NDIAG = 7,NVW=N*(NDIRV+5)+NDIRV*(NDIRV+1))
      REAL*8 A(K,NDIAG),X(N),B(N),VW(NVW),SOLEX(N)
      INTEGER NOFST(NDIAG)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)

      PRINT*,'EXPECTED SOLUSIONS'
      PRINT*,'X(1)= ',SOLEX(1),' X(N)=',SOLEX(N)
C
      PRINT *
      PRINT *,' MGCR METHOD'
      PRINT *,' DIAGONAL FORMAT'
C
      VA1=3D0
      VA2=1D0/3D0
      VA3=5D0
      VC=1.0
      XL=1.0
      YL=1.0
      ZL=1.0
C
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,A,NOFST,
     &     NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      CALL GET_BANDWIDTH_DIAG(NOFST,NDIAG,NBANDL,NBANDR)
      DO 110 I = 1,N
      VW(I+NBANDL) = SOLEX(I)
  110 CONTINUE
      CALL DVMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON)
      PRINT*,'DVMVSD ICON= ',ICON
      ITMAX=2000
      IGUSS=0
      EPS = 1D-10
      CALL DVCRD(A,K,NDIAG,N,NOFST,B,ITMAX,EPS,IGUSS,NDIRV,
     &           X,ITER,VW,ICON)
```

```
C
      PRINT* ,'ITER = ',ITER
      PRINT* ,'DVCRD ICON= ',ICON
      PRINT*, 'COMPUTED VALUES'
      PRINT*, 'X(1)= ',X(1),' X(N)= ',X(N)
      STOP
      END
```

(4) Method

For the MGCR method, see [25]. The algorithm is a modification of the generalized conjugate residuals method. The algorithm is robust and is always faster than the GMRES method. (See [34].)

A72-22-0101 VCRE, DVCRE

System of linear equations with an unsymmetric or indefinite sparse matrix (MGCR method, ELLPACK storage format)
CALL VCRE (A, K, IWIDT, N, ICOL, B, ITMAX, EPS, IGUSS, NDIRV, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations

$\boldsymbol{Ax = b}$

using the modified generalized conjugate residuals (MGCR) method, where an $n \times n$ asymmetrical or indefinite sparse matrix is treated as a coefficient matrix.

The $n \times n$ coefficient matrix is stored with the ELLPACK storage format using two arrays.

$\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 “Iterative linear equation solvers and Convergence,” in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, IWIDT).
For the ELLPACK storage format, see Part I, “Overview,” Section 3.2.1.1, “Storage method for general sparse matrices.”

K..... Input. Size of adjustable dimension ($\geq n$) of A and ICOL.

IWIDT..... Input. The maximum number of non-zero elements in row vector direction on the coefficient matrix A. Size of the second dimension of ICOL and A.

N..... Input. Order n of matrix $\boldsymbol{A}$.

ICOL..... Input. Stores the column indices of the elements stored in the array A using the ELLPACK format, indicating which column vectors the corresponding elements in the array A belong to.
Two-dimensional array of size ICOL (K, IWIDT).

B..... Input. One-dimensional array of size n. Stores the constant vector specified in the right-hand-side term of the linear equation in B.

ITMAX..... Input. The upper limit of iterations in the MGCR method (> 0).

EPS..... Input. Criterion value in judgment of convergency.
If EPS = 0.0 or less, EPS is set to 10^{-6} in double-precision routines. EPS is set to 10^{-4} in single-precision routines.
(See item (3), “Comments on use,” b., “Comments,” 1).)

IGUSS..... Input. Control information about whether to start iteration computation from the approximate value of the solution vector specified in the array X.
IGUSS = 0 Approximate value of the solution vector is not set.
IGUSS ≠ 0 Iteration computation starts from the approximate value of the solution vector specified in the array X.

NDIRV..... Input. The number of search direction vectors used in the MGCR method (≥ 1).
Generally, a small number between 10 and 100.

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of the solution vector.
Output. The solution vector is stored.

ITER..... Output. Number of iterations actually performed using the MGCR method.

VW..... Work area. One-dimensional array of the size $N \times (NDIRV + 5) + NDIRV \times (NDIRV + 1)$.

ICON..... Output. Condition code.
See Table VCRE-1, "Condition codes."

Table VCRE-1 Condition codes

Code	Description	Processing
0	No error	
20001	Reached the maximum number of iterations.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
30000	K < 1, IWIDT < 0, N < 1 or N > K, ITMAX ≤ 0	Processing is stopped.
30004	NDIRV<1	

(3) Comments on use

a. Subprograms used

SSL II: AMACH, URIPA, URITI, URITT, URMEG, URMVE, URMGE, URRCI, URRAN, USSCP, URSTE, URSTI, USVAD, USVCN, USVCP, USVSC, USVSU, USVUP, USVN1, USVN2, USVNM, URELT, MGSSL

b. Comments

1) In the MGCR method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged.

The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of matrix A and EPS.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0, 1] \times [0, 1] \times [0, 1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of partial differential operator is described in Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." For INIT_MAT_ELL, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." INIT_SOL is the routine that generates the solution vectors to be sought in random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000,NDIRV = 50)
      PARAMETER (IWIDT = 7,NX=20,NY=20,NZ=20,N=NX*NY*NZ)
      PARAMETER (NVW=N*(NDIRV+5)+NDIRV*(NDIRV+1))
      REAL*8 A(K,IWIDT),X(N),B(N),VW(NVW),SOLEX(N)
      INTEGER ICOL(K,IWIDT)
C
      XL=1.0
      YL=1.0
      ZL=1.0
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUTION'
      PRINT*,'X(1)= ',SOLEX(1),' X(N)= ',SOLEX(N)
      PRINT*,'    MGCR METHOD'
      PRINT*,'    ELLPACK FORMAT'
C
      VA1=3D0
      VA2=1D0/3D0
      VA3=5D0
      VC=5D0
C
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL,NX,NY,NZ,
     &    XL,YL,ZL,IWIDT,N,K)
      CALL DVMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON)
      PRINT*,'DVMVSE ICON = ',ICON
      IGUSS =0
      EPS = 1D-10
      ITMAX=800
      CALL DVCRE(A,K,IWIDT,N,ICOL,B,ITMAX,EPS,IGUSS,NDIRV
     &     ,X,ITER,VW,ICON)
C
      PRINT*,'DVCRE ICON = ',ICON
      PRINT*,'COMPUTED VALUE'
      PRINT*,'X(1)= ',X(1),' X(N)= ',X(N)
      STOP
      END
```

(4) Method

For the MGCR method, see [25]. The algorithm is a modification of the generalized conjugate residuals method. The algorithm is robust and is always faster than the GMRES method. (See [34]).

B71-13-0101 VHEVP,DVHEVP

Eigenvalues and eigenvectors of a Hermitian matrix (tridiagonalization, multisection method, and inverse iteration)
CALL VHEVP (AR, AI, K, N, NF, NL, IVEC, ETOL, CTOL, NEV, E, MAXNE, M, EVR, EVI, VW, IW, ICON)

(1) Function

This subroutine calculates specified eigenvalues and, optionally, eigenvectors of an *n*-dimensional Hermitian matrix.

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \qquad (1.1)$$

(2) Parameters

AR ..... Input. The real part of Hermitian matrix *A* whose eigenvalues and eigenvectors are to be calculated is stored in AR(1:N,1:N).

Two-dimensional array AR(K,N) .

AI ..... Input. The imaginary part of Hermitian matrix ***A*** whose eigenvalues and eigenvectors are to be calculated is stored in AI(1:N,1:N).

Two-dimensional array AI(K,N) .

K ..... Input. Size of first-dimension of array AR or of array AI. (K ≥ N)

N ..... Input. Order *n* of Hermitian matrix ***A***

NF ..... Input. Number assigned to the first eigenvalue to be acquired by numbering eigenvalues in ascending order. (Multiple eigenvalues are numbered so that one number is assigned to one eigenvalue.)

NL ..... Input. Number assigned to the last eigenvalue to be acquired by numbering eigenvalues in ascending order. (Multiple eigenvalues are numbered so that one number is assigned to one eigenvalue.)

IVEC ..... Input. Control information.

IVEC=1: Both the eigenvalues and the corresponding eigenvectors are sought.

IVEC≠1: Only the eigenvalues are sought.

ETOL ..... Input. A criterion value required to determine whether an eigenvalue is distinct or numerically multiple based on expression (3.1). The default value is 3.0D-16 for double precision (2.0D-7 for single precision) when this value is set to less than.

(See 1) in b, “Notes,” in (3), “Comments on use.”)

CTOL ..... Input. A criterion value required to determine whether adjacent eigenvalues are approximately multiple i.e. clustered according to expression (3.1). CTOL ≥ ETOL
When CTOL is less than ETOL, CTOL = ETOL is set.
(See 1) in b, “Notes,” in (3), “Comments on use.”)

NEV ..... Output. Number of eigenvalues calculated.

Details are given below.

NEV (1) indicates the number of distinct eigenvalues.
NEV (2) indicates the number of distinct clusters.
NEV (3) indicates the total number of eigenvalues including multiplicities.
One-dimensional array NEV (3).

E ..... Output. The eigenvalues calculated are stored in E(1:NEV(3)).

One-dimensional array E(MAXNE)

MAXNE .... Input. Maximum number of eigenvalues that can be computed.
Size of the array E.

When NEV(3) is greater than MAXNE, eigenvectors cannot be computed.

(See 2) in b, "Notes," in (3), "Comments on use.")

M ..... Output. Information about the multiplicity of the computed eigenvalues.
M (*i*, 1) indicates the multiplicity of the *i*-th eigenvalue λ_i. M (*i*, 2) indicates the size of the *i*-th cluster of eigenvalues.

(See 1) in b, "Notes," in (3), "Comments on use.")

Two-dimensional array M(MAXNE,2).

EVR ..... Output. When IVEC = 1, the real part of the eigenvectors corresponding to the eigenvalues is stored in EVR.

The eigenvectors are stored in EVR(1:N,1:NEV(3)).

Two-dimensional array EVR(K,MAXNE).

EVI ..... Output. When IVEC = 1, the imaginary part of the eigenvectors corresponding to the eigenvalues is stored in EVI.

The eigenvectors are stored in EVI(1:N,1:NEV(3)).

Two-dimensional array EVI(K,MAXNE).

VW ..... Work area. One-dimensional array of size $17 \times K$.

IW ..... Work area. One-dimensional array of size $9 \times$ MAXNE+128.

ICON ..... Output. Condition code.

See Table VHEVP-1.

Table VHEVP-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	During calculation of clustered eigenvalues, the total number of eigenvalues exceeded MAXNE.	Processing is discontinued. The eigenvectors cannot be calculated, but the different eigenvalues themselves are already calculated. (See 2) in b, "Notes," in (3), "Comments on use.")
30000	NF < 1, NL > N, NL < NF, K < N, N < 1, or MAXNE < NL - NF + 1.	Processing is discontinued.
30100	The input matrix may not be a Hermitian matrix.	

(3) Comments on use

a. Subprograms used

SSLII ..... UHEVP, UIBBS, UIBFC, UIBFE, UIBSL, UITBS, UITFC, UITFE, UITSL, UTDEX, UTDEY, UTMLS, UTRZB, UTRZV, UZRDM, MGSSL, UMGSL, UMGSL2

b. Notes

1) This routine pays special attention to a clustered eigenvalue.

With ε is equal to ETOL, suppose that the eigenvalues $\lambda_j, j = s, s+1, \ldots$, and $s+k$ ($k \geq 0$) are such that

$$\frac{|\lambda_i - \lambda_{i-1}|}{1+\max(|\lambda_{i-1}|,|\lambda_i|)} \leq \varepsilon \qquad (3.1)$$

While (3.4) is not satisfied for $i = s$-1 and $i = s + k + 1$, then eigenvalues λ_j, $j = s - 1, s, \ldots, s + k$ are considered to be identical, i.e., a single multiple eigenvalue of multiplicity $k + 1$.

The default value of ETOL is 3.0D-16 for double precision (2.0D-7 for single-precision). Using this value, eigenvalues are refined to machine precision.

When (3.1) is not satisfied for ε = ETOL, λ_{i-1} and λ_i are assumed to be distinct eigenvalues.

If (3.1) holds with ε = CTOL (but not with ε = ETOL) for eigenvalues λ_m, $m=t, t+1, \ldots, t+k$ but not for λ_{t-1} and λ_{t+k+1}, these eigenvalues are considered to be approximately multiple, i.e. clustered, though distinct (not numerically multiple). In order to obtain an invariant subspace, eigenvectors corresponding to clustered eigenvalues are computed using orthogonal starting vectors and are reorthogonalized. Of course CTOL ≥ ETOL; if this condition is not satisfied, CTOL is set to be equal to ETOL.

2) Assume *r* eigenvalues are requested. Note that if the first or last requested eigenvalue has a multiplicity greater than 1 then more than *r* eigenvalues, are obtained. The corresponding eigenvectors can be computed only when the corresponding eigenvector storage area is sufficient.

   The maximum number of computable eigenvalues can be specified in MAXNE. If the total number of eigenvalues exceeds MAXNE, ICON = 20000 is returned. The corresponding eigenvectors cannot be computed. In this case, the eigenvalues are returned, but they are not stored repeatedly according to multiplicities.

   When all eigenvalues are distinct, it is sufficient to set MAXNE = NL-NF+1. When the total number of eigenvalues to be sought exceeds MAXNE, the necessary value for MAXNE for seeking eigenvalues again is returned in NEV(3).

3) This routine is faster than HEIG2.

c. Example

This example calculates the specified eigenvalues and eigenvectors of a Hermitian matrix.

```
C      ** EXAMPLE PROGRAM **
       IMPLICIT REAL*8(A-H,O-Z)
       PARAMETER (K=512,N=K,NF=1,NL=28,MAXNE=NL-NF+1)
       PARAMETER (NVW=19*K,NIW=9*MAXNE+128)
       PARAMETER (ETOL=1.0D-14,CTOL=5.0D-12)
       REAL*8    AR(K,N),AI(K,N)
       REAL*8    E(MAXNE), EVR(K,MAXNE), EVI(K,MAXNE)
       INTEGER   NEV(3), M(MAXNE,2)
       REAL*8    VW(NVW)
       INTEGER   IW(NIW)
       IVEC=1

       WRITE (*,*)' Number of data points = ',N
       WRITE (*,*)' Parameter k =',K
       WRITE (*,*)' Eigenvalue calculation tolerance = ',ETOL
       WRITE (*,*)' Cluster tolerance =',CTOL
       WRITE (*,*)' First eigenvalue to be found is ',NF
       WRITE (*,*)' Last eigenvalue to be found is ',NL

C Set up real and imaginary parts of matrix in AR and AI
       DO 100 J=1,N
          DO 98 I=1,N
             AR(I,J) = DBLE(I+J)/DBLE(N)
             IF (I.EQ.J) THEN
                AI(I,J) = 0.0D0
                AR(I,J) = DBLE(J)
              ELSE
                 AI(I,J) = DBLE(I*J)/DBLE(N*N)
              ENDIF
 98       CONTINUE
 100   CONTINUE

       DO 99 J=1,N
           DO 99 I=1,N
               IF (I.GT.J) AI(I,J) = -AI(I,J)
```

```
 99   CONTINUE

C Call complex eigensolver

      CALL DVHEVP(AR,AI,K,N,NF,NL,IVEC,ETOL,CTOL,NEV,
     *            E,MAXNE,M,EVR,EVI,VW,IW,ICON)

      WRITE (*,*)' ************************************'
      WRITE (*,*)'      VHEVP OUTPUT'
      IF(ICON.NE.0) THEN
         WRITE (*,*)' Error parameter icon = ',ICON,
     *              ' VHEVP failed'
         GOTO 5000
      ENDIF
      WRITE (*,*)' Number of Hermitian eigenvalues'
      WRITE (*,*) NEV(3)
      WRITE (*,*)' Eigenvaluse of complex Hermitian matrix'
      WRITE (*,*)(E(I),I=1,NEV(3))
 5000 STOP
      END
```

(4) Method

The $n \times n$ Hermitian matrix A = AR + *i*AI must satisfy $AR = AR^T$ and $AI = -AI^T$

The Householder method is used to reduce the Hermitian matrix to a Hermitian tridiagonal matrix. Then, the diagonal unitary transformation is applied to further reduce the matrix to a real tridiagonal matrix. For details of the Householder calculations, see [44] in Appendix B, "References," or see "TRIDH" in *Fujitsu SSL II User's Guide*.

The eigenvalues and eigenvectors of the tridiagonal matrix are calculated using techniques of multisectioning and inverse iteration (see "VTDEV" and [32] in Appendix B, "References").

In the final step, the eigenvectors of the Hermitian matrix are constructed from the eigenvectors of the tridiagonal matrix.

B71-11-0101 VLAND, DVLAND

Eigenvalues and eigenvectors of a real symmetric sparse matrix (Lanczos method, diagonal storage format)
CALL VLAND (A, K, NDIAG, N, NOFST, IVEC, IX, EPS, NMIN, NMAX, NLMIN, NLMAX, KR, MAXC, E, INDX, NCMIN, NCMAX, EV, WV, IW, ICON)

(1) Function

This routine computes a few of the largest and/or smallest eigenvalues and corresponding eigenvectors in a large-scale real symmetric sparse matrix *A* using the Lanczos method.

(2) Parameters

A..... Input. Non-zero elements of the real symmetric sparse matrix.
Uses the diagonal storage format for general sparse matrices to store diagonals of *A* containing non-zero elements.
Two-dimensional array A (1:N, 1:NDIAG)
For the diagonal storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices," b., "Diagonal Storage Format for General Sparse Matrices."

K..... Input. Size of the first dimension of array A (≥ N)

NDIAG..... Input. The number of diagonals of coefficient matrix *A* including non-zero elements.

N..... Input. Order *n* of matrix *A*

NOFST..... Input. Stores the offset from the main diagonal of the corresponding non-zero diagonal stored in array A. Superdiagonals have positive offsets. Subdiagonals have negative offsets.

IVEC..... Input. Control information indicating whether an initial vector is specified in EV (1:N,1).
IVEC=1 The vector stored in EV (1:N,1) is used as the initial vector.
IVEC≠1 The initial vector is generated randomly.
(See item (3), "Comments on use," b., 1).)

IX..... Input. Seed value used to generate a random number sequence when an initial vector is generated randomly for IVEC ≠ 1. An integer value from 1 to 100,000.
(See item (3), "Comments on use," b., 1).)

EPS..... Input. Tolerance used to decide whether the computed eigenpair (λ_i , V_i) is to be accepted. If EPS is less than or equal to the default value 0, 10^{-6} (10^{-3} for single precision) then it is set to the default value for double precision.
(See item (3), "Comments on use," b., 3).)

NMIN..... Input. The number of smallest eigenvalues and corresponding eigenvectors to be computed (≥ 0). Smaller number. May be 0 if NMAX ≥ 1.

NMAX..... Input. The number of largest eigenvalues and corresponding eigenvectors to be computed (≥ 0). Smaller number. May be 0 if NMIN ≥ 1.

NLMIN..... Input. The number of eigenvalues to be used in the search for the NMIN smallest eigenvalues. (≥ NMIN)
2 × NMIN in many cases.
(See item (3), "Comments on use, "b., 5).)

NLMAX..... Input. The number of eigenvalues to be used in the search for the NMAX largest eigenvalues.
($\geq$ NMAX)
$2 \times$ NMAX in many cases.
(See item (3), "Comments on use, "b., 5).)

KR..... Input. The maximum on the dimension of the Krylov subspace generated in the Lanczos method. ($\geq$ NLMIN + NLMAX) NLMIN + NLMAX in many cases.
(See item (3), "Comments on use, "b., 4).)

MAXC..... Input. The maximum number of eigenvalues in a cluster. For example, 10.
(See item (3), "Comments on use, "b., 2).)

E ..... Output. One-dimensional array of E (NEVL).
The largest and smallest eigenvalues are stored in ascending order using the indirect index list INDX.
NEVL = NLMIN + NLMAX.
The smallest eigenvalues are stored in E (INDX (1:NCMIN)); the largest ones in E (INDX (NEV - NCMAX:NEV). NEV = NMIN + NMAX.

INDX..... Output. One-dimensional array INDX (NEV). Stores indirect indices of arrays E and EV.
The eigenvector corresponding to eigenvalue E (INDX (I)) is stored in EV (1:N, INDX(I)).
I = 1, ..., NEV, with NEV = NMIN + NMAX.

NCMIN..... Output. The number of smallest eigenvalues and corresponding eigenvectors which have been computed.

NCMAX..... Output. The number of largest eigenvalues and corresponding eigenvectors which have been computed.

EV..... Input. When IVEC = 1 an initial vector is stored in EV (1:N, 1) in EV.
Output. Computed eigenvectors are stored. Eigenvectors can be referred using the indirect index list INDX as eigenvalues.
Two-dimensional array EV (K, NEVL), NEVL = NLMIN + NLMAX.

WV..... Work area. One-dimensional array of the size (MAXC + MNL) $\times$ (KR + 2) + MD $\times$ (KR + 1) + 7 $\times$ K + 14 $\times$ (KR + 1).
Here, MNL = MAX (NLMIN, NLMAX), MD = NLMIN + NLMAX.

IW..... Work area. One-dimensional array of the size 11 $\times$ (MAXC + MNL) + MD + 128.
Here, MNL = MAX (NLMIN, NLMAX), MD = NLMIN + NLMAX.

ICON..... Output. Condition code
See Table VLAND-1, "Condition codes."

Table VLAND-1 Condition codes

Code	Description	Processing
0	No error	-
20000	The total number of eigenvalues in a cluster exceeded MAXC. Eigenvectors cannot be computed.	Processing is stopped.
30000	N<1, N>K, NDIAG<1, IX<1, IX>100000, NLMIN<NMIN, NLMAX<NMAX, NMIN<0, NMAX<0, NMIN=NMAX=0	Processing is stopped.
30004	KR<NLMIN+NLMAX	
32001	|NOFST (I)|>N-1	
39001	The initial vector is 0 or near 0.	
39006	The input matrix is not symmetric.	

(3) Comments on use

a. Subprograms used

SSLII: UZBBM, UZGSD, UZGUD, UZGBD, UZISE, UZLCD, UZLPD, UZMLS, UZSRZ, UZSTE, UZS3D, UZTDC, UZTDE, USMN1, USSPS, UIBBS, UIBFC, UIBFE, UIBSL, UITBS, UITFC, UITFE, UITSL, AMACH, URIPA, URMVD, URPER, URPRE, URPFP, URPIP, UZRDM, USSCP, URSTE, USVAD, USVCN, USVCP, USVSC, USVSU, USVUP, USVN1, USVN2, USVNM, MGSSL

b. Comments

1) The Lanczos method is not a deterministic procedure, and hence is not as robust as, for example, the method based on the tridiagonalization by Householder reduction.

The results obtained using the Lanczos method depends on choice of initial vector. If the initial vector contains large components in the directions of the requested eigenvectors, then good approximations to the requested eigenvalues and eigenvectors will be computed. If these components are small or absent then the desired eigenpairs may not be obtained; however, the returned value are good approximations to some eigenpairs of the matrix *A*.

In most cases, a good initial vector will not be known a priori and in these instances the initial vector is generated randomly.

2) A cluster is a set of very close eigenvalues for which the distance (relative to eigenvalue magnitude) between adjacent eigenvalues of order machine epsilon.

3) When the eigen pair (λ_i , V_i) satisfies $\|AV_i - \lambda V_i\| \leq n\varepsilon|\lambda_i|$, it is accepted as an eigenvalue and eigenvector of matrix *A*. Otherwise, this pair is rejected.

Here, ε = EPS, n = KR, and KR indicate the dimension of the Krylov subspace.

The dependence on the value of EPS is rather mild. However, if EPS is too large, the computed eigenvalues and eigenvectors may not have high accuracy.

4) Making KR larger enables the user to obtain better approximate eigenvalues and eigenvectors; however, higher memory and computational cost are entailed, KR should therefore be chosen as small as possible. In some cases, it is impossible to choose KR smaller than N (for example, the one-dimensional discrete Laplacian). KR should exceed N. When KR is equal to N, this routines works correctly but may be unacceptably slow.

   The quality of the computed eigenvalues and eigenvectors depends considerably on the dimension KR of the Krylov sub space and the initial vector.

5) In the Lanczos method, - spurious eigenvalues and eigenvectors - not belonging to the original matrix A may be obtained. These values are rejected. The number of eigenvalues and eigenvectors used in the search, must therefore be increased. These values should be determined carefully.

   In most cases, NLMIN = NMIN, NLMAX = NMAX are insufficient NLMIN and NLMAX values.

   NMLIN = 2 × NMIN, NLMAX = 2 × NMAX are generally suffice.

c. Example of use

In this example, we find the three smallest and largest eigenvalues and corresponding eigenvectors for the matrix *A* resulting from the finite difference approximation of the following elliptic operator *L*.

$Lu = -\Delta u + a\nabla u + u$

With zero boundary conditions on a cube where $a = (a_1, a_2, a_3)$ with a_1, a_2 and a_3 constants.

