FUJITSU Software

FUJITSU

SSL II Thread-Parallel Capabilities

User's Guide
(Scientific Subroutine Library)

J2UL-2486-02ENZ0(00)
March 2020

Preface

This manual describes the functions and usage of the Scientific Subroutine Library II Thread-Parallel Capabilities.

SSL II Thread-Parallel Capabilities provide the computational functionality to efficiently compute or solve large-scale problems on a shared-memory parallel computer with scalar processors. New algorithms for parallel processing have been adopted.

The interfaces of subroutines are generally different from those used in the SSL II, SSLII/VP, SSL II/VPP or SSL II/HPF. This manual describes the usage of these subroutines.

Additionally, the SSL II Thread-Parallel Capabilities include some thread-parallelized routines derived from existing sequential SSL II double precision routines. The initial characters of these thread-parallelized routine names start with "DM_" to be distinguished from the sequential double precision routine names starting with "D". These routines can be used with the identical arguments with the sequential version. For the list of the thread-parallelized routines, refer to the section 2, "Thread-Parallelized routines from sequential SSL II functions" in "SSL II Thread-Parallel Capabilities Subroutines list" in this manual. And for each usage of them, refer to SSL II manuals of the sequential version.

This manual consists of two parts.

Part I General Description

General rules which should be kept in mind when using SSL II Thread-Parallel Capabilities are outlined.

Part II Usage of Subroutines

The functions and usage of each subroutine are described in alphabetical order of their subroutine names.

Readers of this manual are assumed to be familiar with the OpenMP Fortran. For details of the OpenMP Fortran specification, refer to "OpenMP Application Program Interface Version 2.5 May 2005."

For details of Fujitsu OpenMP Fortran compiler, refer to the Fortran User's Guide.

For how to store a sparse matrix and convergence of iterative methods, refer to the FUJITSU SSL II Extended Capabilities User's Guide II.

SSL II Thread-Parallel Capabilities include some functions using codes and algorithms, with appropriate modifications, which have been developed for SSL II/VPP. SSL II/VPP is the library developed in collaboration with the Australian National University (ANU). Development at the ANU has been led by professors Mike Osborne and Richard Brent and coordinated by Dr. Bob Gingold, Head, ANU Supercomputer Facility. The following is a complete list of those ANU experts involved in the design and implementation of SSL II/VPP. Fujitsu acknowledges their cooperation.

Professor Richard Peirce Brent
Dr Andrew James Cleary
Dr Murray Leslie Dow
Mr Christopher Robert Dun
Dr Lutz Grosz

Dr David Lawrence Harrar II
Dr Markus Hegland
Ms Judith Helen Jenkinson
Dr Margaret Helen Kahn
Dr Zbigniew Leyk
Mr David John Miron
Professor Michael Robert Osborne
Dr Peter Frederick Price
Dr Stephen Gwyn Roberts
Dr David Barry Singleton
Dr David Edward Stewart
Dr Bing Bing Zhou

Note

The asterisks in the table of contents and the subroutine list of this manual indicate items added or changed from the previous edition.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Date of Publication and Version

Version	Manual code
March 2020, 19th Version	J2UL-2486-02ENZ0(00)
January 2020, 18th Version	J2UL-2486-01ENZ0(00)
June 2016, 17th Version	J2UL-2070-02ENZ0(00)
September 2015, 16th Version	J2UL-2070-01ENZ0(00)
October 2014, 15th Version	J2UL-1906-01ENZ0(00)
March 2014, 14th Version	J2UL-1793-02ENZ0(00)
June 2013, 13th Version	—
February 2012, 12th Version	—
December 2011, 11th Version	—
February 2009, 10th Version	—
February 2008, 9th Version	—
April 2007 8th Version	—
March 2006 7th Version	—
June 2005, 6th Version	—
September 2004, 5th Version	—
December 2003, 4th Version	—
December 2002, 3rd Version	—
March 2002, 2nd Version	—
December 2000, 1st Version	—

Copyright

Update History

Changes	Location	Version
Option name “-KOMP” is corrected to “-Kopenmp”	2.4 How to Use SSL II Thread Parallel Capabilities	13th Version
A note related to the Neumann preconditioner is appended.	dm_vcgd, dm_vcge	
The following routine was added. • DM_VSRLU • DM_VSRLUX • DM_VSRS	SSL II Thread-Parallel Capabilities Subroutine List, Usage of Subroutines	14th Version
Rework format	Cover, Preface	15th Version
The following routine was added. • DM_VMVSCCC • DM_VRANU5 • DM_VSCLU • DM_VSCLUX • DM_VSCS	SSL II Thread-Parallel Capabilities Subroutine List, Usage of Subroutines	16th Version
Correction lack of description	DM_VSCHOL, DM_VSSPS	
Correction of a slip of the pen	DM_VSRS, DM_VSRLU, DM_VSRLUX	
The following routine was added. • DM_VSSSLU • DM_VSSSLUX • DM_VSSSS	SSL II Thread-Parallel Capabilities Subroutine List, Usage of Subroutines	17th Version
Correction of a wrong word	DM_VRANU5, DM_VSCLUX, DM_VSRLUX	
Correction of wrong sentences	SSL II Thread-Parallel Capabilities Subroutine List, DM_VAMLID, DM_VHTRID, DM_VLCSPSXCR1, DM_VLSPAXCR2, DM_VMLBIFE, DM_VRADAU5, DM_VRANU4, DM_VRANU5, DM_VSCLU, DM_VSCS, DM_VSRLU, DM_VSRS, DM_VSSSLU, DM_VSSSLUX, DM_VSSSS, References	18th Version
The explanation for the size of stack area for each thread is updated.	2.4 How to Use SSL II Thread Parallel Capabilities	
Changed the look according to product upgrades.	-	19th Version

SSL II Thread-Parallel Capabilities Subroutine List

1. Thread-Parallel routines adopting parallel algorithms for SMP

This manual describes the usage of the following thread-parallel routines adopting parallel algorithms suited for SMP machines.

Matrix operations

Subroutine name	Item	Page
DM_VMGGM	Matrix multiplication (real matrix)	II-149
DM_VMVSCC	Multiplication of a real sparse matrix and a real vector (compressed column storage method)	II-167
DM_VMVSCCC	Multiplication of a complex sparse matrix and a complex vector (compressed column storage method)	II-171
DM_VMVSD	Multiplication of a real sparse matrix and a real vector (diagonal format storage method)	II-174
DM_VMVSE	Multiplication of a real sparse matrix and a real vector (ELLPACK format storage method)	II-177

Linear equations (Direct method)

Subroutine name	Item	Page
DM_VLAX	A system of linear equations with real matrices (blocked LU decomposition method)	II-104
DM_VALU	LU decomposition of real matrices (blocked LU decomposition method)	II-1
DM_VLUX	A system of linear equations with LU-decomposed real matrices	II-146
DM_VLSX	A system of linear equations with symmetric positive definite matrices (blocked modified Cholesky decomposition method)	II-142
DM_VSLDL	LDL^T decomposition of symmetric positive definite matrices (blocked modified Cholesky decomposition method)	II-318
DM_VLDLX	A system of linear equations with LDL^T-decomposed symmetric positive definite matrices	II-125
DM_VLCX	A system of linear equations with complex matrices (blocked LU decomposition method)	II-122
DM_VCLU	LU decomposition of complex matrices (blocked LU decomposition method)	II-61

Linear equations (Iterative method)

Inverse matrices

Eigenvalue problem

Fourier transforms

Random numbers

2. Thread-Parallelized routines from sequential SSL II functions

This section lists thread-parallelized routines derived from sequential SSL II. The names of these routines begin with "DM_" instead of the initial letter "D" of the sequential SSL II double-precision routine names. These routines can be used with the identical arguments with the sequential version.

2.1 from Standard capabilities

Refer to the FUJITSU SSL II User's Guide when using the following routines.

Matrix manipulation

Subroutine name	Item
DM_VMAV	Multiplication of a real matrix by a real vector
DM_VMCV	Multiplication of a complex matrix by a complex vector

Least squares solution

Subroutine name	Item
DM_VLAXL	Least squares solution with a real matrix (Householder transformation)
DM_VLAXLM	Least squares minimal norm solution with a real matrix (Singular value decomposition method)
DM_GINV	Generalized Inverse of a real matrix (Singular value decomposition method)
DM_ASVD1	Singular value decomposition of a real matrix (Householder and QR methods)

Eigenvalues and eigenvectors

Subroutine name	Item
DM_EIG1	Eigenvalues and corresponding eigenvectors of a real matrix (double QR method)
DM_HSQR	Eigenvalues of a real Hessenberg matrix (double QR method)
DM_HVEC	Eigenvectors of a real Hessenberg matrix (Inverse iteration method)
DM_CHVEC	Eigenvectors of a complex Hessenberg matrix (Inverse iteration method)
DM_BLNC	Balancing of a real matrix
DM_CBLNC	Balancing of a complex matrix
DM_HES1	Reduction of a real matrix to a real Hessenberg matrix (Householder method)
DM_CHES2	Reduction of a complex matrix to a complex Hessenberg matrix (Stabilized elementary transformation)
DM_HBK1	Back transformation and normalization of the eigenvectors of a real Hessenberg matrix to the eigenvectors of the original matrix

Subroutine name	Item
DM_CHBK2	Back transformation of the eigenvectors of a complex Hessenberg matrix to the eigenvectors of the original matrix
DM_NRML	Normalization of eigenvectors
DM_CNRML	Normalization of eigenvectors of a complex matrix

2.2 from Extended capabilities

Refer to the FUJITSU SSL II Extended Capabilities User's Guide II when using the following routines

Eigenvalues and eigenvectors

Subroutine name	Item
DM_VLAND	Eigenvalues and eigenvectors of a real symmetric sparse matrix (Lanczos method, diagonal storage format)

Transforms

Subroutine name	Item
DM_VMCST	Discrete cosine transform
DM_VMSNT	Discrete sine transform
DM_VCCVF	Discrete convolution or correlation of complex data
DM_VRCVF	Discrete convolution or correlation of real data

Contents

Appendixes

Part I
General Descriptions

Chapter 1
Outline

SSLII Thread-Parallel Capabilities is a parallel mathematical subroutine library to execute on a shared-memory parallel computer with scalar processors. The library provides subroutines to efficiently compute such large-scale problems by parallel processing that are intractable on a single processor.

Each subroutine in the library is supplied as a subroutine written in OpenMP Fortran and can be called by a CALL statement in OpenMP Fortran environment.

The mechanism of "Thread-Parallel" means that multiple execution flows, each of which is called a thread, share the calculation where each thread is responsible for undertaking pieces of calculation using one CPU in the shared memory system. If the number of created threads is less or equal to the number of CPU available, the process can be executed by threads in parallel with all threads carried out by separated CPU. This Thread-Parallel mechanism enables a calculation to be divided into multiple parallel executions (as far as the algorithm could be parallelized).

Each subroutine of SSL II Thread-Parallel Capabilities creates multiple threads internally and solves the problem with a parallel algorithm with these threads. Where, the creation and extinction of the threads, work-sharing constructs and synchronization are directed with OpenMP Fortran specifications. Therefore SSL II Thread-Parallel Capabilities need the run-time execution environment of the OpenMP Fortran.

The number of the threads used by a subroutine of SSL II Thread-Parallel Capabilities can be assigned by the user with OpenMP environment variables or run-time library routines. With these, the subroutine can be executed by as any number of threads as specified.

The scope of functionality, subroutine names, and calling interface of SSL II Thread-Parallel Capabilities are different from those used in the mathematical library SSL II, SSLII/VP, SSL II/VPP or SSL II/HPF.

Chapter 2
General rules

2.1 Precision of Subroutines

SSL II Thread-Parallel Capabilities provides subroutines of double precision only.

2.2 Subroutine Names

The subroutine names that are callable by the user begin with DM_V, and the names of slave subroutines which are called internally begin with DM_U or DL_. And there is an auxiliary subroutine DMACH.

This manual describes the usage of the subroutines which are callable by the user.

2.3 Parameters

(1) Order of parameter sequence

In general, the order of parameter sequence is the same as that in standard SSL II:

(input and output parameter list, input parameter list, output parameter list, ICON)

(2) Parameter types

Parameters beginning with I, J, K, L, M, or N are of 4-byte integer type. Parameters beginning with other characters are of double precision type or double precision complex type.

2.4 How to Use SSL II Thread Parallel Capabilities

(1) Positions of the CALL statements

SSL II Thread-Parallel Capabilities consist of OpenMP subroutines which can be called from both inside and outside of the OpenMP parallel regions in user programs. And these subroutines also can be called from serial programs without OpenMP directives, and also they can be called from programs that are auto-parallelized by the Fortran compiler.

In cases where the subroutine is called from inside of the parallel region, it is necessary that every actual argument as input and output, out put and work areas which is dealt with by each thread must be mapped to different memory area respectively.

In every calling case above, the frt command option "-Kopenmp" must be specified at the time the compiled user program is to be linked with SSL II Thread-Parallel Capabilities. The load module can be OpenMP executable with this option. Refer to "Fortran User's Guide" for details.

(2) How to specify the number of threads

A subroutine of SSL II Thread-Parallel Capabilities is executed by multiple threads in parallel within parallel region which is created internal of the subroutine. The number of threads used by the subroutine can be assigned by the user with an OpenMP environment variable "OMP_NUM_THREADS" or a run-time library routine

"OMP_SET_NUM_THREADS( )". Usually, specify the number of threads in the former way.

The run-time library routine can be used in situations where the user wants to assign a specific number of threads for the parallel region. Specifying the number of threads with this run-time routine just before the SSL II Thread-Parallel subroutine makes it possible to execute the subroutine with a specific number of threads.

Refer to "Fortran User's Guide" and " OpenMP Application Program Interface Version 2.5 May 2005." for details about OpenMP environment variables and run-time library routines.

(3) Size of stack area for each thread

Some subroutines of SSL II Thread-Parallel Capabilities takes work area internally as auto allocatable array on "stack" area for each thread. Suppose that the number of threads to be generated is NT and the total available memory size is M, it is recommended to set the environmental variable OMP_STACKSIZE to about M/(5 × NT) as the stack size for each thread before the execution. When compiler option -Nfjomplib is specified, the environmental variable THREAD_STACK_SIZE can be set as the stack size. Refer to "Fortran User's Guide" for details about setting the stack size for OpenMP executables.

(4) Example programs

a. To call a subroutine from outside of the parallel region

The example program below solves a system of linear equations with input of a real coefficient matrix of 4000 × 4000. If the environment variable OMP_NUM_THREADS is set to be 4 on the system of 4 processors, execution will be with 4 threads in parallel.

```
      implicit real*8 (a-h,o-z)
      parameter(nord=4000,ld=nord+1)
c
      real*8    a(ld,nord),b(nord)
      integer ip(nord),is
c
      c=sqrt(2.0d0/dble(1+nord))
      t=datan(1.0d0)*4./(1+nord)
c
      do j=1,nord
      do i=1,nord
      a(i,j)=c*sin(t*i*j)
      enddo
      enddo
c
      do i=1,nord
      s=0.
      do j=1,nord
      s=s+sin(t*i*j)
      b(i)=s*c
      enddo
      enddo
c
      k=ld
      n=nord
      epsz=0.0d0
```

```
      isw=1
      call dm_vlax(a,k,n,b,epsz,isw,is,ip,icon)
      print*,'icon=',icon
      print*,'n=',n,', b(1)=',b(1),', b(n)=',b(n)
      stop
      end
```

b. To call subroutines from inside of the parallel region

The example program below solves two independent systems of linear equations. One input of a real coefficient matrix is 4000 × 4000, and the other is 4200 × 4200. If the environment variable OMP_NUM_THREADS is set to be 2 and OMP_NESTED is set to be TRUE on the system of 4 processors, each system of linear equation is solved with 2 threads respectively. The execution will be parallelized with 4 threads total.

```
      implicit real*8 (a-h,o-z)
      parameter(nord1=4000,ld1=nord1+1)
      parameter(nord2=4200,ld2=nord2+1)
c
      real*8    a1(ld1,nord1),b1(nord1),
     &          a2(ld2,nord2),b2(nord2),epsz1,epsz2
      integer ip1(nord1),ip2(nord2),is1,is2,
     &        icon1,icon2,n1,n2,k1,k2,num,
     &        omp_get_thread_num
c
      c=sqrt(2.0d0/dble(1+nord1))
      t=datan(1.0d0)*4./(1+nord1)
c
      do j=1,nord1
      do i=1,nord1
      a1(i,j)=c*sin(t*i*j)
      enddo
      enddo
c
      do i=1,nord1
      s=0.
      do j=1,nord1
      s=s+sin(t*i*j)
      b1(i)=s*c
      enddo
      enddo
c
      c=sqrt(2.0d0/dble(1+nord2))
      t=datan(1.0d0)*4./(1+nord2)
c
      do j=1,nord2
      do i=1,nord2
      a2(i,j)=c*sin(t*i*j)
      enddo
      enddo
c
```

```
      do i=1,nord2
      s=0.
      do j=1,nord2
      s=s+sin(t*i*j)
      b2(i)=s*c
      enddo
      enddo
c
!$OMP PARALLEL default(shared)
!$OMP+      private(num)
      num=omp_get_thread_num()
      if(num.eq.0)then
      k1=ld1
      n1=nord1
      epsz1=0.0d0
      isw1=1
      call dm_vlax(a1,k1,n1,b1,epsz1,isw1,is1,ip1,icon1)
      print*,'icon1=',icon1
      else
      k2=ld2
      n2=nord2
      epsz2=0.0d0
      isw2=1
      call dm_vlax(a2,k2,n2,b2,epsz2,isw2,is2,ip2,icon2)
      print*,'icon2=',icon2
      endif
!$OMP END PARALLEL
      print*,'n1=',n1,', b1(1)=',b1(1),', b1(n1)=',b1(n1)
      print*,'n2=',n2,', b2(1)=',b2(1),', b2(n2)=',b2(n2)
      stop
      end
```

2.5 Condition Codes

The parameter ICON is prepared to indicate the status after the execution of SSLII Thread-Parallel Capabilities.

A value between 0 and 39999 is set as the condition code. The values are classified as shown below depending on whether the result is guaranteed.

Table 2.1 Condition codes

Code	Meaning	Integrity of the result	Classification
0	Processing has ended normally.	The results are correct.	Normal
1 to 9999	Processing has ended normally, but auxiliary information was included.		
10000 to 19999	Processing has ended with the placing of internal restrictions during execution.	The results are correct on the restrictions.	Warning
20000 to 29999	Processing was discontinued due to abnormal conditions which had occurred during execution.	The results are not correct.	Abnormal
30000 to 39999	Processing was discontinued due to invalid input parameter.		

Part II
Usage of Subroutines

DM_VALU

LU decomposition of real matrices (blocked LU decomposition method)
CALL DM_VALU(A,K,N,EPSZ,IP,IS,ICON)

(1) Function

An $n \times n$ non-singular real matrix $\boldsymbol{A}$ is decomposed by blocked outer product type Gaussian elimination.

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

where, $\boldsymbol{P}$ is the permutation matrix which exchanges the rows of $\boldsymbol{A}$ by partial pivoting, $\boldsymbol{L}$ is the lower triangular matrix, and $\boldsymbol{U}$ is the unit upper triangular matrix ($n \geq 1$).

(2) Parameters

A ..... Input. Store matrix $\boldsymbol{A}$ in A(1:N,1:N).

Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored in A(1:N,1:N).

See Figure DM_VALU-1.

This is a two-dimensional double precision real array A(K,N).

K ..... Input. Size of the first dimension of the storage array A.

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

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, a standard value is assumed. (See note 1) in (3), "Comments on use".)

IP ..... Output. The transposition vector indicating the history of the exchange of rows by partial pivoting. One-dimensional array of size n. (See note 2) in (3), "Comments on use.")

IS ..... Output. Information to calculate the determinant of matrix A.

The determinant is obtained by multiplying the product of the n diagonal elements of array A by the value of IS after the calculation.

ICON ..... Output. Condition code.

See Table DM_VALU-1.

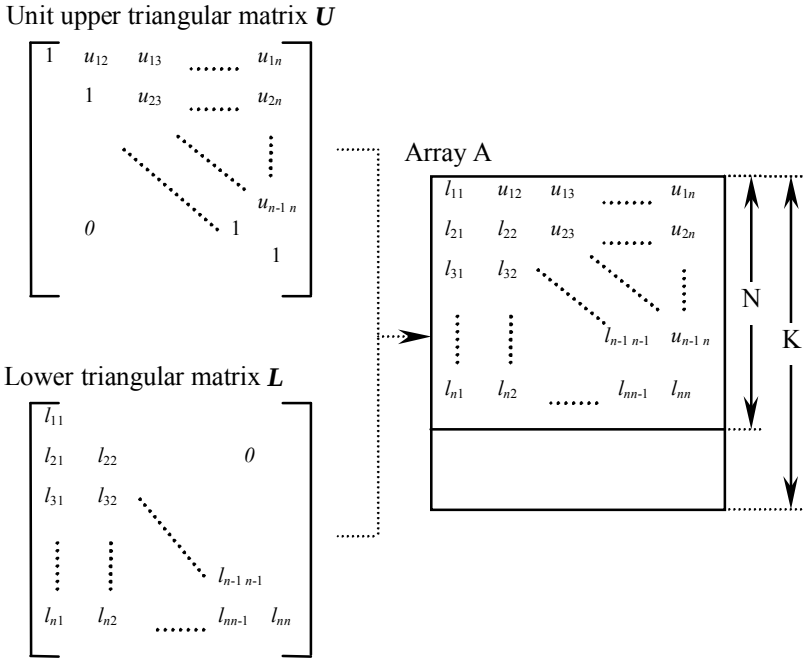


Figure DM_VALU-1 Storing *L* and *U* in array A after the calculation

After LU decomposition, matrix ***L*** and the upper triangular part (except the diagonal elements) of matrix ***U*** are stored in array A(1:N,1:N).

Table DM_VALU-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All elements in some row of array ***A*** were zero, or the pivot became relatively zero. Matrix ***A*** may be singular.	Processing is discontinued.
30000	K < N, N < 1, or EPSZ < 0.0	

(3) Comments on use

a. Notes

1) If a value is set for EPSZ, the value has the following meaning: if the absolute value of the selected pivot is less than EPSZ, the pivot is assumed to be zero and processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is 16*u*.

When computation is to be continued even if the pivot becomes small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) The transposition vector corresponds to the permutation matrix ***P*** in LU decomposition ***PA*** = ***LU*** with partial pivoting.

In this subroutine, the contents of array A are actually exchanged by partial pivoting. That is, when the I-th row (I ≥ J) is selected as the pivot row in the J-th

stage (J = 1, ..., *n*) of decomposition, the contents of the I-th row and J-th row of array A are exchanged. To indicate this exchange, I is stored in IP (J).

3) The linear equation can be solved by calling subroutine DM_VLUX following this subroutine. Normally, the linear equation can be solved in one step by calling subroutine DM_VLAX.

b. Example

LU decomposition is executed by inputting a real 4000 × 4000 matrix.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(4001,4000)
      DIMENSION IP(4000)
C
C
      N=4000
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A,N)
      DO J=1,N
      DO I=1,N
      A(I,J)=MIN(I,J)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO
C
      K=4001
      CALL DM_VALU(A,K,N,0.0D0,IP,IS,ICON)
      WRITE(6,610)ICON
      IF(ICON.GE.20000)STOP

      S=1.0D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A,N)
!$OMP+         REDUCTION(*:S)
      DO 20 I=1,N
      S=S*A(I,I)
   20 CONTINUE
!$OMP END PARALLEL DO