(The matrix *A* is generated with init_mat_diag and stored using the diagonal storage format.)

```
C     ** EXAMPLE PROGRAM **
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER REP
      PARAMETER (REP = 2)
      PARAMETER (NX = 20,NY = 20,NZ = 20)
      PARAMETER (K = NX*NY*NZ, N = K)
      PARAMETER (NMAX = 3, NMIN = 3)
      PARAMETER (IVEC=0,IX=123)
      PARAMETER (EPS1 = 1D-6)
      PARAMETER (NLMIN = 2*NMIN, NLMAX = 2*NMAX)
      PARAMETER (MD = NLMIN+NLMAX,NEVL=MD)
      PARAMETER (MNL = NLMIN)    ! MNL = MAX(NLMIN,NLMAX)
      PARAMETER (NEV = NMIN+NMAX)
      PARAMETER (KR = (NX*NY*NZ)/REP)
      PARAMETER (NDIAG = 7)
      PARAMETER (MAXC = 10)
      PARAMETER (NWV = (MAXC+MNL)*(KR+2)+MD*(KR+1)+
     &                 7*K+14*(KR+1))
      PARAMETER (NIW = 11*(MAXC+MNL)+MD+128)

      REAL*8 A(K,NDIAG),EV(K,NEVL),E(NEVL),VW(NWV)
      INTEGER NOFST(NDIAG),INDX(NEV),IW(NIW)

C     Initialize matrix A
      CALL MAT_DIAG(0D0,0D0,0D0,0D0,2D0,-1D0,A,NOFST,
     &           NX,NY,NZ,NDIAG,K)
```

```
      EPS = EPS1

      CALL DVLAND(A,K,NDIAG,N,NOFST,IVEC,IX,EPS,NMIN,
     &           NMAX,NLMIN,NLMAX,KR,MAXC,E,INDX,NCMIN,
     &           NCMAX,EV,VW,IW,ICON)

      IF (ICON.LT. 20000) THEN
         PRINT*,' Real eigenvalues (MIN:MAX)'
         WRITE (*,901) (E(INDX(I)),I=1,NCMIN)
         WRITE (*,901) (E(INDX(I)),I=NEV-NCMAX+1,NEV)
      ENDIF

  901 FORMAT(D23.16)
      STOP
      END
```

(4) Method

For the Lanczos method, see [14] and the bibliography therein, also [8]. The algorithm used for this routine generates a tridiagonal matrix *T* of size less than (or equal) to that of the matrix *A*. Next the eigenvalues and eigenvectors of this tridiagonal matrix are computed using a multisection Sturm count procedure and inverse iteration, respectively. (See VTDEV.) Finally the eigenvectors of the matrix *A* are recovered from those of *T* using the Krylov subspace basic vectors generated by the Lanczos process.

A53-11-0301 VLBX, DVLBX

System of linear equations with a banded real matrix (Gaussian elimination)
CALL VLBX (A, N, NH1, NH2, B, EPSZ, ISW, IS, IP, VW, ICON)

(1) Function

This routine solves real coefficient linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

using the Gaussian elimination method, where $\boldsymbol{A}$ is an $n \times n$ banded matrix with the lower bandwidth h_1 and upper bandwidth h_2.

$\boldsymbol{b}$ is an n-dimensional real constant vector. $\boldsymbol{x}$ is an n-dimensional solution vector.

$n > h_1 \geq 0$, $n > h_2 \geq 0$ must be obtained.

(2) Parameters

A..... Input. One-dimensional array of size $(2 \times h_1 + h_2 + 1) \times n$ that stores the banded coefficient matrix $\boldsymbol{A}$.
For the storage method for matrix $\boldsymbol{A}$, see Figure VLBX-1, "Storage method for banded matrix in array A."
Output. Stores the LU decomposed $\boldsymbol{L}$ and $\boldsymbol{U}$. The storage method is the same as the input storage method.
For the storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{U}$, see Figure VLBX-2, "Storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ in array A."

N..... Input. Order of matrix $\boldsymbol{A}$.

NH1..... Input. Lower bandwidth h_1 of matrix $\boldsymbol{A}$.

NH2..... Input. Upper bandwidth h_2 of matrix $\boldsymbol{A}$.

B..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

EPSZ..... Input. Value of pivot judgment of relative zero (≥ 0.0). When EPSZ = 0.0, a standard value is selected.
(See item (3), "Comments on use," b., 1).)

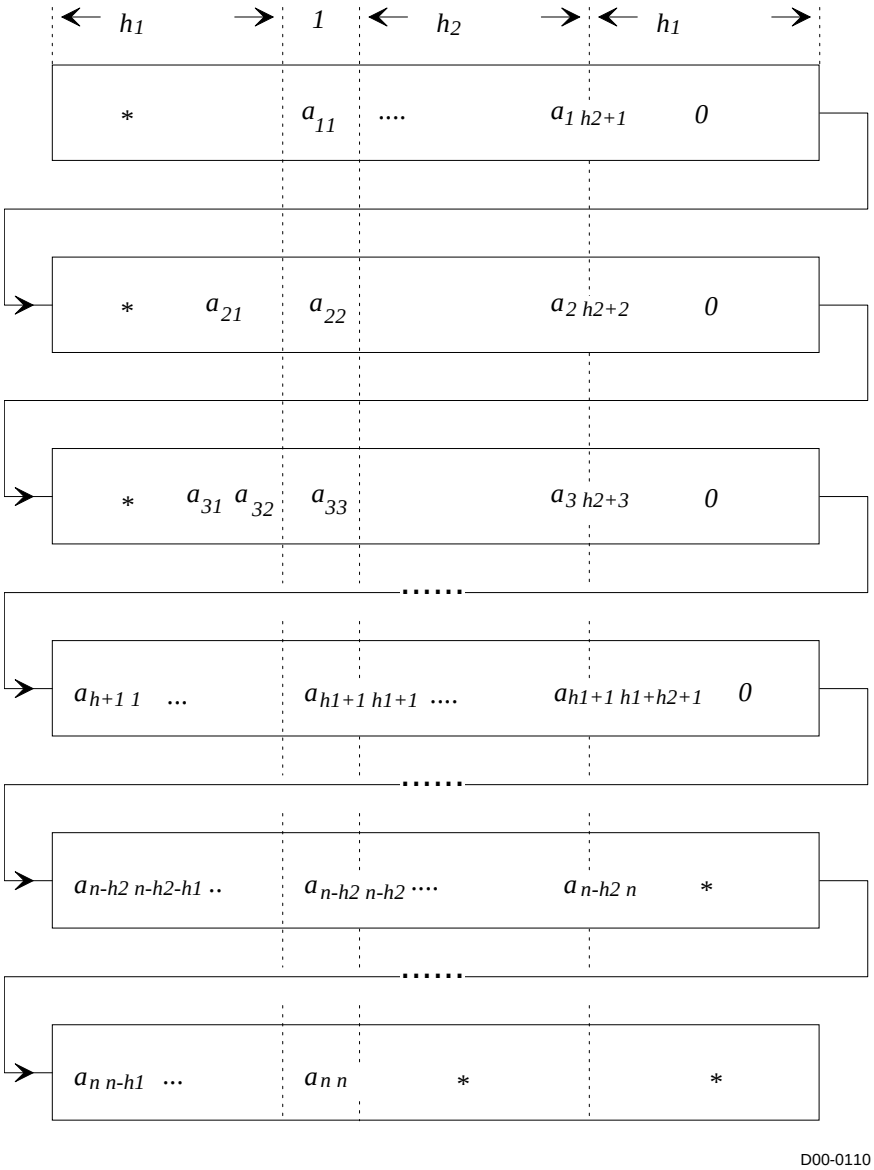
ISW..... Input. Control information.
When solving k ($k \geq 1$) equation sets with the same coefficient matrix, set ISW as follows.
If ISW = 1, first-set equations are solved.
If ISW = 2, second-set and subsequent equations are solved.
All parameters other than B whose value is changed to the value of a new constant vector $\boldsymbol{b}$, should be used unchanged.
(See item (3), "Comments on use," b., 2).)

IS..... Output. Information used to look for the determinant of matrix $\boldsymbol{A}$.
(See item (3), "Comments on use," b., 3).)

IP..... Output. Transposition vector that shows the history of the exchange of rows performed through partial pivoting. One-dimensional array of size n.

VW..... Work area. One-dimensional array of size n.

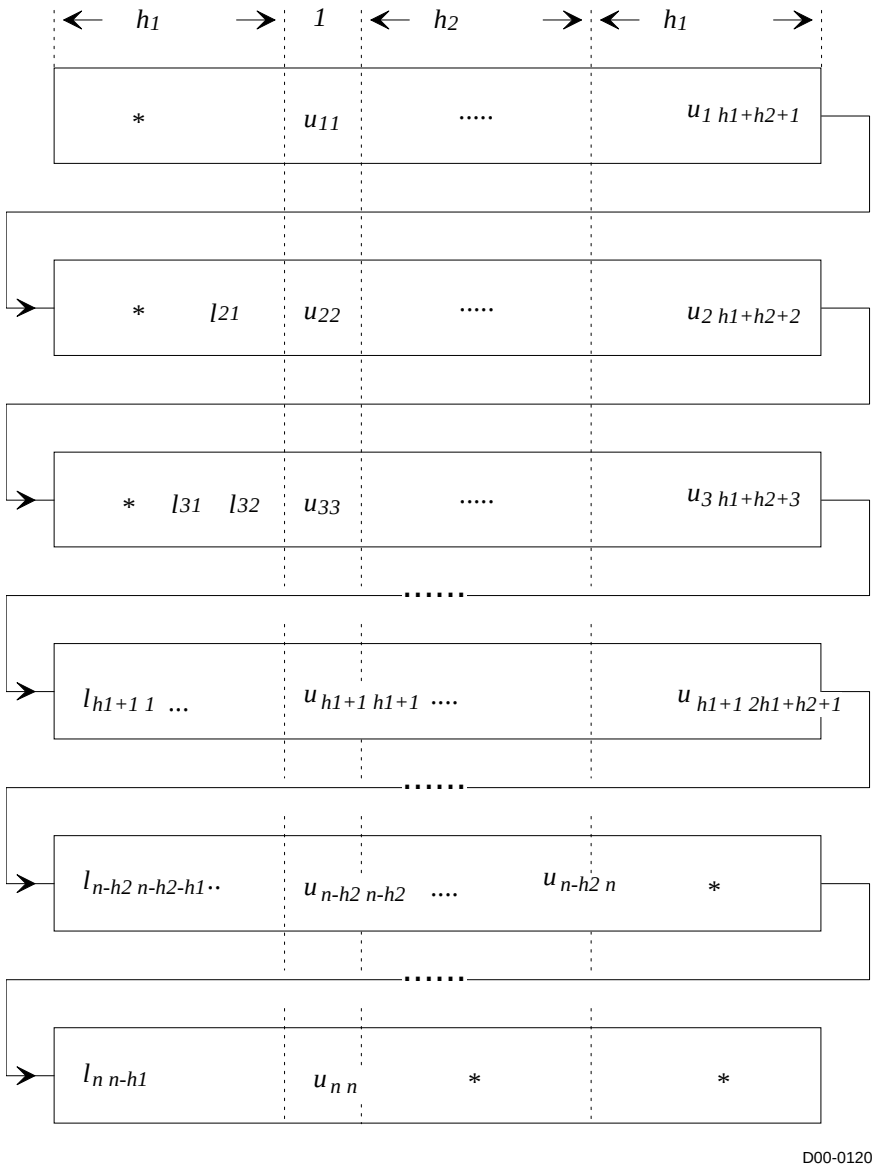
ICON..... Output. Condition code.
See Table VLBX-1, “Condition codes.”



* (asterisk): Undefined value

Figure VLBX-1 Storage method for banded matrix in array A

The *i*-th row vector of the coefficient matrix ***A*** is stored consecutively in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + 1$: $(2 \times h_1 + h_2 + 1) \times i$). Diagonal elements a_{ii} are stored in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$). Outer coefficient matrix elements of the banded part are set to zero.



* (asterisk): Undefined value

Figure VLBX-2 Storage method for matrices *L* and *U* in array A

The *i*-th row vector without diagonal elements of matrix ***L*** is stored in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + 1$: $(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1$). The *i*-th row vector of matrix ***U*** is stored consecutively from the diagonal elements in A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1$: $(2 \times h_1 + h_2 + 1) \times i$).

Table VLBX-1 Condition codes

Code	Description	Processing
0	No error	
20000	All the elements of a row of matrix ***A*** are zero, or pivot is relative zero. Strong possibility that matrix ***A*** is singular.	Processing is stopped.
30000	N ≤ NH1, N ≤ NH2, NH1 < 0, NH2 < 0 or EPSZ < 0.0.	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, VBLU, VBLUX, MGSSL

b. Comments

1) In this subroutine, the case of the pivot value being less than EPSZ is considered relative zero, and processing is stopped with ICON = 20,000.

The standard value of EPSZ is $16 \times u$ where u is the round off unit.

2) When several linear equations with the same coefficient matrix are solved consecutively, solve those equations with ISW = 2 on subsequent calls after the initial call. Then, the computation time diminishes as the LU decomposition process of the coefficient matrix ***A*** is omitted. In this case, the contents in matrix ***A*** are guaranteed as the result of initial call with ISW=1.

3) Elements of matrix ***U*** are stored in array A, as shown in Figure VBLU-2. Therefore, the determinant is obtained by multiplying the IS value by n diagonal elements, that is, the multiplication of A ($(2 \times h_1 + h_2 + 1) \times (i - 1) + h_1 + 1)$, $i = 1, \ldots, n$.

4) In order to save space in the data storage area, this subroutine stores banded matrices by taking advantage of their characteristics. However, depending on bandwidth size, a data storage area that is larger than VALU may be required. In such cases, space in the data storage area can be saved by using VALU.

Characteristics of this subroutine can be exploited when $n > 2 \times h_1 + h_2 + 1$.

c. Example of use

In this example, a linear equation is solved, which takes the unsymmetric banded matrix with bandwidth $h_1 = h_2 = 160$, $n = 160 \times 160$.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(NH=80)
      PARAMETER(NH1=NH)
      PARAMETER(NH2=NH)
      PARAMETER(N=NH*NH)
      PARAMETER(ALPHA=0.5/(NH1+1),BETA=-ALPHA)
      DIMENSION A((2*NH1+NH2+1)*2*N),B(N)
      DIMENSION C(2*NH1+NH2+1,N),IP(N),VW(N)
      EQUIVALENCE(A,C)
C
C     Zero clear
C
      DO 10 I=1,N*(3*NH+1)
```

```
      A(I)=0.0
   10 CONTINUE
C
      DO 15 I=1,N
      B(I)=0.0
      IP(I)=0
   15 CONTINUE
C
C     Coefficient Matrix is built
C
      DO 20 I=1,N
      C(NH1+1,I)=4.0
      B(I)=B(I)+4.0

      IF(I.GT.NH)THEN
      C(1,I)=-1.0+ALPHA
      B(I)=B(I)-1.0+ALPHA
      ENDIF

      IF(I+NH.LE.N)THEN
      C(1+NH1+NH2,I)=-1.0+BETA
      B(I)=B(I)-1.0+BETA
      ENDIF

      IF(I.GT.1.AND.MOD(I-1,NH).NE.0)THEN
      C(NH1,I)=-1.0+ALPHA
      B(I)=B(I)-1.0+ALPHA
      ENDIF

      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(NH1+2,I)=-1.0+BETA
      B(I)=B(I)-1.0+BETA
      ENDIF

   20 CONTINUE
C
C     Solve Banded linear equation
C
      EPSZ=0.0D0
      ICON=0
      ISW=1
      CALL DVLBX(A,N,NH1,NH2,B,EPSZ,ISW,IS,IP,VW,ICON)
      PRINT*,'ICON= ',ICON
C
      PRINT*,'B(1)= ',B(1)
      PRINT*,'B(N)= ',B(N)
      STOP
      END
```

(4) Method

After LU decomposition of the outer product type (see [14]) is performed, the equation

$\boldsymbol{Ax = b}$

is solved through forward-substitution and back-substitution.

A22-61-0402 VLDIV, DVLDIV

The inverse of a positive-definite symmetric matrix decomposed into the factors L,D and L^T
CALL VLDIV (FA, N, VW, ICON)

(1) Function

The inverse matrix $\boldsymbol{A}^{-1}$ of an $n \times n$ positive-definite symmetric matrix $\boldsymbol{A}$ given in decomposed form $\boldsymbol{A} = \boldsymbol{LDL}^T$ is computed.

$$\boldsymbol{A}^{-1} = (\boldsymbol{L}^T)^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1} \qquad (1.1)$$

$\boldsymbol{L}$ and $\boldsymbol{D}$ are, respectively, an $n \times n$ unit lower triangular and a diagonal matrices.

(2) Parameters

FA..... Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$.
For the storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$, see Fig. VSLDL-1 in "Fujitsu SSL II Extended Capabilities User's Guide".
Output. Inverse $\boldsymbol{A}^{-1}$. Lower triangular part of $\boldsymbol{A}^{-1}$ stored by columns.
For the storage method for a symmetric matrix, see Fig. VSLDL-1 in "Fujitsu SSL II Extended Capabilities User's Guide".
One-dimensional array of size $n(n+1)/2$.

N..... Input. Order n of the matrices $\boldsymbol{L}$ and $\boldsymbol{D}$.

VW..... Work Area. One-dimensional array of size n.

ICON..... Output. Condition code.
See Table VLDIV-1, "Condition codes."

Table LDIV-1 Condition codes

Code	Meaning	Processing
0	No error	-
10000	Matrix was not a positive-definite.	Continued
30000	N<1	Bypassed

(1) Comments on use

a. Subprograms used

SSL II.....MGSSL

b. Comments

1) Prior to calling this subroutine, LDL^T-decomposed matrix must be obtained by subroutine VSLDL 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 VSLDL.

2) The subroutine VLSX 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.

c. Example of use

The inverse of a positive symmetric matrix is obtained. $n = 10$.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(55),VW(10)
      DIMENSION VW2(20)
      INTEGER IVW(10)
      N=10
      EPS=0.0D0
      L = 1
      DO J=1,N
         A(l)=N*(N+1)/2-J*(J-1)/2+10.0D0
         L=L+1
         DO I=J+1,N
            A(l)=N+1-I
            L=L+1
         ENDDO
      ENDDO
      WRITE(*,*) 'INPUT MATRIX'
      DO I=1,N
         WRITE(*,1000) (A(((2*N+1-J)*J)/2-N+I),J=1,I)
      ENDDO
      CALL DVSLDL(A,N,EPS,VW2,IVW,ICON)
      IF(ICON.GE.20000)STOP
      CALL DVLDIV(A,N,VW,ICON)
      WRITE(*,*) 'DVLDIV ICON = ',ICON
      WRITE(*,*) 'OUTPUT MATRIX'
      DO I=1,N
         WRITE(*,1000) (A(((2*N+1-J)*J)/2-N+I),J=1,I)
      ENDDO
 1000 FORMAT(X,10E11.3)
      END
```

(4) Method

For further information on the algorithm used consult the entry for LDIV in the Fujitsu *SSL II User's Guide*, and [28]. Note that the storage format used in LDIV is different from that used in this routine, but the underlying algorithm is the same.

A53-31-0301 VLSBX, DVLSBX

System of linear equations with a symmetric positive definite banded matrix (modified Cholesky decomposition)
CALL VLSBX (A, N, NH, B, EPSZ, ISW, ICON)

(1) Function

This routine solves real coefficient linear equations

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad (1.1)$$

using the modified Cholesky method, where $\boldsymbol{A}$ is an $n \times n$ symmetric positive definite banded matrix with upper and lower bandwidths h. $\boldsymbol{b}$ is an n-dimensional real constant vector. $\boldsymbol{x}$ is an n-dimensional solution vector.

$n > h \geq 0$ must be obtained.

In order to exploit performance on vector computers, this subroutine uses the storage method in the order of column vectors.

(2) Parameters

A..... Input. One-dimensional array of size $(h + 1) \times n$.
Stores the diagonal elements of the coefficient matrix A and the lower triangular part of the banded matrix.
For storage method for matrix $\boldsymbol{A}$, see Figure VLSBX-1, "Storage method for matrix $\boldsymbol{A}$ in array A."
Output. Stores LDL^T decomposed $\boldsymbol{D}$ and $\boldsymbol{L}$.
For the storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{D}$, see Figure VLSBX-2, "Storage method for matrices $\boldsymbol{L}$ and $\boldsymbol{D}$ in array A."

N..... Input. Order n of matrix $\boldsymbol{A}$.

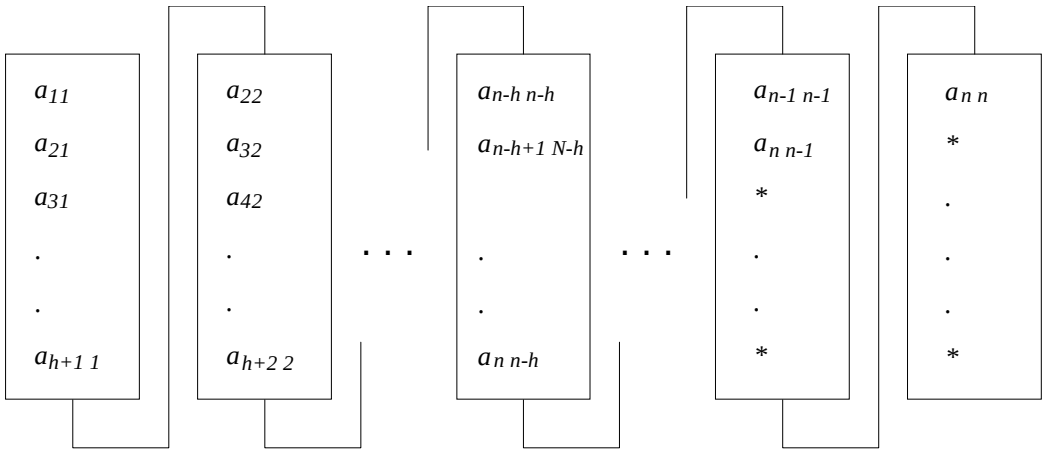
NH..... Input. Lower bandwidth h.

B..... Input. Constant vector $\boldsymbol{b}$.
Output. Solution vector $\boldsymbol{x}$.
One-dimensional array of size n.

EPSZ..... Input. Value of pivot judgment of relative zero (≥ 0.0). If EPSZ = 0.0, a standard value is selected.
(See item (3), "Comments on use," b., 1).)

ISW..... Input. Control information.
When solving k (≥ 1) equation sets with the same coefficient matrix, set ISW as follows.
If ISW = 1, first-set equations are solved.
If ISW = 2, second-set and subsequent equations are solved.
All parameters other than B whose value is changed to the value of new constant vector $\boldsymbol{b}$, should be used unchanged.
(See item (3), "Comments on use," b., 2).)

ICON..... Output. Condition code.
See Table VLSBX-1, "Condition codes."



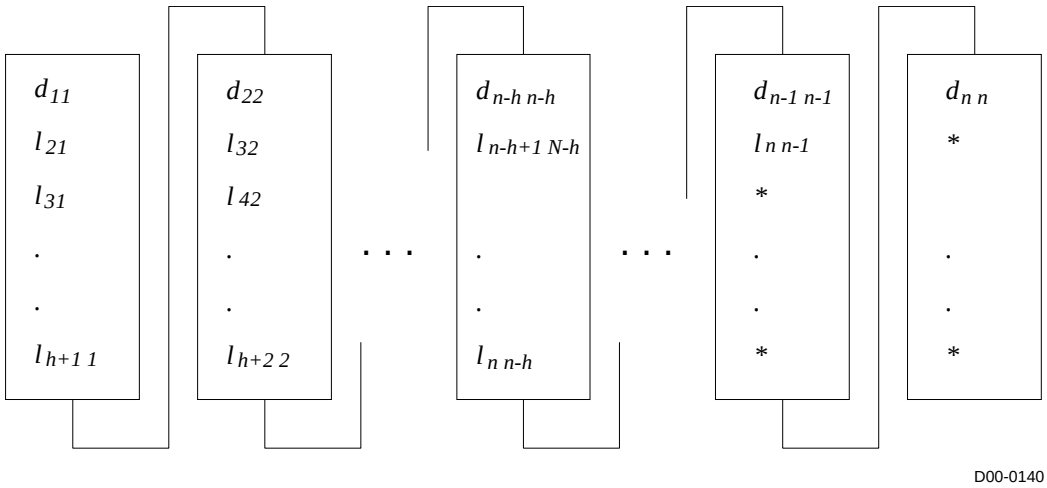
* (asterisk): Undefined value

Figure VLSBX-1 Storage method for matrix *A* in array A

The *i*-th row vector of the lower banded matrix ***A*** is stored according to

$A\ ((h + 1) \times (i - 1) + j - i + 1) = a_{ji}$

where $j = i, \ldots, i + h,\ i = 1, \ldots, n$.



* (asterisk): Undefined value

Figure VLSBX-2 Storage method for matrices *L* and *D* in array A

d_{ii} is stored in $A\ ((h + 1) \times (i - 1) + 1)$.

l_{ji} is stored in $A\ ((h + 1) \times (i - 1) + j - i + 1)$

where $j = i + 1, \ldots, i + h,\ i = 1, \ldots, n$.

Table VLSBX-1 Condition codes

Code	Description	Processing
0	No error	
10000	Pivot is negative. Matrix ***A*** is not positive definite.	Processing continues.
20000	Pivot is relative zero. Strong possibility that matrix ***A*** is singular.	Processing is stopped.
30000	NH < 0, NH ≥ N, or EPSZ < 0.0. ISW ≠ 1, 2.	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, UBLTS, UBUTS, VBLDL, VBLDX, MGSSL

b. Comments

1) In this subroutine, the case of the pivot value being less than EPSZ is considered relative zero and processing is stopped with ICON = 20,000.

The standard value of EPSZ is $16 \times u$ where u is the round off unit.

2) When several linear equations with the same coefficient matrix are solved consecutively, solve those equations with ISW = 2 on the subsequent calls after the initial call. Then, the computation time diminishes as the LDL^T decomposition process of the coefficient matrix ***A*** is omitted.

3) If the pivot becomes negative during the decomposition process, the coefficient matrix is not positive definite. In this subroutine, processing continues, but ICON is set as 10,000.

4) Elements of matrix ***L*** are stored in array A, as shown in Figure VLSBX-2. Therefore, the determinant is obtained by multiplying the n diagonal elements, that is, the multiplication of A ($(h + 1) \times (i - 1) + 1$), $i = 1, \ldots, n$.

c. Example of use

In this example, a linear equation of symmetric positive definite matrix with bandwidth 256 is solved.