C
      DET=IS*S

   40 CONTINUE
      WRITE(6,620)DET
  610 FORMAT(1H0,10X,16HCONDITION CODE =,I5)
  620 FORMAT(1H0,10X,
     *27HDETERMINANT OF THE MATRIX =,D23.16)
      END
```

(4) Method

For details of the outer product type blocked LU decomposition method, see [1], [30], [54], [55], [56], and [70] in Appendix A, "References."

DM_VAMLID

System of linear equations with sparse matrices of M-matrix (Algebraic multilevel iteration method[AMLI Method], diagonal format storage method)
CALL DM_VAMLID (A, K, NDIAG, N, NOFST, B, ISW, IGUSS, INFO, EPSOT, EPSIN, X, W, NW, IW, NIW, ICON)

(1) Function

This subroutine solves, using the iterative method, a system of linear equations with sparse matrices of M-matrix as coefficient matrices. (See 1) in a, "Notes," in (3), "Comments on use.")

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

The $n \times n$ coefficient matrix is stored using the diagonal format storage method. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

The solution method is ORTHOMIN if $\boldsymbol{A}$ is symmetric and GMRES if $\boldsymbol{A}$ is non-symmetric. The iteration (called outer iteration) is preconditioned by the algebraic multilevel iteration method (called AMLI) which requires the solution of small linear system that is also solved iteratively (called inner iteration) , and stable. (In the preconditioner of the algebraic multilevel iteration method, the generated linear system becomes smaller as the level is deeper.)

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix $\boldsymbol{A}$ are stored in A.

The coefficient matrix is stored in A(1:N,1:NDIAG).

Two-dimensional array A(K,N).

For an explanation of the diagonal format storage method, see b, "Diagonal format storage method of general sparse matrices," in Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the *SSL II Extended Capability User's Guide II*.

K ..... Input. Size of first-dimension of array A($K \geq N$).

NDIAG ..... Input. Number of columns in array A and size of array NOFST. Must be equal to the number of nonzero diagonals in matrix $\boldsymbol{A}$.

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

NOFST ..... Input. Offsets of diagonals of $\boldsymbol{A}$ stored in array A. Main diagonal has offset 0, subdiagonals have negative offsets, and superdiagonals have positive offsets.

One-dimensional array NOFST(NDIAG).

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N) .

ISW ..... Input. Control information.

ISW=1 Initial calling.

ISW=2 Second or subsequent calling.

The values of A, IW and W must not be changed if the routine is called again with ISW=2.

(See 2) in a, “Notes,” in (3), “Comments on use.”)

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VAMLID.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

INFO ..... Input. The control information of the iteration.

One dimensional array of INFO(14).

For example, for symmetric coefficient matrix ***A***, INFO is set as follows;

INFO(1)=-1

INFO(2)=NTHRD×100

INFO(3)=0

INFO(5)=1

INFO(6)=2000

INFO(10)=1

INFO(11)=1000

For example, for unsymmetric coefficient matrix A, INFO is set as follows;

INFO(1)=-1

INFO(2)=NTHRD×100

INFO(3)=0

INFO(5)=2

INFO(6)=2000

INFO(7)=5

INFO(8)=20

INFO(10)=2

INFO(11)=1000

INFO(12)=10

INFO(13)=0

Where NTHRD is the number of threads which are executed in parallel.

INFO(1)=MAXLVL

Input. Maximal number of levels in the algebraic multilevel iteration method.

MAXLVL<0 The optimal level evaluated internally is used.

MAXLVL=0 The multi-level method is not used.

MAXLVL>0 The coarser level than the specified depth is not used.

(See 6),7) in a, “Notes,” in (3), “Comments on use.”)

INFO(2)=MINUK

Input. Minimal number of unknowns for the smallest linear system in the deepest level in the inner iteration. It is recommendable to set MINUK very larger than the number of threads NTHRD and very smaller than N. For example, 100×NTHRD.

INFO(3)=NORM

Input. The type of normalization.

NORM<1 The matrix is normalized from the right and the left by the inverse of the square root of the main diagonal of A. This effects that the main diagonal of the normalized matrix A is equal to one and the matrix is symmetric if A is symmetric.

It is recommendable to use symmetrical normalization. However, in some cases the non-symmetrical normalization can produce faster convergence. Criterion value for judgment of convergency.

(See 4) in a, “Notes,” in (3), “Comments on use.”)

NORM≥1 The matrix is normalized from the left by the inverse of the main diagonal of A. This effects that the main diagonal is equal to one but the normalized matrix will be non-symmetric even if the matrix A is symmetric.

(See 5) in a, “Notes,” in (3), “Comments on use.”)

INFO(4)

Output. Number of levels.

INFO(5)=METHOT

Input. The iterative method used in the outer iteration.

METHOT=1 Preconditioned ORTHOMIN is used. It should be used if the matrix ***A*** is symmetric and a symmetrical normalization is used.

METHOT≠1 Restarted and truncated GMRES is used. It should be used if the matrix ***A*** is non-symmetric or a non-symmetrical normalization is used.

INFO(6)=ITMXOT

Input. The maximal number of iteration steps in the outer iteration, for example 2000. If the maximum iteration number of outer iteration is reached the processing is terminated and the returned solution does not fulfill the stopping criterion.

INFO(7)=NRESOT

Input. The number of residuals in the orthogonalization procedure of the outer iteration, i.e. truncation after NRESOT residuals. For example, 5. Only used if GMRES is applied.

(See 5) in a, “Notes,” in (3), “Comments on use.”)

INFO(8)=NRSTOT

Input. Input. After NRSTOT iteration steps the outer iteration is restarted. For example , 20. If it is NRSTOT<1 there is no restart. Only used if GMRES is applied.

(See 5) in a, “Notes,” in (3), “Comments on use.”)

INFO(9)=ITEROT

Output. The number of iteration steps in the outer iteration procedure.

INFO(10)=METHIN

Input. The iterative method used in the inner iteration.

METHIN=1 Preconditioned ORTHOMIN is used. It should be used if the matrix A is symmetric and a symmetrical normalization is used.

METHIN≠1 Restarted and truncated GMRES is used. It should be used if the matrix ***A*** is non-symmetric or a non-symmetrical normalization is used.

INFO(11)=ITMXIN

Input. The maximal number of iteration steps in the inner iteration, for example 1000.

If ITMXIN is reached the processing is continued on the outer iteration.

INFO(12)=NRESIN

Input. The number of residuals in the orthogonalization procedure of the inner iteration, ie. truncation after NRESIN residuals. For example, 10. Only used if GMRES is applied.

(See 5) in a, "Notes," in (3), "Comments on use.")

INFO(13)=NRSTIN

Input. After NRSTIN iteration steps the inner iteration is restarted.

Only used if GMRES is applied. If it is NRSTIN<1 there is no restart.

(See 5) in a, "Notes," in (3), "Comments on use.")

INFO(14)

Output. The average number of the inner iteration.

EPSOT ..... Input. The desired accuracy for the solution. The outer iteration is stopped in the k-th iteration step if the normalized $\hat{r}_k = \hat{A}x_k - \hat{b}_k$ residual of the current approximation $\boldsymbol{x_k}$ satisfies the condition

$$\|\hat{r}_k\| \le \mathrm{EPSOT}\|\hat{b}\|$$

where $\|y\|^2 = y^{\mathrm{T}}y$ denotes the Euclidean norm $\hat{\boldsymbol{A}}$ and $\hat{b}$ are the coefficient matrix and the right hand side of the normalized linear system.

EPSIN ..... Input. The tolerance for the inner iteration. Normally 10^{-3} is optimal.

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X.

One-dimensional array X(N).

W ..... Work area. One-dimensional array W(NW) .

NW ..... Input. Size of the work array W.

NW ≥ NT×(3×NAMAX+5) +3×(NLVL+1)×NBAND×MAXT
+max(NAMAX ×NT, 7×NT+LR0(NT)),

where, NT=N+MAXT, and MAXT is the maximum number of threads which are created in this routine. NBAND is the maximum of the lower and upper

bandwidth of the matrix, NLVL is the number of levels in the algebraic multilevel iteration method. When INFO(1)<0, NLVL is 10. NAMAX≥NDIAG

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

LR0(NT)=4×NT if ORTHOMIN is used and

LR0(NT)=(2×NRES+1)×NT if GMRES with truncation after NRES residuals is used (See section 'Comment on use').

It is sufficient to set NRES=MAX(NRESOT,NRESIN).

IW ..... Work area. One-dimensional array IW(NIW).

NIW ..... Input. Size of the work array IW.

NIW ≥ MAXT×((6×MAXT+12×NAMAX)×(NLVL+1)+8×NBAND+3000)
+4×(N+MAXT),

where, MAXT is the maximum number of threads which are created in this routine. NBAND is the maximum of the lower and upper bandwidth of the matrix, NLVL is the number of levels in the algebraic multilevel iteration method. When INFO(1)<0, NLVL is 10. NAMAX≥NDIAG

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

ICON ..... Output. Condition code.

See Table DM_VAMILD-1.

Table DM_VAMILD-1 Condition codes

Code	Meaning	Processing
0	No error	–
10700	Vector $\boldsymbol{v}^{pos}$ could not be found.	Processing is used with $\boldsymbol{v}^{pos}$=1.
10800	Curable break down in GMRES.	Processing is continued.
20001	Stopping criterion could not be reached within the given number of iteration steps.	Processing is discontinued. The approximate value obtained is output in array X, but the precision is not assured.
20003	Non-curable break down in GMRES.	Processing is discontinued.
20005	Non-curable break down in ORTHOMIN by $\boldsymbol{p}^{T}\boldsymbol{A}\boldsymbol{p}$=0 with $\boldsymbol{p}\neq 0$.	
20006	Non-curable break down in ORTHOMIN by $\boldsymbol{p}^{T}\boldsymbol{r}$=0.	
30000	N<1,N>KA,NDIAG<1, ISW<1, ISW>2.	
30104	Incorrect diagonal offset NOFST.	
30105	Main diagonal is missed.	
30200	Matrix is not an M-matrix.	

Table DM_VAMILD-1 Condition codes

Code	Meaning	Processing
30210	Matrix condensation fails by non-positive value.	Processing is discontinued.
30212	There is a zero entry on the main diagonal.	
30310	Too small integer work array.	
30320	Too small real work array.	

(3) Comments on use

a. Notes

1) A coefficient matrix arising from order two finite difference discretization or, in some cases, from order one finite element discretization of an elliptical boundary value problem is an M-matrix. It can be produced using the routines for discretization of a boundary value problem for second order partial differential equation (DM_VPDE2D, DM_VPDE3D).

To be an M-matrix means that

- All main diagonal entries are positive $a_{i,i}>0$ for all $i=1,...,N$ and all other entries are non-positive $a_{i,j} \leq 0$ for all $i,j=1,...,N$ with $i \neq j$.
- There is a positive vector $\boldsymbol{v}^{pos}$ so $\boldsymbol{A}\boldsymbol{v}^{pos}$ is positive.

If the first condition is not fulfilled, processing is not continued with ICON = 30200. This routine can not find the vector $\boldsymbol{v}^{pos}$ (ICON = 10700) it is set $\boldsymbol{v}^{pos}$ =(1,...,1) the matrix $\boldsymbol{A}$ is assumed and processing is continued with the risk of a breakdown in AMLI with ICON = 30212, 30210 or slow convergence or breakdowns in the outer or inner iteration.

To define the coarse levels the rectangular grid used to assemble the coefficient matrix is recovered. If the recovering is not successful there can be a breakdown in AMLI with ICON=30212, 30210, a disproportionately increase of the number of diagonals in the coarser levels or slow convergence or breakdowns in the outer or inner iteration.

2) When multiple linear equations with the same coefficient matrix but different right hand side vectors are solved set ISW=1 in the first call and ISW=2 in the second and all subsequent calls. Then the coarse level matrices assembled in the first call are reused.

3) Normally it is sufficient to set NAMAX=NDIAG in the formulas for the length for the work arrays. It can happen that the number of diagonals in the coarse level matrices is larger than the number of diagonals in the given matrix. In this case NAMAX has to be increased.

4) It is always recommendable to use ORTHOMIN if possible. This requires that the matrix is symmetric. As this routine removes easily computable unknowns from the matrix before the iteration starts it can happen that the actual iteration matrix is symmetric even if the given matrix is not. Therefore it is recommendable to try ORTHOMIN with symmetrical normalization first if there is a chance that the iteration matrix is symmetric.

5) If the matrix is non-symmetric it is recommendable to use the non-symmetric normalization together with GMRES. Normally it is sufficient to truncate after NRESOT=5 residuals and to restart after 20 steps in the outer iteration. In the inner iteration it can be necessary to select a higher value for the truncation NRESIN and to restart after a larger number of iteration steps or even to forbid a restart. If NRESIN is increased it can happen that more real work space is required. Then it is necessary to increase NRES in the formula for the length workspace NW but, NRES can be set to a smaller value than NRESOT. In general the convergence of GMRES method becomes better as NRESIN and NRESOT are set to larger. But it requires longer computation time and larger amount of memory.

6) This routine tries to find the optimal number of levels. In some rare applications the computing time can be reduced by setting the number of levels by hand but normally the improvements are not significant.

7) The preconditioner bases on a nested incomplete block factorizations using the Schur complement. The matrix A_n, n=1,...,MAXLVL−1 of each level can be blocked as follows choosing the sets of eliminated unknown from the coordination in a virtual grid:

$$A_n = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

And define a matrix $S = A_{22} - A_{21}\, A_{11}^{-1}\, A_{12}$, which is called Schur complement. A_n can be factorized as follows:

$$A_n = \begin{bmatrix} A_{11} & 0 \\ A_{21} & I \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & S \end{bmatrix}.$$

The matrix $A_{n+1}$of next level n+1 can be regarded as a Schur complement matrix with approximating the A_{11}^{-1} to a diagonal matrix. These incomplete factorization are used for preconditioning in this routine.

b. Example

The partial differential equation

$$-\left(\frac{\partial^2 u}{\partial^2 x_1} + \frac{\partial^2 u}{\partial^2 x_2}\right) + cu = 1$$

is solved on the domain $[0,1]^2$. Dirichlet boundary conditions are set to

u=0.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE
      INTEGER   MAXT,N1,N2,KA,NA,NLVL,L1,L2,NW,NIW

      PARAMETER(MAXT=4,N1=1281,N2=1537,NLVL=10,
     &          L1=N1,L2=N2,
     &          KA=N1*N2,NA=5,
```

```
     &          NW=(3*NA+5)*(KA+MAXT)+3*(NLVL+1)*N1*MAXT
     &             +11*(KA+MAXT),
     &          NIW=((6*MAXT+12*NA)*(NLVL+1)
     &             +8*N1+2000)*MAXT+4*(KA+MAXT))

      INTEGER          NOFST(NA),INFO(100),IW(NIW)
      DOUBLE PRECISION X1(L1),X2(L2),
     &                 A1(L1,L2),A2(L1,L2),B1(L1,L2),B2(L1,L2),
     &                 C(L1,L2),F(L1,L2),
     &                 W(NW),EPSIN,EPSOT

      DOUBLE PRECISION A(KA,NA),B(KA),U(KA),SOL(3*N1*N2),
     &                 RHS(N1*N2),RHSC(N1*N2),TMP

      INTEGER Z1,Z2,NDIAG,N,ICON,ISW,IGUSS,I,Z,NBAND
C
C---------------------------------------------------------------
C
C***** CREATE NODE COORDINATES
C
      DO 11 Z1=1,N1
        X1(Z1)=DBLE(Z1-1)/DBLE(N1-1)
11    CONTINUE
      DO 12 Z2=1,N2
        X2(Z2)=DBLE(Z2-1)/DBLE(N2-1)
12    CONTINUE
C
C***** COEFFICIENTS IN THE PARTIAL DIFFERENTIAL EQUATION :
C
      DO 2000 Z2=1,N2

        DO 20 Z1=1,N1
          A1(Z1,Z2)=1
          A2(Z1,Z2)=1
          B1(Z1,Z2)=0
          B2(Z1,Z2)=0
          C (Z1,Z2)=1
          F (Z1,Z2)=1
20    CONTINUE
C
C***** DIRICHLET BOUNDARY CONDITIONS:
C
      C(1,Z2)=1
      F(1,Z2)=0
      C(N1,Z2)=1
      F(N1,Z2)=0
      IF (Z2.EQ.1) THEN
        DO 25 Z1=1,N1
          C(Z1,1)=1
          F(Z1,1)=0
 25     CONTINUE
      END IF
      IF (Z2.EQ.N2) THEN
```

```
        DO 26 Z1=1,N1
          C(Z1,N2)=1
          F(Z1,N2)=0
 26     CONTINUE
      END IF
2000  CONTINUE

      N=N1*N2
      CALL DM_VPDE2D(A1,L1,N1,N2,A2,X1,X2,B1,B2,C,F,A,KA,NA,N,
     &               NDIAG,NOFST,B,ICON)
      PRINT*,'ICON OF DM_VPDE2D =',ICON
      IF (ICON.GT.29999) GOTO 9999
C
      DO Z=1,N
      RHS(Z)=B(Z)
      ENDDO
      NBAND=0
      DO I=1,NDIAG
      NBAND=MAX(NBAND,ABS(NOFST(I)))
      ENDDO
C
C---------------------------------------------------------------
C
C**** CALL DAMLI:
C
      ISW=1
      IGUSS=0
C
      INFO(1)=-1
      INFO(2)=MAXT*100
      INFO(3)=0
      INFO(5)=1
      INFO(6)=2000
      INFO(10)=1
      INFO(11)=1000
C
      EPSOT=1.D-6
      EPSIN=1.D-3

      CALL DM_VAMLID(A,KA,NDIAG,N,NOFST,B,ISW,IGUSS,
     &     INFO,EPSOT,EPSIN,U,W,NW,IW,NIW,ICON)
      PRINT*,'ICON OF DM_VAMLID = ',ICON
      IF (ICON.GT.29999) GOTO 9999
C
9999  CONTINUE
C
      DO I=1,NBAND
      SOL(I)=0.0D0
      SOL(NBAND+N+I)=0.0D0
      ENDDO
      DO Z=1,N
      SOL(NBAND+Z)=U(Z)
      ENDDO
```

```
      CALL DM_VMVSD(A,KA,NDIAG,N,NOFST,NBAND,SOL,RHSC,ICON)
      TMP=0
      DO Z=1,N
      TMP=MAX(TMP,ABS((RHS(Z)-RHSC(Z))/(RHS(Z)+1.0)))
      ENDDO
C
      PRINT*,' ERROR = ',TMP
C
      END
```

(4) Method

Before the calculation starts the linear system is normalized to achieve that the main diagonal contains only the entries one. Moreover rows containing only zero entries outside the main diagonal (typically arising from Dirichlet boundary conditions) are removed from the matrix. The normalized system is solved by preconditioned ORTHOMIN or GMRES method see [79] in Appendix A, “References.” The AMLI preconditioner bases on a nested block incomplete factorizations using approximative Schur complements, see [6] in Appendix A, “References.” The set of simultaneously eliminated unknowns are defined by alternating direction technique after a virtual grid has been recovered from the diagonals of the matrix. The linear system on the coarsest level is normalized and is iteratively solved by ORTHOMIN or GMRES.

DM_VBCSCC

System of linear equations with unsymmetric or indefinite sparse matrices (Bi-Conjugate Gradient Stabilized (*l*) [BICGSTAB(*l*)] method, compressed column storage method)
CALL DM_VBCSCC (A, NZ, NROW, NFCNZ, N, B, ITMAX, EPS, IGUSS, L, X, ITER, W, IW, ICON)

(1) Function

This subroutine solves, using the BICGSTAB(*l*) method, Bi-Conjugate Gradient Stabilized(*l*) method, a system of linear equations with unsymmetric or indefinite sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the compressed column storage method. 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, "Outline," in the SSL II Extended Capability User's Guide II.

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A.

The coefficient matrix is stored in A(1:NZ).

One-dimensional array A(NZ)

For an explanation of the compressed column storage method, see Figure DM_VMVSCC-1 in the description of a DM_VMVSCC routine, "Multiplication of a real sparse matrix and a real vector (compressed column storage method)".

NZ ..... Input. The total number of the nonzero elements belong to a coefficient matrix ***A***.

NROW ..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ ..... Input. The position of the first nonzero element stored in an array A by the compressed column storage method which stores the nonzero elements column by column. NFCNZ(N+1) = NZ + 1.

One-dimensional array NFCNZ(N+1).

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

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N).

ITMAX ..... Input. Upper limit of iterative count for BICGSTAB(*l*) method. The value of ITMAX should usually be set to about 2000.

EPS ..... Input. Criterion value for judgment of convergence.

When the value of EPS is 0.0 or smaller, EPS is set to 10^{-6}.

(See 1) in a, "Notes," in (2), "Comments on use.")

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VBCSCC.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

L ..... Input. The order of stabilizer in BICGSTAB(*l*) method. $1 \leq L \leq 8$. The value of L should usually be set to 1 or 2.

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

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X.

One-dimensional array X(N).

ITER ..... Output. Actual iterative count for BICGSTAB(*l*) method.

W ..... Work area. One-dimensional array W(NZ).

IW ..... Work area. Two-dimensional array IW(2, NZ).

ICON ..... Output. Condition code.

See Table DM_VBCSCC-1.

Table DM_VBCSCC-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A breakdown state occurred.	Processing is discontinued.
20001	The iteration count reached the maximum limit.	Processing is discontinued. The already calculated approximate value is output to array X, but its precision is not assured.
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, ITMAX ≤ 0, L < 1, or L > 8.	Processing is discontinued.

(3) Comments on use

a. Notes

1) When the residual Euclidean norm is equal to or smaller than the product of the first residual Euclidean norm and the value of EPS, it is assumed that the solution converged. The error between the correct solution and the calculated approximate solution is roughly equal to the product of the matrix ***A*** condition number and the value of EPS.

2) When L is set to one, the algorithm is same as that of BICGSTAB method. As the value of L is lager, the cost of one iteration becomes larger however the total

number of iteration is reduced. Consequently in some cases it becomes faster with larger L.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

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

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine init_mat_diag. Then it is converted into the storage scheme in compressed column storage.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER (L = 4)

      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
     $          W(K*NDIAG),IW(2,K*NDIAG)
      DIMENSION X(N),B(N),SOLEX(N),Y(N)

      PRINT *,'    BICGSTAB(L) METHOD'
      PRINT *,'    COMPRESSED COLUMN STORAGE'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      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,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
```

```
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
      DO J=1,N
      NTOPCFG=1
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      NUMNZ=NUMNZ+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1

      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,SOLEX,
     $             B,W,IW,ICON)
      ERR1 = ERRNRM(SOLEX,X,N)
C
      X(1:N)=0.0D0
      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,X,
     $             Y,W,IW,ICON)
      ERR2 = ERRNRM(Y,B,N)

      IGUSS = 0
      ITMAX = 2000
      EPS=1.0D-8

      CALL DM_VBCSCC(A,NNZ,NROW,NFCNZ,N,B,ITMAX
     &          ,EPS,IGUSS,L,X,ITER,W,IW,ICON)

      ERR3 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,X,
```

```
     $             Y,W,IW,ICON)
      ERR4 = ERRNRM(Y,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    DM_VBCSCC ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *,'    ITER MAX = ',ITMAX
      PRINT *,'    ITER = ',ITER
      PRINT *
      PRINT *,'    EPS = ',EPS
      PRINT *
      PRINT *,'    INITIAL ERROR = ',ERR1
      PRINT *,'    INITIAL RESIDUAL ERROR = ',ERR2
      PRINT *,'    CRITERIA RESIDUAL ERROR = ',ERR2*EPS
      PRINT *
      PRINT *,'    ERROR = ',ERR3
      PRINT *,'    RESIDUAL ERROR = ',ERR4
      PRINT *
      PRINT *

      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &              ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
```

```
          K0 = (JS-1)/NXY+1
          IF (K0 .GT. NZ) THEN
      PRINT*,'ERROR; K0.GH.NZ '
      GOTO 100
      ENDIF
          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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C ======================================
* ABSOLUTE ERROR
* | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
```

```
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The BICG algorithm is described in [72] in Appendix A, "References." The BICGSTAB(*l*) algorithm is a modification of the BICGSTAB method, see [77] and[32] in Appendix A, "References."

DM_VBCSD

System of linear equations with unsymmetric or indefinite sparse matrices (Bi-Conjugate Gradient Stabilized (*l*) [BICGSTAB(*l*)] method, diagonal format storage method)
CALL DM_VBCSD (A, K, NDIAG, N, NOFST, B, ITMAX, EPS, IGUSS, L, X, ITER, ICON)

(1) Function

This subroutine solves, using the BICGSTAB(*l*) method, Bi-Conjugate Gradient Stabilized(*l*) method, a system of linear equations with unsymmetric or indefinite sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the diagonal format storage method. 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, "Outline," in the SSL II Extended Capability User's Guide II.

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A.

The coefficient matrix is stored in A(1:N,1:NDIAG).

Two-dimensional array A(K,NDIAG)

For an explanation of the diagonal format storage method, see b, "Diagonal format storage method of general sparse matrices," in Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the SSL II Extended Capability User's Guide II.

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

NDIAG ..... Input. Number of columns in array A and size of array NOFST. Must be greater than or equal to the number of nonzero diagonals in matrix $\boldsymbol{A}$. Size of second-dimension of array A.

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

NOFST ..... Input. Offsets of diagonals of $\boldsymbol{A}$ stored A. Main diagonal has offset 0, subdiagonals have negative offsets, and superdiagonals have positive offsets.

One-dimensional array NOFST(NDIAG)

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N).

ITMAX ..... Input. Upper limit of iterative count for BICGSTAB(*l*) method. The value of ITMAX should usually be set to about 2000.

EPS ..... Input. Criterion value for judgment of convergence.

When the value of EPS is 0.0 or smaller, EPS is set to 10^{-6}.

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

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VBCSD.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

L ..... Input. The order of stabilizer in BICGSTAB(*l*) method. $1 \leq L \leq 8$. The value of L should usually be set to 1 or 2.

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

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X.

One-dimensional array X(N).

ITER ..... Output. Actual iterative count for BICGSTAB(*l*) method.

ICON ..... Output. Condition code.

See Table DM_VBCSD-1.

Table DM_VBCSD-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A breakdown state occurred.	Processing is discontinued.
20001	The iteration count reached the maximum limit.	Processing is discontinued. The already calculated approximate value is output to array X, but its precision is not assured.
30000	N < 1, N > K, NDIAG < 1, ITMAX ≤ 0, L < 1, or L > 8.	Processing is discontinued.
32001	|NOFST(I)| > N – 1	

(3) Comments on use

a. Notes

1) When the residual Euclidean norm is equal to or smaller than the product of the first residual Euclidean norm and the value of EPS, it is assumed that the solution converged. The error between the correct solution and the calculated approximate solution is roughly equal to the product of the matrix ***A*** condition number and the value of EPS.

2) Conditions for using the diagonal format

The external diagonal vector element of coefficient matrix A must be set to 0. The order in which diagonal vectors (refer to Section 3.2.1.1, "Storage method for general sparse matrices" in the SSL II Extended Capabilities User's Guide II) are stored into array A is not restricted.

The merit of this method is that a matrix vectors can be calculated without using an indirect index. The demerit of this method is that a matrix without a diagonal structure cannot be stored efficiently.

3) When L is set to one, the algorithm is same as that of BICGSTAB method. As the value of L is lager, the cost of one iteration becomes larger however the total number of iteration is reduced. Consequently in some cases it becomes faster with larger L.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

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

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine init_mat_diag.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (EPS = 1D-8)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER (L = 4)
      PARAMETER(NVW=3*K)

      DIMENSION NOFST(NDIAG)
      DIMENSION A(K,NDIAG)
      DIMENSION X(N),B(N),SOLEX(N),Y(N)
      DIMENSION VW(NVW)

      PRINT *,'    BICGSTAB(L) METHOD'
      PRINT *,'    DIAGONAL FORMAT'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      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)
 NBANDL=0
 NBANDR=0
 DO I=1,NDIAG
 IF(NOFST(I).LT.0)THEN
 NBANDL=MAX(NBANDL,-NOFST(I))
 ELSE
 NBANDR=MAX(NBANDR,NOFST(I))
 ENDIF
 ENDDO

 VW(1+NBANDL:N+NBANDL) = SOLEX(1:N)
 CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON2)

 X(1:N)=0.0D0
 ERR1 = ERRNRM(SOLEX,X,N)
 VW(1+NBANDL:N+NBANDL) = X(1:N)
 CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,Y,ICON2)
 ERR2 = ERRNRM(Y,B,N)

 IGUSS = 0
 ITMAX = 2000

 CALL DM_VBCSD(A,K,NDIAG,N,NOFST,B,ITMAX
&          ,EPS,IGUSS,L,X,ITER,ICON)

 ERR3 = ERRNRM(SOLEX,X,N)
 VW(1+NBANDL:N+NBANDL) = X(1:N)
 CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,Y,ICON2)
 ERR4 = ERRNRM(Y,B,N)

 PRINT *,'    COMPUTED VALUES'
 PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
 PRINT *
 PRINT *,'    DM_VBCSD ICON = ',ICON
 PRINT *
 PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
 PRINT *,'    NBANDL = ',NBANDL,', NBANDR = ',NBANDR
 PRINT *,'    ITER MAX = ',ITMAX
 PRINT *,'    ITER = ',ITER
 PRINT *
 PRINT *,'    EPS = ',EPS
 PRINT *
 PRINT *,'    INITIAL ERROR = ',ERR1
 PRINT *,'    INITIAL RESIDUAL ERROR = ',ERR2
 PRINT *,'    CRITERIA RESIDUAL ERROR = ',ERR2*EPS
 PRINT *
 PRINT *,'    ERROR = ',ERR3
 PRINT *,'    RESIDUAL ERROR = ',ERR4
 PRINT *
 PRINT *
```

```
      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF

      STOP
      END

C =====================================
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &               ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
        DO 100 J = 1,LEN
          JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
          K0 = (JS-1)/NXY+1
          IF (K0 .GT. NZ) THEN
      PRINT*,'ERROR; K0.GH.NZ '
      GOTO 100
      ENDIF
          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
 100    CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

        RETURN
        END

C =======================================
* ABSOLUTE ERROR
* | X1 - X2 |
C =======================================
        REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
        IMPLICIT  REAL*8 (A-H,O-Z)
        DIMENSION X1(*),X2(*)
C
        S = 0D0
        DO 100 I = 1,LEN
          SS = X1(I) - X2(I)
          S = S + SS * SS
 100    CONTINUE
C
        ERRNRM = SQRT( S )
        RETURN
        END
```

(4) Method

The BICG algorithm is described in [72] in Appendix A, "References." The BICGSTAB(*l*) algorithm is a modification of the BICGSTAB method, see [77] and[32] in Appendix A, "References."

DM_VBCSE

System of linear equations with unsymmetric or indefinite sparse matrices (Bi-Conjugate Gradient Stabilized (*l*) [BICGSTAB(*l*)] method, ELLPACK format storage method)
CALL DM_VBCSE (A, K, IWIDT, N, ICOL, B, ITMAX, EPS, IGUSS, L, X, ITER, ICON)

(1) Function

This subroutine solves, using the BICGSTAB(*l*) method, Bi-Conjugate Gradient Stabilized (*l*) method, a system of linear equations with unsymmetric or indefinite sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the ELLPACK format storage method. 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, "Outline," in the SSL II Extended Capability User's Guide II.

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A(1:N,1:IWIDT).

Two-dimensional array A(K,IWIDT)

For an explanation of the ELLPACK format storage method, see Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the SSL II Extended Capability User's Guide II.

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

IWIDT ..... Input. Maximum number of row-vector-direction nonzero elements of coefficient matrix $\boldsymbol{A}$. Size of second-dimension of A and ICOL.

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

ICOL ..... Input. Column index used in ELLPACK format. Used to indicate to which column vector the corresponding element of A belongs.

Two-dimensional array ICOL(K,IWIDT)

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N)

ITMAX ..... Input. Upper limit of iterative count for BICGSTAB(*l*) method. The value of ITMAX should usually be set to about 2000.

EPS ..... Input. Criterion value for judgment of convergence.

When the value of EPS is 0.0 or smaller, EPS is set to 10^{-6}.

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

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VBCSE.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

L ..... Input. The order of stabilizer in BICGSTAB(*l*) method. $1 \leq L \leq 8$. The value of L should usually be set to 1 or 2.

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

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X(1:N).

One-dimensional array X(N)

ITER ..... Output. Iterative count for BICGSTAB(*l*) method.

ICON ..... Output. Condition code.

See Table DM_VBCSE-1.

Table DM_VBCSE-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A breakdown state occurred.	Processing is discontinued.
20001	The iteration count reached the maximum limit.	Processing is discontinued. The already calculated approximate value is output to array X, but its precision is not assured.
30000	K < 1, IWIDT < 1, N < 1, ITMAX ≤ 0, N > K, L < 1, or L > 8.	Processing is discontinued.
30001	The band width is zero.	

(3) Comments on use

a. Notes

1) When the residual Euclidean norm is equal to or smaller than the product of the first residual Euclidean norm and the EPS, it is assumed that the solution converged. The error between the correct solution and the calculated approximate solution is roughly equal to the product of the matrix A condition number and the EPS.

2) When L is set to one, the algorithm is same as that of BICGSTAB method. As the value of L is lager, the cost of one iteration becomes larger however the total number of iteration is reduced. Consequently in some cases it becomes faster with larger L.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

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

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Ellpack format is generated by the subroutine init_mat_ell.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (EPS = 1D-8)
      PARAMETER (NORD=60,NX =NORD ,NY = NORD,NZ = NORD,
     &           N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (IWIDT = 7)
      PARAMETER (L = 4)
      DIMENSION ICOL(K,IWIDT)
      DIMENSION A(K,IWIDT)
      DIMENSION X(N),B(N),SOLEX(N),Y(N)

      PRINT *,'    BICGSTAB(L) METHOD'
      PRINT *,'    ELLPACK FORMAT'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL
     &          ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)

      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON2)

      X(1:N)=0.0D0
      ERR1 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,X,Y,ICON2)
      ERR2 = ERRNRM(Y,B,N)

      IGUSS = 0
      ITMAX = 2000

      CALL DM_VBCSE(A,K,IWIDT,N,ICOL,B,ITMAX
```

```
     &          ,EPS,IGUSS,L,X,ITER,ICON)

      ERR3 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,X,Y,ICON2)
      ERR4 = ERRNRM(Y,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    DM_VBCSE ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *,'    ITER MAX = ',ITMAX
      PRINT *,'    ITER = ',ITER
      PRINT *
      PRINT *,'    EPS = ',EPS
      PRINT *
      PRINT *,'    INITIAL ERROR = ',ERR1
      PRINT *,'    INITIAL RESIDUAL ERROR = ',ERR2
      PRINT *,'    CRITERIA RESIDUAL ERROR =',ERR2*EPS
      PRINT *
      PRINT *,'    ERROR = ',ERR3
      PRINT *,'    RESIDUAL ERROR = ',ERR4
      PRINT *
      PRINT *

      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF

      STOP
      END

C =======================================
C INITILIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_ELL(VA1,VA2,VA3,VC,A_L,ICOL_L,NX,NY,NZ
     &          ,XL,YL,ZL,IWIDTH,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION A_L(NDIVP,IWIDTH)
      DIMENSION ICOL_L(NDIVP,IWIDTH)
C
      IF (IWIDTH .LT. 1) THEN
         WRITE (*,*) 'SUBROUTINE INIT_MAT_ELL:'
         WRITE (*,*) ' IWIDTH SHOULD BE GREATER THAN OR EQUAL TO 1'
         RETURN
      ENDIF
!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+  SHARED(VA1,VA2,VA3,VC,A_L,ICOL_L,NX,NY,NZ
!$OMP+        ,XL,YL,ZL,IWIDTH,LEN,NDIVP)
```

```
C IWIDTH CANNOT BE GREATER THAN 7
      IWIDTH_LOC = IWIDTH
      IF (IWIDTH .GT. 7) IWIDTH_LOC = 7

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

!$OMP DO
      DO J = 1,IWIDTH
      DO I = 1,NDIVP
      A_L(I,J) = 0.0
      ICOL_L(I,J) = I
      ENDDO
      ENDDO
!$OMP ENDDO

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        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) THEN
    PRINT*,' ERROR; K0.GT.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
C ABSOLUTE ERROR
C | X1 - X2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
```

```
      END
```

(4) Method

The BICG algorithm is described in [72] in Appendix A, "References." The BICGSTAB(*l*) algorithm is a modification of the BICGSTAB method, see [77] and[32] in Appendix A, "References."

DM_VBLU

LU decomposition of banded real matrices (Gaussian elimination)
CALL DM_VBLU(A,K,N,NH1,NH2,EPSZ,IS,IP,ICON)

(1) Function

This subroutine executes LU decomposition for banded matrix $\boldsymbol{A}$ of $n \times n$, lower bandwidth h_1, and upper bandwidth h_2 using Gaussian elimination.

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

where, $\boldsymbol{P}$ is the permutation matrix of the row vector, $\boldsymbol{L}$ is the unit lower banded matrix, and $\boldsymbol{U}$ is the upper banded matrix.

$n > h_1 \geq 0$, $n > h_2 \geq 0$

(2) Parameters

A ..... Input. Store banded coefficient matrix $\boldsymbol{A}$.

Matrix $\boldsymbol{A}$ is stored in A(NH1 + 1:2 × NH1 + NH2 +1,1:N). For A(1:NH1, 1:N), set zero for the elements of matrix $\boldsymbol{A}$ outside the band.

See Figure DM_VBLU-1.

Output. LU-decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored.

See Figure DM_VBLU-2.

This is a double precision real two-dimensional array A(K,N).

The value of A(2×NH1+NH2+2:K, 1:N) is not assured after operation.

K ..... Input. The size of first dimension of array A(≥ 2×NH1+NH2+1).

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

NH1 ..... Input. Lower bandwidth size h_1.

NH2 ..... Input. Upper bandwidth size h_2.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is set.

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

IS ..... Output. Indicates row vector exchange count.

When IS is 1, exchange count is even.

When IS is -1, exchange count is odd.

(See 4) in (3), “Comments on use.”)

IP ..... Output. One-dimensional array of size n. The transposition vector to contain row exchange information is stored.

(See note 2) in (3), “Comments on use.”)

ICON ..... Output. Condition code.

See Table DM_VBLU-1.

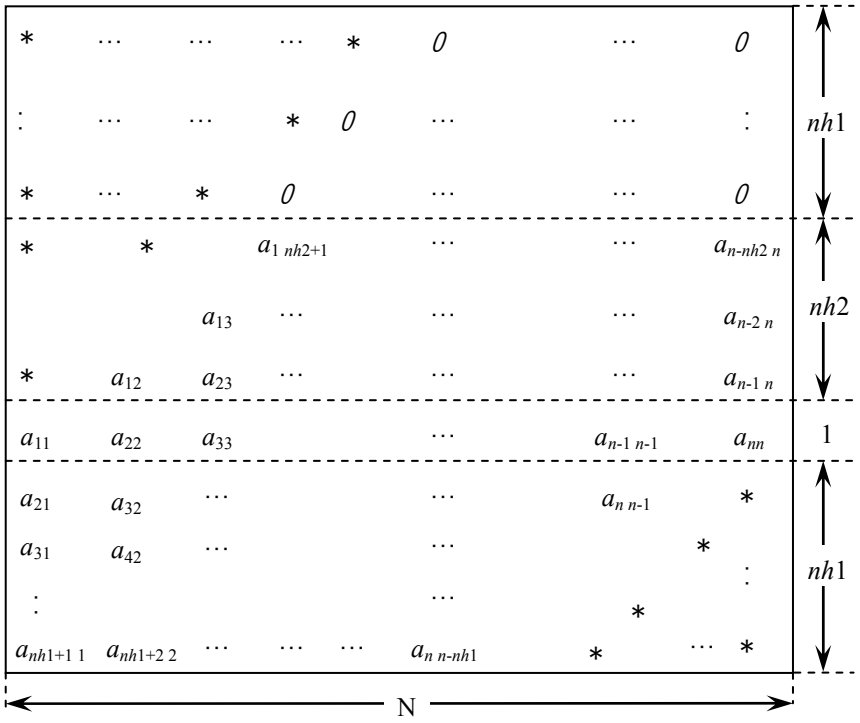


Figure DM_VBLU-1 Storing matrix *A* in array A

The column vector of matrix ***A*** is continuously stored in columns of array A in the same manner as diagonal elements of banded matrix ***A*** a_{ii}, i = 1, ..., n, are stored in A($nh_1 + nh_2 + 1$,1:n).

Upper banded matrix part

$a_{j-i,j}$, i = 1, ..., nh_1, j = 1, ..., n, $j - i \geq 1$ is stored in A ($nh_1 + 1$:$nh_1 + nh_2$, + 1,1:n).

Lower banded matrix part

$a_{j+i,j}$, i = 1, ... , nh_1, j = 1, ... , n, $j + i \leq n$ is stored in A($nh_1 + nh_2 + 2$:$2 \times nh_1 + nh_2 + 1$, 1:n). For A(1:nh_1,1:n), set zero for the elements of matrix ***A*** outside the band.

* indicates undefined values.

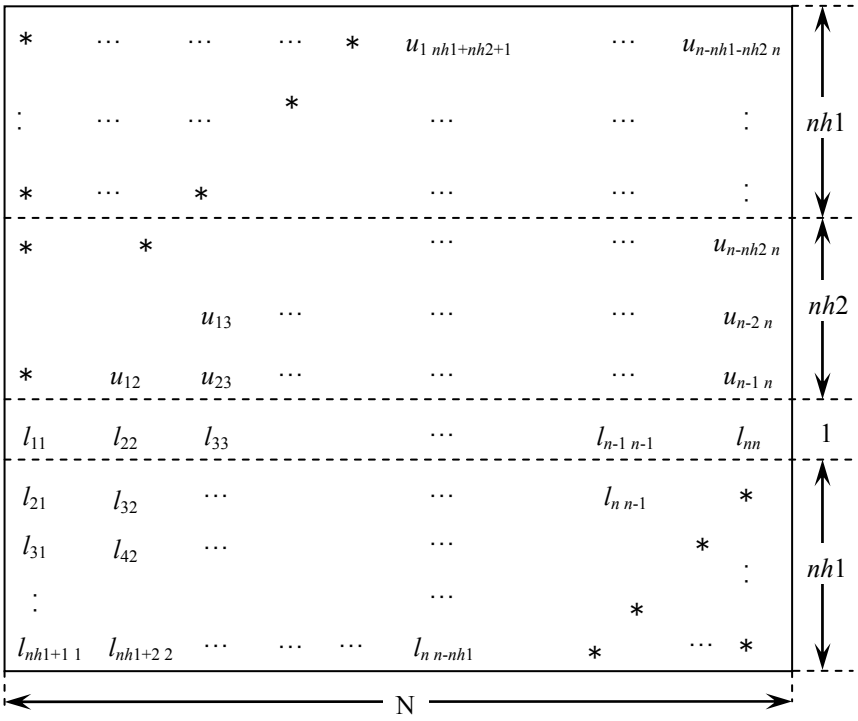


Figure DM_VBLU-2 Storing LU-decomposed matrix *L* and *U* in array A

LU-decomposed unit upper banded matrix except diagonal elements

$u_{j-i+1,j}$, $i = 1, \ldots, h_1 + h_2, j = 1, \ldots, n, j - i + 1 \geq 1$ is stored in A($1{:}h_1 + h_2, 1{:}n$).

Lower banded matrix part

$l_{j+i,j}$, $i = 0, \ldots, h_2, j = 1, \ldots, n, j + i \leq n$ is stored in A($h_1 + h_2 + 1{:}2 \times h_1 + h_2 + 1, 1{:}n$).

* indicates undefined values.

Table DM_VBLU-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	All elements in some row of array ***A*** were zero, or the pivot became relatively zero. Matrix ***A*** may be singular.	Processing is discontinued.
30000	N < 1, NH1 ≥ N, NH1 < 0, NH2 ≥ N, NH2 < 0, K < 2×NH1+NH2+1, EPSZ < 0.	

(3) Comments on use

a. Notes

1) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$.

When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) In this subroutine, the row vector is exchanged using partial pivoting. That is, when the I-th row (I ≥ J) is selected as the pivot row in the J-th stage (J = 1, ..., n) of decomposition, the contents of the I-th row and J-th row are exchanged. To indicate this exchange, I is stored in IP (J).

3) The linear equation can be solved by calling subroutine DM_VBLUX following this subroutine. Normally, the linear equation can be solved in one step by calling subroutine DM_VLBX.

4) The determinant can be obtained by multiplying IS and A($h_1 + h_2 + 1$,i), where i = 1, ... , n.

b. Example

The system of linear equations with banded matrices is solved with the input of a banded real matrix of n = 10000, nh_1 = 2000, nh_2 = 3000.

```
      implicit real*8(a-h,o-z)
      parameter(nh1=2000,nh2=3000,n=10000)
      parameter(ka=2*nh1+nh2+1,n2=n)
      real*8 a(ka,n2),b(n),dwork(4500)
      integer ip(n)

c
      ix=123
      nwork=4500
      nn=nh1+nh2+1
      do i=1,n
      call dvrau4(ix,a(nh1+1,i),nn,dwork,nwork,icon)
      do j=1,nh1+nh2+1
      enddo
      enddo
c
c     zero clear
c
      print*,'nh1=',nh1,',nh2=',nh2,',n=',n
```

```
c
c     a(1:nh1,n)=0.0d0
c
      do j=1,n
      do i=1,nh1
      a(i,j)=0.0d0
      enddo
      enddo
c
c     left upper triangular part
c
      do j=1,nh2
      do i=1,nh2+1-j
      a(i+nh1,j)=0.0d0
      enddo
      enddo
c
c    right rower triangular part
c
      nbase=2*nh1+nh2+1
      do j=1,nh1
      do i=1,j
      a(nbase-i+1,n-nh1+j)=0.0d0
      enddo
      enddo
c
c     set right hand constant vector
c
      do i=1,n
      b(i)=0.0d0
      enddo
c
      do i=1,n
      nptr=i-1
      do j=max(nptr+1-nh2,1),min(n,nptr+nh1+1)
      b(j)=b(j)+a(j-i+nh1+nh2+1,i)
      enddo
      enddo

c
      epsz=0.0d0
      call gettod(tt1)
      call dm_vblu(a,ka,n,nh1,nh2,epsz,is,ip,icon )
      call gettod(tt2)
      print*,'factor time (wall clock)=',(tt2-tt1)*1.0d-6
c
      call gettod(tt1)
      call dm_vblux(b,a,ka,n,nh1,nh2,ip,icon)
      call gettod(tt2)
      print*,'solve time (wall clock)=',(tt2-tt1)*1.0d-6

c
      tmp=0.0d0
      do i=1,n
      tmp=max(tmp,dabs(b(i)-1))
      enddo
c
      print*,'maximum error =',tmp
c
      stop
      end
```

(4) Method

LU-decomposition is executed using outer product type Gaussian elimination.

DM_VBLUX

A system of linear equations with LU-decomposed banded real matrices
CALL DM_VBLUX(B,FA,K,N,NH1,NH2,IP,ICON)

(1) Function

This subroutine solves a linear equation having an LU-decomposed banded matrix as coefficient.

$\boldsymbol{LUx} = \boldsymbol{b}$

where, $\boldsymbol{L}$ is a unit lower banded matrix of lower bandwidth h_1, $\boldsymbol{U}$ is an upper banded matrix of upper bandwidth h (= min ($h_1 + h_2$, n - 1)), and $\boldsymbol{b}$ is an n-dimensional real constant vector. The order of matrix $\boldsymbol{A}$ before LU decomposition, lower bandwidth, and upper bandwidth is n, h_1, and h_2.

$n > h_1 \geq 0$, $n > h_2 \geq 0$

(2) Parameters

B ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

Double precision real one-dimensional array B(N).

FA ..... Input. LU-decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored.

See Figure DM_VBLUX-1.

This is a Double precision real two-dimensional array FA(K,N).

The value of FA(2×NH1+NH2+2:K, 1:N) is not assured after operation.

K ..... Input. The size of first dimension of array FA(≥ 2×NH1+NH2+1).

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

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

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

IP ..... Input. One-dimensional array of size n. Transposition vector which indicates the history of row vector exchange.

ICON ..... Output. Condition code.

See Table DM_VBLUX-1.

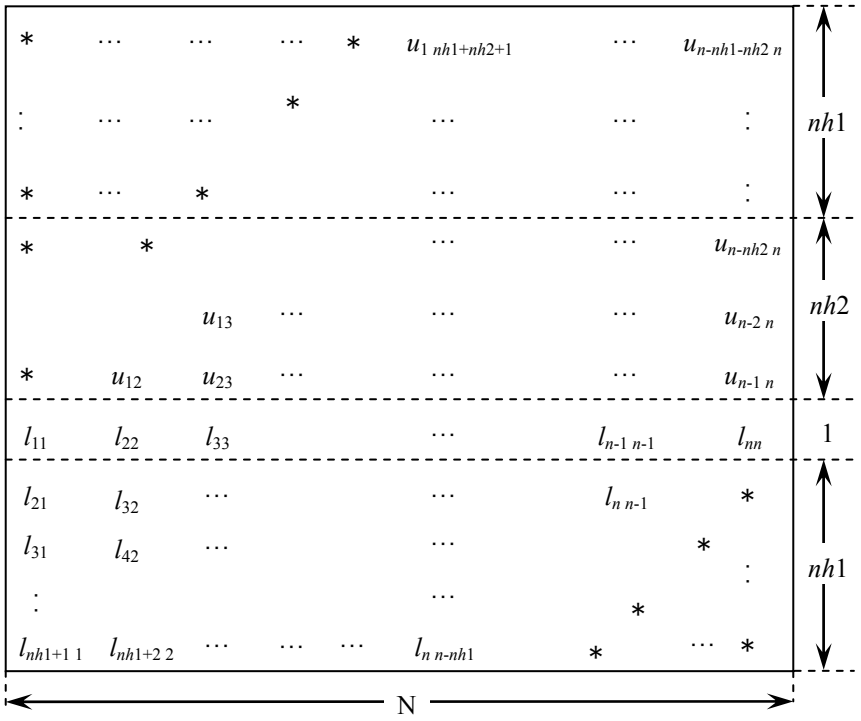


Figure DM_VBLUX-1 Storing LU-decomposed matrices *L* and *U* into array FA

LU-decomposed unit upper banded matrix except diagonal elements

$u_{j-i+1,j}$, $i = 1, \ldots, h_1 + h_2, j = 1, \ldots, n, j - i + 1 \geq 1$ is stored in FA($1{:}h_1 + h_2, 1{:}n$).

Lower banded matrix part

$l_{j+i,j}$, $i = 0, \ldots, h_2, j = 1, \ldots, n, j + i \leq n$ is stored in FA($h_1 + h_2 + 1{:}2 \times h_1 + h_2 + 1, 1{:}n$).

* indicates undefined values.

Table DM_VBLUX-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	N < 1, NH1 ≥ N, NH1 < 0, NH2 ≥ N, NH2 < 0, K < 2×NH1+NH2+1. Diagonal element of lower banded matrix was zero. Contents of IP are invalid.	Processing is discontinued.

(3) Comments on use

a. Notes

1) A system of linear equations with banded matrices can be solved by calling this subroutine following the subroutine DM_VBLU. In this case, specify the output parameters of the subroutine DM_VBLU without modification of the input parameters (except the constant vector) of this subroutine. Normally, a solution can be obtained in one step by calling the subroutine DM_VLBX.

b. Example

The system of linear equations with banded matrices is solved with the input of a banded real matrix of $n = 10000$, $h_1 = 2000$, $h_2 = 3000$.

```
      implicit real*8(a-h,o-z)
      parameter(nh1=2000,nh2=3000,n=10000)
      parameter(ka=2*nh1+nh2+1,n2=n)
      real*8 a(ka,n2),b(n),dwork(4500)
      integer ip(n)

c
      ix=123
      nwork=4500
      nn=nh1+nh2+1
      do i=1,n
      call dvrau4(ix,a(nh1+1,i),nn,dwork,nwork,icon)
      do j=1,nh1+nh2+1
      enddo
      enddo
c
c     zero clear
c
      print*,'nh1=',nh1,',nh2=',nh2,',n=',n
c
c     a(1:nh1,n)=0.0d0
c
      do j=1,n
      do i=1,nh1
      a(i,j)=0.0d0
      enddo
      enddo
c
c     left upper triangular part
c
      do j=1,nh2
      do i=1,nh2+1-j
      a(i+nh1,j)=0.0d0
      enddo
      enddo
```

```
c
c    right rower triangular part
c
      nbase=2*nh1+nh2+1
      do j=1,nh1
      do i=1,j
      a(nbase-i+1,n-nh1+j)=0.0d0
      enddo
      enddo
c
c     set right hand constant vector
c
      do i=1,n
      b(i)=0.0d0
      enddo
c
      do i=1,n
      nptr=i-1
      do j=max(nptr+1-nh2,1),min(n,nptr+nh1+1)
      b(j)=b(j)+a(j-i+nh1+nh2+1,i)
      enddo
      enddo

c
      epsz=0.0d0
      call gettod(tt1)
      call dm_vblu(a,ka,n,nh1,nh2,epsz,is,ip,icon )
      call gettod(tt2)
      print*,'factor time (wall clock)=',(tt2-tt1)*1.0d-6
c
      call gettod(tt1)
      call dm_vblux(b,a,ka,n,nh1,nh2,ip,icon)
      call gettod(tt2)
      print*,'solve time (wall clock)=',(tt2-tt1)*1.0d-6

c
      tmp=0.0d0
      do i=1,n
      tmp=max(tmp,dabs(b(i)-1))
      enddo
c
      print*,'maximum error =',tmp
c
      stop
      end
```

(4) Method

The linear equation with LU-decomposed matrices as coefficient is solved by forward and back-substitution.

DM_VCGD

A system of linear equations with symmetric positive definite sparse matrices (preconditional CG method, diagonal format storage method)
CALL DM_VCGD(A,K, NW, N,NDLT,B,IPC,ITMAX,ISW,OMEGA,EPS, IGUSS,X,ITER,RZ,W,IW,ICON)

(1) Function

This subroutine solves a linear equation having an $n \times n$ normalized symmetric positive definite sparse matrix as coefficient matrix using the preconditioned CG method.

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

The $n \times n$ matrix coefficient is normalized so that its diagonal elements are 1, and non-zero elements except the diagonal elements are stored using the diagonal format spares matrix storage method.

For the normalization of a linear equation with a symmetric positive definite sparse matrix as its coefficient matrix and the diagonal format storage method, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2, "Storage method for symmetric positive definite sparse matrix." For the diagonal format storage method, it is assumed that non-zero elements of the coefficient matrix ***A*** exist only in vectors on some diagonal lines parallel to the main diagonal vector.

When the linear equation is obtained by discretizing a partial differential equation on the lattices parallel to the boundary of the specifically defined domain, it has the structure described above.

In this case, information indicating the position (column vector of coefficient matrix) of each element is not necessary. Only the distance from the main diagonal vector is required. This enables efficient execution.

(2) Parameters

A ..... Input. The normalized sparse matrix is stored in A(1:N,1:NW).

The value of A(N + 1:K,*) is not assured after operation.

Two-dimensional array A(K,NW).

Non-zero elements of the coefficient matrix of normalized symmetric positive definite sparse matrix are stored in diagonal format.

For the diagonal format storage method for normalized symmetric positive definite sparse matrices, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2, "Storage method for symmetric positive definite sparse matrix," b., "Diagonal format storage method for symmetric positive definite sparse matrix."

K ..... Input. The size of the first dimension of array A ($\geq n$).

NW ..... Input. Number of vectors in the diagonal direction where the coefficient matrix ***A*** is stored using the diagonal format storage method. Even number. The size of the second dimension of array A.

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

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

For the diagonal format storage method for normalized symmetric positive definite sparse matrices, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2, "Storage method for symmetric positive definite sparse matrix," b., "Diagonal format storage method for symmetric positive definite sparse matrix."

B ..... Input. The constant vector of right-hand-side terms of linear equation is stored in B(1:N).

One-dimensional array B(N).

IPC ..... Input. Preconditioner control information.

When 1: No preconditioner.

When 2: Neumann preconditioner.

When 3: Preconditioner using block incomplete Cholesky decomposition.
In this case, OMEGA needs to be specified.

(See note 3) in (3), "Comments on use.")

ITMAX ..... Input. Upper limit of the iteration count (≥ 0).

ISW ..... Input. Control information.

1: Initial calling.

2: Second or subsequent calling.

The values of A, NDLT, W, and IW must not be changed because the values set at the initial calling are used for these parameters.

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

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

This is used when IPC = 3.

(See note 3) in (3), "Comments on use.")

EPS ..... Input. Value used for convergency judgment.

When 0 is set, $\varepsilon |\boldsymbol{b}|$ is set as EPS. For ε, 10^{-6} is set.

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

IGUSS ..... Input. Sets the information indicating whether the iteration is started from an approximate value of the solution vector specified in array X.

When 0 is set, the approximate value of the solution vector is not specified.

When non-zero is set, the iterative computation is started from an approximate value of the solution vector specified in array X.

X ..... Input. An approximate value of the solution vector of the linear equation can be specified in X(1:N).

Output. The solution vector linear equation is stored in X(1:N).

This is a one-dimensional array X(N).

ITER..... Output. The actual iteration count.

RZ ..... Output. The square root of the residual *rz* after the convergency judgment.

(See note 2) in (3), “Comments on use.”)

W ..... Work area.

When IPC=3, two-dimensional array of size W(N+MAXT, NW+8).

When IPC≠3, two-dimensional array of size W(N+MAXT, 7),

where MAXT is the maximum number of threads executed in parallel.

IW ..... Work area.

When IPC=3, two-dimensional array of size IW(N+2×MAXT,4).

When IPC≠3, two-dimensional array of size IW(MAXT,2),

where MAXT is the maximum number of threads executed in parallel.

ICON ..... Output. Condition code. See Table DM_VCGD-1.

Table DM_VCGD-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	Diagonal vectors in A were reordered as U/L in ascending distance order.	Processing is continued.
20001	The upper iteration count limit was reached.	Processing is discontinued. The approximate value obtained is output in array X, but the precision is not assured.
20003	Breakdown occurred.	
30003	ITMAX ≤ 0	Processing is discontinued.
30005	K < N	
30006	Incomplete LL^T decomposition could not be performed.	
30007	The pivot became minus.	
30089	NW is not an even number.	
30091	NBAND = 0	
30092	NW ≤ 0	
30093	K ≤ 0, n ≤ 0	
30096	OMEGA < 0, OMEGA > 1	
30097	IPC < 1, IPC > 3	
30102	The upper triangular part is not correctly stored.	
30103	The lower triangular part is not correctly stored.	

Table DM_VCGD-1 Condition codes

Code	Meaning	Processing
30104	The number of diagonal vectors in the upper triangular does not equal that in the lower triangular.	Processing is discontinued.
30105	ISW ≠ 1, 2	
30200	|NDLT(i) | > n–1 or NDLT(i) = 0	

(3) Comments on use

a. Notes

1) When multiple sets of the linear equations having the same coefficient matrix and different constant vectors are solved with IPC = 3:

- First, they are solved with ISW = 1.
- Second, they are solved with ISW = 2.

In the second and subsequent operations, the linear equations are solved by reusing the result of incomplete Cholesky decomposition obtained in the first calling.

2) Convergency judgment

A judgment on whether the n-th iteration solution converged is made when $RZ = \sqrt{(rz)} < EPS$ is satisfied.

Where, $rz = \boldsymbol{r}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{r}$, and $\boldsymbol{r}$ is the residual vector $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_n$, $\boldsymbol{M}$ is the preconditioner matrix.

3) Preconditioner

Two types of preconditioners and a function without a preconditioner are provided.

To solve an elliptic partial differential equation by use a discretization, use a preconditioner derived by the incomplete Cholesky method.

When $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{N}$, the preconditioner $\boldsymbol{M}$ of 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 Block incomplete Cholesky method $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$, where $\boldsymbol{M}$ is the preconditioner matrix which is constituted from incomplete Cholesky decomposed matrices of the each blocked matrix of $\boldsymbol{A}$ that is partitioned by the number of threads executed in parallel.

When IPC=2, the preconditioner also must be a positive definite matrix. For example, diagonal dominance of the matrix ($\boldsymbol{I} + \boldsymbol{N}$) is a sufficient condition for the positive definiteness. Additionally, note that using a preconditioner may not improve the convergence when the preconditioner does not approximate the inverse matrix of $\boldsymbol{A}$ in some situations such that the maximum absolute value of the eigenvalues of the matrix $\boldsymbol{N}$ is larger than one.
When IPC=3, the user must specify a value for OMEGA($0 \leq \mathrm{OMEGA} \leq 1$).

When OMEGA = 0, the incomplete Cholesky method is used. When OMEGA = 1, a modified incomplete Cholesky decomposition method is used.

For linear equations obtained from the discretization of a partial differential equation, it has been proved that the optimal value of OMEGA is between 0.92 and 1.00.