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER(NH=128)
       PARAMETER(N=128*128)
       DIMENSION A((NH+1)*N),B(N),C(NH+1,N)
       EQUIVALENCE(A,C)
C
C      Zero clear
C
       DO 10 I=1,N*(NH+1)
       A(I)=0.0
   10 CONTINUE
C
       DO 15 I=1,N
       B(I)=0.0
   15 CONTINUE
C
```

```
C     Coefficient Matrix is built
C     b = A*y , where y=(1,1,....,1)
C
      DO 20 I=1,N
      C(1,I)=4.0
      B(I)=B(I)+4.0

      IF(I+NH.LE.N)THEN
      C(NH+1,I)=-1.0
      B(I+NH)=B(I+NH)-1.0
      B(I)=B(I)-1.0
      ENDIF

      IF(I+1.LE.N.AND.MOD(I,NH).NE.0)THEN
      C(2,I)=-1.0
      B(I+1)=B(I+1)-1.0
      B(I)=B(I)-1.0
      ENDIF

   20 CONTINUE
C
C     Solve Symmetric Positive Definite linear equation
C
      EPSZ=0.0D0
      ISW=1
      CALL DVLSBX(A,N,NH,B,EPSZ,ISW,ICON)
      PRINT*,'ICON=',ICON
      IF(ICON.NE.0)STOP
C
      PRINT*,'B(1)= ',B(1)
      PRINT*,'B(N)= ',B(N)
      STOP
      END
```

(4) Method

After LDL^T decomposition of the outer product type (see [31]) is performed, the equation is solved through forward-substitution and back-substitution.

A53-41-0101 VLTQR, DVLTQR

System of linear equations with a real tridiagonal matrix (QR factorization)
CALL VLTQR (SU, D, SL, N, B, VW, ICON)

(1) Function

This routine solves a system of linear equations with a real tridiagonal matrix using QR factorization.

$$\boldsymbol{Tx} = \boldsymbol{b} \qquad (1.1)$$

T is a $n \times n$ non-singular real tridiagonal matrix. b is a n-dimensional real constant vector. X is a n-dimensional solution vector. n must be greater than or equal to 1. Suppose elements of matrix T are t_{ij}, diagonal elements are $d_i = t_{i,i}$; lower sub-diagonal elements $l_i = t_{i,i-1}$; upper sub-diagonal elements $u_i = t_{i,i+1}$. However, $l_1 = 0$ and $U_n = 0$ must hold.

(2) Parameters

SU ..... Input. Stores an upper sub-diagonal matrix u_i in a one-dimensional array of SU (N) in SU (1:N-1). SU (N) = 0.

D..... Input. Stores diagonal element d_i in a one-dimensional array of SU (N).

SL..... Input. Stores lower sub-diagonal matrix l_i in a one-dimensional array of SL (N) in SL (2:N). SL (1) = 0.

N..... Input. Order n of tridiagonal matrix T.

B..... Input. Constant vector b.

Output. Solution vector x.

VW..... Work area. One-dimensional array of size $7 \times$ N.

ICON..... Output. Condition code.

See Table VLTQR-1, “Condition codes.”

Table VLTQR-1 Condition codes

Code	Meaning	Processing
0	No error	-
10000	The matrix is near “singular.”	
20000	The matrix is near “singular.”	Processing is stopped.
30000	N<1	

(3) Comments on use

a. Subprograms used

SSL II: UQBBS, UQBFC, UQBFE, UQBSL, UQTBS, UQTFC, UQTFE, UQTSL, AMACH, MGSSL

b. Comments

None.

c. Example of use

Solve a linear equation system with the following tridiagonal matrix:

$$\boldsymbol{T} = \begin{bmatrix} 0 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & . & . & . & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix}$$

D01-0010

NOTE: n must be even, otherwise T is singular.

This matrix is not diagonally dominant. The example below determines a constant vector in the right-hand side so that the following value is obtained as the solution. It then tests the accuracy of the solution.

$x_i = (i - 1) / n, i = 1, \ldots, n$

This routine returns ICON = 20000 when the matrix is singular; therefore, the solution is not obtained. When the matrix is near "singular," this routine returns ICON = 10000, and the correct solution is obtained.

```
C     ** EXAMPLE PROGRAM **
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER(MAXN=300000)
      REAL*8 SU(MAXN),D(MAXN),SL(MAXN),B(MAXN),X(MAXN)
      REAL*8 VW(7*MAXN)
      REAL*8 ERR
      INTEGER I,ICON

C Data initialization
      N = 256*1024
      DO 9000 I=1,N
        SU(I)=1
        D(I)=0
        SL(I)=-1
        X(I) = (I-1.0)/N
 9000 CONTINUE
      SU(N)=0
      SL(1)=0

C Calculate the right hand side.
      B(1)=X(1)*D(1)+X(2)*SU(1)
      DO 9002 I=2,N-1
        B(I)=SL(I)*X(I-1)+D(I)*X(I)+SU(I)*X(I+1)
 9002 CONTINUE
      B(N)=SL(N)*X(N-1)+D(N)*X(N)

C Call subroutine
      CALL DVLTQR(SU,D,SL,N,B,VW,ICON)
```

```
C Calculate the relative error
      ERR=0.0D0
      DO 9004 I=1,N
        CONTINUE
      IF(X(I).NE.0.AND.B(I).NE.0)THEN
        ERR=MAX(ABS((X(I)-B(I))/X(I)),ERR)
      ELSE
        ERR=MAX(ABS(X(I)-B(I)),ERR)
      ENDIF
 9004 CONTINUE
      WRITE(*,*)'ERROR:',ERR

      END
```

(4) Method

The multifrontal method is used first to reduce coefficient matrices in a system to a block bidiagonal form. This reduced system is then solved using a recursive wrap-around partitioning algorithm. The partitioning of the unknowns is such that there is no restriction on the size of the matrix in either the reduction to block-bidiagonal form, or the recursive elimination.

This method does not suffer from memory bank conflicts.

The underlying method is Householder's QR factorization.

For details, see [14] and [18].

A53-11-0101 VMBV, DVMBV

Multiplication of a real band matrix and a real vector.
CALL VMBV (A, N, NH1, NH2, X, Y, ICON)

(1) Function

This subroutine performs multiplication of an $n \times n$ band real matrix $\boldsymbol{A}$ with lower band width h_1 and upper band width h_2 by a vector $\boldsymbol{x}$.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

where, $\boldsymbol{x}$ and $\boldsymbol{y}$ are both an n-dimensional vectors.

Also, $n > h_1 \geq 0$ and $n > h_2 \geq 0$.

(2) Parameters

A..... Input. Matrix $\boldsymbol{A}$.
One-dimensional array of size $(2 \times h_1 + h_2 + 1) \times n$.
The storage method for matrix $\boldsymbol{A}$ is shown in the Figure VLBX-1 for subroutine VLBX.

N..... Input. Order n of the matrix $\boldsymbol{A}$.
(See item (3), "Comments on use," b.)

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 Table VMBV-1, "Condition codes."

Table VMBV-1 Condition codes

Code	Meaning	Processing
0	No error	-
30000	N=0, $\|N\| \leq$ NH1, $\|N\| \leq$ NH2, NH1 < 0 or NH2 < 0	Bypassed

(3) Comments on use

a. Subprograms used

SSL II ..... MGSSL

b. Comments on use

This subroutine mainly consists of the computation

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \qquad (3.1)$$

but it can be changed to another type of computation

$$\boldsymbol{y} = \boldsymbol{y'} - \boldsymbol{Ax}$$

by specifying N= -N and giving an arbitrary vector $\boldsymbol{y'}$ to the parameter Y.

In practice, this method can be used to compute a residual vector of linear equations.

c. Example of use

This program multiplies a banded matrix $\boldsymbol{A}$ by a vector $\boldsymbol{x}$.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8  (A-H,O-Z)
      PARAMETER (K=1000,KH1=100,KH2=100)
      DIMENSION A((2*KH1+KH2+1)*K),X(K),Y(K)

      DO 10 I=1,(2*KH1+KH2+1)*K
         A(I)=0.0
 10   CONTINUE
      WRITE(*,*)'INPUT N,NH1,NH2'
      READ(*,*) N,NH1,NH2
      WRITE(*,*)'INPUT A'
      DO 20 I=1,N
         DO 30 J=1,NH1+NH2+1
            IF((J-NH1+(I-1).GE.1).AND.
     &         (J-NH1+(I-1).LE.N))THEN
               WRITE(*,*)'A(',I,',',J-NH1+(I-1),')= '
               READ(*,*) A(J+(2*NH1+NH2+1)*(I-1))
            ENDIF
 30      CONTINUE
 20   CONTINUE
      WRITE(*,*)'INPUT X'
      READ(*,*) (X(I),I=1,N)

      CALL DVMBV(A,N,NH1,NH2,X,Y,ICON)
      PRINT*,'ICON= ',ICON
      PRINT*,'Y(1)= ',Y(1)
      PRINT*,'Y(2)= ',Y(2)
      PRINT*,'...'
      PRINT*,'Y(N)= ',Y(N)
      END
```

(4) Method

This routine performs the multiplication $\boldsymbol{y} = (y_i)$ of an $n \times n$ real band matrix $\mathbf{A} = (a_{ij})$ ($\mathbf{A}$ with lower bandwidth h_1 and upper bandwidth h_2) by a vector $\boldsymbol{x} = (x_j)$ given by:

$$y_i = \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1,...,n$$

However, as $\mathbf{A}$ is a band matrix, the actual calculation is given by:

$$y_i = \sum_{j=\max(1,i-h_1)}^{\min(i+h_2,n)} a_{ij} x_j, \quad i = 1,...,n$$

F17-12-0101 VMCF2, DVMCF2

Singlevariate, multiple and multivariate discrete complex Fourier transform (complex array, mixed radix)
CALL VMCF2 (Z, N, M, ISN, ICON)

(1) Function

This subroutine performs singlevariate, multiple and multivariate discrete complex Fourier transforms using complex array.

For each dimension, it is possible to specify whether the Fourier transform is to be performed, and whether it is normal or inverse.

The size of each dimension can be an arbitrary number, but the transform is fast when the size has factors 2, 3 or 5.

a. Multivariate Fourier transform

By inputting m-dimensional data $\{x_{j1\,j2\ldots jm}\}$ and performing the transform defined in (1.1), $\{\alpha_{k1\,k2\ldots km}\}$ is obtained.

$$\alpha_{k1k2\ldots km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \ldots \sum_{jm=0}^{nm-1} x_{j1j2\ldots jm} . \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \ldots \omega_{nm}^{-jmkmrm} \qquad (1.1)$$

, $k1 = 0, 1, \ldots, n1 - 1$

, $k2 = 0, 1, \ldots, n2 - 1$

....

, $km = 0, 1, \ldots, nm - 1$

, $\omega_{n1} = \exp(2\pi i/n1)$

, $\omega_{n2} = \exp(2\pi i/n2)$

....

, $\omega_{nm} = \exp(2\pi i/nm)$,

where, $n1, n2, \ldots, nm$ is the size of each dimension.

When $ri = 1$, the transform is normal. When $ri = -1$, the transform is inverse.

If $\boldsymbol{r} = (1, 1, 1)$ for example, the following three-dimensional transform is obtained:

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} . \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

b. Multiple transform

For $ri = 0$, the summation $\sum_{ji=0}^{ni-1}$ is omitted, and index ji of x in (1.1) is changed to ki.

For example, a singlevariate multiple transform has only one summation. When performing the following transform with respect to only the second dimension of a three-dimensional data, specify $r = (0, 1, 0)$.

$$\alpha_{k1k2k3} = \sum_{j2=0}^{n2-1} x_{k1j2k3} . \omega_{n2}^{-j2k2}$$

(2) Parameters

Z..... Input. Complex variable $\{x_{j1\,j2...jm}\}$.
Output. Complex variable $\{\alpha_{k1\,k2...km}\}$.
M-dimensional complex array Z($n1$, $n2$,..., nm).

N..... Input. One-dimensional array of size M. N (i) is the size of i-th dimension, where i = 1, ..., M.

M..... Input. Dimension order M of the multivariate Fourier transform.

ISN..... Input. One-dimensional array of size M.
ISN (i) shows the direction ri of the Fourier transform of each dimension.
For ISN = 1, normal transform.
For ISN = 0, no transform.
For ISN = -1, inverse transform.

ICON..... Output. Condition code.
See Table VMCF2-1, "Condition codes."

Table VMCF2-1 Condition codes

Code	Description	Processing
0	No error	
30000	M≤0,	Processing is stopped.
30002	ISN (i) > 1 or ISN (i) < -1	
30003	N (i) < 1 was specified.	
30004	ISN (i) were all zero.	

(3) Comments on use

a. Subprograms used

SSL II: UZACM, UZCOM, UZFB2, UZFB3, UZFB4, UZFB5, UZFB8, UZFB6, UZFBL, UZFBR, UZFBS, UZFCT, UZFF2, UZFF3, UZFF4, UZFF5, UZUU8, UZFF6, UZFMR, UZFOC, UZUPB, UZFPF, UZFRC, UZFRP, UZFS, UZFT, UZFT2, UZFT3, UZFT5, UZFTB, UZFTF, UZFUB, UZFUF, UZFUS, UZFUW, UZSCL, UZTR2, UZTRN, UZUNI, UNXRD, UFCT, MGSSL

b. Comments

1) General definition of Fourier transform

The multivariate discrete complex Fourier transform and inverse transform are generally defined in (3.1) and (3.2).

$$\alpha_{k1k2k...km} = \frac{1}{n1\,n2..nm} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \cdots \sum_{jm=0}^{nm-1} x_{j1\,j2...jm} \cdot \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \cdots \omega_{nm}^{-jmkm} \qquad (3.1)$$

, $k1$ = 0, 1, ..., $n1$-1

, $k2$ = 0, 1, ..., $n2$-1

...

, km = 0, 1, ..., nm-1

$$x_{j1j2\ldots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \cdots \sum_{km=0}^{nm-1} \alpha_{k1k2\ldots km} \cdot \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \cdots \omega_{nm}^{-jmkm} \qquad (3.2)$$

, $j1 = 0, 1, \ldots, n1\text{-}1$

, $j2 = 0, 1, \ldots, n2\text{-}1$

...

, $jm = 0, 1, \ldots, nm\text{-}1$

where,

$\omega_{n1} = \exp(2\pi i/n1)$

, $\omega_{n2} = \exp(2\pi i/n2)$

...

, $\omega_{nm} = \exp(2\pi i/nm)$

The subroutine calculates $\{n1\ n2\ ..\ nm\ \alpha_{k1k2\ldots km}\ \}$ or $\{x_{j1\ j2\ ..jm}\}$ corresponding to the left-hand-side terms of (3.1) and (3.2). The user must normalize the results, if necessary.

2) Stack size

This subroutine exploits work area internally as an auto allocatable array on stack area. Therefore an abnormal termination could be occured when the stack area runs out. The necessary size for the auto allocatable array is shown below.

If *ni* can be expressed as products of powers of 2, 3, and 5, then the work area size is $8 \times \max\{ni \mid i = 1, \ldots, M \text{ and ISN } (i) \neq 0.\}$ *byte* for single precision, and twice for double precision.

If there are numbers among *ni* that cannot be expressed as products of powers of 2, 3, and 5, then the work area size is $40 \times \max\{ni \mid i = 1, \ldots, M \text{ and ISN } (i) \neq 0.\}$ *byte* at most case for single precision, and twice for double precision.

It is recommended to specify the sufficiently large stacksize with "limit" or "ulimit" command under consideration that the stack area could be used for another work area of fixed size and for user's program also.

c. Example of use

In this example, a singlevariate fast Fourier transform is computed.

```
C     **EXAMPLE**
      INTEGER NMAX
      PARAMETER (NMAX=100000,NDIM=1)
      COMPLEX*16 Z(NMAX)
      REAL*8 ERR,PI,THETA
      INTEGER N(NDIM),ISN(NDIM),N1,L,M,NVAL(6),IN
      DATA NVAL/16199,16200,16201,16383,16384,16385/
      PI=4D0*ATAN(1D0)
      DO 40 IN=1,6
        N1=NVAL(IN)
        N(1)=N1
        L=79
        DO 10 I=1,N1
          Z(I)=(0D0,0D0)
```

```
10   CONTINUE
     Z(L+1)=(1D0,0D0)
     ISN(1)=1
     M=1
     CALL DVMCF2 (Z,N,M,ISN,ICON)
     IF (ICON.NE.0) WRITE (6,*) 'ICON=',ICON
     ERR=0D0
     DO 20 K=0,N1-1
       THETA=2*PI*L*K/DBLE(N1)
       ERR=MAX(ERR,ABS(Z(K+1)-
  &        DCMPLX(COS(THETA),-SIN(THETA))))
20   CONTINUE
     WRITE (6,30) N1,ERR
30   FORMAT (' N=',I6,' ERROR = ',D10.3)
40 CONTINUE
   STOP
   END
```

(4) Method

This subroutine performs either multiple transforms of complex Fourier transforms, or multivariate complex Fourier transforms efficiently on a scalar CPU.

Multivariate transforms are computed by transforming the multiple one-dimensional transform on each dimension in turn. The singlevariate transform is performed with an appropriate method according to the value of *ni*. If the value of *ni* is large in respect to the size of the cache, a variant of two-sided splitting algorithm is used for blocking. (Refer to [17] in Appendix A, "References.")

F17-11-0101 VMCFT, DVMCFT

Singlevariate, multiple and multivariate discrete complex Fourier transform (real and imaginary array seperated, mixed radix)
CALL VMCFT (XR, XI, N, M, ISN, W, IW, ICON)

(1) Function

This subroutine performs singlevariate, multiple and multivariate discrete complex Fourier transforms.

For each dimension, it is possible to specify whether the Fourier transform is to be performed, and whether it will be normal or inverse.

The size of any dimension can be an arbitrary number, but the transform is fast with factors 2, 3 or 5.

a. Multivariate Fourier transform

By inputting $\{x_{j1\ j2...jm}\}$ and performing the transform defined in (1.1), $\{\alpha_{k1\ k2...km}\}$ is obtained.

$$\alpha_{k1k2...km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \cdots \sum_{jm=0}^{nm-1} x_{j1j2...jm} . \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \cdots \omega_{nm}^{-jmkmrm} \qquad (1.1)$$

, $k_1 = 0, 1, ..., n_1 - 1$

, $k_2 = 0, 1, ..., n_2 - 1$

....

, $k_m = 0, 1, ..., n_m - 1$

, $\omega_{n1} = \exp(2\pi i/n_1)$

, $\omega_{n2} = \exp(2\pi i/n_2)$

....

, $\omega_{nm} = \exp(2\pi i/n_m)$

When $r_i = 1$, the transform is normal. When $r_i = -1$, the transform is inverse.

For $r_i = 0$, the summation $\sum_{ji=0}^{ni-1}$ is omitted, and j_i is changed to k_i. where j_i is an index of x in equation (1.1).

If $\boldsymbol{r} = (0, 1, 1)$,

the following equation is obtained:

$$\alpha_{k1k2k3} = \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{k1j2j3} . \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

b. Multiple transform

A multiple transform has only one summation. When performing the second-dimension transform, the following is obtained:

$$\alpha_{k1k2k3} = \sum_{j2=0}^{n2-1} x_{k1j2k3} . \omega_{n2}^{-j2k2}$$

(2) Parameters

XR..... Input. Real part of $\{x_{j1\ j2...jm}\}$.
Output. Real part of $\{\alpha_{k1\ k2...km}\}$.
One-dimensional array of size $n_1 \times n_2 \ldots \times n_m$.

XI..... Input. Imaginary part of $\{x_{j1\ j2...jm}\}$.
Output. Imaginary part of $\{\alpha_{k1\ k2...km}\}$.
One-dimensional array of size $n_1 \times n_2 \times \ldots \times n_m$.

N..... Input. One-dimensional array of size M. N (I) is the size of I-th dimension, where I = 1, ..., M.

M..... Input. Dimension order M of the multivariate Fourier transform.

ISN..... Input. One-dimensional array of size M.
ISN (I) shows the direction r_i of the Fourier transform of each dimension.
For ISN = 1, normal transform.
For ISN = 0, no transform.
For ISN = -1, inverse transform.

W..... Work area.
One-dimensional array of size IW.

IW..... Input. Size of the work area.
If n_i can be expressed as products of powers of 2, 3, and 5, then the work area size is $2 \times \mathrm{MAX}\ \{n_i \mid i = 1, \ldots, \mathrm{M}$ and ISN $(i) \neq 0.\}$
If there are numbers among n_i that cannot be expressed as products of powers of 2, 3, and 5, then the work area size exceeds $2 \times n_1 \times \ldots \times n_m$.
In such a case, the size of the work area can be determined by calling the subroutine with IW = 0.
For the procedure to determine the size of the work area, see item "(3), "Comments on use," b., 2)."
Output. If the size of the work area is smaller than required, returns the required size of the work area.

ICON..... Output. Condition code.
See Table VMCFT-1, "Condition codes."

Table VMCFT-1 Condition codes

Code	Description	Processing
0	No error	
30000	M≤0,	Processing is stopped.
30001	Insufficient work area	
30002	ISN (I) > 1 or ISN (I) < -1	
30003	N (I) < 1 was specified.	
30004	ISN (I) were all zero.	

(3) Comments on use

a. Subprograms used

SSL II: UACOM, UCOMR, UFT, UFTBS, UCFS, UCF16, UCFT2, UCFT3, UCFT4, UCFT5, UCFT8, UCFMR, UCRU, UCTRF, URUNI, USCAL, UTRAN, UTRTW, UTWID, UGCD, UNXRD, UFCT, MGSSL

b. Comments

1) General definition of Fourier transform

The multivariate discrete complex Fourier transform and inverse transform are generally defined in (3.1) and (3.2).

$$\alpha_{k1k2k\ldots km} = \frac{1}{n_1 n_2 .. n_m} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \ldots \sum_{jm=0}^{nm-1} x_{j1j2\ldots jm} . \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \ldots \omega_{nm}^{-jmkm} \qquad (3.1)$$

$k_1 = 0, 1, ..., n_1\text{-}1$

$, k_2 = 0, 1, ..., n_2\text{-}1$

...

$, k_m = 0, 1, ..., n_m\text{-}1$

$$x_{j1j2\ldots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \ldots \sum_{km=0}^{nm-1} \alpha_{k1k2\ldots km} . \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \ldots \omega_{nm}^{-jmkm} \qquad (3.2)$$

$j_1 = 0, 1, ..., n_1\text{-}1$

$, j_2 = 0, 1, ..., n_2\text{-}1$

...

$, j_m = 0, 1, ..., n_m\text{-}1$

$\omega_{n1} = \exp(2\pi i/n_1)$

$, \omega_{n2} = \exp(2\pi i/n_2)$

...

$, \omega_{nm} = \exp(2\pi i/n_m)$

The subroutine calculates $\{n_1\ n_2\ ..\ n_m\ \alpha_{k1k2\ldots km}\ \}$ or $\{x_{j1\,j2\,..jm}\}$ corresponding to the left-hand-side terms of (3.1) and (3.2). The user must normalize the results, if necessary.

2) Size of work area

Symbols used are defined as follows.

RADIX = {*n*: positive integer that can be expressed as the product of powers of 2, 3, and 5}

NORAD = natural number - *RADIX*

minrad (*n*) is the minimum natural number *m*, where $n<m$ and $m \in RADIX$.

relfac (*n*) is the minimum natural number *q*, where $n = p \times q$ and $p \in RADIX$, $q \in NORAD$.

$NP = n_1 \times n_2 \times ... \times n_m$

In this case, the size of the work area is determined using the following procedure.

{*I* | 1, ..., M} and {ISN (I) ≠ 0}

$\underset{i \in I}{Max}(SIZE_i)$ is the size of the work area.

$SIZE_i$ is determined as follows:

a) When $n_i \in RADIX$, $SIZE_i = 2 \times n_i$

b) When *relfac* (n_i) equals n_i

$SIZE_i = 2 \times \mathrm{NP} \times \mathrm{minrad}\ (n_i)/n_i + 4 \times \mathrm{minrad}\ (n_i)$

c) Otherwise,

$SIZE_i = 2 \times NP \times minrad\ (relfac\ (n_i))/relfac\ (n_i) + max\ (4 \times minrad\ (relfac\ (n_i)), 2 \times n_i)$

c. Example of use

In this example, a singlevariate fast Fourier transform is computed.

```
C     **EXAMPLE**
      INTEGER NMAX,NW
      PARAMETER (NMAX=100000,NW=200000)
      REAL*8 XR(NMAX),XI(NMAX),W(NW),PI
      REAL*4 ERR
      INTEGER N(3),ISN(3),IW,N1,L,M,NVAL(6),IN
      DATA NVAL/16199,16200,16201,16383,16384,16385/
      PI=4D0*ATAN(1D0)
      DO 40 IN=1,6
        N1=NVAL(IN)
        N(1)=N1
        L=79
        DO 10 I=1,N1
          XR(I)=0D0
   10     XI(I)=0D0
        XR(L+1)=1D0
        ISN(1)=1
        M=1
        IW=NW
        CALL DVMCFT (XR,XI,N,M,ISN,W,IW,ICON)
        IF (ICON.NE.0) WRITE (6,*) 'ICON=',ICON
        ERR=0D0
        DO 20 K=0,N1-1
          ERR=MAX(ERR,XR(K+1)-COS(2*PI*L*K/DBLE(N1)))
   20     ERR=MAX(ERR,XI(K+1)+SIN(2*PI*L*K/DBLE(N1)))
        WRITE (6,30) N1,ERR
   30   FORMAT (' N=',I6,' ERROR = ',E10.3)
   40 CONTINUE
      STOP
      END
```

(4) Method

This subroutine performs either multiple transforms of singlevariate complex Fourier transforms, or multivariate complex Fourier transforms.