When IPC = 3, the equation is rearranged in order of wavefront, to increase the efficiency of the preconditioner.

b. Example

This example solves a system of linear equations with symmetric positive definition matrices in which n = 51200.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER ND,N,KA,WMAX,NDIAG
      PARAMETER (ND=80,MAXT=4,N=ND**3,KA=N)
      PARAMETER (WMAX=8)
      REAL*8  A(KA,WMAX),B(KA),X(KA),OMEGA,EPS
      REAL*8  IW(MAXT,2),W(N+MAXT,7)
      INTEGER DELTA(WMAX),IPREC,ITER,ITMAX
C
      CALL LAP3D(A,DELTA,KA,N,ND,WMAX,NDIAG)
C
      CALL RHS(A,N,KA,NDIAG,W,DELTA,B)
C
      EPS=1D-6
      ITMAX=2000
      ISW=1
      IGUSS=0
      IPREC=2
C
      CALL DM_VCGD(A,KA,NDIAG,N,DELTA,B,IPREC,ITMAX,ISW,OMEGA,
     &     EPS,IGUSS,X,ITER,RZ,W,IW,ICON)
      PRINT*,'ICON=',ICON
      PRINT*,'X(1)=',X(1)
      PRINT*,'X(N)=',X(N)
      STOP
      END
C
      SUBROUTINE LAP3D(A,DELTA,KA,N,ND,NDMAX,NDIAG)
      INTEGER NDMAX,NDIAG,N,I,J,L
      INTEGER DELTA(NDMAX),ND,NX,NY
      REAL*8  A(KA,NDMAX)

      DO J=1,NDMAX
      DO I=1,KA
      A(I,J)=0D0
      ENDDO
```

```
      ENDDO

      DO J=1,NDMAX
      DELTA(J)=0
      ENDDO

C 3D PROBLEM
      NDIAG=6
      NX=ND
      NY=ND
      DO I=1,N
      L=I
      IF((L/NX)*NX.NE.L.AND.L.LE.N-1) THEN
        A(I,1)=-1.D0/6.D0
      ENDIF
      ENDDO
      DO I=1,N
      L=I
      IZ=(L-1)/(NX*NY)
      IY=(L-1-IZ*NX*NY)/NX
      IF(L.LE.N-NX.AND.IY.NE.NY-1) THEN
        A(I,2)=-1.D0/6.D0
      ENDIF
      ENDDO
      DO I=1,N
      L=I
      IF(L.LE.N-NX*NY) THEN
        A(I,3)=-1.D0/6.D0
      ENDIF
      ENDDO
      DO I=1,N
      L=I
      IF(((L-1)/NX)*NX.NE.L-1.AND.L.GE.2.AND.L.LE.N) THEN
        A(I,4)=-1.D0/6.D0
      ENDIF
      ENDDO
      DO I=1,N
      L=I
      IZ=(L-1)/(NX*NY)
      IY=(L-1-IZ*NX*NY)/NX
      IF(L.GE.NX+1.AND.L.LE.N.AND.IY.NE.0) THEN
        A(I,5)=-1.D0/6.D0
      ENDIF
      ENDDO
      DO I=1,N
      L=I
      IF(L.GE.NX*NY+1.AND.L.LE.N) THEN
        A(I,6)=-1.D0/6.D0
      ENDIF
      ENDDO
      DELTA(1)=1
      DELTA(2)=NX
      DELTA(3)=NX*NY
```

```
      DELTA(4)=-1
      DELTA(5)=-NX
      DELTA(6)=-NX*NY
      RETURN
      END
C
      SUBROUTINE RHS(A,N,KA,NDIAG,DP,DELTA,B)
      IMPLICIT NONE
      INTEGER N,KA,NDIAG,I,J,DSHIFT
      REAL*8 DP(*),A(KA,*),B(KA)
      INTEGER DELTA(*),ICON
C
      DSHIFT=0
      DO J=1,NDIAG
      DSHIFT=MAX(DSHIFT,ABS(DELTA(J)))
      ENDDO
      DO I=1,3*N
      DP(I)=0
      ENDDO
      DO I=1,N
      DP(I+DSHIFT)=1.D0
      ENDDO
      CALL DM_VMVSD(A,KA,NDIAG,N,DELTA,DSHIFT,DP,B,ICON)
      DO I=1,N
      B(I)=B(I)+DP(DSHIFT+I)
      ENDDO
      RETURN
      END
```

(4) Method

The standard conjugate gradient method algorithm is used. (See [30] in Appendix A, “References.”) For the incomplete Cholesky method preconditioner, see [58] in Appendix A, “References.” For vectorization by wavefront ordering, see [45] in Appendix A, “References.” For the diagonal format storage method for sparse matrices, see [59], [52] in Appendix A, “References.”

(5) Acknowledgement

Fujitsu is grateful to the authors of ITPACK and NSPCG who permitted the use of the routines of modified incomplete Cholesky decomposition and wavefront ordering.

DM_VCGE

A system of linear equations with symmetric positive definite sparse matrices (preconditional CG method, ELLPACK format storage method)
CALL DM_VCGE(A,K, NW, N,ICOL,B,IPC,ITMAX,ISW,OMEGA,EPS, IGUSS,X,ITER,RZ,W,IW,ICON)

(1) Function

This subroutine solves a linear equation having an $n \times n$ normalized symmetric positive definite sparse matrix as a coefficient matrix using the preconditioned CG method.

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

The $n \times n$ coefficient matrix is normalized so that the diagonal elements are 1, and the non-zero elements except the diagonal elements are stored by the ELLPACK format storage method.

For the normalization of linear equations with symmetric positive definite sparse matrices as coefficient matrices, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2, "Storage method for symmetric positive definite sparse matrices."

(2) Parameters

A ..... Input. The normalized sparse matrix is stored in A(1:N,1:NW).

This is a two-dimensional array A(K, NW).

For the ELLPACK format storage method for normalized symmetric positive definite sparse matrices, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2·a, "ELLPACK format storage method for symmetric positive definite sparse matrix."

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

K ..... Input. Size of the first dimension of arrays A and ICOL (≥ N). Multiple of NTHRD.

NW ..... Input.

When the maximum numbers of non-zero elements of row vectors of upper and lower triangular matrices are NSU and NSL, respectively, $2 \times max$ (NSU, NSL).

For details, refer to the SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2·a, "ELLPACK format storage method for symmetric positive definite sparse matrix."

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

IOCL ..... Input. The information on the column vector to which non-zero elements belong is stored in ICOL(1:N,1:NW).

This is a two-dimensional array ICOL(K, NW).

B ..... Input. The constant vector of right-hand-side terms of the linear equation is stored in B(1:N). This is a one-dimensional array B(N).

IPC ..... Input. Preconditioner control information.

When 1: No preconditioner.

When 2: Neumann preconditioner.

When 3: Preconditioner using block incomplete Cholesky decomposition. In this case, OMEGA needs to be specified.

(See note 4) in (3), "Comments on use.")

ITMAX ..... Input. Upper limit of the iteration count. Positive integer.

ISW ..... Input. Control information. (> 0)

1: Initial calling.

2: Second or subsequent calling. The values of A, ICOL, W, and IW must not be changed because the values set at the initial calling are used in second or subsequent calls.

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

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

EPS ..... Input. Value used for convergency judgment.

When RZ < EPS, it is assumed that convergency occurred.

When EPS = 0, $\varepsilon |\boldsymbol{b}|$ is set as EPS. For ε, 10^{-6} is set.

(See note 3) in (3), "Comments on use.")

IGUSS ..... Input. Sets the information indicating whether an iteration is started from an approximate value of the solution vector specified in array X.

When 0 is set, no approximate value of the solution vector is specified.

When non-zero is set, an iterative computation is started from an approximate value of the solution vector specified in array X.

X ..... Input. An approximate value of the solution vector of the linear equation can be specified in X(1:N).

After operation, output. The solution vector of the linear equation is stored in X(1:N).

This is a one-dimensional array X(N).

ITER ..... Output. The actual iteration count.

RZ ..... Output. The square root of residual *rz* after the convergency judgment. (See note 2) in (3), "Comments on use.")

W ..... Work area.

When IPC=3, two-dimensional array of size W(N+MAXT, NW+8).

When IPC≠3, two-dimensional array of size W(N+MAXT, 7),

where MAXT is the maximum number of threads executed in parallel.

IW ..... Work area.

When IPC=3, two-dimensional array of size IW(N+2×MAXT, NW+5).

When IPC≠3, two-dimensional array of size W(MAXT, 2),

where MAXT is the maximum number of threads executed in parallel.

ICON ..... Output. condition code.

See Table DM_VCGE-1.

Table DM_VCGE-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	Elements of A and ICOL are rearranged as U/L.	Processing is continued.
20001	The iteration count reaches the upper limit.	Processing is discontinued. The approximate values that have been obtained are output in array X, but precision is not assured.
20003	Breakdown occurred.	
30003	ITMAX ≤ 0	Processing is discontinued.
30005	K < N	
30006	Incomplete $\boldsymbol{LL}^{\mathrm{T}}$ decomposition could not be executed.	
30007	Pivot became minus.	
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	The upper triangular part or the lower triangular part is not correctly stored.	
Negative number	The non-diagonal element is present in the –icon row.	

(3) Comments on use

a. Notes

1) The sparse matrix is stored using the ELLPACK format storage method. (See Appendix A, "References," [45], [62].)

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

When IPC is other than 3 (when a preconditioner other than that using the incomplete Cholesky decomposition is specified), a less constrained storage method than those described in the following is accepted: SSL II Extended Capability User's Guide II, Part I, "Overview," Section 3.2.1.2·a, "ELLPACK format storage method for symmetric positive definite sparse matrix." That is, the following sparse matrix is also accepted as input: A normalized symmetric positive definite sparse matrix excluding diagonal elements stored using the

ELLPACK format storage method for general sparse matrix. In this case, the value of NW need not be 2 × *max*(NSU,NSL).

2) When multiple sets of linear equations having the same coefficient matrix and different constant vectors are solved with IPC = 3:

   - The primary is solved using ISW = 1.
   - The secondary, is solved using ISW = 2.

   In the second and subsequent operations, the linear equations are solved by reusing the result of the incomplete Cholesky decomposition obtained in the first calling.

3) Convergency judgment

   A judgement on whether the *n*-th iteration solution has converged is made when $RZ = \sqrt{(rz)} < EPS$ is satisfied.

   Where, $rz = \boldsymbol{r}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{r}$, and $\boldsymbol{r}$ is a residual vector $\boldsymbol{r} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_n$, and $\boldsymbol{M}$ is a preconditioner matrix.

4) Preconditioner

   Two types of preconditioners and a function without a preconditioner are provided.

   When $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{N}$, 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 Block incomplete Cholesky method $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$, where $\boldsymbol{M}$ is the preconditioner matrix which is constituted from incomplete Cholesky decomposed matrices of the each blocked matrix of $\boldsymbol{A}$ that is partitioned by the number of threads executed in parallel.

   When IPC=2, the preconditioner also must be a positive definite matrix. For example, diagonal dominance of the matrix ($\boldsymbol{I} + \boldsymbol{N}$) is a sufficient condition for the positive definiteness. Additionally, note that using a preconditioner may not improve the convergence when the preconditioner does not approximate the inverse matrix of $\boldsymbol{A}$ in some situations such that the maximum absolute value of the eigenvalues of the matrix $\boldsymbol{N}$ is larger than one.
   When IPC=3, the user must specify a value for OMEGA($0 \leq \mathrm{OMEGA} \leq 1$).

   When OMEGA = 0, the incomplete Cholesky method is used. When OMEGA = 1, a modified incomplete Cholesky decomposition method is used.

   For a linear equation obtained from the discretization of the partial differential equation, it is proved that the optimal value of OMEGA is 0.92 to 1.00.

   When IPC = 3, the equation is rearranged in order of wavefront, to increase the efficiency of the preconditioner.

b. Example

   This example solves the system of linear equations with symmetric positive definition matrix with $n = 51200$.

   The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4

when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER NMAX,N,WMAX,W
      PARAMETER (MAXT=4,NORD=80,WMAX=6)
      PARAMETER (NMAX=NORD**3,N=NMAX)

      REAL*8  RS(NMAX),X(NMAX),EPS,OMEGA,AP(NMAX),RZ
      REAL*8  A(NMAX,WMAX),XW(NMAX+MAXT,7)
      INTEGER ICOL(NMAX,WMAX),XIW1(MAXT,2),IPREC,I,ITMAX,ITER
C
      CALL SET(A,ICOL,NMAX,N,NORD,WMAX)
      DO I=1,N
      AP(I)=1.0D0
      ENDDO
      W=6
      CALL DM_VMVSE(A,NMAX,W,N,ICOL,AP,RS,ICON)
      DO I=1,N
      RS(I)=RS(I)+1.0D0
      ENDDO
      ITMAX=2000
      EPS=1D-6
      ISW=1
      IPREC=2
      IGUSS=0
      CALL DM_VCGE(A,NMAX,W,N,ICOL,RS,IPREC,ITMAX,ISW,OMEGA,EPS,
     &             IGUSS,X,ITER,RZ,XW,XIW1,ICON)
      PRINT*,'ICON = ',ICON
C
      PRINT*,'X(1)=',X(1)
      PRINT*,'X(N)=',X(N)
C
      STOP
      END
C
      SUBROUTINE SET(A,ICOL,NMAX,N,NORD,WMAX)
      INTEGER WMAX,N,I,J
      INTEGER ICOL(NMAX,WMAX),NORD
      REAL*8  A(NMAX,WMAX)
      N=N
      DO J=1,WMAX
      DO I=1,N
       A(I,J)=0D0
       ICOL(I,J)=I
      ENDDO
      ENDDO
C 3D PROBLEM
      NX=NORD
      NY=NORD
C
      DO I=1,N
```

```
       IF((I/NX)*NX.NE.I.AND.I.LE.N-1) THEN
        A(I,1)=-1.0D0/6.0D0
        ICOL(I,1)=I+1
       ENDIF
      ENDDO
C
      DO I=1,N
       IZ=(I-1)/(NX*NY)
       IY=(I-1-IZ*NX*NY)/NX
       IF(I.LE.N-NX.AND.IY.NE.NY-1) THEN
        A(I,2)=-1.0D0/6.0D0
        ICOL(I,2)=I+NX
       ENDIF
      ENDDO
C
      DO I=1,N
       IF(I.LE.N-NX*NY) THEN
        A(I,3)=-1.0D0/6.0D0
        ICOL(I,3)=I+NX*NY
       ENDIF
      ENDDO
C
      DO I=1,N
       IF(((I-1)/NX)*NX.NE.I-1.AND.I.GE.2.AND.I.LE.N) THEN
        A(I,4)=-1.0D0/6.0D0
        ICOL(I,4)=I-1
       ENDIF
      ENDDO
C
      DO I=1,N
       IZ=(I-1)/(NX*NY)
       IY=(I-1-IZ*NX*NY)/NX
       IF(I.GE.NX+1.AND.I.LE.N.AND.IY.NE.0) THEN
        A(I,5)=-1.0D0/6.0D0
        ICOL(I,5)=I-NX
       ENDIF
      ENDDO
C
      DO I=1,N
       IF(I.GE.NX*NY+1.AND.I.LE.N) THEN
        A(I,6)=-1.0D0/6.0D0
        ICOL(I,6)=I-NX*NY
       ENDIF
      ENDDO
      RETURN
      END
```

(4) Method

The algorithm of the standard conjugate gradient method is used. (See [30] in Appendix A, “References.”) To precondition using the incomplete Cholesky method, see [58] in Appendix A, “References.” For vectorization by wavefront ordering, see [45] in Appendix A, “References.”

(5) Acknowledgement

Fujitsu is grateful to the authors of ITPACK and NSPCG who permitted the use of the modified incomplete Cholesky decomposition and wavefront ordering routines.

DM_VCLU

LU decomposition of complex matrices (blocked LU decomposition method)
CALL DM_VCLU(ZA,K,N,EPSZ,IP,IS,ICON)

(1) Function

This subroutine executes LU decomposition for non-singular complex $n \times n$ matrices using blocked outer product type Gaussian elimination.

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

where, ***P*** is the permutation matrix which exchanges rows by partial pivoting, ***L*** is the lower triangular matrix, and ***U*** is unit upper triangular matrix ($n \geq 1$).

(2) Parameters

ZA ..... Input. Store matrix ***A*** in ZA(1:N,1:N).

Output. Matrices ***L*** and ***U*** are stored in ZA(1:N,1:N).

See Figure DM_VCLU-1.

This is a double precision complex two-dimensional array ZA(K,N).

K..... Input. The size of the first array dimension for storage ZA (≥ N).

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

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed. (See note 1) in (3), “Comments on use.”)

IP ..... Output. The transposition vector indicating the history of row exchange by partial pivoting. One-dimensional array of size *n*. (See note 2) in (3), “Comments on use.”)

IS ..... Output. Information to obtain the determinant of matrix ***A***. The determinant is obtained by multiplying the *n* diagonal elements of array ZA by the value of IS after the operation.

ICON ..... Output. Condition code.

See Table DM_VCLU-1.

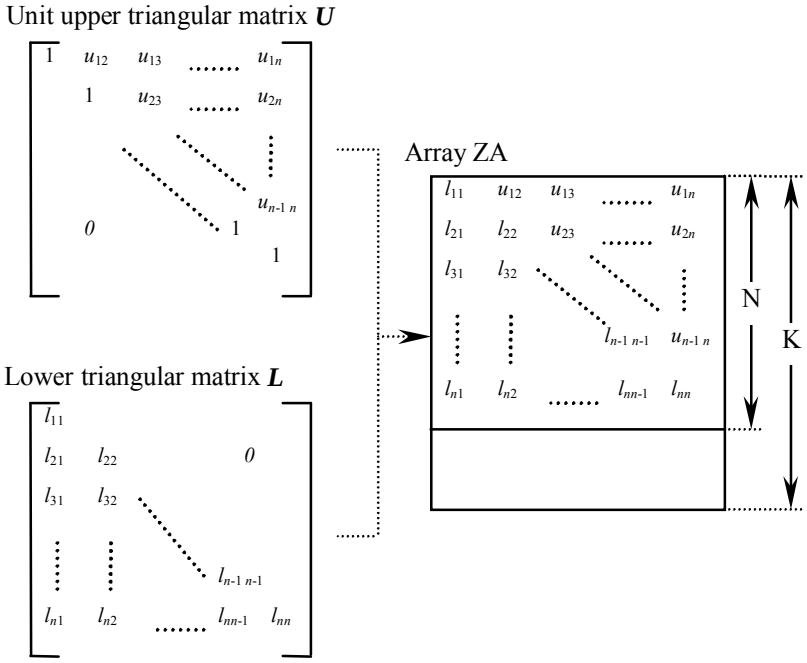


Figure DM_VCLU-1 Storing *L* and *U* in array ZA after the operation

After LU decomposition of matrices ***L*** and ***U***, the upper triangular part (except diagonal elements) of matrix ***U*** and ***L*** are stored in array ZA(I:N,I:N).

Table DM_VCLU-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All elements in some row of array ***A*** were zero, or the pivot became relatively zero. Matrix ***A*** may be singular.	Processing is discontinued.
30000	K < N, N < 1, or EPSZ < 0.0.	

(3) Comments on use

a. Notes

1) If a value is set for EPSZ, the value has the following meaning: if the absolute value of the selected pivot is less than EPSZ, the pivot is assumed to be zero and processing is discontinued when ICON = 20000. When unit round off is u, the standard value of EPSZ is $16u$. When the computation is to be continued even if the pivot becomes small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) The transposition vector corresponds to the permutation matrix ***P*** in LU decomposition ***PA*** = ***LU*** with partial pivoting.

In this subroutine, the contents of array ZA are exchanged using partial pivoting. That is, when the I-th row (I ≥ J) is selected as the pivot row in the J-th stage (J =

I, ..., *n*) of decomposition, the contents of the I-th row and J-th row of array ZA are exchanged. To indicate this exchange, I is stored in IP (J).

3) The linear equation can be solved by calling subroutine DM_VCLUX following this subroutine. Normally, the linear equation can be solved in one step by calling subroutine DM_VLCX.

b. Example

A system of linear equations with a complex coefficient matrix is LU-decomposed and solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N=2000,K=N+1)
C
      COMPLEX*16 A(K,N),B(N)
      REAL*8     C
      INTEGER    IP(N),IS
C
      C=SQRT(1.0D0/DBLE(1+N))
      T=DATAN(1.0D0)*8./(1+N)
C
      DO 100 J=1,N
      DO 100 I=1,N
      A(I,J)=DCMPLX(C*COS(T*I*J),C*SIN(T*I*J))
 100  CONTINUE
C
      DO 200 I=1,N
      S=(0.,0.)
      DO 200 J=1,N
      S=S+DCMPLX(COS(T*I*J),SIN(T*I*J))
      B(I)=S*C
 200  CONTINUE
C
      EPSZ=0.0D0
      CALL DM_VCLU(A,K,N,EPSZ,IP,IS,ICON)
C
      CALL DM_VCLUX(B,A,K,N,IP,ICON)
      PRINT*,'ICON=',ICON

      ERROR=0.0D0
      DO I=1,N
      ERROR=MAX(ERROR,ABS(1.0D0-B(I)))
      ENDDO
      PRINT*,'ERROR =',ERROR

      PRINT*,'ORDER=',N,' B(1)=',B(1),'B(N)=',B(N)
      STOP
      END
```

(4) Method

For details of the blocked LU decomposition method for outer product type, see [1], [30], [54], [55], [56], and [70] in Appendix A, “References.”

DM_VCLUX

A system of linear equations with LU-decomposed complex matrix
CALL DM_VCLUX(ZB,ZFA,KFA,N,IP,ICON)

(1) Function

This subroutine solves a linear equation with an LU-decomposed complex coefficient matrices.

$$\boldsymbol{LUx} = \boldsymbol{Pb}$$

where, $\boldsymbol{L}$ is a lower triangular matrix of $n \times n$, $\boldsymbol{U}$ is a unit upper triangular matrix of $n \times n$, and $\boldsymbol{P}$ is a permutation matrix. (Rows are exchanged by partial pivoting when the coefficient matrix is LU-decomposed.) $\boldsymbol{b}$ is an n-dimensional complex constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector ($n \geq 1$).

(2) Parameters

ZB ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

A double precision complex one-dimensional array of size n.

ZFA ..... Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored in ZFA(1:N,1:N).

See Figure DM_VCLUX-1.

This is a double precision complex two-dimensional array ZFA(KFA,N).

KFA ..... Input. The size of the first dimension of storage array ZFA (≥ N).

N ..... Input. Order n of matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.

IP ..... Input. The transposition vector which indicates the history of row exchange by partial pivoting. A one-dimensional array of size n.

(See note 2) in (3), "Comments on use," for subroutine DM_VCLU.)

ICON ..... Output. Condition code.

See Table DM_VCLUX-1.

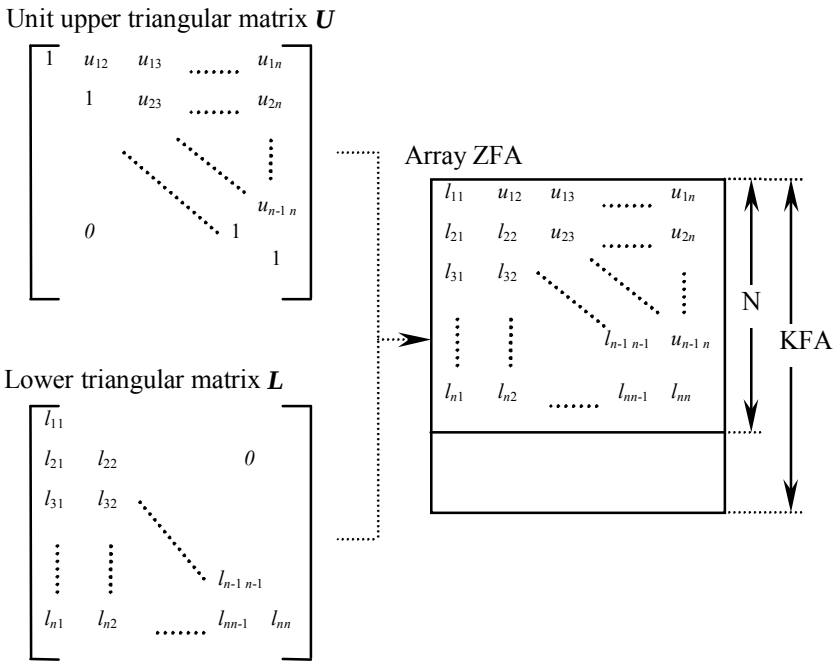


Figure DM_VCLUX-1 Storing *L* and *U* in array ZFA

For LU-decomposed matrices ***L*** and ***U***, ***L*** and the upper triangular part (except diagonal elements) of ***U*** are stored in array ZFA(1:N,1:N).

Table DM_VCLUX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The coefficient matrix was singular.	Processing is discontinued.
30000	KFA <N, N < 1, or IP was invalid.	

(3) Comments on use

a. Notes

1) The linear equations can be solved by calling subroutine DM_VCLU, LU-decomposing the coefficient matrix, then calling this subroutine. Normally, the solution can be obtained in one step by calling subroutine DM_VLCX.

b. Example

A system of linear equations with a complex coefficient matrix is LU-decomposed and solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C       **EXAMPLE**
```

```
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N=2000,K=N+1)
C
      COMPLEX*16 A(K,N),B(N)
      REAL*8     C
      INTEGER    IP(N),IS
C
      C=SQRT(1.0D0/DBLE(1+N))
      T=DATAN(1.0D0)*8./(1+N)
C
      DO 100 J=1,N
      DO 100 I=1,N
      A(I,J)=DCMPLX(C*COS(T*I*J),C*SIN(T*I*J))
 100  CONTINUE
C
      DO 200 I=1,N
      S=(0.,0.)
      DO 200 J=1,N
      S=S+DCMPLX(COS(T*I*J),SIN(T*I*J))
      B(I)=S*C
 200  CONTINUE
C
      EPSZ=0.0D0
      CALL DM_VCLU(A,K,N,EPSZ,IP,IS,ICON)
C
      CALL DM_VCLUX(B,A,K,N,IP,ICON)
      PRINT*,'ICON=',ICON

      ERROR=0.0D0
      DO I=1,N
      ERROR=MAX(ERROR,ABS(1.0D0-B(I)))
      ENDDO
      PRINT*,'ERROR =',ERROR

      PRINT*,'ORDER=',N,' B(1)=',B(1),'B(N)=',B(N)
      STOP
      END
```

(4) Method

The linear equation with LU-decomposed complex matrix as its coefficient is solved by forward and back-substitution. (See [54] in Appendix A, “References.”)

DM_VCMINV

Inverse of complex matrix (blocked Gauss-Jordan method)
CALL DM_VCMINV(ZA,K,N,EPSZ,ICON)

(1) Function

This subroutine obtains the inverse $\boldsymbol{A}^{-1}$ of the $n \times n$ non-singular complex matrix $\boldsymbol{A}$ using the Gauss-Jordan method.

(2) Parameters

ZA ..... Input. Matrix $\boldsymbol{A}$ is stored in ZA(1:N,1:N).

Output. Matrix $\boldsymbol{A}^{-1}$ is stored in ZA(1:N,1:N).

The double precision complex two-dimensional array ZA(K,N).

K ..... Input. The size of the first dimension of the array ZA. (≥ N)

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

EPSZ ..... Input. Judgment of relative zero of the pivot. (≥ 0.0)

When EPSZ is 0.0, the standard value is assumed.

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

ICON ..... Output. Condition code.

See Table DM_VCMINV-1.

Table DM_VCMINV-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All row elements in matrix $\boldsymbol{A}$ are zero or the pivot becomes a relatively zero. Matrix $\boldsymbol{A}$ may be singular.	Processing is discontinued.
30000	N < 1, K < N, or EPSZ < 0.0.	

(3) Comments on use

a. Notes

1) When the pivot element selected by partial pivoting is 0.0 or the absolute value is less than EPSZ, it is assumed to be relatively zero. In this case, processing is discontinued with ICON=20000. When unit round off is u, the standard value of EPSZ is $16u$. If the minimum value is assigned to EPSZ, processing is continued, but the result is not assured.

b. Example

The inverse of a matrix is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4

when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
cc    **example**
      implicit complex*16 (a-h,o-z)
      parameter(n=2000,k=n+1)
c
      complex*16    a(k,n)
      complex*16    as(k,n),tmpz
      real*8    c,t,tmp2,tmp
c
      c=sqrt(1.0d0/dble(n))
      t=datan(1.0d0)*8.d0/(n)
c
      do 100 j=1,n
      do 100 i=1,n
      a(i,j)=dcmplx(c*cos(t*(i-1)*(j-1)),
     $              c*sin(t*(i-1)*(j-1)))
      as(i,j)=dcmplx(c*cos(t*(i-1)*(j-1)),
     $              -c*sin(t*(i-1)*(j-1)))
 100  continue
c
      epsz=0.0d0
      call  dm_vcminv(a,k,n,epsz,icon)
cc
      tmp=0.0d0
      do j=1,n
      do i=1,n
      tmpz=(a(i,j)-as(i,j))
      tmp2=dabs(dble(tmpz))+dabs(dimag(tmpz))
      if(tmp2.gt.tmp)tmp=tmp2
      enddo
      enddo
      print*,'order=',n,' , error = ',tmp
  99  continue
      stop
      end
```

(4) Method

This subroutine solves an inverse of matrix using the blocked Gauss-Jordan method (see [30] in Appendix A, “References.”).

DM_VGEVPH

Generalized eigenvalue problem for real symmetric matrices (eigenvalues and eigenvectors) (Tridiagonalization, multisection method, and inverse iteration)
CALL DM_VGEVPH (A, K, N, B, EPSZ, NF, NL, IVEC, ETOL, CTOL, NEV, E, MAXNE, M, EV, ICON)

(1) Function

This subroutine obtains all the eigenvalues and eigenvectors to solve a generalized eigenvalue problem.

$$\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$$

where, $\boldsymbol{A}$ is an $n \times n$ real symmetric matrix and $\boldsymbol{B}$ is an $n \times n$ positive definite matrix.

(2) Parameters

A ..... Input. The lower triangular part $\{a_{ij} \mid i \geq j\}$ of real symmetric matrix $\boldsymbol{A}$ is stored in the lower triangular part $\{A(i,j) \mid i \geq j\}$ of A(1:N,1:N).

After calculation, the value of A is not assured.

Two-dimensional double-precision real array A(K,N).

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

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

B ..... Input. The lower triangular part $\{b_{ij} \mid i \geq j\}$ of the positive definite symmetric matrix $\boldsymbol{B}$ is stored in the lower triangular part $\{B(i,j) \mid i \geq j\}$ of B(1:N,1:N).

Output. The LL^T-decomposed matrix is stored.
The lower triangular matrix $\boldsymbol{L}$ $\{ l_{ij} \mid i \geq j\}$ is stored in the lower triangular part $\{ B(i,j) \mid i \geq j \}$ of B(1:N, 1:N).

This is a two-dimensional double-precision real array of B(K,N).

EPSZ ..... Input. The zero judgment value of the pivot when $\boldsymbol{B}$ is LL^T-decomposed. (≥ 0.0)

When EPSZ is 0.0, the standard value is assumed.

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

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.

When the value of IVEC is 1, the eigenvalues and corresponding eigenvectors are calculated.

When the value of IVEC is not 1, only the eigenvalues are calculated.

ETOL ..... Input. Criterion value for checking whether the eigenvalues are numerically different from each other or are multiple. When ETOL is less than 3.0D-16, this value is used as the standard value.

CTOL ..... Input. Criterion value for checking whether the adjacent eigenvalues can be considered to be approximately equal to each other. This check uses formula (3.1). This value is used to assure the linear independence of the eigenvector corresponding to the eigenvalue belonging to approximately multiple eigenvalues (clusters).

The value of CTOL should be generally 5.0D-12. For a very large cluster, a large CTOL value is required.

1.0D-6 ≥ CTOL ≥ ETOL

When condition CTOL > 1.0D-6 occurs, CTOL is set to 1.0D-6.

When condition CTOL < ETOL occurs, CTOL = 10 × ETOL is set as the standard value.

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

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

One-dimensional array NEV(5).

The detail information is as follows:

NEV(1) indicates the number of different eigenvalues calculated.

NEV(2) indicates the number of approximately multiple, different eigenvalues (clusters) calculated.

NEV(3) indicates the total number of eigenvalues (including multiple eigenvalues) calculated.

NEV(4) indicates the number representing the first of the eigenvalues calculated.

NEV(5) indicates the number representing the last of the eigenvalues calculated.

E ..... Output. Eigenvalues are stored in E.

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

One-dimensional array E(MAXNE).

MAXNE .... Input. Maximum number of eigenvalues that can be calculated.

When it can be considered that there are two or more eigenvalues with multiplicity m, MAXNE must be set to a larger value than NL − NF + 1 + 2 × m that is bounded by n. Size of the dimension of array E.

When condition NEV(3) > MAXNE occurs, the eigenvectors cannot be calculated.

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

M ..... Output. Information about the multiplicity of eigenvalues calculated.

M(i,l) indicates the multiplicity of the i-th eigenvalue λ_i. M(i,2) indicates the multiplicity of the i-th cluster when the adjacent eigenvalues are regarded as clusters.

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

Two-dimensional array M(MAXNE,2).

EV ..... Output. When IVEC = 1, the eigenvectors corresponding to the eigenvalues are stored in EV.

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

Two-dimensional array EV(K,MAXNE).

ICON ..... Output. Condition code.

See Table DM_VGEVPH-1.

Table DM_VGEVPH-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The pivot becomes negative at LL^T decomposition of matrix ***B***. Matrix ***B*** is not positive.	Processing is discontinued.
20100	The pivot becomes relatively zero at LL^T decomposition of matrix ***B***. Matrix ***B*** may be singular.	
20200	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. A suitable value for MAXNE to allow calculation to proceed is returned in NEV(3). (See 2) in a, "Notes," in (3), "Comments on use.")
30000	NF < 1, NL > N, NL < NF, N < 1, K < N, MAXNE < NL - NF + 1, or EPSZ < 0.	Processing is discontinued.

(3) Comments on use

a. Notes

1) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LL^T decomposition. In this case, processing is discontinued with ICON=20100. When unit round off is u, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the pivot is small, assign, the minimum value to EPSZ. In this case, however, the result is not assured.

2) This routine calculates eigenvalues independently from each other by dividing them into nonoverlapping, sequenced sets (parallel processing).

When ε = ETOL, the following condition is satisfied for consecutive eigenvalues λ_j ($j = s - 1, s, \ldots, s + k, (k \geq 0)$):

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

If formula (3.1) is satisfied for i when $i = s, s + 1, ..., s + k$ but not satisfied when $i = s - 1$ and $i = s + k + 1$, it is assumed that the eigenvalues λ_j ($j = s - 1, s, ..., s + k$) are numerically multiple.

The standard value of ETOL is 3.0D-16 (about the unit round off). With this value, the eigenvalues are refined up to the maximum machine precision.

If formula (3.1) is not satisfied when ε=ETOL, it can be considered that λ_{i-1} and λ_i are distinct eigenvalues.

When ε=ETOL, assume that consecutive eigenvalues λ_m ($m = t - 1, t, ..., t + k$ (k $\geq$ 0)) are different eigenvalues. Also, when ε = CTOL, assume that formula (3.1) is satisfied for i when $i = t, t + 1, ..., t + k$ but not satisfied when $i = t - 1$ and $i = t + k + 1$. In this case, it is assumed that their different eigenvalues λ_m ($m = t - 1, t, ..., t + k$) are approximately multiple (i.e., form a cluster). In this case, independent starting vectors are generated for inverse iteration, and eigenvectors corresponding to λ_m ($m = t - 1, t, \ldots, t + k$) are reorthogonalized.

3) The maximum number of eigenvalues that can be calculated is specified in MAXNE. When the value of CTOL is increased, the cluster size also increases. Therefore, the total number of eigenvalues calculated might exceed the value of MAXNE. In this case, decrease the value of CTOL or increase the value of MAXNE.

   If the total number of eigenvalues calculated exceeds the value of MAXNE, ICON = 20200 is returned. In this case, the eigenvectors cannot be calculated even if eigenvector calculation is specified. Eigenvalues are calculated, but are not stored repeatedly according to the multiplicity.

   The calculated different eigenvalues are stored in E(1:NEV(1)). The information about the multiplicity of the corresponding eigenvalues is stored in M(1:NEV(1),1).

   When all the eigenvalues are different from each other and there are no approximately multiple eigenvalues, MAXNE can be set to NT (NT=NL-NF+1). However, when there are multiple eigenvalues and the multiplicity can be assumed to be m, then MAXNE must be set to at least NT + 2 × m.

   If the total number of eigenvalues to be calculated exceeds the value of MAXNE, the value required to continue the calculation is returned to NEV(3). The calculation can be continued by allocating the area by using this returned value and by calling the routine again.

b. Example

   This example calculates the specified eigenvalues and eigenvectors of a generalized eigenvalue problem whose eigenvalues and eigenvectors are known.

   The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
cc    **example**
      implicit real*8(a-h,o-z)
      parameter(n=2000 ,k=n+1)
      parameter(nf=1,nl=n,max_nev=nl-nf+1,tau=1.0d0)
      dimension a(k,n),b(k,n),b2(k,n),c(k,n),d(k,n)
      dimension nev(5),mult(max_nev,2)
```

```
      dimension eval(max_nev),evec(k,max_nev)
cc
      pai=4.0d0*datan(1.0d0)
      coef=dsqrt(2.0d0/(n+1))
      do j=1,n
      do i=1,n
      d(i,j)=coef*dsin(pai/(n+1)*i*j)
      enddo
      enddo
cc
      do j=1,n
      do i=1,n
      if(i.eq.j)then
      c(J,J)=DBLE(J)
      else
      c(i,j)=0.0d0
      endif
      enddo
      enddo
cc
cc    d x c -> b
cc
      call dm_vmggm(d,k,c,k,b,k,n,n,n,icon)
cc
cc    b x d -> a
cc
      call dm_vmggm(b,k,d,k,a,k,n,n,n,icon)
cc
cc       B = LL^t , A <- LALt
cc
      do i=1,n
      do j=1,n
      b(j,i)=1.0d0/dsqrt(tau)
      b2(j,i)=min(i,j)/tau
      enddo
      enddo
      call dtrmm('Left','Lower','Not transpose','Not-unit',
     $           n,n,1.0d0,b,k,a,k)
      call dtrmm('Right','Upper','Not transpose','Not-unit',
     $           n,n,1.0d0,b,k,a,k)
cc
      n0x=nf
      n1x=nl
      ivec=1
      etol=1.0d-15
      ctol=1.0d-10
      max_nevx=max_nev
      epsz=0.0d0
      call dm_vgevph( a,k,n,b2,epsz,n0x,n1x,ivec,
     &                     etol,ctol,nev,
     &                     eval,max_nevx,mult,evec,icon )
      do i=1,nev(3),nev(3)/10
      print*,'eigen value in eval(',i,') = ',eval(i)
```

```
      enddo
      stop
      end
```

(4) Method

The generalized eigenvalue problem (4.1) is solved. Here, $\boldsymbol{B}$ is a positive definite matrix so that Cholesky decomposition can be executed.

$$\boldsymbol{Ax} = \lambda\boldsymbol{Bx} \tag{4.1}$$

$$\boldsymbol{LL}^{\mathrm{T}} = \boldsymbol{B} \tag{4.2}$$

Multiply (4.1) by $\boldsymbol{L}^{-1}$ from left-side.

$$\boldsymbol{L}^{-1}\boldsymbol{Ax} = \lambda\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} \tag{4.3}$$

$$\boldsymbol{y} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{x} \tag{4.4}$$

Then

$$\boldsymbol{x} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{y} \tag{4.5}$$

Substitute (4.5) into (4.3).

$$\boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}}\boldsymbol{y} = \lambda\boldsymbol{y} \tag{4.6}$$

$$\boldsymbol{C} = \boldsymbol{L}^{-1}\boldsymbol{AL}^{-\mathrm{T}} \tag{4.7}$$

Substituting $\boldsymbol{C}$ we get

$$\boldsymbol{Cy} = \boldsymbol{y} \tag{4.8}$$

(4.8) can be regarded as an eigenvalue problem for a real symmetric matrix. The eigenvalue problem of the real symmetric matrix is solved using DM_VSEVPH. (See the description on (4) Method of DM_VSEVPH.)

DM_VHEVP

Eigenvalues and eigenvectors of Hermite matrices
CALL DM_VHEVP (ZA, K, N, NF, NL, IVEC, ETOL, CTOL, NEV, EH, MAXNE, M, ZEV, ICON)

(1) Function

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

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \tag{1.1}$$

(2) Parameters

ZA ..... Input. The lower triangular part $\{a_{ij} \mid i \geq j\}$ of Hermite matrix $\boldsymbol{A}$ whose eigenvalues and eigenvectors are to be calculated is stored in the lower triangular part $\{ZA(i,j) \mid i \geq j\}$ of ZA(1:N,1:N). The value of ZA is not assured after operation.

Two-dimensional double-precision real array ZA(K,N).

K ..... Input. Size of first-dimension of array ZA (K ≥ N).

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

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.

When IVEC is 1, the eigenvalues and the corresponding eigenvectors are calculated.

When IVEC is not 1, only the eigenvalues are calculated.

ETOL ..... Input. Criterion value for checking whether the eigenvalues are different from each other or equal to each other. This check uses formula (3.1). When ETOL is less than 3.0D-16, this value is used as the standard value.

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

CTOL ..... Input. Criterion value for checking whether the adjacent eigenvalues are approximately equal to each other. This check uses formula (3.1). CTOL is used to assure the linear independence of the eigenvector corresponding to the eigenvalue belonging to approximately multiple eigenvalues (clusters).

The CTOL value should generally be 5.0D-12. For a very large cluster, a large CTOL value is required.

1.0D-6 ≥ CTOL ≥ ETOL

When condition CTOL > 1.0D-6 occurs, CTOL is set to 1.0D-6.

When condition CTOL < ETOL occurs, CTOL = 10 × ETOL is set as the standard value.

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

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

One-dimensional array NEV(5).

Details are given below.

NEV(1) indicates the number of different eigenvalues calculated.

NEV(2) indicates the number of approximately multiple different eigenvalues (different clusters) calculated.

NEV(3) indicates the total number of eigenvalues (including multiple eigenvalues) calculated.

NEV(4) indicates the number representing the first of the eigenvalues calculated.

NEV(5) indicates the number representing the last of the eigenvalues calculated.

EH ..... Output. Eigenvalues are stored in EH.

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

One-dimensional double-precision array EH(MAXNE).

MAXNE .... Input. Maximum number of eigenvalues that can be calculated. Size of the first-dimension of array EH.

When it can be assumed that there are two or more eigenvalues with multiplicity m, MAXNE must be a larger value than $NL - NF + 1 + 2 \times m$ that is bounded by n.

When condition NEV(3) > MAXNE occurs, the eigenvectors cannot be calculated. (See 2) in a, "Notes," in (3), "Comments on use.")

M ..... Output. Information about the multiplicity of eigenvalues calculated.

M(i,l) indicates the multiplicity of the i-th eigenvalue λ_i calculated. M(i,2) indicates the multiplicity of the i-th cluster calculated when the adjacent eigenvalues are regarded as approximately multiple eigenvalues (clusters).

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

Two-dimensional array M(MAXNE,2).

ZEV ..... Output. When IVEC = 1, the eigenvectors corresponding to the eigenvalues are stored in ZEV.

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

Two-dimensional double-precision array ZEV(K,MAXNE).

ICON ..... Output. Condition code.

See Table DM_VHEVP-1.

Table DM_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. A suitable value for MAXNE to allow calculation to proceed is returned in NEV(3). (See 2) in a, "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.

(3) Comments on use

a. Notes

1) This routine calculates eigenvalues independently from each other by dividing them into nonoverlapping, sequenced sets (parallel processing).

When ε = ETOL, the following condition is satisfied for consecutive eigenvalues λ_j ($j = s - 1, s, ..., s + k$, ($k \geq 0$)):

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

If formula (3.1) is satisfied for i when $i = s, s + 1, ..., s + k$ but not satisfied when $i = s - 1$ and $i = s + k + 1$, it is assumed that the eigenvalues λ_j ($j = s - 1, s, ..., s + k$) are numerically multiple.

The standard value of ETOL is 3.0D-16 (about the unit round off). In this case, the eigenvalues are refined up to the maximum machine precision.

If formula (3.1) is not satisfied when ε = ETOL, it can be considered that λ_{i-1} and λ_i are distinct eigenvalues.

When ε = ETOL, assume that consecutive eigenvalues λ_m ($m = t - 1, t, ..., t + k$ ($k \geq 0$)) are different eigenvalues. Also, when ε = CTOL, assume that formula (3.1) is satisfied for i when $i = t, t + 1, ..., t + k$ but not satisfied when $i = t - 1$ and $i = t + k + 1$. In this case, it is assumed that the distinct eigenvalues λ_m ($m = t - 1, t, ..., t + k$) are approximately multiple (i.e., form a cluster). In this case, independent starting vectors are generated for inverse iteration, and eigenvectors corresponding to λ_m ($m = t - 1, t, \ldots, t + k$) are reorthogonalized.

2) The maximum number of eigenvalues calculated can be specified in MAXNE. When the CTOL value is increased, the cluster size also increases. Therefore, the total number of eigenvalues calculated might exceed the MAXNE value. In this case, decrease the CTOL value or increase the MAXNE value.

If the total number of eigenvalues calculated exceeds the MAXNE value, ICON = 20000 is returned. In this case, the eigenvectors cannot be calculated even if eigenvector calculation is specified. Eigenvalues are calculated, but are not stored repeatedly according to the multiplicity.

The calculated different eigenvalues are stored in EH(1:NEV(1)). The multiplicity of the corresponding eigenvalues is stored in M(1:NEV(1),1).

When all the eigenvalues are different from each other and there are no approximately multiple eigenvalues, the MAXNE value can be NT (NT=NL-NF+1 is the total number of eigenvalues calculated). However, when there are multiple eigenvalues and the multiplicity is m, the MAXNE value must be at least NT + 2 × m.

If the total number of eigenvalues to be calculated exceeds the MAXNE value, the value required to continue the calculation is returned to NEV(3). The calculation can be continued by allocating the area by using this returned value and by calling the routine again.

b. Example

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

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N=2000,K=N,NE=N,MAX_NEV=NE)
      COMPLEX*16 A(K,N),B(K,N),C(K,N),D(K,N),DH(K,N),ALPHA,BETA,
     &           EVECH(K,MAX_NEV)
      DIMENSION  NEV(5),MULT(MAX_NEV,2)
      DIMENSION  EVAL(MAX_NEV)
CC
      PAI2=8.0D0*DATAN(1.0D0)
      COEF=DSQRT(1.0D0/(N))
      DO J=1,N
      DO I=1,N
      PART1 =COEF*DCOS(PAI2/N*(I-1)*(J-1))
      PART2 =COEF*DSIN(PAI2/N*(I-1)*(J-1))
      D(I,J)=DCMPLX(PART1,PART2)
      DH(I,J)=DCMPLX(PART1,-PART2)
      ENDDO
      ENDDO
CC
      DO J=1,N
      DO I=1,N
      IF(I.EQ.J)THEN
      C(I,J)=DCMPLX(DBLE(I),0.0D0)
      ELSE
      C(I,J)=(0.0D0,0.0D0)
      ENDIF
      ENDDO
      ENDDO
```

```
CC
CC    D X C -> B
CC

      ALPHA=(1.0D0,0.0D0)
      BETA=(0.0D0,0.0D0)
      CALL ZGEMM('NO TRANSPOSE','NO TRANSPOSE',N,N,N,ALPHA,
     $            D,K,C,K,BETA,B,K)
CC
CC    B X D^H -> A
CC
      CALL ZGEMM('NO TRANSPOSE','NO TRANSPOSE',N,N, N,ALPHA,
     &            B,K,DH,K,BETA,A,K)
CC
      IVEC=1
      NF=1
      NL=NE
      EVAL_TOL=1.0D-15
      CLUS_TOL=1.0D-10
      CALL  DM_VHEVP( A,K,N,NF,NL,IVEC,EVAL_TOL,CLUS_TOL,NEV,
     &                EVAL,MAX_NEV,MULT,EVECH,ICON )
      DO I=1,NE,100
      PRINT*,'EIGEN VALUE IN EVEC(',I,') = ',EVAL(I)
      ENDDO
      STOP
      END
```

(4) Method

The $n \times n$ Hermite matrix A = AR + iAI must satisfy AR = AR^T and AI = $-AI^T$

The blocking Householder method is used to reduce the Hermite matrix to a Hermite tridiagonal matrix. Then, the diagonal unitary transformation is applied to further reduce the matrix to a real tridiagonal matrix.

The eigenvalues and eigenvectors of the tridiagonal matrix are calculated using techniques of multisectioning and inverse iteration (see “DM_VTDEVC” and [61] in Appendix A, “References”).

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

DM_VHTRID

Tridiagonalization of Hermite matrices
CALL DM_VHTRID (ZA, K, N, D, SL, ZS, ICON)

(1) Function

This subroutine reduces an Hermite matrix into an Hermite tridiagonal matrix and this matrix is transformed into a real tridiagonal matrix using diagonal unitary transform.

$$\boldsymbol{H} = \boldsymbol{P}^{*}\boldsymbol{A}\boldsymbol{P}$$

$$\boldsymbol{T} = \boldsymbol{V}^{*}\boldsymbol{H}\boldsymbol{V}$$

$\boldsymbol{A}$ is an $n \times n$ Hermite matrix, $\boldsymbol{P}$ is an $n \times n$ unitary matrix. $\boldsymbol{V}$ is an $n \times n$ diagonal unitary matrix and $\boldsymbol{T}$ is a real tridiagonal matrix..

(2) Parameters

ZA ..... Input. The lower triangular part $\{a_{ij} \mid i \geq j\}$ of Hermite matrix $\boldsymbol{A}$ is stored in the lower triangular part $\{ZA(i,j) \mid i \geq j\}$ of ZA(1:N,1:N).

Two-dimensional double-precision complex array ZA(K,N).

Output. The information on Householder transforms used for Hermite tridiagonalization is stored in the lower triangular part $\{ZA(i,j) \mid i \geq j\}$ of ZA(1:N,1:N). The values in the upper triangular part of ZA is not assured after operation.

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

K ..... Input. Size of first-dimension of array ZA (K ≥ N).

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

D ..... Input. The diagonal elements of the reduced tridiagonal matrix are stored in real double-precision one-dimensional array D(N).

SL ..... Input. The subdiagonal elements of reduced tridiagonal matrix are stored in SL(2:N) of real double-precision one-dimensional array SL(N). SL(1) = 0.

ZS ..... Output. Diagonal elements of the diagonal unitary matrix are stored in ZS(1:N).

One-dimensional double-precision complex array ZS(N).

ICON ..... Output. Condition code.

See Table DM_VHTRID-1.

Table DM_VHTRID-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	K < N, N < 2.	Processing is discontinued.

(3) Comments on use

a. Notes

1) Hermite tridiagonalization is performed by the repeated transforms varying k = 1, ... , n - 2.

$$\boldsymbol{A}^k = \boldsymbol{P}_k^* \boldsymbol{A}^{k-1} \boldsymbol{P}_k \ , \quad \boldsymbol{A}^0 = \boldsymbol{A}$$

Put $\boldsymbol{b}^{\mathrm{T}} = (0, \ldots , 0, \boldsymbol{A}^k(k{+}1{:}n, k)^{\mathrm{T}})$.

$\boldsymbol{b}^* \cdot \boldsymbol{b} = S^2$ and put $\boldsymbol{w}^{\mathrm{T}} = (0, \ldots , 0, b_{k+1}\left(1 + \dfrac{|S|}{|b_{k+1}|}\right), b_{k+2}, \ldots , b_n)$.

Then the transform matrix is represented as follows.

$$\boldsymbol{P}_k = \mathrm{I} - \alpha \boldsymbol{w} \cdot \boldsymbol{w}^* , \alpha = \frac{1}{S^2 + |b_{k+1} S|} .$$

$\boldsymbol{w}(k{+}1{:}n)$ and α are stored in $\boldsymbol{A}(k{+}1{:}n, k)$ and $\boldsymbol{A}(k, k)$ respectively.

b. Example

This example calculates the tridiagonalization of a Hermite matrix with the known eigenvalues.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
c     **example**
      implicit real*8(a-h,o-z)
      parameter(n=2000,k=n,ne=n,max_nev=ne)
      complex*16 a(k,n),b(k,n),c(k,n),d(k,n),
     &          dh(k,n),alpha,beta,
     &          tr(n)
      dimension nev(5),mult(max_nev,2)
      dimension eval(max_nev),evec(k,max_nev),dd(n),sld(n),sud(n)
cc
      pai2=8.0d0*datan(1.0d0)
      coef=dsqrt(1.0d0/(n))
      do j=1,n
      do i=1,n
      part1 =coef*dcos(pai2/n*(i-1)*(j-1))
      part2 =coef*dsin(pai2/n*(i-1)*(j-1))
      d(i,j)=dcmplx(part1,part2)
      dh(i,j)=dcmplx(part1,-part2)
      enddo
      enddo
cc
      do j=1,n
      do i=1,n
      if(i.eq.j)then
      c(i,j)=dcmplx(dble(i),0.0d0)
      else
      c(i,j)=(0.0d0,0.0d0)
```

```
      endif
      enddo
      enddo
cc
cc    d x c -> b
cc

      alpha=(1.0d0,0.0d0)
      beta=(0.0d0,0.0d0)
      call zgemm('No transpose','No transpose',n,n,
     $           n,alpha,d,k,c,k,beta,b,k)
cc
cc    b x d^h -> a
cc
      call zgemm('No transpose','No transpose',n,n,
     $           n,alpha,b,k,dh,k,beta,a,k)
cc
      call  dm_vhtrid(a,k,n,dd,sld,tr,icon)
      if(icon.ne.0)then
      print*,' icon of dm_vhtrid =',icon
      stop
      endif
c
      do i=2,n
      sud(i-1)=sld(i)
      enddo
      sud(n)=0.0d0
c
      nf=1
      nl=n
      ivec=0
      eval_tol=1.0d-15
      clus_tol=1.0d-10
      call dm_vtdevc(dd,sld,sud,n,nf,nl,ivec,
     &               eval_tol,clus_tol,nev,
     &               eval,max_nev,evec,k,mult,icon )
      do i=1,ne,n/20
      print*,'eigen value in eval(',i,') = ',eval(i)
      enddo

      stop
      end
```

(4) Method

The $n \times n$ Hermite matrix A = AR + iAI must satisfy AR = AR^T and AI = $-AI^T$

The blocking Householder method is used to reduce the Hermite matrix to a Hermite tridiagonal matrix. Then, the diagonal unitary transformation is applied to further reduce the matrix to a real tridiagonal matrix.

DM_VJDHECR

Eigenvalues and eigenvectors of an Hermitian sparse matrix (Jacobi-Davidson method, compressed row storage method)
CALL DM_VJDHECR(ZH, NZ, NCOL, NFRNZ, N, ITRGT, DTRGT, NSEL, NEV, ITMAX, ITER, IFLAG, DPRM, DEVAL, ZEVEC, KV, DHIS, KH, ICON)

(1) Function

This subroutine computes a few of selected eigenvalues and corresponding eigenvectors of an Hermitian sparse eigenvalue problem

$\boldsymbol{Ax} = \lambda \boldsymbol{x}$

using the Jacobi-Davidson method, where $\boldsymbol{A}$ is an $n \times n$ Hermitian sparse matrix, the lower triangular part of which is stored using the compressed row storage method, and $\boldsymbol{x}$ is an n-dimensional vector.

(2) Parameters

ZH ..... Input. The non-zero elements of the lower triangular part of the sparse matrix $\boldsymbol{A}$ are stored.

One-dimensional complex array ZH(NZ).

For the compressed row storage method, refer to Figure DM_VJDHECR-1.

NZ..... Input. The total number of the nonzero elements which belong to the lower triangular part of the matrix $\boldsymbol{A}$.

NCOL..... Input. The column indices used in the compressed row storage method, which indicate the column number of each nonzero element stored in the array ZH.

One-dimensional array NCOL(NZ).

NFRNZ..... Input. The position of the first nonzero element of each row stored in the array ZH in the compressed row storage method which stores the lower part of the nonzero elements row by row. Specify NFRNZ(N+1)=NZ+1.

One-dimensional array NFRNZ(N+1).

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

ITRGT..... Input. Select a way of specifying the eigenvalues to be sought ($0 \leq$ ITRGT ≤ 4).

Specify ITRGT=0 to compute eigenvalues closest to a target value DTRGT.

Specify ITRGT=1 to compute eigenvalues with largest magnitude.

Specify ITRGT=2 to compute eigenvalues with smallest magnitude.

Specify ITRGT=3 to compute eigenvalues with largest real part.

Specify ITRGT=4 to compute eigenvalues with smallest real part.

(See note 1) and 2) in (3), "Comments on use.")

DTRGT..... Input. The target value τ is specified when ITRGT=0. In the following cases, the convergence might be improved by specifying a value near the seeking eigenvalue even when ITRGT$\neq$0.

1) The value τ is used as a shift of the test subspace <$\boldsymbol{W}$>=<($\boldsymbol{A}-\tau\boldsymbol{I}$)$\boldsymbol{V}$> when DPRM(3)=1 which indicates that the harmonic algorithm is to be used. (See note 2) in (3), "Comments on use.")

2) When DPRM(9)≥1, the value τ is used as an approximated eigenvalue in the Jacobi-Davidson correction equation while the initial phase of the iteration is proceeding. (See note 5) in (3), "Comments on use.")

3) When DPRM(15) ≥1, the value τ is used as a shift value of the preconditioner for the Jacobi-Davidson correction equation. (See note 7) in (3), "Comments on use.")

In other cases, DTRGT is not referred in this subroutine.

NSEL..... Input. The number of eigenvalues to be computed (1≤NSEL≤N). (See note 1) in (3), "Comments on use.")

NEV..... Output. The number of eigenvalues converged.

ITMAX..... Input. Upper limit of iterative count for the Jacobi-Davidson method (≥0).

ITER..... Output. Actual iterative count for the Jacobi-Davidson method.

IFLAG..... Input. Control information array specifying whether the auxiliary parameter is specified explicitly in DPRM array.

When IFLAG(i)≠0, the parameter specified in DPRM(i) is to be used.

When IFLAG(i)=0, a default parameter is used and DPRM(i) is not referred.

Set IFLAG(16:32) to be all zero since these area are preserved for future enhanced functionality.

One-dimensional array IFLAG(32).

DPRM..... Input. Auxiliary parameters are specified as for the IFLAG(i) denotes that the user specified value is to be used.

For definition of each parameter in the algorithm, see (4), "Method."

If all of IFLAG(1:32) are set to be zero, DPRM(1:32) are not referred and default parameters are used. Changing the parameter is recommended when the iteration did not converge with default parameters.

One-dimensional array DPRM(32).

DPRM(1): The dimension $m_{\min}$ of shrunk subspace when restarting (1≤ $m_{\min}$ <N). The default value is $m_{\min}$=50.

DPRM(2): Upper limit of the dimension $m_{\max}$ of subspace ($m_{\min} < m_{\max} \leq$N). The default value is $m_{\max}= m_{\min} +30$.
(See note 8) in (3), "Comments on use.")

DPRM(3): The type of the algorithm, which is associated with setting of a test subspace.
When DPRM(3)=0, the standard algorithm is adopted. The algorithm is appropriate for seeking the extreme eigenvalues in the spectrum.
When DPRM(3)=1, the harmonic algorithm is adopted. The algorithm is appropriate for seeking the internal eigenvalues in the spectrum.
The default value is the harmonic algorithm for ITRGT=0 or 2, or the standard algorithm in other cases.

DPRM(4): The criterion value for judgment of acceptable convergence. The default value is 10^{-6}. (See note 4) in (3), "Comments on use.")

DPRM(5): The way how to calculate the residual norm with respect to the approximated eigenvalue θ and eigenvector $\boldsymbol{u}$.
When DPRM(5)=0, the residual norm relative to the absolute value of approximated eigenvalue $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\theta|$ is adopted.
When DPRM(5)=1, the residual norm relative to the 1-norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_1$ is adopted.
When DPRM(5)=2, the residual norm relative to the Frobenius norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_F$ is adopted.
When DPRM(5)=3, the residual norm relative to the infinity-norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_\infty$ is adopted.
When DPRM(5)=4, the absolute residual norm $|\boldsymbol{Au}-\theta\boldsymbol{u}|$ is adopted.
The default is DPRM(5)=0. (See note 3) in (3), "Comments on use.")

DPRM(6): A criterion value for a delay-deflation scheme (≤1.0).
The default value is DPRM(6)=0.9. (See note 4) in (3), "Comments on use.")

DPRM(7): Control information indicating whether the iteration is started from a vector specified in the array ZEVEC(1:N,1).
When DPRM(7)=0, the iteration is started from a random vector generated in this subroutine internally.
When DPRM(7)=1, set an initial vector in the array ZEVEC(1:N,1).
The default setting is using a random vector.

DPRM(8): A seed to generate a random vector (≥1.0). The default value is 1.

DPRM(9): While the iteration count is less or equal to DPRM(9), the process is regarded as an initial phase of the iteration. Then the fixed value of τ is used as an approximated eigenvalue instead of the value of θ in the Jacobi-Davidson correction equation.
When DPRM(3)=0, the default value is DPRM(9)=0.
When DPRM(3)=1, the default value is DPRM(9)= $m_{\max}$.
(See note 5) in (3), "Comments on use.")