A singlevariate transform is performed as follows:

A. Splitting the order of the transform into factors $n = p\ q$, where the factor of p can be expressed as the product of powers of 2, 3, and 5, and the factor of q is a number

mutually prime to 2, 3, and 5. (In the following, the set 2, 3, and 5 is referred to as the radix set).

B. After implementing the basic factorization of the order into $n = p\,q$, the following four-step algorithm is performed.

$$z^{(1)}_{j_1+k_0q} = \sum_{j_0=0}^{p-1} \omega_p^{k_0 j_0} x_{j_1+j_0q} \qquad j_1 = 0, ..., q\text{-}1,\ k_0 = 0, ..., p\text{-}1 \qquad (4.1)$$

$$z^{(2)}_{j_1+k_0q} = \omega_n^{k_0 j_1} z^{(1)}_{j_1+k_0q} \qquad k_0 = 0, ..., p\text{-}1,\ j_1 = 0, ..., q\text{-}1 \qquad (4.2)$$

$$z^{(3)}_{k_0+j_1p} = z^{(2)}_{j_1+k_0q} \qquad k_0 = 0, ..., p\text{-}1,\ j_1 = 0, ..., q\text{-}1 \qquad (4.3)$$

$$y_{k_0+k_1p} = \sum_{j_1=0}^{q-1} \omega_q^{k_1 j_1} z^{(3)}_{k_0+j_1p} \qquad k_0 = 0, ..., p\text{-}1,\ k_1 = 0, ..., q\text{-}1 \qquad (4.4)$$

Step 1 and step 4 are multiple Fourier transforms of order p and q respectively. The factor p is a product of powers of the radices, and step 1 is computed using a mixed radix fast Fourier transform.

For details about this algorithm, see [17] and [19]. The mixed radix algorithm consists of a transform of low orders, a unitary scaling operation, and a transposition.

Steps 2 and 3 are fairly simple and are performed in a straight forward way.

The factor q is mutually prime to the radix set, so step 4 is performed using a variant of Bluestein's algorithm. (See [40].)

Multivariate transforms are computed by transforming the previous multiple one-dimensional transform on each dimension in turn. During the process, the data is permuted to maintain long vector lengths and continuous data access, though the returned result is in the correct order.

F17-12-0201 VMRF2, DVMRF2

Singlevariate, multiple and multivariate discrete real Fourier transform (mixed radix)
CALL VMRF2 (X, N, M, ISIN, ISN, ICON)

(1) Function

This subroutine performs singlevariate, multiple and multivariate discrete real Fourier transforms.

Whether the Fourier transform is to be performed, and its direction, can be specified for each dimension.

For the 1-st dimension, "no transform" cannot be specified, and the size of the 1-st dimension must be an even number. The sizes of all other dimension can be arbitrary numbers, but the transform is fast with the sizes which can be expressed as products of the powers of 2, 3, and 5.

The result of a multiple and multivariate discrete real Fourier transform has a complex conjugate relation. For the 1-st dimension, the first $n_1 / 2 + 1$ complex elements are stored.

a. Multivariate Fourier transform

1) Transform

Inputting m-dimensional data $\{x_{j1j2...jm}\}$ and performing the transform defined in (1.1) obtains $\{\alpha_{k1k2...km}\}$.

$$\alpha_{k1k2...km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \cdots \sum_{jm=0}^{nm-1} x_{j1j2...jm} \cdot \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \cdots \omega_{nm}^{-jmkmrm} \qquad (1.1)$$

$$, k1 = 0, 1, \ldots , n1 - 1$$

$$, k2 = 0, 1, \ldots , n2 - 1$$

$$\cdots$$

$$, km = 0, 1, \ldots, nm - 1$$

$$, \omega_{n1} = \exp(2\pi i / n1)$$

$$, \omega_{n2} = \exp(2\pi i / n2)$$

$$\cdots$$

$$, \omega_{nm} = \exp(2\pi i / nm) ,$$

where, $n1, n2, \ldots, nm$ is the size of each dimension.

$ri = 1$ or $ri = -1$ can be specified for the transform direction.

If $r = (1, 1, 1)$ for example, the following three-dimensional Fourier transform is obtained:

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \cdot \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

2) Inverse transform

Inputting $\{\alpha_{k1k2\ldots km}\}$ and performing the transform defined in (1.2), obtains $\{x_{j1j2\ldots jm}\}$.

$$x_{j1j2\ldots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \cdots \sum_{km=0}^{nm-1} \alpha_{k1k2\ldots km} \cdot \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \cdots \omega_{nm}^{-jmkmrm} \qquad (1.2)$$

$$, j1 = 0, 1, \ldots , n1 -1$$

$$, j2 = 0, 1, \ldots , n2 -1$$

$$\ldots .$$

$$, jm = 0, 1, \ldots , nm -1$$

$$, \omega_{n1} = \exp(2\pi i / n1)$$

$$, \omega_{n2} = \exp(2\pi i / n2)$$

$$\ldots .$$

$$, \omega_{nm} = \exp(2\pi i / nm) ,$$

where, $n1, n2, \ldots, nm$ is the size of each dimension.

In an inverse transform, a direction that is inverse to that specified in the transform must be specified.

ri = -1 or ri = 1

b. Multiple transform

When ri = 0 is specified, the summation $\sum_{ji=0}^{ni-1}$ is omitted.

In the case of real-to-complex transform, index ji of x in (1.1) is changed to ki.

In the case of complex-to-real transform, index ki of α in (1.2) is changed to ji.

For example, singlevariate multiple transform has only one summation. When performing the following transform with respect to only the first-dimension of a three-dimensional data, speciry r=(1,0,0).

$$\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} x_{j1k2k3} \cdot \omega_{n1}^{-j1k1}$$

(2) Parameters

X..... m-dimensional array X ($n1$+2, $n2$, ..., nm).

[For ISN = 1 (transform from real to complex): **]**

Input. The real data $\{x_{j1j2\ldots jm}\}$ is stored in X (1:$n1$, 1:$n2$, ... , 1:nm).

Output. The real and imaginary part of $\{\alpha_{k1k2..km}\}$ are stored in X (1:$n1$+2, 1:$n2$, ... , 1:nm) by turns.

$k1$ = 0, 1, ... , $n1$/ 2,

$k2$ = 0, 1, ... , $n2$-1,

. . .

km = 0, 1, ..., *nm*, -1

[For ISN = -1 (transform from complex to real): **]**

Input. The real and imaginary part of { $\alpha_{k1k2...km}$ } are stored in X (1:*n*1+2, 1:*n*2, ... , 1:*nm*) by turns.

*k*1 = 0, 1, . . ., *n*1 / 2,

*k*2 = 0, 1, . . ., *n*2 - 1,

. . .

km = 0, 1, . . ., *nm*, - 1

Output. The real data {$x_{j1j2...jm}$} is stored in X (1:*n*1, 1:*n*2, ..., 1:*nm*).

N..... Input. One-dimensional array of size M. *ni* is stored in N (*i*) (*i* = 1, ..., M), where *ni* is the size of the I-th dimension. The size of the 1-st dimension must be an even number.

M..... Input. The size of dimension *m* of the multivariate Fourier transform.

ISIN..... Input. One-dimensional array of size M.

ISIN (*i*) shows the direction r_i of the Fourier transform of each dimension.

ISIN (1) cannot be 0.

For ISIN = 1, *ri* = 1.

For ISIN = 0, there is no transform.

For ISIN = -1, *ri* = - 1.

ISN..... Input.

For ISN = 1, normal transform (real to complex).

For ISN = - 1, inverse transform (complex to real).

ICON..... Output. Condition code.

See Table VMRF2-1, “Condition codes.”

Table VMRF2-1 Condition codes

Code	Description	Processing
0	No error	-
30001	N(*i*) ≤ 0 or M ≤ 0	Processing is stopped.
30016	ISIN (*i*) < -1, ISIN (*i*) > 1, or ISIN (1) = 0	
30032	ISN ≠ 1 and ISN ≠ -1	
30512	The size of first dimension is odd number.	

(3) Comments on use

a. Subprograms used

SSLII: UMRFF, UMRFB, VMCF2, UZACM, UZCOM, UZFB2, UZFB3, UZFB4, UZFB5, UZFB8, UZFB6, UZFBL, UZFBR, UZFBS, UZFCT, UZFF2, UZFF3, UZFF4, UZFF5, UZUU8, UZFF6, UZFMR, UZFOC, UZUPB, UZFPF, UZFRC, UZFRP, UZFS, UZFT, UZFT2, UZFT3, UZFT5, UZFTB, UZFTF, UZFUB, UZFUF, UZFUS, UZFUW, UZSCL, UZTR2, UZTRN, UZUNI, UNXRD, UFCT, MGSSL

b. Comments

1) General definition of Fourier transform

The multivariate discrete Fourier transform and inverse transform are generally defined as in (3.1) and (3.2).

$$\alpha_{k1k2\ldots km} = \frac{1}{n1\,n2\ldots nm} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \ldots \sum_{jm=0}^{nm-1} x_{j1j2\ldots jm} . \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \ldots \omega_{nm}^{-jmkm} \quad (3.1)$$

$$, k1 = 0, 1, \ldots, n1\text{-}1$$

$$, k2 = 0, 1, \ldots, n2\text{-}1$$

$$\ldots .$$

$$, km = 0, 1, \ldots, nm\text{-}1$$

$$x_{j1j2\ldots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \ldots \sum_{km=0}^{nm-1} \alpha_{k1k2\ldots km} . \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{nm}^{jmkm} \quad (3.2)$$

$$, j1 = 0, 1, \ldots, n1 - 1$$

$$, j2 = 0, 1, \ldots, n2 - 1$$

$$\ldots .$$

$$, jm = 0, 1, \ldots, nm - 1$$

where

$$\omega_{n1} = \exp(2\pi i / n1)$$

$$, \omega_{n2} = \exp(2\pi i / n2)$$

$$\ldots .$$

$$, \omega_{nm} = \exp(2\pi i / nm)$$

The subroutine calculates $\{n1\ n2..nm\ \alpha_{k1k2..km}\}$ or $\{x_{j1j2..jm}\}$ corresponding to the left-hand terms of (1.1) and (1.2). For i, where ISIN (i) = 0, ni is replaced with 1. If necessary, the user must normalize the results.

2) The result of the multivariate discrete real Fourier transform has the following complex conjugate relation:

$$\alpha_{k1\ k2\ \ldots\ km} = \overline{\alpha_{n1\text{-}k1\ n2\text{-}k2\ \ldots\ nm\text{-}km}}$$

$$k1 = 0, \ldots, n1\ /2$$

$$, k2 = 0, \ldots, n2\ \text{-}1$$

$$\ldots .$$

, km = 1, ..., nm -1

In the case of ki=0, ni-ki is regarded as 0.

For h, where ISIN (h) = 0, the h-th index in the right-hand terms is still kh.

The rest of terms can be calculated using this relation.

2) Stack size

This subroutine exploits work area internally as an auto allocatable array on stack area. Therefore an abnormal termination could be occured when the stack area runs out. The necessary size for the auto allocatable array is shown below.

If ni can be expressed as products of powers of 2, 3, and 5, then the work area size is $12 \times \max\{ni \mid i = 1, ..., M \text{ and ISN } (i) \neq 0.\}$ *byte* for single precision, and twice for double precision.

If there are numbers among ni that cannot be expressed as products of powers of 2, 3, and 5, then the work area size is $40 \times \max\{ni \mid i = 1, ..., M \text{ and ISN } (i) \neq 0.\}$ *byte* at most case for single precision, and twice for double precision.

It is recommended to specify the sufficiently large stacksize with "limit" or "ulimit" command under consideration that the stack area could be used for another work area of fixed size and for user's program also.

c. Example of use

In this example, a two-dimensional real Fourier transform is calculated.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(N1=1024,N2=1024,M=2)
      DIMENSION X(N1+2,N2),N(M),ISIN(M)
C
      DO 100 I=1,N2
      DO 100 J=1,N1
      X(J,I)=FLOAT(J)+FLOAT(N1)*(I-1)
  100 CONTINUE
C
      N(1)=N1
      N(2)=N2
      ISIN(1)=1
      ISIN(2)=1
      ISN= 1
C
C     REAL TO COMPLEX TRANSFORM
C
      CALL DVMRF2(X,N,M,ISIN,ISN,ICON)
      PRINT*,'ICON=',ICON
C
      N(1)=N1
      N(2)=N2
      ISIN(1)=-1
      ISIN(2)=-1
      ISN=-1
C
C     COMPLEX TO REAL TRANSFORM
C
      CALL DVMRF2(X,N,M,ISIN,ISN,ICON)
```

```
      PRINT*,'ICON=',ICON

      ERROR=0.0D0
      DO 200 I=1,N2
      DO 200 J=1,N1
      ERROR=MAX(ABS(X(J,I)/(N1*N2)-
     &    (FLOAT(J)+FLOAT(N1)*(I-1))),ERROR)
  200 CONTINUE
C
      PRINT*,'ERROR=',ERROR
      STOP
      END
```

(4) Method

This subroutine performs either real-to-complex or complex-to-real multiple multivariate discrete Fourier transforms efficiently on a scalar CPU.

A real Fourier transform of the first dimension can be done without redundant calculations by exploiting inherent properties of a complex transform. For further information on the algorithm, refer to the description of RFT routine in the Fujitsu *SSL II User's Guide*. For the transforms of the other dimensions, the multivariate discrete complex Fourier transform routine VMCF2 is used for complex data straightforward.

F17-11-0201 VMRFT, DVMRFT

Multiple and multivariate discrete real Fourier transform (mixed radices of 2, 3, and 5)
CALL VMRFT (X, N, M, ISIN, ISN, W, ICON)

(1) Function

This subroutine performs multiple and multivariate discrete real Fourier transforms.

Whether the Fourier transform is to be performed, and its direction, can be specified for each dimension. All dimensions on which a transform is to be performed must have sizes which can be expressed as products of the powers of 2, 3, and 5.

At least one of the first *m*-1 dimensions must be an even number. For the *m*-th dimension, "no transform" cannot be specified.

The result of a multiple and multivariate discrete real Fourier transform has a complex conjugate relation. For the *m*-th dimension, the first $n_m / 2 + 1$ elements are stored.

a. Multivariate Fourier transform

1) Transform

Inputting $\{x_{j1j2\cdots jm}\}$ and performing the transform defined in (1.1) obtains $\{n_1\, n_2 \ldots n_m\, \alpha_{k1k2\ldots km}\}$.

$$n_1\, n_2 \ldots n_m\, \alpha_{k1k2\ldots km} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \cdots \sum_{jm=0}^{nm-1} x_{j1j2\ldots jm} \cdot \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \cdots \omega_{nm}^{-jmkmrm} \qquad (1.1)$$

$$, k_1 = 0, 1, \ldots, n_1 - 1$$

$$, k_2 = 0, 1, \ldots, n_2 - 1$$

$$\ldots\ldots$$

$$, k_m = 0, 1, \ldots, n_m - 1$$

$$, \omega_{n1} = \exp(2\pi i / n_1)$$

$$, \omega_{n2} = \exp(2\pi i / n_2)$$

$$, \omega_{nm} = \exp(2\pi i / n_m)$$

$r_i = 1$ or $r_i = -1$ can be specified for the transform direction.

For $r_i = 0$, the summation $\sum_{ji=0}^{ni-1}$ is omitted, and j_i is changed to k_i, where j_i is an index of x in equation (1.1).

n_i in the left-hand term of equation (1.1) is replaced with 1.

For $r = (0, 1, 1)$, the following equation is obtained:

$$n_2 n_3 \alpha_{k1k2k3} = \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{k1j2j3} \cdot \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$

2) Inverse transform

Inputting $\{\alpha_{k1k2...km}\}$ and performing the transform defined in (1.2), obtains $\{X_{j1j2...jm}\}$.

$$x_{j1j2...jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \cdots \sum_{km=0}^{nm-1} \alpha_{k1k2...km} \cdot \omega_{n1}^{-j1k1r1} \omega_{n2}^{-j2k2r2} \cdots \omega_{nm}^{-jmkmrm} \qquad (1.2)$$

$$, j_1 = 0, 1, \ldots , n_1 - 1$$

$$, j_2 = 0, 1, \ldots , n_2 - 1$$

$$\ldots .$$

$$, j_m = 0, 1, \ldots , n_m - 1$$

$$, \omega_{n1} = \exp(2\pi i / n_1)$$

$$, \omega_{n2} = \exp(2\pi i / n_2)$$

$$\ldots .$$

$$, \omega_{nm} = \exp(2\pi i / n_m)$$

In an inverse transform, a direction that is inverse to that specified in the transform must be specified.

$r_i = -1$ or $r_i = 1$

For $r_i = 0$, the summation $\sum_{ji=0}^{ni-1}$ is omitted and k_i is changed to j_i, where k_i is an index of α in equation (1.2).

b. Multiple transform

A multiple transform has only one summation. With a three-dimensional transform, the following is obtained:

$$n_3\alpha_{k1k2k3} = \sum_{j3=0}^{n3-1} x_{k1k2j3} \cdot \omega_{n3}^{-j3k3r3}$$

(2) Parameters

X..... For ISN = 1 (transform from real to complex):

Input. The real data $\{x_{j1j2...jm}\}$ is stored in X (1:n_1, 1:n_2, ... , 1:n_m).

Output. The real part of $\{n_1\ n_2 ... n_m\ \alpha_{k1k2..km}\}$ is stored in X (1:n_1, 1:n_2, ... , 1:n_m /2+1). The imaginary part of $\{n_1\ n_2 ... n_m\ \alpha_{k1k2...km}\}$ is stored in X (1:n_1, 1:n_2, ..., n_m / 2 + 2:2 × (n_m / 2+1)).

$k_1 = 0, 1, \ldots , n_1-1,$

$k_2 = 0, 1, \ldots , n_2-1,$

. . .

$k_m = 0, 1, ..., n_m / 2,$

For ISIN (i) = 0, n_i in $\{n_1\ n_2 ... n_m\ \alpha_{k1k2...km}\}$ is replaced with 1.

For ISN = -1 (transform from complex to real):

Input. The real part of $\{ \alpha_{k1k2...km} \}$ is stored in X (1:n_1, 1:n_2, ... , 1:n_m / 2+1). The imaginary part of $\{ \alpha_{k1k2...km} \}$ is stored in X (1:n_1, 1:n_2, ..., n_m / 2+2:2 × (n_m / 2+1)).

k_1 = 0, 1, . . ., n_1 - 1,

k_2 = 0, 1, . . ., n_2 - 1,

. . .

k_m = 0, 1, . . ., n_m / 2,

Output. The real data $\{x_{j1j2...jm}\}$ is stored in X (1:n_1, 1:n_2, ..., 1:n_m).

One-dimensional array of size $n_1 \times n_2 \times ... \times (2 \times (n_m / 2+1))$, or *m*-dimensional array X (n_1, n_2, ..., $(2 \times (n_m / 2+1))$).

N..... Input. One-dimensional array of size M. n_i is stored in N (I) (I = 1, ..., M), where n_i is the size of the I-th dimension. If ISIN(I) is nonzero, N(I) must be able to be expressed as a product of powers of 2, 3, and 5. At least one of the first M-1 elements of N must be an even number.

M..... Input. The size of dimension *m* of the multivariate Fourier transform.

ISIN..... Input. One-dimensional array of size M.

ISIN (I) shows the direction r_i of the Fourier transform of each dimension.

ISIN (M) cannot be 0.

For ISIN = 1, r_i = 1.

For ISIN = 0, there is no transform.

For ISIN = -1, r_i = -1.

ISN..... Input.

For ISN = 1, normal transform (real to complex).

For ISN = -1, inverse transform (complex to real).

W..... Work area.

One-dimensional array of size $2 \times \max (n_1, n_2, ..., n_m) + n_1 \times n_2 \times ... \times n_{m-1} \times (2 \times (n_m / 2+1))$.

ICON..... Output. Condition code.

See Table VMRFT-1, “Condition codes.”

Table VMRFT-1 Condition codes

Code	Description	Processing
0	No error	-
30001	N(I) ≤ 0 or M < 2	Processing is stopped.
30008	For ISIN (I) ≠ 0, N (I) is not an integer expressed as a product of powers of 2, 3, and 5.	
30016	ISIN (I) < -1, ISIN (I) > 1, or ISIN (M) = 0	
30032	ISN ≠ 1 and ISN ≠ -1	Processing is stopped.
30512	The first M-1 elements of N are odd numbers.	

(3) Comments on use

a. Subprograms used

SSLII: UASSM, USEPR, UJOIN, USPLT, UCTRV, UCFS, UCF16, UCFT2, UCFT3, UCFT4, UCFT5, UCFT8, UFMRW, UCRU, UCTRF, MGSSL

b. Comments

1) General definition of Fourier transform

The multivariate discrete Fourier transform and inverse transform are generally defined as in (3.1) and (3.2).

$$\alpha_{k1k2\ldots km} = \frac{1}{n_1 n_2 \ldots n_m} \times \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \ldots \sum_{jm=0}^{nm-1} x_{j1j2\ldots jm} . \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \ldots \omega_{nm}^{-jmkm} \qquad (3.1)$$

$$k_1 = 0, 1, \ldots, n_1\text{-}1$$
$$, k_2 = 0, 1, \ldots, n_2\text{-}1$$
$$\ldots .$$
$$, k_m = 0, 1, \ldots, n_m\text{-}1$$

$$x_{j1j2\ldots jm} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \ldots \sum_{km=0}^{nm-1} \alpha_{k1k2\ldots km} . \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{nm}^{jmkm} \qquad (3.2)$$

$$j_1 = 0, 1, \ldots, n_1 \text{-}1$$
$$, j_2 = 0, 1, \ldots, n_2 \text{-}1$$
$$\ldots .$$
$$, j_m = 0, 1, \ldots, n_m \text{-}1$$

where

$$\omega_{n1} = \exp(2\pi i / n_1)$$
$$, \omega_{n2} = \exp(2\pi i / n_2)$$
$$\ldots .$$

$$, \ \omega_{nm} = \exp(2\pi i / n_m)$$

The subroutine calculates $\{n_1\ n_2..n_m\ \alpha_{k1k2..km}\}$ or $\{x_{j1j2..jm}\}$ corresponding to the left-hand terms of (1.1) and (1.2). For i, where ISIN (i) = 0, n_i is replaced with 1. If necessary, the user must normalize the results.

2) The result of the multivariate discrete real Fourier transform has the following complex conjugate relation:

$$\alpha_{k1\ k2\ ...\ km} = \overline{\alpha_{n1-k1\ n2-k2\ ...\ nm-km}}$$

$$k_1 = 0, ..., n_1 - 1$$

$$, k_2 = 0, ..., n_2 - 1$$

$$. . . .$$

$$, k_m = 1, ..., n_m / 2$$

In the case of k_i=0, n_i-k_i is regarded as 0.

For h, where ISIN (h) = 0, the h-th index in the right-hand terms is still k_h.

The rest of terms can be calculated using this relation.

c. Example of use

In this example, a two-dimensional real Fourier transform is calculated.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(N1=1024,N2=1024,M=2)
      PARAMETER(NS=2*(N2/2+1))
      DIMENSION X(N1,NS),N(M),W(2*N1+N1*NS),ISIN(M)
C
      DO 100 I=1,N2
      DO 100 J=1,N1
      X(J,I)=FLOAT(J)+FLOAT(N1)*(I-1)
  100 CONTINUE
C
      N(1)=N1
      N(2)=N2
      ISIN(1)=1
      ISIN(2)=1
      ISN= 1
C
C     REAL TO COMPLEX TRANSFORM
C
      CALL DVMRFT(X,N,M,ISIN,ISN,W,ICON)
      PRINT*,'ICON=',ICON
C
      N(1)=N1
      N(2)=N2
      ISIN(1)=-1
      ISIN(2)=-1
      ISN=-1
C
C     COMPLEX TO REAL TRANSFORM
C
      CALL DVMRFT(X,N,M,ISIN,ISN,W,ICON)
```

```
      PRINT*,'ICON=',ICON

      ERROR=0.0D0
      DO 200 I=1,N2
      DO 200 J=1,N1
      ERROR=MAX(ABS(X(J,I)/(N1*N2)-
     &    (FLOAT(J)+FLOAT(N1)*(I-1))),ERROR)
  200 CONTINUE
C
      PRINT*,'ERROR=',ERROR
      STOP
      END
```

A71-01-0101 VMVSD, DVMVSD

Multiplication of a real sparse matrix and a real vector (diagonal storage format)
CALL VMVSD (A, K, NDIAG, N, NOFST, NLB, X, Y, ICON)

(1) Function

This routine computes the product

$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$

of an $n \times n$ sparse matrix and a vector.

Sparse matrix $\boldsymbol{A}$ is stored using the diagonal storage format.

Vector $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-dimensional vectors.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Real-type, two-dimensional array of size A (K, NDIAG). Non-zero elements of the sparse matrix are stored in A (1 : N, NDIAG). For the diagonal storage format, see Part I, "Overview," Section 3.2.1.1, "Storage method for general sparse matrices, b., "Diagonal storage format for general sparse matrices."

K..... Input. Adjusted dimensions ($\geq n$) of array A.

NDIAG..... Input. The number of diagonals that contain non-zero elements of the coefficient matrix stored in matrix $\boldsymbol{A}$.
The size of the second dimension of A.

N..... Input. Order n of matrix $\boldsymbol{A}$.

NOFST..... Input. One-dimensional array NOFST(NDIAG). Stores the offset from the main diagonal corresponding to diagonals stored in A. Superdiagonals, are expressed as positive values. Subdiagonals are expressed as negative values.

NLB..... Input. Lower bandwidth of matrix $\boldsymbol{A}$.

X..... Input. Vector $\boldsymbol{x}$ is stored in X (NLB + 1 : NLB + N).
One-dimensional array of size $n + nlb + nub$.
nlb is the lower bandwidth. nub is the upper bandwidth.