DPRM(10): The method to solve the Jacobi-Davidson correction equation.
When DPRM(10)=0, $\boldsymbol{t}=\boldsymbol{r}$ is set without using the correction equation.
When DPRM(10)=1, the GMRES method is adopted.
When DPRM(10)=2, the BiCGstab(L) method is adopted.
When DPRM(10)=11, the MINRES method is adopted.
The default is using the MINRES method. (See 7) and 8) in (3), "Comments on use.")

DPRM(11): A parameter for the solver of the correction equation.
When the BiCGstab(L) is used, specify the value of L (≤10). The default value is 4.

DPRM(12): Upper limit of the iteration count of the solver for the Jacobi-Davidson correction equation (≥1). The default value is 30.

DPRM(13): A parameter to determine the stopping criterion for the iterative solver of the correction equation (>0.0).
The default value is 0.7. (See 6) in (3), "Comments on use.")

DPRM(14): A parameter to determine the stopping criterion for the iterative solver of the correction equation (0.0<DPRM(14)≤1.0). The stopping criterion is set to DPRM(13)×DPRM(14)^l, where l is an iteration

counter of the outer loop which is reset in each deflation.
The default value is 0.7. (See 6) in (3), "Comments on use.")

DPRM(15): The type of preconditioning of the correction equation (≤1).
When DPRM(15)=0, no preconditioning is used.
When DPRM(15)=1, the diagonal left preconditioning is exploited.
(See 7) in (3), "Comments on use.")
The default is DPRM(15)=0.

DPRM(16:32): Preserved area for future enhanced functionality.

DEVAL..... Output. Detected eigenvalues are stored in DEVAL(1:NEV).

One-dimensional array DEVAL(NSEL).

ZEVEC..... Output. Detected eigenvectors are stored in ZEVEC(1:N,1:NEV).

Two-dimensional complex array ZEVEC(KV,NSEL).

Input. Set the initial vector in ZEVEC(1:N,1) when IFLAG(7)≠0 and DPRM(7)=1.0.

KV..... Input. Size of the first dimension of array ZEVEC (≥N).

DHIS..... Output. The convergence history of the residuals of the eigenproblem are stored in DHIS(1:*min*(KH,ITER),1). The final relative residual norm of the each correction equation are stored in DHIS(1:*min*(KH,ITER),2).

Two-dimensional array DHIS(KH,2).

KH..... Input. Size of the first dimension of array DHIS (≥0). Setting KH=ITMAX is enough. If KH=0 is set, the outputs to the array DHIS are suppressed.

ICON..... Output. Condition code.

(See Table DM_VJDHECR-1.)

Table DM_VJDHECR-1 Condition codes

Code	Meaning	Processing
0	No error	–
1000	Breakdown occurred in the iterative linear equations solver.	Processing is continued with the approximated solution until the point.
2000	A null vector is detected in a sort of process of the orthogonalization.	Processing is continued with the subspace expanded by a random vector.
3000	A recovery procedure is activated in a sort of restorative process of the delay deflation.	Processing is continued
10000	The iteration count reached the maximum limit before NSEL-th eigenvalue is obtained.	The calculated eigenpairs up to NEV are correct.
20000	The projected dense eigenproblem can not be solved.	Processing is discontinued. The calculated eigenpairs up to NEV are correct if NEV>0.

Code	Meaning	Processing
21000	The iteration count reached the maximum limit without a single convergence.	Processing is discontinued. The approximate values obtained up to this point are output in array DEVAL(1) and ZEVEC(1:N,1), but their precision cannot be guaranteed.
29000	An internal error occurred.	Processing is discontinued.
30000	N<1, ITRGT<0, ITRGT>4, NSEL<1, NSEL>N, ITMAX<0, KV<N or KH<0.	
30001 to 30032	The value of IFLAG or DPRM is not correct.	
31000	The value of NZ, NCOL or NFRNZ is not correct.	

$$A = \begin{bmatrix} 1 & 2+4i & 0 & 0 \\ 2-4i & 5 & 7-3i & 6+9i \\ 0 & 7+3i & 8 & 0 \\ 0 & 6-9i & 0 & 10 \end{bmatrix}$$

$$\mathbf{NFRNZ} = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 6 \\ 8 \end{bmatrix}, \quad \mathbf{ZH} = \begin{bmatrix} 1 \\ 2-4i \\ 5 \\ 7+3i \\ 8 \\ 6-9i \\ 10 \end{bmatrix}, \quad \mathbf{NCOL} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 2 \\ 4 \end{bmatrix}$$

Figure DM_VJDHECR-1 Storing a matrix A in compressed row storage method

(3) Comments on use

a. Notes

1) Robustness of the Jacobi-Davidson algorithm
The Jacobi-Davidson algorithm is not a decisive procedure, and hence is not as robust as the method for dense matrices based on the reduction of matrix elements. The results obtained using the Jacobi-Davidson method depends on choice of the initial vector, and the order of obtained eigenvalues are not guaranteed to be the

order of precedence user specified. This method is applicable when the seeking eigenvalues are only a few of the entire spectrum.

The convergence behavior of this routine is affected by various auxiliary parameters. For description of these parameters, refer to "Comments on use."

2) ITRGT and DTRGT parameter
The default value of DPRM(3), which specifies a type of algorithm, is switched automatically according to the setting of ITRGT, which specifies a way of selecting eigenvalues. However, an explicit specification of the value in DPRM(3) by setting IFLAG(3)≠0 is prior to the default value of course. Which means that the standard algorithm can be used with ITRGT=0 or 2, and that the harmonic algorithm can be used with ITRGT=1,3,4,5 or 6, as long as user knows its adaptivity.

Note that the DTRGT parameter is referred as a shift of the test subspace for the default harmonic algorithm when just setting ITRGT=2, which specifies to compute eigenvalues with smallest magnitude. Define the DTRGT to be 0.0D0 if other appropriate value is not known.

3) Calculating the residual norm
In the default setting, convergence of the eigenproblem is judged based on the residual norm relative to absolute value of the approximated eigenvalue. When the absolute value of the seeking eigenvalue is far smaller than the norm of the matrix, however, it is difficult to satisfy the convergence condition $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\theta|<$DPRM(4). In that case, adjust the convergence criterion DPRM(4), or change the way of calculating the residual norm which can be specified by DPRM(5) parameter.

4) Delay deflation procedure
This subroutine adopts an ingenious scheme to improve the precision of the results. After the residual becomes below the convergence criterion, this subroutine still continues some more iteration without deflation while the decrease ratio of the residual remains valid. This procedure is called *delay-deflation* here. The decrease ratio is regarded valid if the ratio of the residual norm relative to the preceding residual is less than the parameter DPRM(6). If the residual deteriorates while this extra iteration, the better previous variables are restored and the deflation with the vector takes place. With setting DPRM(6)=0.0, this delay-deflation does not act and then the parameter DPRM(4) is regarded as an ordinary convergence criterion.

5) Approximated eigenvalue in the correction equation
In the initial few steps of the process, the values of θ are usually poor approximations of the wanted eigenvalue. This subroutine takes the target value τ specified in the DTRGT as an approximated eigenvalue instead of θ in the initial phase, since the validity of the expansion vector $\boldsymbol{t}$ is affected by the closeness to the approximated eigenvalue in the Jacobi-Davidson correction equation. The process is regarded as the initial phase of the iteration while the iteration count is less than or equal to DPRM(9). However, the default value of this parameter is DPRM(9)=0 when DPRM(3)=0 is adopted, because it is difficult to determine a value of τ in advance when the standard algorithm is specified.

6) Stopping criterion for inner iteration
The Jacobi-Davidson correction equation is solved by some iterative method in this subroutine, thus the whole algorithm consists of two nested iterations. In the outer iteration the approximation for the eigenproblem is constructed , and in the inner iteration the correction equation is approximately solved. If the residual of the eigenproblem still not be small in the outer iteration, solving accurately the

correction equation in the inner iteration might be unnecessary. Therefore, the stopping criterion for the inner iteration can be varied according to a counter associated with the outer iteration. The criterion is set to be DPRM(13)×DPRM(14)^l, where l is the outer iteration counter which is reset to zero at each deflation. Incidentally, the upper limit count for the inner iteration is specified by DPRM(12).

7) Precondition for the correction equation
It is known that a good preconditioner improves the convergence of the iterative method for linear equations. The preconditioner to be applied is controlled by the parameter DPRM(15) in this subroutine. Note that the value of DTRGT is used for constructing a matrix M ≅ (A−τI), which approximates a part of the coefficient matrix in some way. The preconditioner is derived from the inverse procedure of the matrix M and projections on both sides. If the preconditioner does not approximate the coefficient matrix of the correction equation properly or the parameter DTRGT is far from the seeking eigenvalue, the convergence may deteriorate. Additionally, DPRM(10) must specify a kind of the iterative method that is applicable to nonsymmetric linear systems, because the coefficient matrix becomes nonsymmetric with a left preconditioner adopted in this routine.

8) Memory usage
This subroutine exploits work area internally as auto allocatable arrays. Therefore an abnormal termination could occur when the available area of the memory runs out. The necessary size for the outer iteration is at least n×(2×m_{max}+2×NSEL) ×16 bytes for the standard algorithm and n×(3×m_{max}+2×NSEL) ×16 bytes for the harmonic algorithm. And when the GMRES method is used as the solver of the correction equation, the additional necessary area is n×DPRM(12)×16 bytes for the inner iteration.

b. Example

Ten largest eigenvalues in magnitude and corresponding eigenvectors of an eigenproblem $\boldsymbol{Ax}=\lambda\,\boldsymbol{x}$ are sought, where $\boldsymbol{A}$ is a 10000×10000 example Hermitian matrix of the random sparsity pattern with about 20 nonzero entries in each row.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on a system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE
      INTEGER NNZMAX,NMAX,LDK,NZC
      PARAMETER (NMAX=10000,NZC=20)
      PARAMETER (NNZMAX=NMAX*NZC)
      PARAMETER (LDK=10)
      COMPLEX*16 ZH(NNZMAX),ZEVEC(NMAX,LDK)
      COMPLEX*16 RVEC(NMAX),ZW(NMAX)
      REAL*8 DTRGT,DEVAL(LDK),DERR,DPRM(32),DHIS(NMAX,2)
      INTEGER NZ,NCOL(NNZMAX),NFRNZ(NMAX+1),N,ITRGT
      INTEGER IFLAG(32),NSEL,NEV,ITMAX,ITER,LDX,LDH,ICON
      INTEGER I,J,K,NCOLJ

      N=NMAX
      CALL MKSPMAT(N,NZC,ZH,NCOL,NFRNZ)
      NZ=NFRNZ(N+1)-1
```

```
      ITMAX = 500
      NSEL = 10
      DO I = 1,32
       IFLAG(I)=0
      ENDDO
      LDX = NMAX
      LDH = NMAX
      DTRGT = 0.0D0
      ITRGT = 1

      CALL DM_VJDHECR(ZH,NZ,NCOL,NFRNZ,N,ITRGT,DTRGT,NSEL,
     &                NEV,ITMAX,ITER,IFLAG,DPRM,
     &                DEVAL,ZEVEC,LDX,DHIS,LDH,ICON)

      PRINT *,'DM_VJDHECR ICON=',ICON
      PRINT *,'ITER=',ITER
      DO K = 1,NEV
!$OMP PARALLEL PRIVATE(I,J,ZW,NCOLJ)
       ZW(1:N) = (0.0D0,0.0D0)
!$OMP DO
       DO I=1,N
        RVEC(I)=(0.0D0,0.0D0)
        DO J=NFRNZ(I),NFRNZ(I+1)-1
         NCOLJ=NCOL(J)
         RVEC(I)=RVEC(I)+ZH(J)*ZEVEC(NCOLJ,K)
         IF(I.NE.NCOLJ)THEN
          ZW(NCOLJ)=ZW(NCOLJ)+DCONJG(ZH(J))*ZEVEC(I,K)
         ENDIF
        ENDDO
       ENDDO
!$OMP CRITICAL
       DO I=1,N
        RVEC(I)=RVEC(I)+ZW(I)
       ENDDO
!$OMP END CRITICAL
!$OMP END PARALLEL
       DERR=0.0D0
       DO I=1,N
        RVEC(I)=RVEC(I)-DEVAL(K)*ZEVEC(I,K)
        DERR=DERR +DREAL(RVEC(I))**2 +DIMAG(RVEC(I))**2
       ENDDO
       DERR=DSQRT(DERR)
       PRINT*,'EIGEN VALUE',K,'=',DEVAL(K)
       PRINT*,'ERROR=',DERR/DABS(DEVAL(K))
      ENDDO
      END

      SUBROUTINE MKSPMAT(N,NZC,ZH,NCOL,NFRNZ)
      IMPLICIT NONE
      INTEGER N,NZC,NCOL(*),NFRNZ(*)
      COMPLEX*16 ZH(*)
      INTEGER I,IC,ICT,J,K,ISEED,LDW,ICON,NNZ
```

```
      PARAMETER(LDW=1350)
      REAL*8 DWORK(NZC),RNDWORK(LDW)
      ISEED=1
      NNZ=0
      DO I=1,N
       NFRNZ(I)=NNZ+1
10     CALL DVRAU4(ISEED,DWORK,NZC,RNDWORK,LDW,ICON)
       IC=0
       DO J=1,NZC
        ICT=N*DABS(DWORK(J))+1
        IF(ICT.LE.I)THEN
         DO K=1,IC
          IF(ICT.EQ.NCOL(NNZ-K+1))THEN
           NNZ=NNZ-IC
           GO TO 10
          ENDIF
         ENDDO
         NNZ=NNZ+1
         IC=IC+1
         NCOL(NNZ)=ICT
        ENDIF
       ENDDO
      ENDDO
      NFRNZ(N+1)=NNZ+1
      ISEED = 1
      CALL DVRAN4(0.0D0,1.0D0,ISEED,ZH,2*NNZ,RNDWORK,LDW,
     &            ICON)
      DO I=1,N
       DO J=NFRNZ(I),NFRNZ(I+1)-1
        IF(I.EQ.NCOL(J))ZH(J)=DREAL(ZH(J))+DIMAG(ZH(J))
       ENDDO
      ENDDO
      RETURN
      END
```

(4) Method

This subroutine solves large sparse eigenproblems using the Jacobi-Davidson method. In the Jacobi–Davidson approach a small *m*-dimensional search subspace is updated in each iteration, from which an approximated eigenvector of the given *n*-dimensional eigenproblem is sought. In each iteration there are two important phases of procedure, one is expansion in which the subspace is enlarged by adding a new appropriate basis vector to it, and one is extraction in which a sensible approximate eigenpair is sought from the search subspace. For the subspace expansion phase, a correction vector against an approximated eigenvector is calculated as a solution vector of the Jacobi-Davidson correction equation. For the extraction phase, an approximated eigenpair is calculated as a solution of a small projected eigenproblem. Even when the seeking eigenvalues are in the interior of the spectrum, this method extracts them ingeniously using distinct search and test subspaces.

The following shows the overall procedure of the algorithm and describes some auxiliary parameters used in this subroutine.

1. Prepare an expansion vector $\boldsymbol{t}$.
2. Expand the search subspace $<\boldsymbol{V}>$ and the test subspace $<\boldsymbol{W}>$ according to the vector $\boldsymbol{t}$, where $\boldsymbol{V}$ and $\boldsymbol{W}$ are $n \times m$ matrices, and $<\boldsymbol{V}>$ represents a linear subspace spanned by the column vectors of the matrix $\boldsymbol{V}$. The type of the algorithm whether standard or harmonic can be distinguished by the setting of the test subspace, $\boldsymbol{W}=\boldsymbol{V}$ for standard and $<\boldsymbol{W}>=<(\boldsymbol{A}-\tau \boldsymbol{I})\boldsymbol{V}>$ for harmonic. The setting is specified by DPRM(3).
3. Solve a projected m-dimensional eigenvalue problem with a standard technique for dense matrices. The results are called Ritz values or harmonic-Ritz values.
4. Select an eigenvalue from the results of the projected eigenproblem according to the way ITRGT or DTRGT specifies.
5. Extract an approximating eigenvector $\boldsymbol{u}$ of the given large eigenproblem from the search space by expanding the eigenvector corresponds to the selected eigenvalue of the projected eigenproblem, and let θ be its Rayleigh quotient.
6. Set a residual vector $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{u} - \theta \boldsymbol{u}$.
7. Calculate the residual norm according to the way DPRM(5) specifies.
8. Deflate the eigenproblem by the vector $\boldsymbol{u}$, if the residual norm satisfies convergence conditions which are indicated by the convergence criterion DPRM(4) and by the criterion ratio DPRM(6) of the delay-deflation.
9. Reduce the dimensions of the subspaces m to $m_{\min}$ with restarting, when the m is greater than $m_{\max}$. The values of $m_{\max}$ and $m_{\min}$ are specified by DPRM(1) and DPRM(2) respectively.
10. Obtain the next expansion vector $\boldsymbol{t}$ for the subsequent loop, which is the solution of the Jacobi-Davidson correction equation $(\boldsymbol{I}-\boldsymbol{u}\boldsymbol{u}^*)(\boldsymbol{A}-\theta \boldsymbol{I})(\boldsymbol{I}-\boldsymbol{u}\boldsymbol{u}^*)\boldsymbol{t} = -\boldsymbol{r}$ using some iterative method. In this inner iteration, the parameters from DPRM(9) to DPRM(15) are used.

Note that some projection procedures needed after deflation are omitted to explain for simplicity.

For details of the Jacobi-Davidson method, see [7] in Appendix A, "References."

DM_VJDNHCR

Eigenvalues and eigenvectors of a complex sparse matrix (Jacobi-Davidson method, compressed row storage method)
CALL DM_VJDNHCR(ZA, NZ, NCOL, NFRNZ, N, ITRGT, ZTRGT, NSEL, NEV, ITMAX, ITER, IFLAG, DPRM, ZEVAL, ZEVEC, KV, DHIS, KH, ICON)

(1) Function

This subroutine computes a few of selected eigenvalues and corresponding eigenvectors of a complex sparse eigenvalue problem

$\boldsymbol{Ax} = \lambda \boldsymbol{x}$

using the Jacobi-Davidson method, where $\boldsymbol{A}$ is an $n \times n$ complex sparse matrix stored using the compressed row storage method and $\boldsymbol{x}$ is an n-dimensional vector.

(2) Parameters

ZA ..... Input. The non-zero elements of the sparse matrix $\boldsymbol{A}$ are stored.

One-dimensional complex array ZA(NZ).

For the compressed row storage method, refer to Figure DM_VJDNHCR-1.

NZ..... Input. The total number of the nonzero elements of the matrix $\boldsymbol{A}$.

NCOL..... Input. The column indices used in the compressed row storage method, which indicate the column number of each nonzero element stored in the array ZA.

One-dimensional array NCOL(NZ).

NFRNZ..... Input. The position of the first nonzero element of each row stored in the array ZA in the compressed row storage method which stores the nonzero elements row by row. Specify NFRNZ(N+1)=NZ+1.

One-dimensional array NFRNZ(N+1).

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

ITRGT..... Input. Select a way of specifying the eigenvalues to be sought ($0 \leq$ ITRGT ≤ 6).

Specify ITRGT=0 to compute eigenvalues closest to a target value ZTRGT.

Specify ITRGT=1 to compute eigenvalues with largest magnitude.

Specify ITRGT=2 to compute eigenvalues with smallest magnitude.

Specify ITRGT=3 to compute eigenvalues with largest real part.

Specify ITRGT=4 to compute eigenvalues with smallest real part.

Specify ITRGT=5 to compute eigenvalues with largest imaginary part.

Specify ITRGT=6 to compute eigenvalues with smallest imaginary part.

(See note 1) and 2) in (3), "Comments on use.")

ZTRGT..... Input. The target value τ is specified as a complex variable when ITRGT=0. In the following cases, the convergence might be improved by specifying a value near the seeking eigenvalue even when ITRGT≠0.

1) The value τ is used as a shift of the test subspace <$\boldsymbol{W}$>=<($\boldsymbol{A}-\tau\boldsymbol{I}$)$\boldsymbol{V}$> when DPRM(3)=1 which indicates that the harmonic algorithm is to be used. (See note 2) in (3), "Comments on use.")

2) When DPRM(9)≥1, the value τ is used as an approximated eigenvalue in the Jacobi-Davidson correction equation while the initial phase of the iteration is proceeding. (See note 5) in (3), "Comments on use.")

3) When DPRM(15) ≥1, the value τ is used as a shift value of the preconditioner for the Jacobi-Davidson correction equation. (See note 7) in (3), "Comments on use.")

In other cases, ZTRGT is not referred in this subroutine.

NSEL..... Input. The number of eigenvalues to be computed (1≤NSEL≤N). (See note 1) in (3), "Comments on use.")

NEV..... Output. The number of eigenvalues converged.

ITMAX..... Input. Upper limit of iterative count for the Jacobi-Davidson method (≥0).

ITER..... Output. Actual iterative count for the Jacobi-Davidson method.

IFLAG..... Input. Control information array specifying whether the auxiliary parameter is specified explicitly in DPRM array.

When IFLAG(i)≠0, the parameter specified in DPRM(i) is to be used.

When IFLAG(i)=0, a default parameter is used and DPRM(i) is not referred.

Set IFLAG(16:32) to be all zero since these area are preserved for future enhanced functionality.

One-dimensional array IFLAG(32).

DPRM..... Input. Auxiliary parameters are specified as for the IFLAG(i) denotes that the user specified value is to be used.

For definition of each parameter in the algorithm, see (4), "Method."

If all of IFLAG(1:32) are set to be zero, DPRM(1:32) are not referred and default parameters are used. Changing the parameter is recommended when the iteration did not converge with default parameters.

One-dimensional array DPRM(32).

DPRM(1): The dimension $m_{\min}$ of shrunk subspace when restarting (1≤ $m_{\min}$ <N). The default value is $m_{\min}$=50.

DPRM(2): Upper limit of the dimension $m_{\max}$ of subspace ($m_{\min} < m_{\max} \leq$N). The default value is $m_{\max}= m_{\min} +30$.
(See note 8) in (3), "Comments on use.")

DPRM(3): The type of the algorithm, which is associated with setting of a test subspace.
When DPRM(3)=0, the standard algorithm is adopted. The algorithm is appropriate for seeking the extreme eigenvalues in the spectrum.
When DPRM(3)=1, the harmonic algorithm is adopted. The algorithm is appropriate for seeking the internal eigenvalues in the spectrum.
The default value is the harmonic algorithm for ITRGT=0 or 2, or the standard algorithm in other cases.

DPRM(4): The criterion value for judgment of acceptable convergence. The default value is 10^{-6}. (See note 4) in (3), "Comments on use.")

DPRM(5): The way how to calculate the residual norm with respect to the approximated eigenvalue θ and eigenvector $\boldsymbol{u}$.
When DPRM(5)=0, the residual norm relative to the absolute value of approximated eigenvalue $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\theta|$ is adopted.
When DPRM(5)=1, the residual norm relative to the 1-norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_1$ is adopted.
When DPRM(5)=2, the residual norm relative to the Frobenius norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_F$ is adopted.
When DPRM(5)=3, the residual norm relative to the infinity-norm of the matrix $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\boldsymbol{A}|_\infty$ is adopted.
When DPRM(5)=4, the absolute residual norm $|\boldsymbol{Au}-\theta\boldsymbol{u}|$ is adopted.
The default is DPRM(5)=0. (See note 3) in (3), "Comments on use.")

DPRM(6): A criterion value for a delay-deflation scheme (≤1.0).
The default value is DPRM(6)=0.9. (See note 4) in (3), "Comments on use.")

DPRM(7): Control information indicating whether the iteration is started from a vector specified in the array ZEVEC(1:N,1).
When DPRM(7)=0, the iteration is started from a random vector generated in this subroutine internally.
When DPRM(7)=1, set an initial vector in the array ZEVEC(1:N,1).
The default setting is using a random vector.

DPRM(8): A seed to generate a random vector (≥1.0). The default value is 1.

DPRM(9): While the iteration count is less or equal to DPRM(9), the process is regarded as an initial phase of the iteration. Then the fixed value of τ is used as an approximated eigenvalue instead of the value of θ in the Jacobi-Davidson correction equation.
When DPRM(3)=0, the default value is DPRM(9)=0.
When DPRM(3)=1, the default value is DPRM(9)= $m_{\max}$.
(See note 5) in (3), "Comments on use.")

DPRM(10): The method to solve the Jacobi-Davidson correction equation.
When DPRM(10)=0, $\boldsymbol{t}=\boldsymbol{r}$ is set without using the correction equation.
When DPRM(10)=1, the GMRES method is adopted.
When DPRM(10)=2, the BiCGstab(L) method is adopted.
The default is using the GMRES method. (See 8) in (3), "Comments on use.")

DPRM(11): A parameter for the solver of the correction equation.
When the BiCGstab(L) is used, specify the value of L (≤10). The default value is 4.

DPRM(12): Upper limit of the iteration count of the solver for the Jacobi-Davidson correction equation (≥1). The default value is 30.

DPRM(13): A parameter to determine the stopping criterion for the iterative solver of the correction equation (>0.0).
The default value is 0.7. (See 6) in (3), "Comments on use.")

DPRM(14): A parameter to determine the stopping criterion for the iterative solver of the correction equation (0.0<DPRM(14)≤1.0). The stopping criterion is set to DPRM(13)×DPRM(14)^l, where l is an iteration

counter of the outer loop which is reset in each deflation.
The default value is 0.7. (See 6) in (3), "Comments on use.")

DPRM(15): The type of preconditioning of the correction equation (≤1).
When DPRM(15)=0, no preconditioning is used.
When DPRM(15)=1, the diagonal left preconditioning is exploited.
(See 7) in (3), "Comments on use.")
The default is DPRM(15)=0.

DPRM(16:32): Preserved area for future enhanced functionality.

ZEVAL..... Output. Detected eigenvalues are stored in ZEVAL(1:NEV).

One-dimensional complex array ZEVAL(NSEL).

ZEVEC..... Output. Detected eigenvectors are stored in ZEVEC(1:N,1:NEV).

Two-dimensional complex array ZEVEC(KV,NSEL).

Input. Set the initial vector in ZEVEC(1:N,1) when IFLAG(7)≠0 and DPRM(7)=1.0.

KV..... Input. Size of the first dimension of array ZEVEC (≥N).

DHIS..... Output. The convergence history of the residuals of the eigenproblem are stored in DHIS(1:*min*(KH,ITER),1). The final relative residual norm of the each correction equation are stored in DHIS(1:*min*(KH,ITER),2).

Two-dimensional array DHIS(KH,2).

KH..... Input. Size of the first dimension of array DHIS (≥0). Setting KH=ITMAX is enough. If KH=0 is set, the outputs to the array DHIS are suppressed.

ICON..... Output. Condition code.

(See Table DM_VJDNHCR-1.)

Table DM_VJDNHCR-1 Condition codes

Code	Meaning	Processing
0	No error	–
1000	Breakdown occurred in the iterative linear equations solver.	Processing is continued with the approximated solution until the point.
2000	A null vector is detected in a sort of process of the orthogonalization.	Processing is continued with the subspace expanded by a random vector.
3000	A recovery procedure is activated in a sort of restorative process of the delay deflation.	Processing is continued
10000	The iteration count reached the maximum limit before NSEL-th eigenvalue is obtained.	The calculated eigenpairs up to NEV are correct.
20000	The projected dense eigenproblem can not be solved.	Processing is discontinued. The calculated eigenpairs up to NEV are correct if NEV>0.

Code	Meaning	Processing
21000	The iteration count reached the maximum limit without a single convergence.	Processing is discontinued. The approximate values obtained up to this point are output in array ZEVAL(1) and ZEVEC(1:N,1), but their precision cannot be guaranteed.
29000	An internal error occurred.	Processing is discontinued.
30000	N<1, ITRGT<0, ITRGT>6, NSEL<1, NSEL>N, ITMAX<0, KV<N or KH<0.	
30001 to 30032	The value of IFLAG or DPRM is not correct.	
31000	The value of NZ, NCOL or NFRNZ is not correct.	

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 0 & 6 \\ 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 \end{bmatrix}$$

$$\mathbf{NFRNZ} = \begin{bmatrix} 1 \\ 4 \\ 7 \\ 10 \\ 12 \end{bmatrix}, \quad \mathbf{ZA} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{bmatrix}, \quad \mathbf{NCOL} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 4 \\ 2 \\ 3 \\ 4 \\ 3 \\ 4 \end{bmatrix}$$

Figure DM_VJDNHCR-1 Storing a matrix A in compressed row storage method

(3) Comments on use

a. Notes

1) Robustness of the Jacobi-Davidson algorithm
   The Jacobi-Davidson algorithm is not a decisive procedure, and hence is not as robust as the method for dense matrices based on the reduction of matrix elements. The results obtained using the Jacobi-Davidson method depends on choice of the initial vector, and the order of obtained eigenvalues are not guaranteed to be the order of precedence user specified. This method is applicable when the seeking eigenvalues are only a few of the entire spectrum.

   The convergence behavior of this routine is affected by various auxiliary parameters. For description of these parameters, refer to "Comments on use."

2) ITRGT and ZTRGT parameter
   The default value of DPRM(3), which specifies a type of algorithm, is switched automatically according to the setting of ITRGT, which specifies a way of selecting eigenvalues. However, an explicit specification of the value in DPRM(3) by setting IFLAG(3)≠0 is prior to the default value of course. Which means that the standard algorithm can be used with ITRGT=0 or 2, and that the harmonic algorithm can be used with ITRGT=1,3,4,5 or 6, as long as user knows its adaptivity.

   Note that the ZTRGT parameter is referred as a shift of the test subspace for the default harmonic algorithm when just setting ITRGT=2, which specifies to compute eigenvalues with smallest magnitude. Define the ZTRGT to be (0.0,0.0) if other appropriate value is not known.

3) Calculating the residual norm
   In the default setting, convergence of the eigenproblem is judged based on the residual norm relative to the absolute value of the approximated eigenvalue. When the absolute value of the seeking eigenvalue is far smaller than the norm of the matrix, however, it is difficult to satisfy the convergence condition $|\boldsymbol{Au}-\theta\boldsymbol{u}|/|\theta|<$DPRM(4). In that case, adjust the convergence criterion DPRM(4), or change the way of calculating the residual norm which can be specified by DPRM(5) parameter.

4) Delay deflation procedure
   This subroutine adopts an ingenious scheme to improve the precision of the results. After the residual becomes below the convergence criterion, this subroutine still continues some more iteration without deflation while the decrease ratio of the residual remains valid. This procedure is called *delay-deflation* here. The decrease ratio is regarded valid if the ratio of the residual norm relative to the preceding residual is less than the parameter DPRM(6). If the residual deteriorates while this extra iteration, the better previous variables are restored and the deflation with the vector takes place. With setting DPRM(6)=0.0, this delay-deflation does not act and then the parameter DPRM(4) is regarded as an ordinary convergence criterion.

5) Approximated eigenvalue in the correction equation
   In the initial few steps of the process, the values of θ are usually poor approximations of the wanted eigenvalue. This subroutine takes the target value τ specified in the ZTRGT as an approximated eigenvalue instead of θ in the initial phase, since the validity of the expansion vector $\boldsymbol{t}$ is affected by the closeness to the approximated eigenvalue in the Jacobi-Davidson correction equation. The process is regarded as the initial phase of the iteration while the iteration count is less than or equal to DPRM(9). However, the default value of this parameter is DPRM(9)=0 when DPRM(3)=0 is adopted, because it is difficult to determine a value of τ in advance when the standard algorithm is specified.

6) Stopping criterion for inner iteration
The Jacobi-Davidson correction equation is solved by some iterative method in this subroutine, thus the whole algorithm consists of two nested iterations. In the outer iteration the approximation for the eigenproblem is constructed , and in the inner iteration the correction equation is approximately solved. If the residual of the eigenproblem still not be small in the outer iteration, solving accurately the correction equation in the inner iteration might be unnecessary. Therefore, the stopping criterion for the inner iteration can be varied according to a counter associated with the outer iteration. The criterion is set to be DPRM(13)×DPRM(14)^{l}, where l is the outer iteration counter which is reset to zero at each deflation. Incidentally, the upper limit count for the inner iteration is specified by DPRM(12).

7) Precondition for the correction equation
It is known that a good preconditioner improves the convergence of the iterative method for linear equations. The preconditioner to be applied is controlled by the parameter DPRM(15) in this subroutine. Note that the value of ZTRGT is used for constructing a matrix $M \cong (A-\tau I)$, which approximates a part of the coefficient matrix in some way. The preconditioner is derived from the inverse procedure of the matrix M and projections on both sides. If the preconditioner does not approximate the coefficient matrix of the correction equation properly or the parameter ZTRGT is far from the seeking eigenvalue, the convergence may deteriorate.

8) Memory usage
This subroutine exploits work area internally as auto allocatable arrays. Therefore an abnormal termination could occur when the available area of the memory runs out. The necessary size for the outer iteration is at least $n\times(3\times m_{max}+2\times NSEL)\times 16$ bytes for the standard algorithm and $n\times(4\times m_{max}+2\times NSEL)\times 16$ bytes for the harmonic algorithm. And when the GMRES method is used as the solver of the correction equation, the additional necessary area is $n\times DPRM(12)\times 16$ bytes for the inner iteration.

b. Example

Ten largest eigenvalues in magnitude and corresponding eigenvectors of an eigenproblem $\boldsymbol{Ax}=\lambda\boldsymbol{x}$ are sought, where $\boldsymbol{A}$ is a 10000×10000 example matrix of the random sparsity pattern with 20 nonzero entries in each row.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on a system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE
      INTEGER NNZMAX,NMAX,LDK,NZC
      PARAMETER (NMAX=10000,NZC=20)
      PARAMETER (NNZMAX=NMAX*NZC)
      PARAMETER (LDK=10)
      COMPLEX*16 ZA(NNZMAX),ZTRGT,ZEVAL(LDK),ZEVEC(NMAX,LDK)
      COMPLEX*16 RVEC(NMAX)
      REAL*8 DERR,DPRM(32),DHIS(NMAX,2)
      INTEGER NZ,NCOL(NNZMAX),NFRNZ(NMAX+1),N,ITRGT,IFLAG(32)
      INTEGER NSEL,NEV,ITMAX,ITER,I,J,K,ICON,LDX,LDH
```

```
      N=NMAX
      CALL MKSPMAT(N,NZC,ZA,NCOL,NFRNZ)
      NZ=NFRNZ(N+1)-1

      ITMAX = 500
      NSEL = 10
      DO I = 1,32
       IFLAG(I)=0
      ENDDO
      LDX = NMAX
      LDH = NMAX
      ZTRGT = (0.0D0,0.0D0)
      ITRGT = 1

      CALL DM_VJDNHCR(ZA,NZ,NCOL,NFRNZ,N,ITRGT,ZTRGT,NSEL,NEV,
     &          ITMAX,ITER,IFLAG,DPRM,ZEVAL,ZEVEC,LDX,DHIS,LDH,ICON)

      PRINT *,'DM_VJDNHCR ICON=',ICON
      PRINT *,'ITER=',ITER
      DO K = 1,NEV
       RVEC(1:N)=(0.0D0,0.0D0)
!$OMP PARALLEL DO PRIVATE(J)
       DO I=1,N
        DO J=NFRNZ(I),NFRNZ(I+1)-1
         RVEC(I)=RVEC(I)+ZA(J)*ZEVEC(NCOL(J),K)
        ENDDO
        RVEC(I)=RVEC(I)-ZEVAL(K)*ZEVEC(I,K)
       ENDDO
       DERR=0.0D0
       DO I=1,N
        DERR=DERR +DREAL(RVEC(I))**2 +DIMAG(RVEC(I))**2
       ENDDO
       DERR=DSQRT(DERR)
       PRINT*,'EIGEN VALUE',K,'=',ZEVAL(K)
       PRINT*,'ERROR=',DERR/CDABS(ZEVAL(K))
      ENDDO
      STOP
      END

      SUBROUTINE MKSPMAT(N,NZC,ZA,NCOL,NFRNZ)
      IMPLICIT NONE
      INTEGER N,NZC,NCOL(*),NFRNZ(*)
      COMPLEX*16 ZA(*)
      INTEGER I,IC,ICT,J,K,ISEED,LDW,ICON
      PARAMETER(LDW=1350)
      REAL*8 DWORK(NZC),RNDWORK(LDW)
      ISEED=1
      CALL DVRAN4(0.0D0,1.0D0,ISEED,ZA,2*N*NZC,RNDWORK,LDW,ICON)
      ISEED=1
      DO I=1,N
       NFRNZ(I)=(I-1)*NZC+1
 10    CALL DVRAU4(ISEED,DWORK,NZC,RNDWORK,LDW,ICON)
       IC=(I-1)*NZC
```

```
 DO J=1,NZC
  ICT=N*DABS(DWORK(J))+1
  DO K=1,J-1
   IF(ICT.EQ.NCOL(IC-K+1))GO TO 10
  ENDDO
  IC=IC+1
  NCOL(IC)=ICT
 ENDDO
ENDDO
NFRNZ(N+1)=IC+1
RETURN
END
```

(4) Method

This subroutine solves large sparse eigenproblems using the Jacobi-Davidson method. In the Jacobi–Davidson approach a small m-dimensional search subspace is updated in each iteration, from which an approximated eigenvector of the given n-dimensional eigenproblem is sought. In each iteration there are two important phases of procedure, one is expansion in which the subspace is enlarged by adding a new appropriate basis vector to it, and one is extraction in which a sensible approximate eigenpair is sought from the search subspace. For the subspace expansion phase, a correction vector against an approximated eigenvector is calculated as a solution vector of the Jacobi-Davidson correction equation. For the extraction phase, an approximated eigenpair is calculated as a solution of a small projected eigenproblem. Even when the seeking eigenvalues are in the interior of the spectrum, this method extracts them ingeniously using distinct search and test subspaces.

The following shows the overall procedure of the algorithm and describes some auxiliary parameters used in this subroutine.

1. Prepare an expansion vector $\boldsymbol{t}$.
2. Expand the search subspace $<\boldsymbol{V}>$ and the test subspace $<\boldsymbol{W}>$ according to the vector $\boldsymbol{t}$, where $\boldsymbol{V}$ and $\boldsymbol{W}$ are $n \times m$ matrices, and $<\boldsymbol{V}>$ represents a linear subspace spanned by the column vectors of the matrix $\boldsymbol{V}$. The type of the algorithm whether standard or harmonic can be distinguished by the setting of the test subspace, $\boldsymbol{W}=\boldsymbol{V}$ for standard and $<\boldsymbol{W}>=<(\boldsymbol{A}-\tau \boldsymbol{I})\boldsymbol{V}>$ for harmonic. The setting is specified by DPRM(3).
3. Solve a projected m-dimensional eigenvalue problem with a standard technique for dense matrices. The results are called Ritz values or harmonic-Ritz values.
4. Select an eigenvalue from the results of the projected eigenproblem according to the way ITRGT or ZTRGT specifies.
5. Extract an approximating eigenvector $\boldsymbol{u}$ of the given large eigenproblem from the search space by expanding the eigenvector corresponds to the selected eigenvalue of the projected eigenproblem, and let θ be its Rayleigh quotient.
6. Set a residual vector $\boldsymbol{r}=\boldsymbol{A}\boldsymbol{u}-\theta \boldsymbol{u}$.
7. Calculate the residual norm according to the way DPRM(5) specifies.

8. Deflate the eigenproblem by the vector $\boldsymbol{u}$, if the residual norm satisfies convergence conditions which are indicated by the convergence criterion DPRM(4) and by the criterion ratio DPRM(6) of the delay-deflation.
9. Reduce the dimensions of the subspaces m to $m_{\min}$ with restarting, when the m is greater than $m_{\max}$. The values of $m_{\max}$ and $m_{\min}$ are specified by DPRM(1) and DPRM(2) respectively.
10. Obtain the next expansion vector $\boldsymbol{t}$ for the subsequent loop, which is the solution of the Jacobi-Davidson correction equation $(\boldsymbol{I}-\boldsymbol{u}\boldsymbol{u}^*)(\boldsymbol{A}-\theta\boldsymbol{I})(\boldsymbol{I}-\boldsymbol{u}\boldsymbol{u}^*)\boldsymbol{t} = -\boldsymbol{r}$ using some iterative method. In this inner iteration, the parameters from DPRM(9) to DPRM(15) are used.

Note that some projection procedures needed after deflation are omitted to explain for simplicity.

For details of the Jacobi-Davidson method, see [7] in Appendix A, "References."

DM_VLAX

A system of linear equations with a real matrix (blocked LU decomposition method)
CALL DM_VLAX(A,K,N,B,EPSZ,ISW,IS,IP,ICON)

(1) Function

This subroutine solves a system of real coefficient linear equations using the blocked LU-decomposition method of outer product type.

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

where, $\boldsymbol{A}$ is a non-singular real matrix of $n \times n$, $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector. ($n \geq 1$)

(2) Parameters

A ..... Input. Matrix $\boldsymbol{A}$ is stored in A(1:N,1:N).

Output. Matrices L and U are stored in A(1:N,1:N).

This is a double precision real two-dimensional array A(K,N).

The value of A other than A(1:N,1:N) is not assured after operation.

K ..... Input. The size of first dimension of array for storage A (≥ N).

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

B ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

A double precision one-dimensional array of size N.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed. (See note 1) in (3), "Comments on use.")

ISW ..... Input. Control information.

When solving k (≥ 1) sets of equations having the same coefficient matrix, specify as follows.

Specify ISW = 1 for the first set of equations.

Specify ISW = 2 for the second and subsequent sets. When specifying ISW = 2, change only the value of B into a new constant vector b and do not change other parameters.

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

IS ..... Output. Information to obtain the determinant of matrix A. The determinant is obtained by multiplying the product of the n diagonal elements of array A by the value of IS after decomposition.

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

IP ..... Output. The transposition vector which indicates the history of row exchange by partial pivoting. A one-dimensional array of size n.

ICON ..... Output. Condition code.

See Table DM_VLAX-1.

Table DM_VLAX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All the elements in some row of matrix *A* are zero, or the pivot becomes relatively zero. Matrix *A* may be singular.	Processing is discontinued.
30000	K < N, N < 1 or EPSZ < 0.0	Processing is discontinued

(3) Comments on use

a. Notes

1) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) When several sets of linear equations that have an identical coefficient matrix are successively solved, the value of ISW should be 2 from the second time on. This reduces the execution time because LU decomposition of coefficient matrix ***A*** is bypassed. The value of IS does not change from the time ISW = 1.

3) This subroutine calls DM_VALU and DM_VLUX internally. Therefore, instead of calling this function in a parallel region with specifying the number of threads by run-time library OMP_SET_NUM_THREADS(), call DM_VALU and DM_VLUX directly with specifying the number of threads with OMP_SET_NUM_THREADS() just before the each of them.

b. Example

A system of linear equations having on 4000×4000 coefficient matrix is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(4001,4000)
      DIMENSION IP(4000),B(4000)
C
      N=4000
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(A,B,N)

!$OMP DO
      DO J=1,N
      DO I=1,N
      A(I,J)=MIN(I,J)
      ENDDO
```

```
      ENDDO
!$OMP END DO

!$OMP DO
      DO I=1,N
      B(I)=I*(I+1)/2+I*(N-I)
      ENDDO
!$OMP END DO

!$OMP END PARALLEL
C
      K=4001
      CALL DM_VLAX(A,K,N,B,0.0D0,1,IS,IP,ICON)
      WRITE(6,610)ICON
      IF(ICON.GE.20000)STOP

      S=1.0D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A,N)
!$OMP+          REDUCTION(*:S)
      DO I=1,N
      S=S*A(I,I)
      ENDDO
!$OMP END PARALLEL DO

      DET=IS*S

C
      WRITE(6,620)(I,B(I),I=1,10)
      WRITE(6,630)DET
  610 FORMAT(1H0,10X,16HCONDITION CODE =,I5)
  620 FORMAT(1H0,10X,15HSOLUTION VECTOR
     */(10X,3(1H(,I3,1H),D23.16)))
  630 FORMAT(1H0,10X,
     *27HDETERMINANT OF THE MATRIX =,D23.16)
      END
```

DM_VLBX

A system of linear equations with banded real matrices (Gaussian elimination)
CALL DM_VLBX(A,K,N,NH1,NH2,B,EPSZ,ISW,IS,IP,ICON)

(1) Function

This subroutine solves a system of linear equations with the banded real matrix using Gaussian elimination.

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

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, and $\boldsymbol{x}$ is an n-dimensional solution vector.

$n > h_1 \geq 0$, $n > h_2 \geq 0$

(2) Parameters

A ..... Input. Store a banded coefficient matrix $\boldsymbol{A}$.

Matrix $\boldsymbol{A}$ is stored in A(NH1 + 1:2 × NH1 + NH2 +1,1:N). For A(1:NH1, 1:N), set zero for the elements of matrix $\boldsymbol{A}$ outside the band.

See Figure DM_VLBX-1.

Output. The LU-decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored.

See Figure DM_VLBX-2.

This is a double precision real two-dimensional array A(K,N).

The value of A(2×NH1+NH2+2:K, 1:N) is not assured after operation.

K ..... Input. The size of first dimension of array A(≥ 2×NH1+NH2+1).

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

NH1 ..... Input. Lower bandwidth size h_1.

NH2 ..... Input. Upper bandwidth size h_2.

B ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

A one-dimensional array of size n.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is set. (See note 1) in (3), “Comments on use.”)

ISW ..... Input. Control information.

When solving k ($k \geq 1$) sets of equations having the same coefficient matrix, specify as follows.

Specify ISW = 1 for the first set of equations.

Specify ISW = 2 for the second and subsequent sets of equations. When specifying ISW = 2, change only the value of B into a new constant vector $\boldsymbol{b}$ and do not change any other parameters.

IS ..... Output. Indicates the row vector exchange count.

When IS is 1, the exchange count is even.

When IS is -1, the exchange count is odd.

(See note 3) in (3), “Comments on use.”)

IP ..... Output. A one-dimensional array of size *n*. The transposition vector to contain row exchange information is stored.

(See note 2) in (3), “Comments on use.”)

ICON ..... Output. The condition code.

See Table DM_VLBX-1.

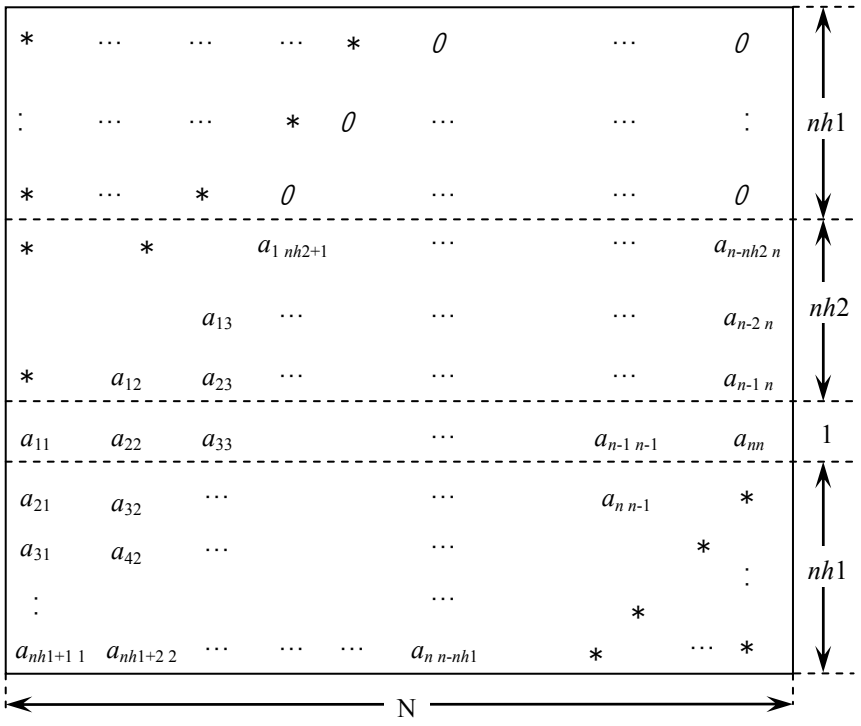


Figure DM_VLBX-1 Storing matrix *A* in array A

The column vector of matrix ***A*** is continuously stored in columns of array A in the same manner as diagonal elements of banded matrix ***A*** a_{ii}, $i = 1, ..., n$, are stored in A($nh_1 + nh_2 + 1,1:n$).

Upper banded matrix part

$a_{j-i,j}$, $i = 1, ..., nh_1$, $j = 1, ..., n$, $j - i \geq 1$ is stored in A ($nh_1 + 1:nh_1 + nh_2, + 1,1:n$).

Lower banded matrix part

$a_{j+i,j}$, $i = 1, ... , nh_1$, $j = 1, ... , n$, $j + i \leq n$ is stored in A($nh_1 + nh_2 + 2:2 \times nh_1 + nh_2 + 1$, $1:n$). For A($1:nh_1,1:n$), set zero for the elements of matrix ***A*** outside the band.

* indicates undefined values.

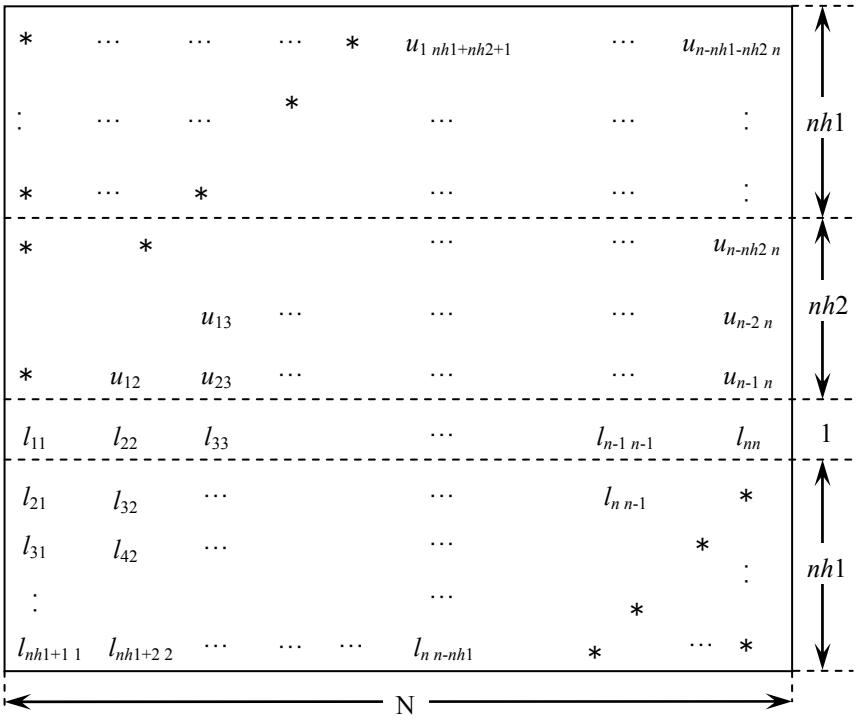


Figure DM_VLBX-2 Storing LU-decomposed matrix *L* and *U* in array A

LU-decomposed unit upper banded matrix except diagonal elements

$u_{j-i+1,j}$, $i = 1, \ldots, h_1 + h_2, j = 1, \ldots, n, j - i + 1 \geq 1$ is stored in A($1{:}h_1 + h_2, 1{:}n$).

Lower banded matrix part

$l_{j+i,j}$, $i = 0, \ldots, h_2, j = 1, \ldots, n, j + i \leq n$ is stored in A($h_1 + h_2 + 1{:}2 \times h_1 + h_2 + 1, 1{:}n$).

* indicates undefined values.

Table DM_VLBX-1 Condition codes

Code	Meaning	Processing
0	No error	
20000	All elements in some row of array A were zero, or the pivot became relatively zero. Matrix ***A*** may be singular.	Processing is discontinued.
30000	N<1, NH1 ≥ N, NH1 < 0, NH2 ≥ N, NH2 < 0, K < 2×NH1+NH2+1, EPSZ < 0.	

(3) Comments on use

a. Notes

1) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is 16 × u. When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) In this subroutine, row vector is exchanged using partial pivoting. That is, when the I-th row (I ≥ J) is selected as the pivot row in the J-th stage (J = 1, ... , n) of decomposition, the contents of the I-th row and J-th row are exchanged. To indicate this exchange, I is stored in IP(J).

3) The determinant can be obtained by multiplying IS and A($h_1 + h_2 + 1$, i), i =1, ..., n.

b. Example

The system of linear equations with banded matrices is solved with the input of a banded real $n = 10000$ matrix, $h_1 = 2000$, $h_2 = 3000$.

```
      implicit real*8(a-h,o-z)
      parameter(nh1=2000,nh2=3000,n=10000)
      parameter(ka=2*nh1+nh2+1,n2=n)
      real*8 a(ka,n2),b(n),dwork(4500)
      integer ip(n)

c
      ix=123
      nwork=4500
      nn=nh1+nh2+1
      do i=1,n
      call dvrau4(ix,a(nh1+1,i),nn,dwork,nwork,icon)
      do j=1,nh1+nh2+1
      enddo
      enddo
c
c     zero clear
c
      print*,'nh1=',nh1,',nh2=',nh2,',n=',n
c
c     a(1:nh1,n)=0.0d0
c
      do j=1,n
      do i=1,nh1
      a(i,j)=0.0d0
      enddo
      enddo
c
c     left upper triangular part
c
      do j=1,nh2
      do i=1,nh2+1-j
      a(i+nh1,j)=0.0d0
      enddo
      enddo
c
c    right rower triangular part
c
      nbase=2*nh1+nh2+1
      do j=1,nh1
      do i=1,j
      a(nbase-i+1,n-nh1+j)=0.0d0
      enddo
      enddo
c
c     set right hand constant vector
c
      do i=1,n
      b(i)=0.0d0
      enddo
c
      do i=1,n
      nptr=i-1
      do j=max(nptr+1-nh2,1),min(n,nptr+nh1+1)
      b(j)=b(j)+a(j-i+nh1+nh2+1,i)
      enddo
      enddo
```