Y..... Output. Stores the result of the multiplication of the matrix and the vector.
One-dimensional array of size n.

ICON..... Output. Condition code.
See Table VMVSD-1, "Condition codes."

Table VMVSD-1 Condition codes

Code	Description	Processing
0	No error	
30000	K < 1, N < 1, N > K, NDIAG < 1, or NLB ≠ MAX (- NOFST (I)), or | NOFST (I) | > N - 1	Processing is stopped.

(3) Comments on use

a. Subprogram used

SSL II: MGSSL

b. Comments

1) Comments on using the diagonal format

The diagonal elements outside of the coefficient Matrix ***A*** must be set to zero.

There is no special restriction on the order in which a diagonal vector column should be stored in array A.

The advantage of this method lies in the fact that the matrix vector product can be calculated without the use of indirect indices. The disadvantage is that matrices without the diagonal structure cannot be stored efficiently with this method.

c. Example of use

In this example using DVCGD, ***Ax*** is sought from matrix A, which does not store diagonal elements that are 1. ***b*** = (***A*** - ***E***) ***x*** + ***x***. For SET, see VCGD, DVCGD, item (3) "Comments on use," c., "Example of use."

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (N=51200,K=N+1)
       PARAMETER (NW=2,IWKS=4,N2=K+1)
       PARAMETER (NVW=K*(NW+6)+10)
       REAL*8 B(N),EPS,OMEGA,RZ,VW(NVW),X(N)
       INTEGER NDLT(NW)
       REAL*8 A(K,NW)
       INTEGER IVW(N2,IWKS)
C
C INITIALISE A
       CALL SET(A,NDLT,K,NW,N)
       ISHIFT=0
       DO 10 J=1,NW
       ISHIFT=MAX(ISHIFT,ABS(NDLT(J)))
   10 CONTINUE
C COMPUTE RHS SO AX=B SO WE KNOW SOLUTION X (X(I)=I)
       DO 30 I=1,N
   30 VW(I+ISHIFT)=I
C
C      B=(A-E)*X+X
       CALL DVMVSD(A,K,NW,N,NDLT,ISHIFT,VW,B,ICON)
       DO 70 I=1,N
       B(I)=B(I)+VW(I+ISHIFT)
   70 CONTINUE
```

```
C
      ITMAX=8*SQRT(N+0.1)
      EPS=1D-10
      OMEGA=0D0
      ISW=1
      IGUSS=0
      DO 100 IPC=1,3
      IF(IPC.EQ.3) OMEGA=0.98
      CALL DVCGD(A,K,NW,N,NDLT,B,IPC,ITMAX,ISW,OMEGA,
     &           EPS,IGUSS,X,ITER,RZ,VW,IVW,ICON)
      IF(ICON.NE.0) WRITE(6,*)'ICON=',ICON
      IF(RZ.LE.EPS) WRITE(6,41)'CONVERGED. ACCURACY=',RZ
      IF(RZ.GT.EPS) WRITE(6,41)'FAILED. ACCURACY=',RZ
      WRITE(6,*)'X'
      DO 60 I=1,MIN(N,16),4
   60 WRITE(6,42) I,(X(M),M=I,I+3)
  100 CONTINUE
   42 FORMAT(1X,I3,4(1X,F20.10))
   41 FORMAT(A,2X,E10.3)
      STOP
      END
```

A71-02-0101 VMVSE, DVMVSE

Multiplication of a real sparse matrix and a real vector (ELLPACK storage format)
CALL VMVSE (A, K, NW, N, ICOL, X, Y, ICON)

(1) Function

This routine computes the product

$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$

of an $n \times n$ sparse matrix and a vector.

The $n \times n$ coefficient matrix is stored using the ELLPACK storage format using two arrays.

$\boldsymbol{y}$ and $\boldsymbol{x}$ are n-dimensional vectors.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, NW).
For the ELLPACK storage format, see Part I, "Overview," Section 3.2.1.1, "Storage methods for general sparse matrices."

K..... Input. Size of adjustable dimensions ($\geq n$) of array *A* and ICOL.

NW..... Input. Maximum number of non-zero elements in each row vector of the matrix $\boldsymbol{A}$ stored in array A.
The size of the second dimension of ICOL and A.

N..... Input. Order n of matrix $\boldsymbol{A}$ stored in array A.

ICOL..... Input. Store column indices of the elements stored in the array A using the ELLPACK format, indicating which column vectors the corresponding elements in the array A belong to.
Two-dimensional array ICOL (K, NW).

X..... Input. Stores vector $\boldsymbol{x}$. One-dimensional array of size *n*.

Y..... Output. Stores the result of the multiplication of the matrix and the vector.
One-dimensional array of size *n*.

ICON..... Output. Condition code.
See Table VMVSE-1, "Condition code."

Table VMVSE-1 Condition codes

Code	Description	Processing
0	No error	
30000	K < 1, N ≤ 0, NW < 1, or N > K	Processing is stopped.

(3) Comments on use

a. Subprogram used

SSL II: MGSSL

b. Comments

1) Comments on using the ELLPACK storage format

Before storing data in the ELLPACK format, it is recommended to initialize each of the arrays A and ICOL with zero and the row number, respectively.

c. Example of use

In this example, when using DVCGE, $\boldsymbol{Ax}$ is sought through $\boldsymbol{b} = (\boldsymbol{A} - \boldsymbol{E})\,\boldsymbol{x} + \boldsymbol{x}$ by storing, in array A, elements other than the diagonal elements of matrix *A*, which are 1. For SET, see subroutine VCGE, DVCGE, (3) "Comments on use," c., "Example of use."

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (NW=2,N=51200,K=N+1)
       REAL*8 B(N),X(N),EPS,OMEGA,RZ,
      &      A(K,NW),VW(K*NW+4*N)
       INTEGER ICOL(K,NW),IVW(K*NW+4*N)
       WRITE(6,*) ' EXAMPLE DVCGE '
C INITIALISE A,ICOL
       CALL SET(A,ICOL,K,NW,N)
C GENERATE RHS B
       DO 10 I=1,N
   10 VW(I)=I
C COMPUTE RHS SO AX=B SO WE KNOW SOLUTION X (X(I)=I)
C
C B = (A-E)*X + E*X
       CALL DVMVSE(A,K,NW,N,ICOL,VW,B,ICON)
       PRINT*,'ERROR CODE =',ICON
       DO 20 I=1,N
       B(I)=B(I)+VW(I)
   20 CONTINUE
C
       ITMAX=4000
       EPS=1D-10
       ISW=1
       IGUSS=0
       DO 30 IPC=1,3
       IF(IPC.EQ.3)OMEGA=0.98
       CALL DVCGE(A,K,NW,N,ICOL,B,IPC,ITMAX,ISW,OMEGA
      &           ,EPS,IGUSS,X,ITER,RZ,VW,IVW,ICON)
C
       PRINT*,'ERROR CODE= ',ICON
       IF(RZ.LE.EPS) WRITE(6,41)'CONVERGED. ACCURACY=',RZ
       IF(RZ.GT.EPS) WRITE(6,41)'FAILED. ACCURACY=',RZ
       WRITE(6,*)'X'
       DO 60 I=1,MIN(N,16),4
   60 WRITE(6,42) I,(X(M),M=I,I+3)
   30 CONTINUE
   42 FORMAT(I3,4(F12.4))
   41 FORMAT(A,2X,E10.3)
```

```
STOP
END
```

A72-23-0101 VQMRD, DVQMRD

System of linear equations with unsymmetric or indefinite sparse matrix (QMR method, diagonal storage format)
CALL VQMRD (A, K, NDIAG, N, NOFST, AT, NTOFST, B, ITMAX, EPS, IGUSS, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations system with an $n \times n$ unsymmetric or indefinite sparse coefficient matrix using the quasi-minimal residual method (QMR).

$Ax = b$

Use two $n \times n$ coefficient matrices A and A^T. They are stored in the diagonal format method. Vectors b and x are n-dimensional vectors.

The iterative calculation may not be continued (break-down) because of the characteristics of the initial vector and coefficient matrices. This is because zero is obtained as the intermediate result in the recursive calculation formula. In this case, use the MGCR method that causes no break-down.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 "Iterative linear equation solvers and Convergence," in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, NDIAG). Stores coefficient matrix A in A (1:N, NDIAG) with a diagonal format.
For the diagonal storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices," b., "Diagonal Storage Format for General Sparse Matrices."

K..... Input. Size of adjustable dimension of array A.

NDIAG..... Input. The number of diagonal vectors in coefficient matrix A that contain non-zero elements.

N..... Input. Order n of matrix A.

NOFST..... Input. Stores offset from the main diagonal corresponding to diagonals stored in A. A superdiagonals have positive values; subdiagonals have negative values.
One-dimensional array NOFST (NDIAG).

AT..... Input. Stores non-zero elements of A^T.
Two-dimensional array AT (K, NDIAG). Stores coefficient matrix A^T in AT (1:N, NDIAG).
For the diagonal storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices," b., "Diagonal Storage Format for General Sparse Matrices."

NTOFST..... Input. Stores a offset from the main diagonal corresponding to a diagonal stored in array AT. Superdiagonals have positive values; subdiagonals have negative values.
One-dimensional array NOFST (NDIAG).

B..... Input. One-dimensional array of size n. Stores a constant vector of the right-hand side term of a linear equation system.

ITMAX..... Input. The upper limit of iteration steps in QMR method (> 0).

EPS..... Input. A criterion value used for convergence criterion.
If EPS is 0.0 or less, it is set to 10^{-6} in double-precision routines and 10^{-4} in single-precision routines.
(See item (3), "Comments on use," b., 1).)

IGUSS..... Input. Set control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=1: An approximate value of the solution vector is not specified.
IGUSS≠1: The iterative computation starts from the approximate value of the solution vector specified in array X.

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of a solution vector.
Output. The solution vector is stored.

ITER..... Output. The real number of iteration steps in QMR method.

VW..... Work area. One-dimensional array of size K × 9 + N + NBANDL + NBANDR. NBANDL indicates a lower bandwidth; NBANDR indicates an upper bandwidth.

ICON..... Output. Condition code
See Table VQMRD-1, "Condition codes."

Table VQMRD-1 Condition codes

Code	Meaning	Processing contents
0	No error	-
20000	Break-down occurred.	Processing is stopped.
20001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate value obtained up to this point in array X is output, but their precision cannot be guaranteed.
30000	N<1, K<1, K<N, NDIAG<1, K<NDIAG, or ITMAX ≤ 0	Processing is stopped.
32001	| NOFST (I) | > N-1, | NTOFST (I) | > N-1	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, URGWD, URIPA, URITI, URITT, URMVD, USSCP, URSTE, USVCN, UXVCP, USVSC, USVSU, USVUP, USVN2, URELT, MGSSL, UQMRR, UQMRD, UQBBM, UQITB

b. Comments

1) In the QMR method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of matrix *A* and EPS.

2) Notes on using the diagonal format

A diagonal vector element outside coefficient matrix *A* must be set to zero.

There is no restriction in the order in which diagonal vectors are stored in array A.

The advantage of this method lies in the fact the matrix vector multiplication can be calculated without the use of an indirect index. The disadvantage is that matrices without the diagonal structure cannot be stored efficiently with this method.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0, 1] \times [0, 1] \times [0, 1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of partial differential operator is described in Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them."

For INIT_MAT_ELL, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." INIT_SOL is the routine that generates the solution vectors to be sought in random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER MACH
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000)
      PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
      PARAMETER (NDIAG = 7,NVW = 9*K+N+400+400)
      REAL*8 A(K,NDIAG),AT(K,NDIAG),X(N),B(N),SOLEX(N)
     &       ,VW(NVW)
      INTEGER NOFST(NDIAG),NTOFST(NDIAG)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUSIONS'
      PRINT*,'X(1) = ',SOLEX(1),'X(N) = ',SOLEX(N)
C
      PRINT *
      PRINT *,'    QMR METHOD'
      PRINT *,'    DIAGONAL FORMAT'
C
      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
```

```
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,A,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      CALL INIT_MAT_TR_DIAG(VA1,VA2,VA3,VC,AT,NTOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      CALL GET_BANDWIDTH_DIAG(NOFST,NDIAG,NBANDL,NBANDR)
         DO 110 I = 1,N
            VW(I+NBANDL) = SOLEX(I)
110      CONTINUE
      CALL DVMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON)
      PRINT*,'DVMVSD ICON = ',ICON
C
      EPS = 1D-10
      IGUSS = 0
      ITMAX = 2000
      CALL DVQMRD(A,K,NDIAG,N,NOFST,AT,NTOFST,B,ITMAX
     &           ,EPS,IGUSS,X,ITER,VW,ICON)
C
      PRINT* ,'ITER = ',ITER
      PRINT* ,'DVQMRD ICON = ',ICON
      PRINT*, 'COMPUTED VALUES'
      PRINT*, 'X(1) = ',X(1),'X(N) = ',X(N)
      STOP
      END
```

(4) Method

For the QMR method, see [13].

A72-24-0101 VQMRE, DVQMRE

System of linear equations with unsymmetric or indefinite sparse matrix (QMR method, ELLPACK storage format)
CALL VQMRE (A, K, IWIDT, N, ICOL, AT, IWIDTT, ICOLT, B, ITMAX, EPS, IGUSS, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations with an $n \times n$ unsymmetric or indefinite sparse coefficient matrix using the quasi-minimal residual method (QMR).

$Ax = b$

Use two $n \times n$ coefficient matrices A and A^T. They are stored in the ELLPACK format method. Vectors b and x are n-dimensional vectors.

The iterative calculation may not be continued (break-down) because of the characteristics of the initial vector and coefficient matrices. This is because zero is really obtained as the intermediate result although non-zero is desired in the recursive calculation formula. In this case, use the MGCR method that causes no break-down.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 "Iterative linear equation solvers and Convergence," in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, IWIDT).
For details on the ELLPACK storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices," b., "Diagonal Storage Format for General Sparse Matrices."

K..... Input. Size of adjustable dimension of A and ICOL. ($\geq n$)

IWIDT..... Input. The maximum number of non-zero elements in row vector direction of the coefficient matrix A.
Two-dimensional size of A and ICOL.

N..... Input. Order n of matrix A.

ICOL..... Input. Stores the column indices of the elements stored in the array A using the ELLPACK format, indicating which column vectors the corresponding elements in the array A belong to.
Two-dimensional array ICOL (K, IWIDT)

AT..... Input. Stores non-zero elements of a transposed coefficient matrix A^T in AT (1:N:IWIDTT).
Two-dimensional array AT (K, IWIDTT).
For details on the ELLPACK storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices," b., "Ellpack Storage Format for General Sparse Matrices."

IWIDTT..... Input. The maximum number of non-zero elements in row vector direction of the transposed coefficient matrix A^T.

ICOLT..... Input. Store the column indices of the element stored in the array AT using the ELLPACK format, indicating which, column vectors the corresponding

elements in the array AT belong to.
Two-dimensional array ICOLT (K, IWIDTT)

B..... Input. One-dimensional array of size *n*. Stores a constant vector specified in the right-hand side term of a linear equation system in B.

ITMAX..... Input. The upper limit of iterations in QMR method (> 0).

EPS..... Input. A criterion value used for convergence criterion.
If EPS is 0.0 or less, it is set to 10^{-6} for double-precision routines and 10^{-4} for single-precision routines.
(See item (3), "Comments on Use," b., 1).)

IGUSS..... Input. Control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=0: An approximate value of the solution vector is not specified.
IGUSS≠0: The iterative computation starts from the approximate value of the solution vector specified in array X.

X..... Input. One-dimensional array of size *n*. Can specified the approximate value of a solution vector.
Output. The solution vector is stored.

ITER..... Output. The real number of iteration steps in QMR method.

VW..... Work area. One-dimensional array of the size $K \times 12$.

ICON..... Output. Condition code
See Table VQMRE-1, "Condition codes."

Table VQMRE-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	Break-down occurred.	Processing is stopped.
20001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
30000	K<1, N<1, K<N, IWIDT<1, IWIDTT<1, K<IWIDT, K<IWIDTT, or ITMAX ≤ 0	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, URIPA, URITI, URITT, URMVE, USSCP, URSTE, USVCN, USVCP, USVSC, USVSU, USVUP, USVN2, URELT, MGSSL, UQMRR, UQMRE, UQBBM, UQITB

b. Comments

1) In the QMR method, if the residual Euclidean norm is equal to or less than the product of initial residual Euclidean norm and EPS, it is judged as having converged.

The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of matrix A and EPS.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0,1] \times [0,1] \times [0,1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of partial differential operator is described in Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." For INIT_MAT_ELL, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." INIT_SOL is the routine that generates the solution vectors to be sought in random numbers.

```
C      **EXAMPLE**
       PROGRAM TEST_ITER_SOLVERS
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (MACH = 0)
       PARAMETER (K = 10000)
       PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
       PARAMETER (IWIDT = 7,IWIDTT = IWIDT,NVW = K*12)
       REAL*8 A(K,IWIDT),AT(K,IWIDTT),X(N),B(N),SOLEX(N)
      &       ,VW(NVW)
       INTEGER ICOL(K,IWIDT),ICOLT(K,IWIDTT)
C
       CALL INIT_SOL(SOLEX,N,1D0,MACH)
       PRINT*,'EAPECTED SOLUTION'
       PRINT*,'X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
C
       PRINT *
       PRINT *,'    QMR METHOD'
       PRINT *,'    ELLPACK FORMAT'
C
       VA1 = 3D0
       VA2 = 1D0/3D0
       VA3 = 5D0
       VC = 5D0
       XL = 1.0
       YL = 1.0
       ZL = 1.0
C
       CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL
      &         ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)
       CALL INIT_MAT_TR_ELL(VA1,VA2,VA3,VC,AT,ICOLT
      &         ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)
       CALL DVMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON)
       PRINT*,'DVMVSE ICON = ',ICON
C
       EPS = 1D-10
       IGUSS = 0
       ITMAX = 800
```

```
      CALL DVQMRE(A,K,IWIDT,N,ICOL,AT,IWIDTT,ICOLT,B,ITMAX
     &           ,EPS,IGUSS,X,ITER,VW,ICON)
C
      PRINT*,'DVQMRE ICON = ',ICON
      PRINT*,'COMPUTED VALUE'
      PRINT*,'X(1) = ',X(1),' X(N) = ',X(N)
      STOP
      END
```

(4) Method

For QMR method, see [13].

F15-31-0401 VRPF3, DVRPF3

Three-dimensional prime factor discrete real Fourier transform
CALL VRPF 3 (A, L, M, N, ISN, VW, ICON)

(1) Function

When three-dimensional (where the size of each dimension is $N1$, $N2$, $N3$) real time series data $\{x_{J1,J2,J3}\}$ is given, this subroutine performs a discrete real Fourier transform or its inverse transform by using the prime factor Fourier transform (prime factor FFT). The size of each dimension must satisfy the following conditions:

- The size must be expressed as a product of mutual prime factor p, selected from the following numbers:

  factor p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)

- The size of the first dimension must be an even number ($2 \times I$), where I satisfies the previous condition.

Calling this subroutine with N = 1 sets a two-dimensional real prime factor fast Fourier transform. Calling this subroutine with N = 1 and M = 1 sets a one-dimensional real prime factor fast Fourier transform.

1) Three-dimensional real Fourier transform

By inputting $\{x_{J1,J2,J3}\}$ and performing the transform defined in (1.1), the three-dimensional Fourier transform looks for $\{N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}\}$.

$$N1 \times N2 \times N3 \times \alpha_{K1,K2,K3} = \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} \cdots \sum_{J3=0}^{N3-1} x_{J1,J2,J3} \omega_1^{-J1.K1} \omega_2^{-J2.K2} \omega_3^{-J3.K3} \qquad (1.1)$$

, $K1 = 0, 1, \ldots, N1-1$

, $K2 = 0, 1, \ldots, N2-1$

, $K3 = 0, 1, \ldots, N3-1$

, $\omega_j = exp(2\pi i/Nj)$, $j = 1, 2, 3$

For a three-dimensional real Fourier transform, approximately half of the computation is performed because $\{x_{J1,J2,J3}\}$ is a real number. That is, for one-dimensional transforms, $K1$ is computed from 0 to $N1/2$.

2) Three-dimensional real Fourier inverse transform

By inputting $\{\alpha_{K1,K2,K3}\}$ and performing the transform defined in (1.2), a three-dimensional Fourier inverse transform looks for $\{x_{J1,J2,J3}\}$.

$$x_{J1,J2,J3} = \sum_{K1=0}^{N1-1} \sum_{K2=0}^{N2-1} \cdots \sum_{K3=0}^{N3-1} \alpha_{K1,K2,K3} \omega_1^{J1.K1} \omega_2^{J2.K2} \omega_3^{J3.K3} \qquad (1.2)$$

, $J1 = 0, 1, \ldots, N1-1$

, $J2 = 0, 1, \ldots, N2-1$

, $J3 = 0, 1, \ldots, N3-1$

, $\omega_j = exp(2\pi i/Nj)$, $j = 1, 2, 3$

For a three-dimensional real Fourier inverse transform, $\{\alpha_{K1,K2,K3}\}$ looks for only *K1* = 0, 1, ..., *N1*/2. Compute the two-dimensional and three-dimensional $\sum$ first. Then, use the conjugate relation for the elements of the first dimension to compute (1.2).

(2) Parameters

A..... Input. Real number $\{x_{J1,J2,J3}\}$ or Fourier transformed complex number $\{\alpha_{K1,K2,K3}\}$.
Output: Fourier transformed complex number $\{\alpha_{K1,K2,K3}\}$ or inverse transformed real number $\{x_{J1,J2,J3}\}$.
Three-dimensional array A (L, M, N).
For a real number (transform input and inverse transform output), data is stored in A (L, M, N).

L - 2 is the number of data items of the first dimension, where L - 2 is an even number.
M is the number of data items of the second dimension. N is the number of data items of the third dimension.
For a complex number (transform output and inverse transform input), the real part is stored in the first half and the imaginary part is stored in the second half of the same array.

PARAMETER (L =, M =, N =, LH = L/2)

DIMENSION A (L, M, N), B (LH, M, N, 2)

EQUIVALENCE (A, B)

Real part B (L/2, M, N, 1) and Imaginary part B (L/2, M, N, 2) are stored separately in the contiguous area. (See item (3), “Comments on use,” b., 3).)

L..... Input. Number of data items + 2 of the first dimension. The number of data items of the first dimension must be an even number.
$(L - 2)/2 \leq 5{,}040$

M..... Input. Number of data items of the second dimension.
$M \leq 5{,}040$

N..... Input. Number of data items of the third dimension.
$N \leq 5{,}040$

ISN..... Input. Specifies either transform or inverse transform.
Transform if ISN ≥ 0 (non-negative integer).
Inverse transform if ISN < 0 (negative integer).

VW..... Work area. Three-dimensional array with the same size as A.

ICON..... Output. Condition code.
See Table VRPF3-1, “Condition codes.”

Table VRPF3-1 Condition codes

Code	Description	Processing
0	No error	
20000	(L - 2)/2, M or N exceeded 5,040. Or this could not be factored into the multiplication of the mutually prime factor in {2, 3, 4, 5, 7, 8, 9, 16}.	Processing stopped.
30000	L - 2 was not an even number. Or L, M, or N is zero or a negative value.	

(3) Comments on use

a. Subprograms used

SSL II: UINI1, UINI2, UTER1, UTER2, UTRSP, UPFT1, UPFT2, UTRR1, UTRR2, MGSSL

b. Comments

1) General definition of three-dimensional Fourier transform

The three-dimensional Fourier transform and its inverse transform are generally defined in (3.1) and (3.2).

$$\alpha_{K1,K2,K3} = \frac{1}{N1 \times N2 \times N3} \sum_{J1=0}^{N1-1} \sum_{J2=0}^{N2-1} \cdots \sum_{J3=0}^{N3-1} x_{J1,J2,J3} \omega_1^{-J1.K1} \omega_2^{-J2.K2} \omega_3^{-J3.K3} \quad (3.1)$$

$$x_{J1,J2,J3} = \sum_{K1=0}^{N1-1} \sum_{K2=0}^{N2-1} \cdots \sum_{K3=0}^{N3-1} \alpha_{K1,K2,K3} \omega_1^{J1.K1} \omega_2^{J2.K2} \omega_3^{J3.K3} \quad (3.2)$$

The subroutine looks for $\{N1 \times N2 \times N3 \times \alpha_{K1,K2,K3}\}$ or $\{x_{J1,J2,J3}\}$ corresponding to the left-hand-side terms of (3.1) and (3.2), respectively. The user must normalize the results, if necessary. If the transform and inverse transform is executed by calling this subroutine consecutively without being normalized, each element of the input data is multiplied by $N1$, $N2$, or $N3$, respectively, and then output.

2) Number of terms

The number of terms is expressed as a product of a mutually prime factor p, selected from the numbers listed as follows.

The maximum number is $5 \times 7 \times 9 \times 16 = 5{,}040$.

factor p ($p \in \{2, 3, 4, 5, 7, 8, 9, 16\}$)

The number of terms of the first dimension can be a value up to a multiple of the mutually prime numbers listed previously.

3) Data storage method

The real data $\{x_{J1,J2,J3}\}$ are stored in a three-dimensional array A. The number of terms N1 of the first dimension is equivalent to L - 2. The terms are stored from 1 up to L - 2.

The real and imaginary parts of the complex data $\{\alpha_{K1,K2,K3}\}$ are stored as an array divided into two contiguous regions by splitting array A. The number of indices of $K1$ of the first dimension is $N1/2 + 1$ (L/2), from zero up to $N1/2$.

The real part is stored in B (LH, M, N, 1), and the imaginary part is stored in B (LH, M, N, 2) where:

LH = L/2

and

PARAMETER (L =, M =, N =, LH = L/2)

DIMENSION A (L, M, N), B (LH, M, N, 2)

EQUIVALENCE (A, B)

c. Example of use

In this example, real time series data $\{x_{J1,J2,J3}\}$ of terms $N1$, $N2$ and $N3$ are input, and a Fourier transform is performed. The results are used to perform a Fourier inverse transform to look for $\{x_{J1,J2,J3}\}$.

Here $N1 = 12$, $N2 = 12$, and $N3 = 12$.

```
C     **EXAMPLE**
      DIMENSION A(12+2,12,12),B(6+1,12,12,2),NI(3)
      DIMENSION VW(12+2,12,12)
      DATA      NI/12,12,12/,L,M,N/14,12,12/
      EQUIVALENCE (A,B)
      READ(5,500) (((A(I,J,K),I=1,NI(1)),
     *             J=1,NI(2)),K=1,NI(3))
      WRITE(6,600) (NI(I),I=1,3),
     *            (((I,J,K,A(I,J,K),I=1,NI(1)),
     *             J=1,NI(2)),K=1,NI(3))
C     NORMAL TRANSFORM
      CALL VRPF3(A,L,M,N,1,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.NE.0) STOP
C     INVERSE TRANSFORM
      CALL VRPF3(A,L,M,N,-1,VW,ICON)
      NT=NI(1)*NI(2)*NI(3)
      DO 10 K=1,NI(3)
      DO 10 J=1,NI(2)
      DO 10 I=1,NI(1)
      A(I,J,K)=A(I,J,K)/FLOAT(NT)
   10 CONTINUE
      WRITE(6,620) (((I,J,K,A(I,J,K),I=1,NI(1)),
     *              J=1,NI(2)),K=1,NI(3))
      STOP
  500 FORMAT(E20.7)
  600 FORMAT('0',10X,'INPUT DATA',5X,
     *       '(',I3,',',I3,',',I3,')'/
     *      (15X,'(',I3,',',I3,',',I3,')',
     *       E20.7))
  610 FORMAT('0',10X,'RESULT ICON=',I5)
  620 FORMAT('0',10X,'OUTPUT DATA'/
     *      (15X,'(',I3,',',I3,',',I3,')',
     *       E20.7))
      END
```

(4) Method

The three-dimensional real Fourier transform is performed by using the fast Fourier transform with the factored mutual prime factor as the radix (prime factor FFT).

1) Three-dimensional transform

The three-dimensional transform defined in (1.1) can be performed in the order shown in (4.1) by simplifying common terms. The order for obtaining the sum of *J*1, *J*2, and *J*3 can also be replaced.

$$N1\times N2\times N3\times\alpha_{K1,K2,K3} = \sum_{J1=0}^{N1-1}\omega_1^{-J1.K1}\times\sum_{J2=0}^{N2-1}\omega_2^{-J2.K2}\sum_{J3=0}^{N3-1}x_{J1,J2,J3}\omega_3^{-J3.K3} \qquad (4.1)$$

In (4.1), $\sum_{J3}$ takes $N1 \times N2$ sets of one-dimensional transforms of data item $N3$. $\sum_{J2}$ takes $N1 \times N3$ sets of one-dimensional transforms of data item N2. $\sum_{J1}$ takes $N2 \times N3$ sets of one-dimensional transforms of data item $N1$.

In order to perform a one-dimensional transform for each dimension, this subroutine applies the fast Fourier transform with the mutually prime factor as the radix.

2) Real transform

Because the number of data items is even for the real Fourier transform of the first dimension, complex Fourier transform is computed from zero up to *N1/2* for *K1*. The remaining Fourier transform computation need not be performed due to the complex conjugate relation.

Assume a one-dimensional discrete real Fourier transform of N data items.

$$\alpha_K = \sum_{J=0}^{N-1} x_J \exp(-2\pi iKJ / N) \qquad (4.2)$$

K = 0, ..., N-1

$$\alpha_{N-K} = \alpha_K^* \qquad (4.3)$$

Even if the sequence for obtaining $\sum$ in (4.1) is computed from the first dimension, the results are the same. Therefore, the computation in (4.2) of the one-dimensional transform with fixed terms of second and third dimensions can be performed for K = 0, 1, ..., N1/2.

For a real transform with an even number of data items, a complex transform can be used in the computation of the one-dimensional transform in (4.2). For details, refer to *FUJITSU SSL II User's Guide (Scientific Subroutine Library)* under the "Method" section in RFT.

In addition, the relationship that follows applies to a three-dimensional real Fourier transform. This can be used for looking for other coefficients.

$$\alpha_{K1,K2,K3} = \alpha^*_{N1-K1,N2-K2,N3-K3} \qquad (4.4)$$

3) Prime factor fast Fourier transform

The three-dimensional real Fourier transform can be calculated by performing a multiple set of one-dimensional Fourier transforms three times. The one-dimensional Fourier transforms are performed by using the prime factor fast Fourier transform (prime factor FFT). For an explanation of prime factor fast Fourier transform, see item 2), “Prime factor fast Fourier transform,” under the Method section in VCPF3. For more information, see [6] and [45].

B71-14-0101 VSEVP,DVSEVP

Eigenvalues and eigenvectors of a real symmetric matrix (tridiagonalization, multisection method, and inverse iteration)
CALL VSEVP (A, K, N, NF, NL, IVEC, ETOL, CTOL, NEV, E, MAXNE, M, EV, VW, IW, ICON)

(1) Function

This subroutine calculates specified eigenvalues and, optionally, eigenvectors of *n*-dimensional real symmetric matrix ***A***.

First, the matrix is reduced to tridiagonal form using the Householder reductions. Then, the specified eigenvalues are obtained by the multisection method. The eigenvectors are obtained by the inverse iteration.

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \qquad (1.1)$$

where, ***A*** is an $n \times n$ real symmetric matrix.

(2) Parameters

A ..... Input. Real symmetric matrix ***A*** is stored in A(1:N,1:N).
After calculation, the value of A is not assured.
Two-dimensional array A(K,N).

K ..... Input. Size of first-dimension of array A. (K ≥ N).

N ..... Input. Order *n* of real symmetric matrix ***A***

NF ..... Input. Number assigned to the first eigenvalue to be acquired by numbering eigenvalues in ascending order. (Multiple eigenvalues are numbered so that one number is assigned to one eigenvalue.)

NL ..... Input. Number assigned to the last eigenvalue to be acquired by numbering eigenvalues in ascending order. (Multiple eigenvalues are numbered so that one number is assigned to one eigenvalue.)

IVEC ..... Input. Control information.

IVEC=1: Both the eigenvalues and corresponding eigenvectors are sought.
IVEC≠1: Only the eigenvalues are sought.

ETOL ..... Input. A criterion value required to determine whether an eigenvalue is distinct or numerically multiple based on expression (3.4). The default value is 3.0D-16 for double precision (2.0D-7 for single precision) when this value is set to less than.

CTOL ..... Input. A criterion value required to determine whether adjacent eigenvalues are approximately multiple i.e. clustered according to expression (3.1). CTOL ≥ ETOL
When CTOL is less than ETOL, CTOL = ETOL is set.
(See 1) in b, "Notes," in (3), "Comments on use.")

NEV ..... Output. Number of eigenvalues calculated.

The detail information is as follows:
NEV (1) indicates the number of distinct eigenvalues.
NEV (2) indicates the number of distinct clusters.

NEV (3) indicates the total number of eigenvalues including multiplicities.
One-dimensional array NEV (3).

E ..... Output. Eigenvalues are stored in E.
The eigenvalues are stored in E(1:NEV(3)).

One-dimensional array E(MAXNE).

MAXNE .... Input. The maximum number of eigenvalues that can be computed.
Dimension of array E.

When NEV (3) is greater than MAXNE, eigenvectors cannot be computed.
(See 2) in b, “Notes,” in (3), “Comments on use.”)

M ..... Output. Information about the multiplicity of the computed eigenvalues.
M (*i*, 1) indicates the multiplicity of the *i*-th eigenvalue λ_i. M (*i*, 2) indicates the size of the *i*-th cluster of eigenvalues.
(See 1) in b, “Notes,” in (3), “Comments on use.”)

Two-dimensional array M(MAXNE,2).

EV ..... Output. When IVEC = 1, the eigenvectors corresponding to the computed eigenvalues are returned in EV (1:N, 1:NEV(3)).
Two-dimensional array EV (K, MAXNE).

VW ..... Work area. One-dimensional array of size $17 \times K$

IW ..... Work area. One-dimensional array of size $9 \times$ MAXNE+128

ICON ..... Output. Condition code.

See Table VSEVP-1.

Table VSEVP-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	During calculation of clustered eigenvalues, the total number of eigenvalues exceeded the value of MAXNE.	Processing is discontinued. The eigenvectors cannot be calculated, but the different eigenvalues themselves are already calculated. (See 2) in b, “Notes,” in (3), “Comments on use.”)
30000	NF < 1, NL > N, NL < NF, N < 1, K < N, or MAXNE < NL - NF + 1.	Processing is discontinued.
30100	The input matrix may not be a symmetric matrix.	

(3) Comments on use

a. Subprograms used

SSLII ..... UIBBS, UIBFC, UIBFE, UIBSL, UITBS, UITFC, UITFE, UITSL, USEVP, UTDEX, UTDEY, UTMLS, UTRBK, UTRVM, UZRDM, MGSSL, UMGSL, UMGSL2

b. Notes

1) This routine pays special attention to a clustered eigenvalue.

With ε is equal to ETOL, suppose that the eigenvalues $\lambda_j, j = s, s+1, \ldots$, and $s+k$ ($k \geq 0$) are such that

$$\frac{|\lambda_i - \lambda_{i-1}|}{1+\max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon \qquad (3.1)$$

While (3.4) is not satisfied for $i = s-1$ and $i = s + k + 1$, then eigenvalues λ_j, $j = s - 1, s, \ldots, s + k$ are considered to be identical, i.e., a single multiple eigenvalue of multiplicity $k + 1$.

The default value of ETOL is 3.0D-16 for double precision (2.0D-7 for single-precision). Using this value, eigenvalues are refined to machine precision.

When (3.1) is not satisfied for ε = ETOL , λ_{i-1} and λ_i are assumed to be distinct eigenvalues.

If (3.1) holds with ε = CTOL (but not with ε = ETOL) for eigenvalues λ_m, $m=t, t+1, \ldots, t+k$ but not for λ_{t-1} and λ_{t+k+1}, these eigenvalues are considered to be approximately multiple, i.e. clustered, though distinct (not numerically multiple). In order to obtain an invariant subspace, eigenvectors corresponding to clustered eigenvalues are computed using orthogonal starting vectors and are reorthogonalized. Of course CTOL ≥ ETOL; if this condition is not satisfied, CTOL is set to be equal to ETOL.

2) Assume *r* eigenvalues are requested. Note that if the first or last requested eigenvalue has a multiplicity greater than 1 then more than *r* eigenvalues, are obtained. The corresponding eigenvectors can be computed only when the corresponding eigenvector storage area is sufficient.

The maximum number of computable eigenvalues can be specified in MAXNE. If the total number of eigenvalues exceeds MAXNE, ICON = 20000 is returned. The corresponding eigenvectors cannot be computed. In this case, the eigenvalues are returned, but they are not stored repeatedly according to multiplicities.

When all eigenvalues are distinct, it is sufficient to set MAXNE = NL-NF+1. When the total number of eigenvalues to be sought exceeds MAXNE, the necessary value for MAXNE for seeking eigenvalues again is returned in NEV(3).

3) This routine is faster than SEIG1, SEIG2 and VSEG2.

c. Example

This example calculates the specified eigenvalues and eigenvectors of a real symmetric matrix whose eigenvalues and eigenvectors are already obtained.

```
C     ** EXAMPLE PROGRAM **
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (K=500,N=K,NF=1,NL=100,MAXNE=NL-NF+1)
      PARAMETER (NVW=15*K,NIW=9*MAXNE+128)
      REAL*8    A(K,N),AB(K,N)
      REAL*8    E(K),EV(K,MAXNE),VW(NIW)
```

```
      REAL*8    VV(K,N)
      INTEGER   IW(NVW),M(MAXNE,2),NEV(3)
C
      ETOL=3.0D-16
      CTOL=5.0D-12
      IVEC=1

C Generate real symmetric matrix with known eigenvalues
C Initialization
      PI = 4.0D0*DATAN(1.0D0)
      DO 1 J=1,N
             DO 11 I=1,N
             VV(I,J)=DSQRT(2.0D0/DBLE(N+1))*SIN(DBLE(I)*PI
     &                    *DBLE(J)/ DBLE(N+1))
               A(I,J)=0.0D0
 11          CONTINUE
  1   CONTINUE

      DO 22 J=1,N
         A(J,J) = DBLE(-N/2+J)
 22   CONTINUE

      WRITE (6,*)' Input matrix size is ',N
      WRITE (6,*)' Matrix calculations use k =',K
      WRITE (6,*)' Desired eigenvalues are nf to nl ',NF,NL
      WRITE (6,*)' That is, request ',NL-NF+1,
     &           ' eigenvalues.'
      WRITE (6,*)' True eigenvalues are as follows'
      WRITE (6,*)(A(J,J),J=NF,NL)

      CALL DVMGGM(A,K,VV,K,AB,K,N,N,N,ICON)
      CALL DVMGGM(VV,K,AB,K,A,K,N,N,N,ICON)

C Calculate the eigendecomposition of A
      CALL DVSEVP(A,K,N,NF,NL,IVEC,ETOL,CTOL,NEV,
     &                  E,MAXNE,M,EV,VW,IW,ICON)
      IF (ICON.GT.0) THEN
          WRITE (*,*)' VSEVP failed with parameter ',
     &               'icon=',ICON
          STOP
      ENDIF

      WRITE (*,*)' Number of eigenvalues ',
     &                  NEV(3)
      WRITE (*,*)' Number of distinct eigenvalues ',
     &                  NEV(1)
      WRITE (*,*)' Solution to eigenvalues '
      WRITE (*,*)' E  ',(E(I),I=1,NEV(3))
 299  CONTINUE

      STOP
      END
```

(4) Method

This routine solves an eigenvalue problem of a tridiagonal matrix created from a real symmetric matrix. The reduction to a tridiagonal form is the Householder reduction.

The eigenvalue problem of a tridiagonal matrix is calculated using multisectioning to find the eigenvalues and inverse iteration for the eigenvectors. For details, see “VTDEV” and see [32] in Appendix B, “References.”

The eigenvectors of the original matrix are found by multiplying the matrix of eigenvectors of the tridiagonal matrix by the matrix of transformations carried out in the reduction to the tridiagonal.

F17-11-0301 VSRFT, DVSRFT

One-dimensional and multiple discrete real Fourier transform (mixed radices of 2, 3, and 5)
CALL VSRFT (X, M, N, ISIN, ISN, W, ICON)

(1) Function

This subroutine performs one-dimensional discrete real Fourier transforms (for *m* multiplicity).

n must be a number expressed as a product of powers of 2, 3, and 5, where n is the size of the data to be transformed.

1) Transform

Inputting $\{x_{k1j2}\}$ and performing the transform defined in (1.1), obtains $\{n\ \alpha_{k1k2}\}$.

$$n\alpha_{k1k2} = \sum_{j2=0}^{n-1} x_{k1j2}.\omega_n^{-j2k2r} \qquad (1.1)$$

$\omega_n = exp(2\pi i/n)$

r = 1 or r = -1 can be specified for the transform direction.

k_1 = 0, 1, ..., m-1,

k_2 = 0, 1, ..., n-1

2) Inverse transform

Inputting $\{\alpha_{k1k2}\}$ and performing the transform defined in (2.1) obtains $\{x_{k1j2}\}$.

$$x_{k1j2} = \sum_{k2=0}^{n-1} \alpha_{k1k2}.\omega_n^{j2k2r} \qquad (1.2)$$

$\omega_n = exp(2\pi i/n)$

In the inverse transform, the direction inverse to that specified in the transform must be specified.

r = -1 or r = 1

k_1 = 0, 1, ..., m-1,

j_2 = 0, 1, ..., n-1

The result of the real Fourier transform has a complex conjugate relation. The first n/2 + 1 elements of k_2 in $\{n\ \alpha_{k1k2}\}$ are stored. Either m or n must be an even number.

(2) Parameters

X..... For ISN = 1 (transform from real to complex):

Input. Real data $\{x_{k1j2}\}$ is stored in X (1:m, 1:n).

Output. The real part of $\{n\ \alpha_{k1k2}\}$ is stored in X (1:m, 1:n/2 + 1). The imaginary part of $\{n\ \alpha_{k1k2}\}$ is stored in X (1:m, n/2 + 2:2 × (n/2 + 1)).

k_1 = 0, 1, ..., m-1,

k_2 = 0, 1, ..., n/2

For ISN = -1 (transform from complex to real):

Input. The real part of $\{\alpha_{k1k2}\}$ is stored in X (*1*:m, 1:*n*/2 + 1). The imaginary part of $\{\alpha_{k1k2}\}$ is stored in X (1:*m*, *n*/2 + 2:2 × (*n*/2 + 1)).

k_1 = 0, 1, ..., *m*-1,

k_2 = 0, 1, ..., *n*/2

Output. The real data $\{x_{k1j2}\}$ is stored in X (1:*m*, 1:*n*).

Two-dimensional array X (*m*, (*n* + 4 × *int* ($\sqrt{n/2}$))).

X (*m*, 2 × (*n*/2 + 1) + 1:*n* + 4 × *int* ($\sqrt{n/2}$)) is used internally. The operation result is not guaranteed.

M..... Input. *m* of the multiplicity (number of data items) for which one-dimensional discrete real Fourier transform is performed. Either *m* or *n* must be an even number.

N..... Input. *n* of the size of data on which the one-dimensional discrete real Fourier transform is performed. *n* is a number expressed as a product of powers of 2, 3, and 5. Either *m* or *n* must be an even number.

ISIN..... Input. Fourier transform direction r.

For ISIN = 1, *r* = 1.

For ISIN = -1, *r* = -1.

ISN..... Input.

For ISN = 1, normal transform (real to complex).

For ISN = -1, inverse transform (complex to real).

W..... Work area.

One-dimensional array of size 2 × *n* + *m* × (*n* +4 × *int* ($\sqrt{n/2}$)).

ICON..... Output. Condition code.

See Table VSRFT-1, “Condition codes.”

Table VSRFT-1 Condition codes

Code	Description	Processing
0	No error	-
30001	M ≤ 0 or N ≤ 0	Processing is stopped.
30008	N is not an integer expressed as a product of powers of 2, 3, and 5.	
30016	ISIN ≠ 1 and ISIN ≠ -1	
30032	ISN ≠ 1 and ISN ≠ -1	
30512	Both M and N are odd numbers.	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSLII: UASSM, UCTSV, USEPR, UFIX, UJOIN, USPLT, UUFIX, USTUP, UCFS, UCF16, UCFT2, UCFT3, UCFT4, UCFT5, UCFT8, UFMRW, UCRU, UCTRF, MGSSL

b. Comments

1) General definition of Fourier transform

The multiple discrete Fourier transform and inverse transform are generally defined as in (3.1) and (3.2).

$$\alpha_{k1k2} = \frac{1}{n}\sum_{j2=0}^{n-1} x_{k1j2}\omega_n^{-j2k2} \qquad (3.1)$$

$$k_1 = 0, 1, ..., m\text{-}1$$

$$, k_2 = 0, 1, ..., n\text{-}1$$

$$x_{j1j2} = \sum_{k2=0}^{n-1} \alpha_{j1k2}\omega_n^{j2k2} \qquad (3.2)$$

$$j_1 = 0, 1, ..., m\text{-}1$$

$$, j_2 = 0, 1, ..., n\text{-}1$$

where

$$\omega = \exp(2\pi i / n)$$

The subroutine calculates n α_{k1k2} or x_{j1j2} corresponding to the left-hand terms of (1.1) and (1.2). If necessary, the user must normalize the results.

2) The result of the multiple discrete real Fourier transform has the following complex conjugate relation:

$$\alpha_{k1k2} = \overline{\alpha_{k1n-k2}}$$

$$k_1 = 0, ..., m\text{-}1$$

$$, k_2 = 1, ..., n/2$$

The rest of terms can be calculated using this relation.

3) Two methods are used, one for when n is an even number and one for when m is an even number. The method for when n is even has the vector length is about $m\sqrt{n}$. The method for when m is even has vector length $m/2$, but performs less data movement.

The routine performs transforms at maximum speed when m is a large even number.

c. Example of use

In this example, a one-dimensional real FFT of multiplicity m = 500 is calculated.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
```

```
      PARAMETER(M=500,N=2**10)
      PARAMETER(N2=N+4*22)
      DIMENSION X(M,N2),W(2*N+M*N2)
C
      DO 100 J=1,M
      DO 100 I=1,N
      X(J,I)=FLOAT(I)+FLOAT(N)*(J-1)
  100 CONTINUE
C
      ISIN=1
      ISN= 1
C
C     REAL TO COMPLEX TRANSFORM
C
      CALL DVSRFT(X,M,N,ISIN,ISN,W,ICON)
      PRINT*,'ICON=',ICON
C
      ISIN=-1
      ISN=-1
C
C     COMPLEX TO REAL TRANSFORM
C
      CALL DVSRFT(X,M,N,ISIN,ISN,W,ICON)
      PRINT*,'ICON=',ICON
C
      ERROR=0.0D0
      DO 200 J=1,M
      DO 200 I=1,N
      ERROR=MAX(ABS(X(J,I)/N-
     &      (FLOAT(I)+FLOAT(N)*(J-1))),ERROR)
  200 CONTINUE
C
      PRINT*,'ERROR=',ERROR
      STOP
      END
```

B71-12-0101 VTDEV, DVTDEV

Eigenvalues and eigenvectors of real tridiagonal matrix
CALL VTDEV (D, SL, SU, N, NF, IVEC, ETOL, CTOL, NEV, E, MAXNE, EV, K, M, VW, IVW, ICON)

(1) Function

This routine computes the eigenvalues and, optionally, the corresponding eigenvectors of a real tridiagonal matrix.

$$\boldsymbol{Tx} = \lambda \boldsymbol{x} \qquad (1.1)$$

The subdiagonal and superdiagonal elements of the tridiagonal matrix T of dimension n must satisfy the following conditions:

$$l_i \, u_{i-1} > 0, \; i = 2, \ldots, n \qquad (1.2)$$

Where l_1 is equal to $u_n = 0$ and

$$(\boldsymbol{Tv})_i = l_i \, v_{i-1} + d_i \, v_i + u_i \, v_{i+1}, \; i = 1, 2, \ldots, n \qquad (1.3)$$

(2) Parameters

D..... Input. One-dimensional array D (N) containing the diagonal elements of ***T***.

SL..... Input. One-dimensional array SL (N) containing the subdiagonal elements of ***T*** in SL (2:N). SL (1) = 0.

SU..... Input. One-dimensional array SU (N) containing superdiagonal elements of ***T*** in SU (1:N-1). SU (N) = 0.

N..... Input. Order n of tridiagonal matrix.

NF..... Input. The index of the first eigenvalue sought, where eigenvalues are numbered in ascending order. Eigenvalues with indices in the range NF to NF + NEV (1) - 1 are computed.
Output. The index of the first eigenvalue obtained, taking into account the case in which the first obtained eigenvalue is multiple and/or part of a cluster.

IVEC..... Input. Control information.
1: Both the eigenvalues and eigenvectors are sought.
Other than 1: Only the eigenvalues is sought.

ETOL..... Input. A criterion value required to determine whether an eigenvalue is distinct or numerically multiple based on expression (3.4). The default value is 3.0D-16 for double precision (2.0D-7 for single precision) when this value is set to less than.
(See item (3), “Comments on use,” b., 2).)

CTOL..... Input. A criterion value required to determine whether adjacent eigenvalues are approximately multiple i.e. clustered according to expression (3.4). CTOL ≥ ETOL
When CTOL is less than ETOL, CTOL = ETOL is set.
(See item (3), “Comments on use,” b., 2).)

NEV..... Input. NEV (1) indicates the number eigenvalues to be computed.
Output. NEV (1) indicates the number of distinct eigenvalues.
NEV (2) indicates the number of distinct clusters.

NEV (3) indicates the total number of eigenvalues including multiplicities.
One-dimensional array NEV (3).

E..... Output.
The eigenvalues computed are stored in E(1:NEV(3)).
One-dimensional array E (MAXNE).

MAXNE..... Input. The maximum number of eigenvalues that can be computed.
Dimension of array E.
When NEV (3) is greater than MAXNE, eigenvectors cannot be computed.
(See item (3), "Comments on use," b., 3).)

EV..... Output. When IVEC = 1, the eigenvectors corresponding to the computed eigenvalues are returned in EV (1:N, 1:NEV(3)).
Two-dimensional array EV (K, MAXNE).

K..... Input. Leading dimension of array EV (≥ N).

M..... Output. Information about the multiplicity of the computed eigenvalues.
M (***i***, 1) indicates the multiplicity of the ***i***-th eigenvalue $\boldsymbol{\lambda}_i$. M (***i***, 2) indicates the size of the ***i***-th cluster of eigenvalues.
(See item (3), "Comments on use," b., 3).)
Two-dimensional array M (MAXNE, 2).

VW..... Work area. One-dimensional array of size $12 \times N$.

IVW..... Work area. One-dimensional array of size $9 \times MAXNE + 128$.

ICON..... Output. Condition code.
See Table VTDEV-1, "Condition codes."

Table VTDEV-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	The total number of eigenvalues exceeded MAXNE during computation of multiple eigenvalues and/or clustered.	Processing is stopped. The eigenvectors cannot be computed. Eigenvalues are returned but are not stored taking into account multiplicities. (See item (3), "Comments on use," b., 3).)
30000	N<1, K<1, NF<1, NEV(1)<1, NF+NEV (1)>N, N>K	Processing is stopped.
30100	SL (i) × SU (i-1) ≤ 0, The matrix cannot be symmetrized.	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: UTMLS, UZRDM, UTDEY, UTDEX, UIBBS, UIBFC, UIBFE, UIBSL, UITBS, UITFC, UITFE, UITSL, AMACH, MGSSL

b. Comments

1) Problem solved using this routine

This routine requires only that $l_i\, u_{i-1} > 0$. Therefore, this routine can also used to solve the generalized eigenvalue problem in (3.1) by the following replacement:

$T \leftarrow TD^{-1}$

$$Tv = \lambda Dv \qquad (3.1)$$

Where, the diagonal matrix must satisfy $D > 0$.

The eigenvalue problem for T can be reduced to a symmetric generalized eigenvalue problem.

$$(DT - \lambda D)v = 0 \qquad (3.2)$$

Where, $D_1 = 1$ and $D_i = u_{i-1}\, D_{i-1} / l_i$ $i = 2, \ldots, n$.

If D_i can cause a scaling problem, it is preferable to consider the symmetric problem.

$$(D^{1/2}TD^{1/2} - \lambda I)w = 0 \qquad (3.3)$$

Where $w = D^{1/2}\, v$.

2) This routine pays special attention to a clustered eigenvalue.

With ε is equal to ETOL, suppose that the eigenvalues $\lambda_j, j = s, s+1, \ldots$, and $s+k$ ($k \geq 0$) are such that

$$\frac{|\lambda_i - \lambda_{i-1}|}{1 + \max(|\lambda_{i-1}|, |\lambda_i|)} \leq \varepsilon \qquad (3.4)$$

While (3.4) is not satisfied for $i = s - 1$ and $i = s + k + 1$, then eigenvalues λ_j, $j = s - 1, s, \ldots, s + k$ are considered to be identical, i.e., a single multiple eigenvalue of multiplicity $k + 1$.

The default value of ETOL is 3.0D-16 for double precision (2.0D-7 for single-precision). Using this value, eigenvalues are refined to machine precision.

When (3.4) is not satisfied for ε = ETOL, λ_{i-1} and λ_i are assumed to be distinct eigenvalues.

If (3.4) holds with ε = CTOL (but not with ε = ETOL) for eigenvalues λ_m, $m=t, t+1, \ldots, t+k$ but not for λ_{t-1} and λ_{t+k+1}, these eigenvalues are considered to be approximately multiple, i.e. clustered, though distinct (not numerically multiple). In order to obtain an invariant subspace, eigenvectors corresponding to clustered eigenvalues are computed using orthogonal starting vectors and are reorthogonalized. Of course CTOL ≥ ETOL; if this condition is not satisfied, CTOL is set to be equal to ETOL.

3) Assume r eigenvalues are requested. Note that if the first or last requested eigenvalue has a multiplicity greater than 1 then more than r eigenvalues, are obtained. The corresponding eigenvectors can be computed only when the corresponding eigenvector storage area is sufficient.

The maximum number of computable eigenvalues can be specified in MAXNE. If the total number of eigenvalues exceeds MAXNE, ICON = 20000 is returned. The corresponding eigenvectors cannot be computed. In this case, the eigenvalues are returned, but they are not stored repeatedly according to multiplicities.

When all eigenvalues are distinct, it is sufficient to set MAXNE = NEV (1).

c. Example of use

Here, we give a simple calculation to find ne eigenvalues and corresponding eigenvectors for a model problem based on a modification of one due to Wilkinson [44] which is known to have numerically multiple eigenvalues.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      INTEGER K,P1,Q1,N,N0,N1,MAX_CLUS,NE,MAXNE,NVW,NIVW
      REAL*8 ETOL,CTOL
C
      PARAMETER (K=1000)
      PARAMETER (P1=350,Q1=2,N=P1*Q1,N0=584,N1=686,
     &           NE=N1-N0+1)
      PARAMETER (MAX_CLUS=2*Q1,MAXNE=NE+MAX_CLUS)
      PARAMETER (ETOL=3.D-16,CTOL=5.D-12)
      PARAMETER (NVW=12*N,NIVW=9*MAXNE+128)
C
      REAL*8 SL(N),D(N),SU(N),E(MAXNE),EV(K,MAXNE),VW(NVW)
      INTEGER M(MAXNE,2),NEV(3),IVW(NIVW),NF,ICON,NEVAL,I
     &        ,J,KK,IVEC
      LOGICAL EVAL_OUTPUT,DBG_OUTPUT
C
      IVEC=1
C
C     Blocked W ^+_n (Wilkinson):  Pathologically close
C     eigenvalues in each p1 x p1 (p1 odd, small) block,
C     with q1 blocks so that multiplicity of largest
C     eigenvalues is 2*q1.  If maxnev <2*q1 then error
C     condition 20000 is obtained.
C
      J = ( P1 + 1 ) / 2
      D(J) = 0.D0
      DO 10 I=1,J-1                    ! first block
         SL(I+1)  = 1.D0
         SU(I)    = 1.D0
         SL(J+I)  = 1.D0
         SU(J+I-1) = 1.D0
         D(I)     = FLOAT(J-I)
         D(2*J-I) = D(I)
   10 CONTINUE

      SL(1)  = 0.D0
      SU(P1) = 0.D0

      DO 20 KK=2,Q1                    ! subsequent blocks
         II = (KK-1) * P1
         DO 20 I=1,P1
            SL(II+I) = SL(I)
            SU(II+I) = SU(I)
            D(II+I) = D(I)
   20 CONTINUE

      SL(1) = 0.D0
      SU(N) = 0.D0
```

```
      NF     = N0
      NEV(1) = NE
      ICON = 0
C
      CALL DVTDEV(D,SL,SU,N,NF,IVEC,ETOL,CTOL,NEV,E,MAXNE
     &            ,EV,K,M,VW,IVW,ICON)
C
      DBG_OUTPUT = .FALSE.
      IF( ICON .EQ. 20000 ) DBG_OUTPUT = .TRUE.
      EVAL_OUTPUT = .TRUE.
      IF ( ICON .EQ. 30000 .OR. ICON .EQ. 30100 )
     &   EVAL_OUTPUT = .FALSE.
      IF ( EVAL_OUTPUT ) THEN
         NEVAL = NF
         WRITE(*,*)' ICON = ',ICON
         II=1

         DO 30 J=1,NEV(1)
               WRITE(*,900) NEVAL,E(II),M(J,1)
            IF ( DBG_OUTPUT ) THEN
               II = II + 1
            ELSE
               II = II + M(J,1)
            ENDIF
            NEVAL = NEVAL + M(J,1)
   30    CONTINUE
      ENDIF
C
  900 FORMAT(' EIGENVALUE(',I5,')=',E25.18,2X,
     &       ' WITH MULTIPLICITY=',I5)
C
      STOP
      END
```

(4) Method

In the version of the Sturm count-based algorithm used here at least three subintervals are required in the refinement of each interval over which a sign change is detected [36]. Therefore at least $4 \times$ MAXNE points are required. Since this number determines the vector length used in the computation and the minimum vector register length on the VPP series is 64, this routine sets the number of points to be some multiple of 64 which is larger than $4 \times$ MAXNE.

A composite data structure is used: An array structure which facilitates vectorization is combined with an LIFO (*last in, first out*) list structure to keep track of both eigenvalue ordering and multisectioning; this is discussed in [32]. The computation is carried out until the limit of refinement as determined by ETOL is reached. (See [44].) When the default value is selected, the accuracy of the eigenvalue estimate relative to the scale of the matrix should approach machine precision.

The prescription for evaluating the Sturm count, follows [10]; it has the property that the sign count is a monotonic function of the eigenvalue parameter in IEEE floating-point arithmetic.

Eigenvectors are computed by inverse iteration. The initial vector is chosen using the sign structure of the Sturm sequence except when numerically multiple (or approximately multiple) eigenvalues have been detected. In this case additional initial vectors are generated randomly and orthogonalized with respect to the other vectors of the cluster.

Usually one step of inverse iteration suffices. Eigenvectors corresponding to clustered eigenvalues are also reorthogonalized after inverse iteration.

A72-25-0101 VTFQD, DVTFQD

System of linear equations with unsymmetric or indefinite sparse matrix (TFQMR method, diagonal storage format)
CALL VTFQD (A, K, NDIAG, N, NOFST, B, ITMAX, EPS, IGUSS, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations with an $n \times n$ unsymmetric or indefinite sparse matrix using the transpose-free quasi-minimal residual method (TFQMR).

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

The $n \times n$ coefficient is stored with the diagonal format method. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

The iterative calculation may not be continued (break-down) because of the characteristics of the initial vector and coefficient matrices. This is because zero is obtained as the intermediate result in the recursive calculation formula. In this case, use the MGCR method that causes no break-down.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 “Iterative linear equation solvers and Convergence,” in Part I.

(2) Parameters

A..... Input. Stores non-zero elements in a coefficient matrix.
Two-dimensional array A (K, NDIAG). Stores coefficient matrix $\boldsymbol{A}$ in A (1:N, NDIAG) with the diagonal storage format. For the diagonal storage format, see Part I, “Overview,” Section 3.2.1.1, “Storage Methods for General Sparse Matrices,” b., “Diagonal Storage Format for General Sparse Matrices.”

K..... Input. Size of adjustable dimension of array A

NDIAG..... Input. The number of diagonal vectors in coefficient matrix $\boldsymbol{A}$ that contain non-zero elements.
Size of second-dimension of array A.

N..... Input. Order n of matrix $\boldsymbol{A}$.

NOFST.... Input. Stores the distance from the main diagonal vector corresponding to diagonal vectors stored in array A. Superdiagonal vectors have positive values; a subdiagonal vector have negative values.
One-dimensional array NOFST (NDIAG).

B..... Input. One-dimensional array of size n. Stores the constant vector of the right-hand side term of a linear equation system.

ITMAX..... Input. The upper limit of iterations in TFQMR method (> 0).

EPS..... Input. A convergency criterion value in judgment of convergency.
If EPS is 0.0 or less, it is set to 10^{-6} in double-precision routines and 10^{-4} in single-precision routines.
(See item (3), “Comments on use,” b., 1).)

IGUSS..... Input. Sets control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=0: Approximate value of the solution vector is not specified.
IGUSS≠0: The iterative computation starts from the approximate value of the solution vector specified in array X.

X..... Input. One-dimensional array of size *n*. Can specify the approximate value of the solution vector.
Output. The solution vector is stored.

ITER..... Output. Number of iteration performed using the QMR method.

VW..... Work area. One-dimensional array K × 10 + N + NBANDL + NBANDR. NBANDL indicates a lower bandwidth; NBANDR indicates an upper bandwidth.

ICON..... Output. Condition code
See Table VTFQD-1, "Condition codes."

Table VTFQD-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	Break-down occurred	Processing is stopped.
0001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate value obtained up to this point in array X is output, but their precision cannot be guaranteed.
30000	K<1, N<1, K<N, NDIAG<1, K<NDIAG, or ITMAX ≤ 0	Processing is stopped.
32001	| NOFST (I) | > N-1	Processing is stopped.

(1) Comments on use

a. Subprograms used

SSL II: AMACH, URGWD, URIPA, URITI, URITT, URMVD, USSCP, URSTE, USVCN, USVCP, USVSU, USVUP, USVN2, URELT, MGSSL, UTFQD, UTFQR, UQBBM

b. Comments

1) In the QMR method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and the obtained approximation is roughly equal to the product of the condition number of Matrix ***A*** and EPS.

2) Notes on using the diagonal format

A diagonal vector element outside coefficient matrix ***A*** must be set to zero. There is no restriction in the order in which diagonal vectors are stored in array A.

The advantage of this method lies in the fact that the matrix vector multiplication can be calculated without the use of indirect indices. The disadvantage is that

matrices without the diagonal structure cannot be stored efficiently with this method.

c. Example of use

In this example, linear equations of coefficient matrices obtained by discretizing partial differential operators are solved in the region $[0,1] \times [0,1] \times [0,1]$, with the Dirichlet boundary condition (function value zero at the boundary).

This type of partial differential operator is described in Part I, "Overview," Section 3.2.2 "Discretization of partial differential operators and storage examples for them."

For INIT_MAT_DIAG, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them." GET_BANDWIDTH_DIAG is a routine that estimates band width. INIT_SOL is a routine that generates solution vectors to be sought with random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER MACH
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000)
      PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
      PARAMETER (NDIAG = 7,NVW = 10*K+N+400+400)
      REAL*8 A(K,NDIAG),X(N),B(N),SOLEX(N),VW(NVW)
      INTEGER NOFST(NDIAG)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUSIONS'
      PRINT*,'X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
C
      PRINT *
      PRINT *,'    TFQMR METHOD'
      PRINT *,'    DIAGONAL FORMAT'
C
      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0

      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,A,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      CALL GET_BANDWIDTH_DIAG(NOFST,NDIAG,NBANDL,NBANDR)
         DO 110 I = 1,N
            VW(I+NBANDL) = SOLEX(I)
 110     CONTINUE
      CALL DVMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON)
      PRINT*,'DVMVSD ICON=',ICON
C
      EPS = 1D-10
      IGUSS = 0
      ITMAX = 2000
      CALL DVTFQD(A,K,NDIAG,N,NOFST,B,ITMAX
     &           ,EPS,IGUSS,X,ITER,VW,ICON)
```

```
C
      PRINT* ,'ITER = ',ITER
      PRINT* ,'DVTFQD ICON = ',ICON
      PRINT*, 'COMPUTED VALUES'
      PRINT*, 'X(1) = ',X(1),' X(N) = ',X(N)
      STOP
      END
```

(4) Method

For the TFQMR method, see [12].

A72-26-0101 VTFQE, DVTFQE

System of linear equations with unsymmetric or indefinite sparse matrix (TFQMR method, ELLPACK storage format)
CALL VTFQE (A, K, IWIDT, N, ICOL, B, ITMAX, EPS, IGUSS, X, ITER, VW, ICON)

(1) Function

This routine solves linear equations with an $n \times n$ unsymmetric or indefinite sparse matrix using the transpose-free quasi-minimal residual method.

$Ax = b$

Coefficient matrices ($n \times n$) are stored with the ELLPACK format method. Vectors b and x are n-dimensional vectors.

The iterative calculation may not be continued (break-down) because of the characteristics of the initial vector and coefficient matrices. This is because zero is obtained as the intermediate result in the recursive calculation formula. In this case, use the MGCR method that causes no break-down.

Regarding the convergence and the guideline on the usage of iterative methods, see Chapter 4 "Iterative linear equation solvers and Convergence," in Part I.

(2) Parameters

A..... Input. Stores non-zero elements of the coefficient matrix.
Two-dimensional array A (K, IWIDT).
For the ELLPACK storage format, see Part I, "Overview," Section 3.2.1.1, "Storage Method for General Sparse Matrices."

K..... Input. Size of adjustable dimension ($\geq n$) of A and ICOL.

IWIDT..... Input. The maximum number of non-zero-elements in row vector direction on the coefficient matrix *A*.
Two dimensional size of A and ICOL.

N..... Input. Order n of matrix *A*.

ICOL..... Input. Store the column indices of the element stored in the array A using the ELLPACK format, indicating which column vectors the corresponding elements in the array A belong to.
Two-dimensional array ICOL (K, IWIDT)

B..... Input. One-dimensional array of size n. Stores a constant vector of the right-hand-side term of a linear equation system.

ITMAX..... Input. The upper limit of iterations in TFQMR method (> 0).

EPS..... Input. A convergence criterion value in judgment of convergency.
If EPS is 0.0 or less, it is set to 10^{-6} in double-precision routines and 10^{-4} in single-precision routines.
(See item (3), "Comments on use," b., 1).)

IGUSS..... Input. Control information about whether to start the iterative computation from the approximate value of the solution vector specified in array X.
IGUSS=0: Approximate value of the solution vector is not set.

IGUSS≠0: The iterative computation starts from the approximate value of the solution vector specified in array X.

X..... Input. One-dimensional array of size *n*. An approximate value of a solution vector can be specified.
Output. Stores a solution vector.

ITER..... Output. The real number of iteration steps in TFQMR method.

VW..... Work area. One-dimensional array $K \times 13$.

ICON..... Output. Condition code
See Table VTFQE-1, "Condition codes."

Table VTFQE-1 Condition codes

Code	Meaning	Processing
0	No error	-
20000	Break-down occurred	Processing is stopped.
20001	The upper limit of iteration steps was reached.	Processing is stopped. The approximate values obtained up to this point in array X are output, but their precision cannot be guaranteed.
30000	K<1, N<1, K<N, IWIDT<1, K<IWIDT, or ITMAX ≤ 0	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: AMACH, URIPA, URITI, URITT, URMVE, USSCP, URSTE, USVCN, USVCP, USVSU, USVUP, USVN2, URELT, MGSSL, UTFQE, UTFQR, UQBBM

b. Comments

1) In the TFQMR method, if the residual Euclidean norm is equal to or less than the product of the initial residual Euclidean norm and EPS, it is judged as having converged. The difference between the precise solution and obtained approximate solution is equal to the product of the condition number of matrix A and EPS.

c. Example of use

In this example, linear equations of coefficient matrices obtained by descretizing partial differential operators are solved in the region $[0,1] \times [0,1] \times [0,1]$ with the Dirichlet boundary condition (function value zero at the boundary). This type of partial differential operator is described in Part I, "Overview," Section 3.2.2, "Descretization of partial differential operator and storage examples for them."

For INIT_MAT_ELL, see Part I, "Overview," Section 3.2.2, "Discretization of partial differential operators and storage examples for them."

INIT_SOL is the routine that generates the solution vectors to be sought in random numbers.

```
C     **EXAMPLE**
      PROGRAM TEST_ITER_SOLVERS
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (MACH = 0)
      PARAMETER (K = 10000)
      PARAMETER (NX = 20,NY = 20,NZ = 20,N = NX*NY*NZ)
      PARAMETER (IWIDT = 7,NVW = K*13)
      REAL*8 A(K,IWIDT),X(N),B(N),SOLEX(N),VW(NVW)
      INTEGER ICOL(K,IWIDT)
C
      CALL INIT_SOL(SOLEX,N,1D0,MACH)
      PRINT*,'EXPECTED SOLUTION'
      PRINT*,'X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT*
      PRINT*,'     TFQMR METHOD'
      PRINT*,'     ELLPACK FORMAT'
C
      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 5D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
C
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL
     &          ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)
      CALL DVMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON)
      PRINT*,'DVMVSE ICON = ',ICON
C
      EPS = 1D-10
      IGUSS = 0
      ITMAX = 800
      CALL DVTFQE(A,K,IWIDT,N,ICOL,B,ITMAX
     &            ,EPS,IGUSS,X,ITER,VW,ICON)
C
      PRINT*,'DVTFQE ICON = ',ICON
      PRINT*,'COMPUTED VALUE'
      PRINT*,'X(1) = ',X(1),' X(N) = ',X(N)
      STOP
      END
```

(4) Method

For the TFQMR method, see [12].

F18-11-0101 VWFLT, DVWFLT

Wavelet filter generation
CALL VWFLT (F, N, ICON)

(1) Function

This routine generates a filter corresponding to the Daubechies wavelet (order *n*) having a compact support. The filters of order 2, 4, 6, 12, and 20 can be generated.

(2) Parameters

F..... Output. One-dimensional array of size $2 \times N$. Stores a wavelet filter used for transform.
(See item (3), "Comments on use," b., "1).")

N..... Input. The number of wavelet filter coefficients. (2, 4, 6, 12, or 20)

ICON..... Output. Condition code.
See Table VWFLT-1, "Condition codes."

Table VWFLT-1 Condition codes

Code	Meaning	Processing
0	No error	-
30000	N is not 2, 4, 6, 12, or 20.	Processing is stopped.

(3) Comments on use

a. Subprograms used

None.

b. Comments

1) Filter conditions

The orthogonal filter used for this function is described by a vector of size $2 \times N$. F(1) ,..., F(N) defines a low-pass filter; F(N+1), ..., F(2×N) defines a high-pass filter. These coefficients satisfy the following relationships:

$$\sum_{i=1}^{N} F(i)^2 = 1, \quad F(2N+1-i) = (-1)^i \; F(i), i = 1,\ldots,N$$

For details, see [7] and [9].

c. Example of use

This example shows a one-dimensional wavelet transform and inverse transform for data of size n = 1024.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)

C     -------- Constants ------------
      INTEGER MaxK, MaxSize
      PARAMETER (MaxK = 20,MaxSize = 1024)

C     -------- Variables and formats ------------
      INTEGER N,K,i,ISN,ICON,LS
      REAL*8 X(1:MaxSize),T(1:MaxSize),Y(1:MaxSize),
     &       F(1:2*MaxK),
     &       ireal,Emax,diff,temp,Xmax,Erel

C     -------- Generate input ------------
      N  = 1024
      K  = 6
      LS = 3

      DO 100 i= 1,N
        ireal = i
        temp = 0.5 - abs(ireal/N - 0.5)
        X(i) = temp                    ! Input vector
        T(i) = temp                    ! Reference vector
 100  CONTINUE

C     ---------------  Initialize filter ------
      CALL DVWFLT(F,K,ICON)

C     ---------------- Transform Data ----------------

      ISN=1
      CALL DV1DWT(X,N,Y,ISN,F,K,LS,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN 1D Wavelet Transform,ICON = ',ICON
        STOP
      ENDIF

C     ---------------- Transform back ---------------

      ISN=-1
      CALL DV1DWT(X,N,Y,ISN,F,K,LS,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN Inverse of 1D Wavelet Transform,'
     &        ,'ICON = ',ICON
        STOP
      ENDIF

C     ------ Verify result ------

      Emax = 0.0
      Xmax = 0.0
      DO 200 i=1,N
        diff = abs(X(i)-T(i))
        IF (diff .GT. Emax) Emax = diff
        IF (abs(X(i)) .GT. Xmax) Xmax = abs(X(i))
 200  CONTINUE
```

```
Erel = Emax/Xmax
IF (Erel .GT. 1.0e-4) THEN
  PRINT*,'Relative Max error (FWT):',Erel
  STOP
END IF
PRINT*,'1D Wavelet Transform OK'

STOP
END
```

F18-12-0101 V1DWT, DV1DWT

One-dimensional wavelet transform
CALL V1DWT (X, N, Y, ISN, F, K, LS, ICON)

(1) Function

This routine performs a one-dimensional wavelet transform or its inverse. The transform is defined by its high- and low- pass filter coefficients.

(2) Parameters

X..... Input or output. One-dimensional array X(N). Stores vector data to be transformed as input in the case of wavelet transform (ISN = 1); the transformed vector data is stored as output in the case of the inverse transform (ISN = -1).

N..... Input. Length of the transformed data. Must be a power of two. (See item (3), "Comments on use," b., 1).)

Y..... Output or input. One-dimensional array Y(N). The transformed vector data is stored as output in the case of the wavelet transform (ISN = 1); store vector data to be transformed as input in the case of the inverse transform (ISN = -1). (See item (3), "Comments on use," b., 2).)

ISN..... Input. Specify transform or inverse transform.
Transform: ISN = 1
Inverse transform: ISN = -1

F..... Input. One-dimensional array of size $2 \times K$. Stores the wavelet filter used for transform. The user can supply either the filter coefficients F, or call VWFLT before this routine to specify a filter coefficient used for the one-dimensional wavelet transform.
(See item (3), "Comments on use," b., 3).)

K..... Input. A positive even number to indicate the number of the wavelet filter coefficients.

LS..... Input. A positive integer that indicates the depth of transform for vector data. $N \geq 2^{LS}$. When $N = 2^{LS}$, a full wavelet transform is performed.

ICON..... Output. Condition code.
See Table V1DWT-1, "Condition codes."

Table V1DWT-1 Condition codes

Code	Meaning	Processing
0	No error	-
30000	ISN ≠ 1 and ISN ≠ -1	Processing is stopped.
30002	N<2	
30004	N is not a power of 2.	
30008	K is not an even number, or LS < 0, LS > $\log_2 N$	Processing is stopped.

(3) Comments on use

a. Subprograms used

SSL II: UWFT1, UWFI1, UWPT1, UWVI1, UWPI1, UWVT1, MGSSL

b. Comments

1) When the size of the data to be transformed is not a power of two, the wavelet transform can be done storing the data with the remaining data padded with zeros in a larger array with size N of a power of two.

2) Storing the transform result

For vector v_j in one-dimensional input data, the result of the high-pass filter in each partial wavelet transform is stored in v_j ($N\times2^{-i}+1$:$N\times2^{-i+1}$), i=1, ..., LS. The output result of the high-pass filter for partial wavelet transform in the first stage is stored in Y(N/2 + 1:N, M/2 + 1: M).

3) Filter conditions

The orthogonal filter used for this function generally has a vector of size $2 \times K$. F(1), ..., F(K) defines the low-pass filter coefficients; F(K+1), ..., F(2×K) defines the high-pass filter coefficients. These coefficients have the following relationships:

$$\sum_{i=1}^{K} \mathrm{F}(i)^2 = 1,\ \mathrm{F}(2\mathrm{K}+1-i) = (-1)^i\ F(i), i = 1,...,K$$

For details, see [7] and [9].

c. Example of use

For data of size n = 1024, perform the one-dimensional wavelet transform and inverse transform.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)

C     -------- Constants ------------
      INTEGER MaxK, MaxSize
      PARAMETER (MaxK = 20,
     &           MaxSize = 1024)

C     -------- Variables and formats ------------
      INTEGER N,K,i,ISN,ICON,LS
      REAL*8 X(1:MaxSize),T(1:MaxSize),Y(1:MaxSize),
     &       F(1:2*MaxK),
     &       ireal,Emax,diff,temp,Xmax,Erel
C     -------- Generate input ------------
      N  = 1024
      K  = 6
      LS = 3

      DO 100 i= 1,N
        ireal = i

        temp = 0.5 - abs(ireal/N - 0.5)
        X(i) = temp                    ! Input vector
```

```
        T(i) = temp                     ! Reference vector
 100  CONTINUE

C     ---------------  Initialize filter ------
      CALL DVWFLT(F,K,ICON)

C     ---------------- Transform Data ----------------

      ISN=1
      CALL DV1DWT(X,N,Y,ISN,F,K,LS,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN 1D Wavelet Transform,ICON = ',ICON
        STOP
      ENDIF

C     ---------------- Transform back ---------------

      ISN=-1
      CALL DV1DWT(X,N,Y,ISN,F,K,LS,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN Inverse of 1D Wavelet Transform,'
     &        ,'ICON = ',ICON
        STOP
      ENDIF

C     ------ Verify result ------

      Emax = 0.0
      Xmax = 0.0
      DO 200 i=1,N
        diff = abs(X(i)-T(i))
        IF (diff .GT. Emax) Emax = diff
        IF (abs(X(i)) .GT. Xmax) Xmax = abs(X(i))
 200  CONTINUE
        Erel = Emax/Xmax
        IF (Erel .GT. 1.0e-4) THEN
          PRINT*,'Relative Max error (FWT):',Erel
          STOP
        END IF
        PRINT*,'1D Wavelet Transform OK'

        STOP
        END
```

(4) Method

A partial wavelet transform of a vector *s* (usually a signal) of length *N* is obtained by applying a low-pass and a high-pass filters. The subvector w_1, ..., $w_{n/2}$ is obtained by applying the low-pass filter to *s*. The subvector $w_{n/2+1}$, ..., w_n is obtained by applying the high-press filter to s.

A wavelet transform is the recursive application of a partial wavelet transform to the subvector containing the low-pass filtered components, up to $\log_2(n)$ times. Each application involves only half the data of the previous application.

As the first step dominates the amount of computational work, the complexity transform is O ($K \times N$), where K is the order of the wavelet in question and N is the length of the vector being transformed.

The implemented transform treats only periodic data. If applied to non-periodic data artificial discontinuities appear at the endpoints and have an effect on the transform. To minimize this similar techniques as used for fourier transforms (interpolation, padding with mirrored data) may also be applied.

Introductory material on wavelet transforms can be found in [15][39], and further applications are described in [35]. An in-depth treatment of the subject is given in [7][9].

F18-13-0101 V2DWT, DV2DWT

Two-dimensional wavelet transform
CALL V2DWT (X, M, N, Y, ISN, F, K, LSX, LSY, ICON)

(1) Function

This routine performs a two-dimensional wavelet transform or its inverse. The transform is defined by its high- and low-pass filter coefficients.

(2) Parameters

X..... Input or output. Two-dimensional array of X(M, N). Stores the two-dimensional data to be transformed as input in the case of transform (ISN = 1); the two-dimensional data transformed is stored as output in the case of inverse transform (ISN = -1).

M..... Input. The number of rows containing the data to be transformed. A positive integer indicated by a power of two.
(See item (3), "Comments on use," b., 1).)

N..... Input. The number of columns containing the data to be transformed. A positive integer indicated by a power of two.
(See item (3), "Comments on use," b., 1).)

Y..... Output or input. Two-dimensional array Y(N, M). The transformed data as output in the case of transform (ISN = 1) store the data to be transformed as input in the case of inverse transform (ISN = -1).
(See item (3), "Comments on use," b., 2).)

ISN..... Input. Specify transform or inverse transform.
Transform: ISN = 1
Inverse transform: ISN = -1

F..... Input. One-dimensional array of size $2 \times K$. Stores the wavelet filter coefficients used for transform. The user can supply either the filter coefficients themselves or call VWFLT before this routine to specify filter coefficients used for two-dimensional wavelet transform.
(See item (3), "Comments on use," b., 3).)

K..... Input. The number of wavelet filter coefficients.

LSX..... Input. A positive integer that indicates the depth of transform for each column. $M \geq 2^{LSX}$. When $M = 2^{LSX}$, a full wavelet transform is performed.

LSY..... Input. A positive integer that indicates the depth of transform for each row. $N \geq 2^{LSY}$. When $N = 2^{LSY}$, a full wavelet transform is performed.

ICON..... Output. Condition code.
See Table V2DWT-1, "Condition codes."

Table V2DWT-1 Condition codes

Code	Meaning	Processing
0	No error	-
30000	ISN ≠ 1 and ISN ≠ -1	Processing is stopped.
30002	M<2 or N<2	
30004	Either M or N is not a power of two.	
30008	K is not an even number, or LSX < 0, LSX > $\log_2$M, LSY < 0, LSY > $\log_2$N	

(3) Comments on use

a. Subprograms used

SSL II: UWFT2, UWFI2, UWPT2, UWVI2, UWPI2, UWTRP, UWVT2, MGSSL

b. Comments

1) When the size of the data to be transformed is not a power of two, the wavelet transform can be done storing the data with the remaining data padded with zeros in a larger array with size (M, N) of a power of two.

2) Storing the transform result

For column vector c_j and row vector r_j in two-dimensional input data, the result of the high-pass filter in each wavelet transform row are respectively stored in c_j ($M\times2^{-i}+1$:$M\times2^{-i+1}$), i=1, ..., LSX and r_j ($N\times2^{-i}+1$:$N\times2^{-i+1}$), i=1, ..., LSY. The result of the two-dimensional wavelet transform is transposed and stored in array Y.

The output result of the high-pass filter for partial wavelet transform in the first stage is stored in Y(N/2 + 1:N, M/2 + 1:M).

3) Filter conditions

The orthogonal filter used for this function generally has a vector of size $2 \times K$. F(1), ..., F(K) defines the low-pass filter coefficients; F(K+1), ..., F(2×K) defines the high-pass filter coefficients. These coefficients have the following relationships:

$$\sum_{i=1}^{K} \mathrm{F}(i)^2 = 1,\ \mathrm{F}(2\mathrm{K}+1-i) = (-1)^i\ F(i), i = 1,...,K$$

For details, see [7] and [9].

c. Example of use

For two-dimensional data (1024×512), perform the two-dimensional wavelet transform and inverse transform.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)

C     -------- Constants ------------
      INTEGER MaxK,MaxSize
      PARAMETER (MaxK = 20,MaxSize = 512*1024)

C     -------- Variables and formats ------------
      INTEGER M,N,K,i,row,index2D,ISN,ICON,LSX,LSY
      REAL*8 X(1:MaxSize),T(1:MaxSize),Y(1:MaxSize),
     &       F(1:2*MaxK),
     &       ireal,Emax,diff,temp,Xmax,Erel

C     -------- Generate input ------------
      M   = 1024
      N   = 512
      K   = 6
      LSX = 3
      LSY = 4

      DO 99 row = 1,M
        DO 100 i= 1,N
          ireal = i
          temp = 0.5 - abs(ireal/N - 0.5)
          ireal = row
          temp = temp + 0.5 - abs(ireal/M - 0.5)
          index2D = row + (i-1)*M
          X(index2D) = temp              ! Input vector
          T(index2D) = temp              ! Reference vector
 100    CONTINUE
 99   CONTINUE

C     ---------------  Initialize filter ------
      CALL DVWFLT(F,K,ICON)

C     ---------------- Transform Data ----------------

      ISN=1
      CALL DV2DWT(X,M,N,Y,ISN,F,K,LSX,LSY,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN 2D Wavelet Transform,ICON = ',ICON
        STOP
      ENDIF

C     ---------------- Transform back ---------------

      ISN=-1
      CALL DV2DWT(X,M,N,Y,ISN,F,K,LSX,LSY,ICON)
      IF (ICON .NE. 0) THEN
        PRINT*,'ERROR IN Inverse of 2D Wavelet Transform,'
     &        ,'ICON = ',ICON
        STOP
      ENDIF

C     ------ Verify result ------
```

```
      Emax = 0.0
      Xmax = 0.0
      DO 199 row =1,M
        DO 200 i=1,N
          index2D = row + (i-1)*M
          diff = abs(X(index2D)-T(index2D))
          IF (diff .GT. Emax) Emax = diff
          IF (abs(X(index2D)) .GT. Xmax)
     &       Xmax = abs(X(index2D))
200     CONTINUE
199   CONTINUE

      Erel = Emax/Xmax
      IF (Erel .GT. 1.0e-4) THEN
        PRINT*,'Relative Max error (FWT):',Erel
        STOP
      END IF
      PRINT*,'2D Wavelet Transform OK'

      STOP
      END
```

(4) Method

A partial wavelet transform of a vector *s* (usually a signal) of length *n* is obtained by applying a low-pass and a high-pass filters. The subvector w_1, ..., $w_{n/2}$ is obtained by applying the low-pass filter to *s*. The subvector $w_{n/2+1}$, ..., w_n is obtained by applying the high-press filter to *s*.

A wavelet transform is the recursive application of a partial wavelet transform to the subvector containing the low-pass filtered components, up to $\log_2(n)$ times. Each application involves only half the data of the previous application.

As the first step dominates the amount of computational work, the complexity of each 1D transform is $O(K \times N)$, where K is the order of the wavelet in question and N is the length of the vector being transformed.

In the two-dimensional case, a wavelet transform is applied to each column of the matrix with depth LSX and then to each row of the resultant matrix with depth LSY.

The implemented transform treats only periodic data. If applied to non-periodic data artificial discontinuities appear at the endpoints and have an effect on the transform. To minimize this similar techniques as used for fourier transforms (interpolation, padding with mirrored data) may also be applied.

Introductory material on wavelet transforms can be found in [15][39], and two-dimensional transforms are described in [35]. An in-depth treatment of the subject is given in [7][9].

Appendix A References

[1] S.L.Anderson
Random number generators on vector supercomputers and other advanced architectures, SIAM Rev, 32 (1990), 221-251.

[2] R.P.Brent
Uniform random number generators for supercomputers, Proc. Fifth Australian Supercomputer Conference, Melbourne, Dec. 1992, 95-104.

[3] R.P.Brent
Uniform random number generators for vector and parallel computers, Report TR-CS-92-02, Computer Sciences Laboratory, Australian National University, Canberra, March 1992 (can be accessed from the pub/Brent directory at ftp site “dcssoft.anu.edu.au”).

[4] R.P.Brent
Fast normal random number generators on vector processors, Technical Report TR-CS-93-04, Computer Sciences Laboratory, Australian National University, Canberra, March 1993. (Available by ftp from nimbus.anu.edu.au in the directory pub/Brent.)

[5] R.P.Brent
A Fast Vectorised Implementation of Wallace's Normal Random Number Generator, Technical Report, Computer Sciences Laboratory, Australian National University, to appear.(Will be available by ftp from nimbus.anu.edu.au in the directory pub/Brent)

[6] C.Sidney Burrus, Fellow, IEEE, and Peter W. Eschenbacher
“An In Place, In-Order Prime Fact or FFT Algorithm”, IEEE Trans. on Acoust., Speech, Signal Processing, vol. ASSP-29, No.4, pp. 806-817, August 1981.

[7] C.K.Chui
Adv. in Numer. Analysis, vol.II, ch. Wavelets and Splines, Oxford Sci. Publ., 1992.

[8] J.K.Cullum and R.A.Willoughby
“Lanczos algorithm for large symmetric eigenvalue computations”, Birkhauser, 1985.

[9] I.Daubechies
Ten lectures on wavelets, SIAM, 1992.

[10] J.Demmel and W.kahan
Accurate singular values of bidiagonal matrices, SISSC 11, 873--912, 1990.

[11] A.M.Ferrenberg, D.P.Landau and Y.J.Wong,
Monte Carlo simulations: Hidden errors from “good” random number generators, Phys. Rev. Lett. 69 (1992), 3382-3384.

[12] R.Freund
“A transpose-free quasi-minimal residual algorithm for nonhermitian linear systems”, SIAM J.Sci.Comput. 14, 1993, pp. 470-482.

[13] R.Freund and N.Nachtigal
“QMR: a quasi minimal residual method for non-Hermitian linear systems”, Numer. Math. 60, 1991, pp. 315-339.

[14] G.Golub and C.Van Loan
Matrix computations, The Johns Hopkins University Press, Baltimore, 1989.

[15] A.Graps
An introduction to wavelets, IEEE Computational Science and Engineering 9, 1995, no.2.

[16] M.H.Gutknecht
Variants of BiCGStab for matrices with complex spectrum,IPS Research report No. 91-14, 1991.

[17] Markus Hegland
A self-sorting in-place fast Fourier transform algorithm suitable for vector and parallel processing, accepted for publication in Numerische Mathematik, 1994

[18] M.Hegland
On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization, Numerische Mathematik 59, 1991, no.5, 453-472.

[19] Markus Hegland
An Implementation of Multiple and Multivariate Fourier Transforms on Vector Processors" appears in SIAM J. Sci. Comput., Vol. 16, No. 2, pp. 271-288, March 1995

[20] J.R.Heringa, H.W.J.Blöte and A.Compagner.
New primitive trinomials of Mersenne-exponent degrees for random-number generation, International J. of Modern Physics C 3 (1992), 561-564.

[21] M.R. Hestenes and E.Stiefel
Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Standards, 49: 409-435, 1952.

[22] F.James
A review of pseudo-random number generators, Computer Physics Communication 60 (1990), 329-344.

[23] D.Kincaid, T.Oppe
ITPACK on supercomputers, Numerical methods, Lecture Notes in Mathematics 1005 (1982).

[24] D.E.Knuth
The Art of Computer programming, Volume 2: Seminumerical Algorithms (second edition). Addison-Wesley, Menlo Park, 1981, Sec. 3.4.1, Algorithm P.

[25] Z.Leyk
"Modified generalized conjugate residuals for nonsymmetric systems of linear equations" in "Proceedings of the 6th Biennial Conference on Computational Techniques and Applications: CTAC93", D.Stewart, H.Gardner and D.Singleton, eds., World Scientific, 1994, pp. 338-344. Also published as CMA Research Report CMA-MR33-93, Australian National University, 1993.

[26] N.K.Madsen, G.H.Rodrigue, and J.I.Karush
"Matrix multiplication by diagonals on a vector/parallel processor", Information Processing Letters, vol. 5, 1976, pp. 41-45

[27] G.Marsaglia
A current view of random number generators, Computer Science and Statistics: The Interface (edited by L.Billard), Elsevier Science Publishers B.V. (North-Holland), 1985, 3-10.

[28] R.S.Martin, G.Peters and J.H.Wilkinson,
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

[29] T.Oppe, W.Joubert and D Kincaid
An overview of NSPCG: a nonsymmetric preconditioned conjugate gradient package, Computer physics communications 53 p283 (1989).

[30] T.C.Oppe and D.R.Kincaid
"Are there interactive BLAS?", Int.J.Sci.Comput. Modeling (to appear).

[31] J.Ortega
Introduction to parallel and vector solution of linear systems, Plenum Press, 1988

[32] M.R.Osborne
Computing the eigenvalues of tridiagonal matrices on parallel vector processors, Mathematics Research Report No. MRR 044-94, Australian National University, 1994.

[33] J.R.Rice and R.F.Boisvert
Solving Elliptic Problems Using ELLPACK, Springer-Verlang, New York, 1985.

[34] Y.Saad and M.H.Schultz
"GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM J.Sci.Stat. Comput. 7, 1986, p.856-869.

[35] D.H.Salesin, E.J.Stollnitz, T.D.DeRose
Wavelets for computer graphics: A primer, part 1 and 2, IEEE Computer Graphics and Applications 15 (1995).

[36] H.D.Simon
Bisection is not optimal on vector processors, SISSC 10, 205-209, 1989.

[37] G.Sleijpen, D.Fokkema
BICG_STAB(L) for linear equations involving unsymmetric matrices with complex spectrum, Electronic Transactions on Numerical Analysis, Vol 1, p11-32, 1993. (can be accessed from "http://etna.mcs.kent.edu/")

[38] G.L.G Sleijpen, H.A.van der Vorst, and D.R.Fokkema.
BiCGSTAB(l) and other hybrid Bi-CG methods. Numerical Algorithms, 7:75-109, 1994.

[39] G.Strang, T.Nguyen
Wavelets and filter banks, Wellesley-Cambridge Press, 1996

[40] Paul N. Swarztrauber
Bluestein's FFTs for arbitrary N on the hypercube, Parallel Comput. 17 (1991), 607-617.

[41] H.A.Van Der Vorst
"BCG: A fast and smoothly converging variant of BI-CG for the solution of non-symmetric linear systems", SIAM J. Sci. Statist. Comput., 13 p631 1992

[42] C.S.Wallace
"Fast Pseudo-Random Generators for Normal and Exponential Variates", ACM Trans. on Mathematical Software 22 (1996), 119-127.

[43] R.Weiss
Parameter-Free Itarative Linear Solvers. Mathematical Research, vol.97. Akademie Verlag, Berlin, 1996.

[44] J.H.Wilkinson
The Algebraic Eigenvalue Problem, O.U.P., 1965.

[45] S.Winograd
"On computing the discrete Fourier transform", Math. Computation., Vol.32, pp.175-199, Jan. 1978.

Appendix B
Contributors and Their Work

Contributor	Subroutine name	Function
Richard Peirce Brent	DVRAN3	Generation of normal random numbers (double precision)
	DVRAN4	Generation of normal random numbers (double precision, Wallace's method)
Richard Peirce Brent Peter Frederick Price	DVRAU4	Generation of uniform random numbers [0, 1) (double precision)
Andrew James Cleary	VBLDL	LDL^T decomposition for a symmetric positive definite banded matrix (modified Cholesky decomposition)
	VBLDX	System of linear equations with a LDL^T decomposed symmetric positive definite banded matrix
	VBLU	System of linear equations for a banded real matrix (Gaussian elimination)
	VLSBX	System of linear equations with a symmetric positive definite banded matrix (modified Cholesky decomposition)
Murray Leslie Dow	VBCSD	System of linear equations with unsymmetric or indefinite sparse matrix (BICGSTAB(*l*) method, diagonal storage format)
	VBCSE	System of linear equations with unsymmetric or indefinite sparse matrix (BICGSTAB(*l*) method, ELLPACK storage format)
	VCGD	System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, diagonal storage format)
	VCGE	System of linear equations with a symmetric positive definite sparse matrix (preconditioned CG method, ELLPACK storage format)
Markus Hegland Judith Helen Jenkinson Murray Leslie Dow	VMCFT	Singlevariate, multiple and multivariate discrete complex Fourier transform (mixed radix)
Markus Hegland Christopher Robert Dun	VLTQR	System of linear equations with real tridiagonal matrix (QR factorization)

Contributor	Subroutine name	Function
Margaret Helen Kahn	VHEVP	Eigenvalues and eigenvectors of Hermitian matrices (tridiagonalization, multisection method, and inverse iteration)
	VSEVP	Eigenvalues and eigenvectors of real symmetric matrices (tridiagonalization, multisection method, and inverse iteration)
Jeoffrey Keating	VMRFT	Multiple and multivariate discrete real Fourier transform (mixed radices of 2, 3, and 5)
	VSRFT	One-dimensional and multiple discrete real Fourier transform (mixed radices of 2, 3, and 5)
Zbigniew Leyk	VCRD	System of linear equations with an unsymmetric or indefinite sparse real matrix (MGCR method, diagonal storage format)
	VCRE	System of linear equations with an unsymmetric or indefinite sparse real matrix (MGCR method, ELLPACK storage format)
	VMVSD	Multiplication of a real sparse matrix and a real vector (diagonal storage format)
	VMVSE	Multiplication of a real sparse matrix and a real vector (ELLPACK storage format)
Zbigniew Leyk Murray Leslie Dow	VQMRD	System of linear equations with an unsymmetric or indefinite sparse real matrix (QMR method, diagonal storage format)
	VQMRE	System of linear equations with an unsymmetric or indefinite sparse real matrix (QMR method, ELLPACK storage format)
	VTFQD	System of linear equations with an unsymmetric or indefinite sparse real matrix (TFQMR method, diagonal storage format)
	VTFQE	System of linear equations with an unsymmetric or indefinite sparse real matrix (TFQMR method, ELLPACK storage format)
Zbigniew Leyk David Lawrence Harrar II	VLAND	Eigenvalues and eigenvectors of a real symmetric sparse matrix (Lanczos method, diagonal storage format)
Ole Møller Nielsen	VWFLT	Wavelet filter generation

Contributor	Subroutine name	Function
Ole Møller Nielsen Markus Hegland Gavin John Mercer	V1DWT	One-dimensional wavelet transform
Ole Møller Nielsen Markus Hegland	V2DWT	Two-dimensional wavelet transform
Michael Robert Osborne David Lawrence Harrar II	VTDEV	Eigenvalues and eigenvectors of real tridiagonal matrix

Index

A

approximately multiple..... II-65, II-133, II-142

B

Banded matrix linear equation.....I-1
banded real matrix II-29, II-34, II-73
BICG algorithm..... II-17, II-21
BICGSTAB method II-17, II-21
BICGSTAB(*l*) algorithm II-17, II-21
BICGSTAB(*l*) method..... I-1, I-18, II-14, II-18
Bi-Conjugate Gradient Stabilized(*l*) method .. II-14, II-18
Bluestein's algorithm..... II-98

C

CG method I-1
Chinese remainder theorem..... II-54
clustered eigenvalue II-65, II-133, II-142
condition number..... II-15, II-19
conjugate gradient algorithm II-43, II-49
conjugate gradient method.....I-18
convencence criterion..... II-18
convergence.....I-17
convergence criterion II-14, II-118, II-122, II-146, II-150

D

determinant of matrix II-24, II-29, II-32, II-73
diagonal storage format II-14, II-38, II-56, II-68, II-117, II-146
  for general sparse matrices.....I-6
  for symmetric positive definite sparse matrices I-8
diagonal-direction vector.....I-6
discrete complex Fourier transforms II-90, II-94
discrete real Fourier transforms..... II-99, II-105, II-136
discretization of partial differential operators and storage
  examples for them.....I-9
Double-precision random numbers.....I-1

E

eigenvalue problem I-1
eigenvalues and eigenvectors
  of a real symmetric sparse matrix..... II-68
  of Hermitian matrix.....I-1, I-63
  of real symmetric matrix II-131
  of real tridiagonal matrix.....I-1, II-140
eigenvalues distribution of matrix and convergence...I-18
elliptic partial differential equation I-9, II-41
ELLPACK storage format I-5, I-18, II-44, II-60, II-114, II-121, II-150
for symmetric positive definete sparse matrices.....I-7

F

factor..... II-50, II-125
first-order moment..... II-7
Fourier transforms I-2
four-step algorithm II-98

G

Gaussian elimination II-29
generalized Chinese remainder theorem..... II-55
generalized Fibonacci method II-12
GMRES method II-59, II-62

H

Householder method..... II-67
Householder reduction..... II-131

I

incomplete Cholesky decomposition II-38, II-41, II-45
Incomplete Cholesky decomposition..... II-47
in-place II-54
inverse iteration II-63, II-131, II-144
inverse of a positive-definite symmetric matrix..... II-79
iterative method.....I-1

J

judgment of convergency II-39, II-45, II-56, II-60
Judgment of convergency..... II-41, II-47

L

Lanczos method..... II-68, II-69, II-71
LDL^T decomposition II-22, II-26, II-79, II-83
linear equations with real tridiagonal matrix I-1, II-85
linear recurrence II-12
LU decomposition II-29, II-34

M

main diagonal vector I-6
MGCR method I-1, I-18, II-56, II-60
mixed radices..... II-90, II-94, II-99, II-105, II-136
modified Cholesky decomposition II-22, II-81
modified generalized conjugate residuals..... II-56, II-60
modified incomplete Cholesky decomposition.....I-1
multifrontal method II-87
multiple..... II-90, II-94, II-99, II-105

multiple and multivariate discrete real Fourier transform II-99, II-105
multiple eigenvalue II-65, II-133, II-142
multiple transform II-90, II-94
multiplication of a real band matrix and a real vector II-88
multiplication of a real sparse matrix II-111, II-114
multiplicity II-136
multivariate II-90, II-94, II-99, II-105
multivariate fourier transform II-99, II-105
mutual prime factor II-50, II-125, II-129

N

Neumann
  preconditioner II-38, II-45
  series I-1
normal random numbers I-1, II-1, II-5
normalisation I-17
normalization II-38, II-44, II-92, II-96, II-102, II-109, II-127, II-138
normalized symmetric positive definite sparse matrices I-7

O

one-dimension II-136
one-dimensional wavelet transform II-156

P

partial differential coefficient I-10
partial pivoting II-29, II-32, II-34
period II-2, II-12
pivot II-22, II-24, II-29, II-32, II-73, II-76, II-81, II-83
preconditioned CG method II-38, II-44
preconditioner I-1, II-38, II-41, II-44, II-47
prime factor
  fast Fourier transform II-50, II-53, II-130
  Fourier transform II-125
primitive trinomial II-12

Q

QMR method II-117, II-121
QR factorization II-85
quasi-minimal residual method II-117, II-121

R

random number subsequences II-2
real vector multiplication I-1

S

Scaling I-17
second-order moment II-7
second-order moments II-3
singlevariate II-90, II-94
  fast Fourier transform II-92, II-97
sparse matrices I-5
Sparse matrix linear equations I-1
statistical test II-12
storage method(s)
  for general sparse matrices I-5
  for sparse matrices I-5
  for symmetric positive definite sparse matrices I-7
  selection criteria I-9
storing diagonals, method of I-5
Sturm count II-144
symmetric positive definite
  banded matrix II-22, II-26, II-81
  sparse matrix II-38, II-44
symmetric positive matrix I-1

T

testing statistical hypotheses II-13
TFQMR method I-1, I-18, II-146, II-150
transpose-free quasi-minimal residual method II-146, II-150
two-dimensional wavelet transform II-160

U

uniform random numbers II-2, II-13
unsymmetric or indefinite I-1
unsymmetric or indefinite sparse matrix II-14, II-18, II-56, II-60, II-117, II-121, II-146, II-150

W

Wallace's method II-5, II-8
wavefront ordering II-43, II-49
wavelet
  filter generation II-153
  transform I-2, II-156, II-160

X

$\chi 2$ testing II-13