```
c
      epsz=0.0d0
      isw=1
      call gettod(tt1)
      call dm_vlbx(a,ka,n,nh1,nh2,b,epsz,isw,is,ip,icon)
      call gettod(tt2)
      print*,'time (wall clock)=',(tt2-tt1)*1.0d-6
c
      tmp=0.0d0
      do i=1,n
      tmp=max(tmp,dabs(b(i)-1))
      enddo
c
      print*,'maximum error =',tmp
c
      stop
      end
```

DM_VLCSPSXCR1

System of linear equations with non-Hermitian symmetric complex sparse matrices (Conjugate A-Orthogonal Conjugate Residual method with preconditioning by incomplete $\boldsymbol{LDL}^T$ decomposition, symmetric compressed row storage method)
CALL DM_VLCSPSXCR1(ZSA,NZ,NCOL,NFRNZ,N,ZB, ISW, ZX,IPAR,RPAR, ZVW,ICON)

(1) Function

This subroutine solves, using Conjugate A-Orthogonal Conjugate Residual method, *COCR* method, a system of linear equations with non-Hermitian symmetric complex sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix $\boldsymbol{A}$ is stored using the symmetric compressed row storage method. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

(2) Parameters

ZSA ..... Input. The nonzero elements of the coefficient matrix are stored in ZSA(1:NZ). One-dimensional complex array ZSA(NZ). Regarding the symmetric compressed row storage method, see Fig. DM_VCLSPSXCR1-1.

NZ ..... Input. Total number of the nonzero elements belong to the coefficient matrix $\boldsymbol{A}$ ($\geq$ 1).

NCOL ..... Input. The column indices used in the compressed row storage method, which indicate the column number of each nonzero element stored in the array ZSA. One-dimensional array NCOL(NZ).

NFRNZ ..... Input. The position of the first nonzero element stored in array ZSA by the symmetric compressed row storage methods which stores the nonzero elements row by row of upper triangular portion of matrix $\boldsymbol{A}$. NFRNZ(N+1)=NZ+1. One-dimensional array NFRNZ(N+1).

N ..... Input. Order n of the matrix $\boldsymbol{A}$ ($\geq$ 1).

ZB ..... Input. The right-side constant vector of the system of linear equations is stored in ZB(1:N). One-dimensional complex array ZB(N).

ISW ..... Input. Control information.
When solving multiple sets of equations having the same coefficient matrix, specify as follows;
Specify ISW = 1 for the first set of equations.
Specify ISW = 3 for the second and subsequent sets with the same coefficient matrix and different constant vector $\boldsymbol{b}$.
When specifying ISW = 3, change only the value of ZB and ZX into a new constant vector $\boldsymbol{b}$ and initial vector $\boldsymbol{x}$ and do not change other parameters.

ZX ..... Input. The initial value of solution can be specified in ZX(1:N).
Output. The solution vector is stored in ZX(1:N).
One-dimensional complex array ZX(N).

IPAR …… Control parameters having integer values. Some parameters may be modified on output. When specify 0 for any parameter, it will be assumed to specify

default value on it. If no convergence is met by using default parameters, it is recommended to try again by making parameters change. One-dimensional array IPAR(20).
IPAR(1:5): Reserved for future extensions. Specify 0 for each, just in case.

IPAR(6): Input. Specify the upper limit of iteration counts for the *COCR* method (≥ 0). Default value is 2000.

IPAR(7): Output. Actual iteration counts.

IPAR(8): Output. Actual evaluation counts of matrix-vector multiplications $\boldsymbol{Av}$ where $\boldsymbol{A}$ is the coefficient matrix and $\boldsymbol{v}$ is iterative vector in the *COCR* method.

IPAR(9:10): Reserved for future extensions. Specify 0 for each, just in case.

IPAR(11): Input. Specify control parameter how to make compensation for dropped new nonzero elements which are filled in during incomplete $\boldsymbol{LDL^T}$ decomposition. If specify as IPAR(11)=0, no compensation will be made. If specify as IPAR(11)=1, compensation will be made by reflecting dropped entries into diagonal elements. Default value is 0.
For more detail, see note 1) in (3), "Comments on use".

IPAR(12): Output. Actual number of dropped new nonzero elements.

IPAR(13:20): Reserved for future extensions. Specify 0 for each, just in case.

RPAR ..... Control parameters having real values. Some parameters may be modified on output. When specify 0.0 for any parameter, it will be assumed to specify default value on it. If no convergence is met by using default parameters, it is recommended to try again by making parameters change.
One-dimensional array RPAR(20).

RPAR(1): Reserved for future extensions. Specify 0.0 for each, just in case.

RPAR(2): Input. Specify convergence criteria *epst* for iterative solution of given a system of linear equations by *COCR* method (≥ 0.0). Default value is 10^{-8}.

RPAR(3): Output. Relative residual norm for residual vector of the solution.

RPAR(4): Output. Real part of the accumulated sum of dropped new nonzero elements which are filled in during incomplete $\boldsymbol{LDL^T}$ decomposition. For more detail, see note 1) in (3), "Comments on use".

RPAR(5): Output. Imaginary part of the accumulated sum of dropped new nonzero elements which are filled in during incomplete $\boldsymbol{LDL^T}$ decomposition.
For more detail, see note 1) in (3), "Comments on use".

RPAR(6:20) : Reserved for future extensions. Specify 0.0 for each, just in case.

ZVW ..... Work area. Input/Output. One-dimensional array ZVW(NZ).

ICON ..... Output. Condition code.
See Table DM_VLCSPSXCR1-1.

Table DM_VLCSPSXCR1 Condition codes

Code	Meaning	Processing
0	No error.	–

Code	Meaning	Processing
20000	The iteration counts reached the upper limit.	Processing is discontinued. The already calculated approximate value is output to array ZX along with relative residual error.
29000	Matrix ***A*** is singular.	Processing is discontinued.
30000	Parameter error(s). N<1, NZ<1, NZ≠NFRNZ(N+1)-1, ISW<1, ISW=2, ISW>3, IPAR(6)<0, IPAR(11)<0, IPAR(11)>1, RPAR(2)<0.0.	

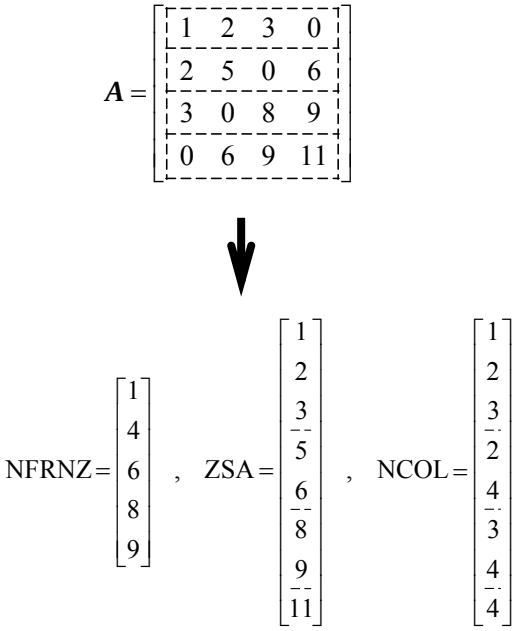


Figure DM_VLCSPSXCR1_1 Storing matrix A in symmetric compressed row storage method

(3) Comments on use

a. Notes

1) About drop of the new nonzero and its compensation
In this subroutine, the new nonzero elements which are filled in during incomplete $\boldsymbol{LDL^T}$ decomposition will be dropped in general. In order to ease up effect of such dropping, this subroutine attempts to compensate such dropping according to IPAR(11). If specify as IPAR(11)=1, it makes compensation for each diagonal elements by adding certain value which is accumulated sum of dropped new nonzero elements which are filled in on the row. By this compensation, it may affect to improve characteristic of the preconditioning matrix.

Further, this subroutine outputs the accumulated sum *zdrp* as an index regardless of IPAR(11) specification. The real part and imaginary part of *zdrp* are stored in RPAR(4) and RPAR(5) respectively.

b. Example

Read a symmetric complex matrix, then solve a linear system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ by this subroutine.
The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C=====================================================
C  TEST PROGRAM FOR KRYLOV ITERATION METHODS
C  FOR SPARSE LINEAR EQUATIONS
C  WITH NON-HERMIT COMPLEX SYMMETRIC MATRIX.
C=====================================================
      PARAMETER (NZMAX=500 000, NMAX=10 000)
      IMPLICIT   REAL*8 (A-H,O-Y)
      IMPLICIT   COMPLEX*16 (Z)
      REAL*8     CNORM2
      DIMENSION  ZSA(NZMAX),NFRNZ(NMAX+1),
     1           NCOL(NZMAX),ZX(NMAX),ZB(NMAX),
     2           ZSAT(NZMAX),NFRNZT(NMAX+1),
     3           NCOLT(NZMAX),ZXT(NMAX),ZBT(NMAX),
     4           ZVW(NZMAX)
     5          ,IPAR(20),RPAR(20)
      CHARACTER  TITLE*72
C-----------------------------------------------------
C     INPUT MATRIX FROM UF SPARSE MATRIX COLLECTION
C-----------------------------------------------------
      CALL CREADMAT(TITLE,ZSAT,N,NFRNZT,NCOLT,ZSA)
      CALL CVECGEN(ZSAT,N,NFRNZT,NCOLT,ZXT,ZBT)
      CALL CMATCOPY(ZSAT,N,NFRNZT,NCOLT,ZXT,ZBT,
     +              ZSA,NFRNZ,NCOL,ZX,ZB)
C
      WRITE(6,600) TITLE
  600 FORMAT(
     */'-------------------------------------------'
     */'TEST MATRIX : '/A36/A36)
C-----------------------------------------------------
      ISW=1
      DO II=1,20
         IPAR(II)=0
         RPAR(II)=0.0D0
      END DO
      NZ=NFRNZ(N+1)-1
      CALL DM_VLCSPSXCR1(ZSA,NZ,NCOL,NFRNZ,N,ZB,
     *                   ISW,ZX,IPAR,RPAR,ZVW,ICON)
C
      IC    =IPAR(7)
      ICMAV =IPAR(8)
      MDRP  =IPAR(11)
```

```
      NZDRP =IPAR(12)
      EPST  =RPAR(2)
      RELRES=RPAR(3)
      DRPR  =RPAR(4)
      DRPI  =RPAR(5)
      REL   =CNORM(ZB,N)
      CALL CMSVCR1(ZSA,N,NFRNZ,NCOL,ZX,ZB,0)
      RELERR=CNORM(ZB,N)/REL
C
      WRITE(6,601)
  601 FORMAT(
     */'-------------------------------------------'
     */' SOLUTION RESULTS BY "DM_VLCSPSXCR1"')
      WRITE(6,605) N,NFRNZ(N+1)-1,MDRP
      WRITE(6,606) ICON,IC,ICMAV,NZDRP,DRPR,
     *             DRPI,EPST,RELRES,RELERR
  605 FORMAT(/' N           =',I12
     *       /' NZ          =',I12
     *       /' MDRP        =',I12)
  606 FORMAT(/' ICON        =',I12
     *       /' IC          =',I12
     *       /' ICMAV       =',I12
     *       /' NZDRP       =',I12
     *       /' DRPR        =',D12.2
     *       /' DRPI        =',D12.2
     *       /' EPST        =',D12.2
     *       /' RELRES      =',D12.2
     *       /' RELERR      =',D12.2
     */'-------------------------------------------')
      IF(RELERR.LE.EPST*1.1D0.AND.ICON.EQ.0)THEN
         WRITE(*,*)' ********** OK **********'
      ELSE
         WRITE(*,*)' ********** NG **********'
      ENDIF
      STOP
      END
C=====================================================
C     READ TEST MATRIX FOR COMPLEX SYMMETRIC MATRIX.
C=====================================================
      SUBROUTINE CREADMAT(TITLE,A,NCOL,IS,JS,W)
C
C THIS ROUTINE READS MATRIX DATA OF RB SPARSE FORM.
C THE FOLLOWING SAMPLE CODE IS ORIGINATED FROM MATRIX
C MARKET;
C
      IMPLICIT NONE
      CHARACTER TITLE*72,KEY*8,MXTYPE*3,RHSTYP*3,
     1         PTRFMT*16,INDFMT*16,VALFMT*20,RHSFMT*20
      INTEGER  TOTCRD,PTRCRD,INDCRD,VALCRD,RHSCRD,
     1         NROW,NCOL,NNZERO,NELTVL,
     2         NRHS,NRHSIX
      INTEGER  IS(*),JS(*),I
      REAL*8   A(*),W(*)
```

```
      INTEGER  IX
C     ------------------------
C     READ IN HEADER BLOCK
C     ------------------------
      READ(5,1000) TITLE,KEY,TOTCRD,PTRCRD,INDCRD,
     +VALCRD,RHSCRD,MXTYPE,NROW,NCOL,NNZERO,NELTVL,
     +PTRFMT,INDFMT,VALFMT,RHSFMT
 1000 FORMAT(A72,A8/5I14/A3,11X,4I14/2A16,2A20)
C
      IF(RHSCRD.GT.0) READ(5,1001) RHSTYP,NRHS,NRHSIX
 1001 FORMAT(A3,11X,2I14)
C     ------------------------
C     READ MATRIX STRUCTURE
C     ------------------------
      READ(5,PTRFMT) (IS(I),I=1,NCOL+1)
      READ(5,INDFMT) (JS(I),I=1,NNZERO)
C
      IF(VALCRD.GT.0) THEN
C     ---------------------
C     READ MATRIX VALUES
C     ---------------------
      IF(MXTYPE(1:1).EQ.'R') THEN
         READ(5,VALFMT) (A(I),I=1,NNZERO)
      ELSE
         READ(5,VALFMT) (A(I),I=1,2*NNZERO)
      END IF
      END IF
      RETURN
      END
C=====================================================
C     COPY COMPLEX MATRIX AND VECTORS.
C=====================================================
      SUBROUTINE CMATCOPY(ZSAT,N,NFRNZT,NCOLT,
     +                ZXT,ZBT,ZSA,NFRNZ,NCOL,ZX,ZB)
      IMPLICIT   REAL*8 (A-H,O-Y)
      IMPLICIT   COMPLEX*16 (Z)
      DIMENSION  ZSAT(*),NFRNZT(*),NCOLT(*),
     +           ZXT(*),ZBT(*),ZSA(*),NFRNZ(*),
     +           NCOL(*),ZX(*),ZB(*)
C
      NZ=NFRNZT(N+1)-1
      DO I=1,N+1
         NFRNZ(I)=NFRNZT(I)
      END DO
      DO I=1,NZ
         ZSA(I)=ZSAT(I)
         NCOL(I)=NCOLT(I)
      END DO
C
      DO I=1,N
         ZX(I)=ZXT(I)
         ZB(I)=ZBT(I)
      END DO
```

```
      RETURN
      END
C=====================================================
C     GENERATE COMPLEX B AND X VECTORS.
C=====================================================
      SUBROUTINE CVECGEN(ZSAT,N,NFRNZT,NCOLT,ZXT,ZBT)
      IMPLICIT   REAL*8 (A-H,O-Y)
      IMPLICIT   COMPLEX*16 (Z)
      DIMENSION  ZSAT(*),NFRNZT(*),NCOLT(*),
     +           ZXT(*),ZBT(*)
C
C     COMPUTE RIGHT HAND SIDE VECTOR B.
      DO II=1,N
         ZXT(II)=1.0D0+DFLOAT(II)/DFLOAT(N)
      END DO
      CALL CMSVCR1(ZSAT,N,NFRNZT,NCOLT,ZXT,ZBT,1)
C
C     SET INITIAL VALUE
      DO II=1,N
         ZXT(II)=0.0D0
      END DO
      RETURN
      END
C=====================================================
C     MATRIX VECTOR MULTIPLICATION.
C     COMPLEX SYMMETRIC MATRIX STORED IN CSR FORM.
C=====================================================
      SUBROUTINE CMSVCR1(ZSA,N,NFRNZ,NCOL,ZX,ZB,ISW)
      IMPLICIT   REAL*8 (A-H,O-Y)
      IMPLICIT   COMPLEX*16 (Z)
      DIMENSION  ZSA(*),ZB(*),ZX(*),NFRNZ(*),NCOL(*)
C
      IF(ISW.EQ.1) THEN !*** MULTIPLICATION (AX=>B)
C
         DO I=1,N
            ZB(I)=0.0D0
         END DO
         DO I=1,N
            K1=NFRNZ(I)
            K2=NFRNZ(I+1)-1
            IF(ZX(I).NE.0.0D0) THEN
               DO J=K1,K2
                  ZB(NCOL(J))=ZSA(J)*ZX(I)+ZB(NCOL(J))
                  IF(NCOL(J).NE.I)
     +               ZB(I)=ZSA(J)*ZX(NCOL(J))+ZB(I)
               END DO
            ELSE
               DO J=K1,K2
                  ZB(I)=ZSA(J)*ZX(NCOL(J))+ZB(I)
               END DO
            END IF
         END DO
C
```

```
      ELSE           !*** RESIDUAL VECTOR (B-AX=>B)
C
         DO I=1,N
            K1=NFRNZ(I)
            K2=NFRNZ(I+1)-1
            IF(ZX(I).NE.0.0D0) THEN
               DO J=K1,K2
                 ZB(NCOL(J))=-ZSA(J)*ZX(I)+ZB(NCOL(J))
                  IF(NCOL(J).NE.I)
     +               ZB(I)=-ZSA(J)*ZX(NCOL(J))+ZB(I)
               END DO
            ELSE
               DO J=K1,K2
                  ZB(I)=-ZSA(J)*ZX(NCOL(J))+ZB(I)
               END DO
            END IF
         END DO
      END IF
      RETURN
      END
C=====================================================
C     L2 NORM OF A COMPLEX VECTOR.
C=====================================================
      FUNCTION  CNORM(ZX,N)
      IMPLICIT  REAL*8 (A-H,O-Y)
      IMPLICIT  COMPLEX*16 (Z)
      DIMENSION ZX(N)
      CNORM=0.0D0
      DO I=1,N
         CNORM=ZX(I)*DCONJG(ZX(I))+CNORM
      END DO
      IF(CNORM.NE.0.0D0) CNORM=DSQRT(CNORM)
      RETURN
      END
```

(4) Method

This subroutine solves a system of linear equations with non-Hermitian symmetric complex sparse matrices as coefficient matrices using Conjugate A-Orthogonal Conjugate Residual method, *COCR* method, with preconditioning by incomplete $\boldsymbol{LDL}^T$ decomposition.

a. Incomplete $\boldsymbol{LDL}^T$ decomposition

Preconditioning method makes the system to more tractable form and reduces total iteration counts. On such point of view, incomplete decomposition method is well known.

This subroutine employs a preconditioning method based on incomplete $\boldsymbol{LDL}^T$ decomposition with dropping new nonzero elements.

b. Conjugate A-Orthogonal Conjugate Residual, *COCR* method

In general, there are popular methods for solving linear systems with non-Hermitian symmetric complex sparse matrix such as *BiCG* and *CGS* method based on Lanczos process, *BiCGSTAB* method based on product type process and *GMRES* method based on Arnoldi process. However, since these methods do not take advantage of symmetric property of the matrix, number of matrix-vector multiplications come to 2 times per iteration in the kernel loop.

This subroutine employs Conjugate A-Orthogonal Conjugate Residual, *COCR* method, which takes advantage of symmetric property, holds a minimal residual property and takes stable convergence property.

c. Algorithm of *COCR* method with preconditioning

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0, \boldsymbol{z}_0 = \boldsymbol{M}^{-1}\boldsymbol{r}_0, \beta_{-1} = 0,$
$for\ k = 0,1,...until\ \|\boldsymbol{r}_k\|_2 \le \varepsilon\|\boldsymbol{b}\|_2\ do;$
$\boldsymbol{p}_k = \boldsymbol{z}_k + \beta_{k-1}\boldsymbol{p}_{k-1},$
$\boldsymbol{A}\boldsymbol{p}_k = \boldsymbol{A}\boldsymbol{z}_k + \beta_{k-1}\boldsymbol{A}\boldsymbol{p}_{k-1},$
$\alpha_k = (\bar{\boldsymbol{z}}_k, \boldsymbol{A}\boldsymbol{z}_k)/(\boldsymbol{M}^{-1}\boldsymbol{A}\bar{\boldsymbol{p}}_k, \boldsymbol{A}\boldsymbol{p}_k),$
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k, \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{A}\boldsymbol{p}_k,$
$\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{r}_{k+1}, \beta_k = (\bar{\boldsymbol{z}}_{k+1}, \boldsymbol{A}\boldsymbol{z}_{k+1})/(\bar{\boldsymbol{z}}_k, \boldsymbol{A}\boldsymbol{z}_k)$
end

Where the inner product (***x,y***) is defined by below.

$$(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \bar{x}_i y_i$$

d. Convergence test

The iterate $\boldsymbol{x}_k$ is accepted as a solution of the system if the residual satisfies $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|_2 \le epst\|\boldsymbol{b}\|_2$.
Where *epst* is a convergence criteria specified in RPAR(2). Default value of *epst* is 10^{-8}.
The final relative residual norm
$\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|_2 / \|\boldsymbol{b}\|_2$
is stored in RPAR(3), even if in the case that the residual does not satisfy convergence test. The residual vector $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$ is computed by using recurrence in the iteration formula.

For details of the algorithms, see [66] and [74] in Appendix A, "References."

DM_VLCX

A system of linear equations with complex matrices (blocked LU decomposition method)
CALL DM_VLCX(ZA,K,N,ZB,EPSZ,ISW, IS,IP, ICON)

(1) Function

This subroutine solves a system of complex coefficient linear equations using blocked LU-decomposition method of an outer product type.

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

where, $\boldsymbol{A}$ is a non-singular $n \times n$ complex matrix, $\boldsymbol{b}$ is an n-dimensional complex constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector ($n \geq 1$).

(2) Parameters

ZA..... Input. Matrix $\boldsymbol{A}$ is stored in ZA(1:N,1:N).

Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored in ZA(1:N,1:N).

This is a two-dimensional double precision complex type array ZA(K,N).

K..... Input. The size of the first dimension of the array ZA. (≥ N).

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

ZB ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector x.

A double precision complex type array ZB(N).

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, a standard value is assumed. (See note 1) in (3), Comments on use.)

ISW ..... Input. Control information.

When solving k (≥ 1) sets of equations having identical coefficient matrices, specify as follows.

Specify ISW = 1 for the first set of equations.

Specify ISW = 2 for the second and the subsequent sets of equations. When specifying ISW = 2, change only the value of ZB into a new constant vector. Do not change any other parameters.

(See note 2) in (3), “Comments on use.”)

IS ..... Output. Information to obtain the determinant of matrix $\boldsymbol{A}$.

The determinant is obtained by multiplying n diagonal elements of array ZA by the value of IS after the operation.

(See note 2) in (3), “Comments on use.”)

IP ..... Output. The transposition vector which indicates the history of the row exchange by partial pivoting. A one-dimensional array of size n.

ICON ..... Output. Condition code.

See Table DM_VLCX-1.

Table DM_VLCX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All the elements in some row of matrix ***A*** are zero, or the pivot becomes relatively zero. Matrix ***A*** may be singular.	Processing is discontinued.
30000	K < N, N < 1, EPSZ < 0.0.	

(3) Comments on use

a. Notes

1) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16u$. When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.

2) When several sets of linear equations with an identical coefficient matrix are successively solved, the value of ISW should be 2 from the second time on. This reduces the execution time because LU decomposition of coefficient matrix ***A*** is bypassed. The value of IS does not change from the time ISW = 1.

b. Example

A system of linear equations having an $n \times n$ complex coefficient matrix is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N=2000,K=N+1)
C
      COMPLEX*16 A(K,N),B(N)
      REAL*8     C
      INTEGER    IP(N),IS
C
      C=SQRT(1.0D0/DBLE(1+N))
      T=DATAN(1.0D0)*8./(1+N)
C
      DO 100 J=1,N
      DO 100 I=1,N
      A(I,J)=DCMPLX(C*COS(T*I*J),C*SIN(T*I*J))
 100  CONTINUE
C
      DO 200 I=1,N
      S=(0.,0.)
      DO 200 J=1,N
      S=S+DCMPLX(COS(T*I*J),SIN(T*I*J))
```

```
      B(I)=S*C
 200  CONTINUE
C
      ISW=1
      EPSZ=0.0D0
      CALL DM_VLCX(A,K,N,B,EPSZ,ISW,IS,IP,ICON)
      PRINT*,'ICON=',ICON

      ERROR=0.0D0
      DO I=1,N
      ERROR=MAX(ERROR,ABS(1.0D0-B(I)))
      ENDDO
      PRINT*,'ERROR =',ERROR

      PRINT*,'ORDER=',N,' B(1)=',B(1),'B(N)=',B(N)
      STOP
      END
```

DM_VLDLX

A system of linear equations with LDL^T-decomposed positive definite matrices
CALL DM_VLDLX(B,FA,KFA,N,ICON)

(1) Function

This subroutine solves a system of linear equations with LDL^T- decomposed symmetric positive definite coefficient matrix.

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

where, $\boldsymbol{L}$ and $\boldsymbol{D}$ are a unit lower triangular matrix and an $n \times n$ diagonal matrix respectively, $\boldsymbol{b}$ is an n-dimensional real constant vector, $\boldsymbol{x}$ is an n-dimensional solution vector, and $n \geq 1$.

This subroutine receives the LDL^T-decomposed matrix from subroutine DM_VSLDL and calculates the solution of a system of linear equations.

(2) Parameters

B ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

A double precision real one-dimensional array of size n.

FA ..... Input. The LDL^T-decomposed matrices $\boldsymbol{L}$, $\boldsymbol{D}^{-1}$, and $\boldsymbol{L}^T$ are stored.

The LDL^T-decomposed matrices are stored in FA(1:N,1:N). That is, FA(i,j) contains

l_{ij} (when $i > j$)

reciprocals of d_{ii} (when $i = j$).

See Figure DM_VLDLX-1.

This is a double precision real two-dimensional array FA(KFA,N).

KFA ..... Input. The size of the first dimension of array FA (≥ N).

N ..... Input. Order n of matrices $\boldsymbol{L}$ and $\boldsymbol{D}$.

ICON ..... Output. Condition code.

See Table DM_VLDLX.

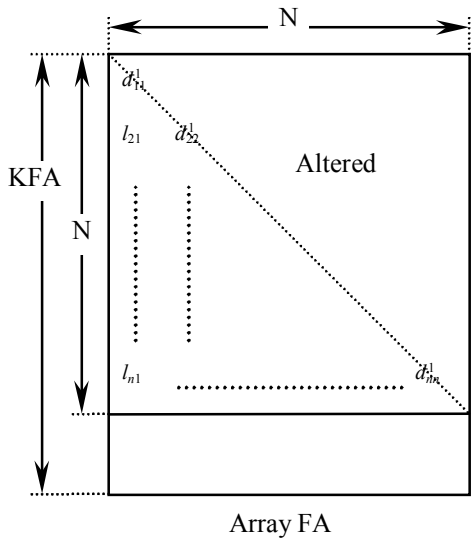


Figure DM_VLDLX-1 Storing matrices *L*, D^{-1} into array FA

After LDL^T decomposition, matrix $\boldsymbol{D}^{-1}$ is stored in diagonal elements and $\boldsymbol{L}$ (excluding the diagonal elements) are stored in the lower triangular part respectively.

Table DM_VLDLX-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	The coefficient matrix is not positive definite.	Processing is continued.
30000	N < 1, KFA < N.	Processing is discontinued.

(3) Comments on use

a. Notes

1) A system of linear equations with a positive definite coefficient matrix can be solved by calling this subroutine after calling subroutine DM_VSLDL. However, subroutine DM_VLSX should be usually used to solve a system of linear equations in one step.

b. Example

A 4000 × 4000 coefficient matrix is decomposed into LDL^T-decomposed matrix, then the system of linear equations is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N=4000,KFA=N+1)
      REAL*8    A(KFA,N)
```

```
      REAL*8    B(N)
C
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(A,B)
!$OMP DO
      DO J=1,N
      DO I=J,N
      A(I,J)=MIN(I,J)
      ENDDO
      ENDDO
!$OMP END DO

!$OMP DO
      DO I=1,N
      B(I)=I*(I+1)/2+I*(N-I)
      ENDDO
!$OMP END DO

!$OMP END PARALLEL
C
      CALL DM_VSLDL(A,KFA,N,1.D-13,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 10

      CALL DM_VLDLX(B,A,KFA,N,ICON)
      WRITE(6,630) (B(I),I=1,10)

      S=1.0D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A)
!$OMP+          REDUCTION(*:S)
      DO I=1,N
      S=S*A(I,I)
      ENDDO
!$OMP END PARALLEL DO

      DET=S
      DET=1.D0/DET
      WRITE(6,620) DET
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(3D22.15)
  600 FORMAT(1H,I5/(10X,3D23.16))
  610 FORMAT(/10X,5HICON=,I5)
  620 FORMAT(/10X
     *,34HDETERMINANT OF COEFFICIENT MATRIX=
     *,D23.16)
  630 FORMAT(/10X,15HSOLUTION VECTOR
     *//(10X,3D23.16))
  640 FORMAT(/10X,12HINPUT MATRIX)
  10  STOP
      END
```

(4) Method

The system of linear equations with LDL^T-decomposed coefficient matrices is solved by forward and back-substitution. (See [54] in Appendix A, "References.")

DM_VLSPAXCR2

System of linear equations with unsymmetric real sparse matrices (Induced Dimension Reduction method with preconditioning by sparse approximate inverse, compressed row storage method)
CALL DM_VLSPAXCR2(A,NZ,NCOL,NFRNZ,N,B,ISW,X,AM,NZM,NCOLM, NFRNZM,NWM,IPAR,RPAR,VW1,IVW1,VW2,IVW2,LMMAX,LNMAX,NUMT,ICON)

(1) Function

This subroutine solves, using IDR method with stabilization, *IDRstab(s,l)* method, a system of linear equations with unsymmetric real sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix $\boldsymbol{A}$ is stored using the compressed row storage method. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors. The parameter s is the order of shadow residual and l is the order of acceleration polynomial.

(2) Parameters

A ..... Input. The nonzero elements of the coefficient matrix are stored in A(1:NZ). The compressed row storage method is to store transposed matrix of the coefficient matrix $\boldsymbol{A}$ in the compressed column storage method. Regarding the compressed column storage method, see Fig. DM_VMVSCC-1.

NZ ..... Input. Total number of the nonzero elements belong to the coefficient matrix (≥ 1).

NCOL..... Input. The column indices used in the compressed row storage method, which indicate the column number of each nonzero element stored in the array A. One-dimensional array NCOL(NZ).

NFRNZ..... Input. The position of the first nonzero element stored in array A by the compressed row storage methods which stores the nonzero elements row by row. NFRNZ(N+1)=NZ+1. One-dimensional array NFRNZ(N+1).

N ..... Input. Order n of the matrix $\boldsymbol{A}$ (≥ 1).

B ..... Input. The right-side constant vector of the system of linear equations is stored in B(1:N). One-dimensional array B(N).

ISW..... Input. Control information.
When solving multiple sets of equations having the same sparse structure and /or the same coefficient matrix, specify as follows;
Specify ISW = 1 for the first set of equations.
Specify ISW = 2 for the second and subsequent sets with the same sparse structure and different coefficient matrix $\boldsymbol{A}$ and constant vector $\boldsymbol{b}$.
Specify ISW = 3 for the second and subsequent sets with different constant vector $\boldsymbol{b}$.
When specifying ISW = 2 or 3, change only the parameters necessary to be changed such as A, B and/or X and do not change other parameters.

X ..... Input. The initial value of solution can be specified in X(1:N).
Output. The solution vector is stored in X(1:N).
One-dimensional array X(N).

AM..... Input. If any, the nonzero elements of the initial approximate inverse matrix $\boldsymbol{M_0}$ are stored in AM(1:NZM) using the compressed row storage method. One-dimensional array AM(NWM).
The compressed row storage method is the same with matrix $\boldsymbol{A}$.
Output. The approximate inverse matrix $\boldsymbol{M}$.

NZM..... Input. If any, total number of the nonzero elements belong to the initial approximate inverse matrix $\boldsymbol{M_0}$ (≥ 1).
If not, specify as NZM=0. In this case, this subroutine employs the unit matrix as the initial approximate inverse internally.
Output. Total number of the nonzero elements of approximate inverse matrix $\boldsymbol{M}$.

NCOLM..... Input. If any, the column indices used in the compressed row storage method, which indicate the column number of each nonzero element stored in the array AM. One-dimensional array NCOLM(NWM).
Output. The column indices of approximate inverse matrix $\boldsymbol{M}$.

NFRNZM.... Input. If any,the position of the first nonzero element stored in array AM by the compressed row storage method which stores the nonzero elements row by row. NFRNZM(N+1)=NZM+1. One-dimensional array NFRNZM(N+1).
Output. The position of the first nonzero element of each row of approximate inverse matrix $\boldsymbol{M}$.

NWM..... Input. Specify the maximum size of areas used for computation of approximate inverse matrix $\boldsymbol{M}$ (≥ 1).
Total number of the nonzero elements of approximate inverse matrix $\boldsymbol{M}$ is calculated by the formula below where nz_k is number of nonzero elements in the k-th column of matrix $\boldsymbol{A}$.

$$nzm = \sum_{k=1}^{n} \max(1, nz_k \times IPAR(2) / 100)$$

Then NWM is specified as follows;

$$NWM = \max(nzm , nz) .$$

For more detail, see note 1) in (3), "Comments on use".

IPAR ..... Control parameters having integer values. Some parameters may be modified on output. When specify 0 for any parameter, it will be assumed to specify default value on it. If no convergence is met by using default parameters, it is recommended to try again by making parameters change.
One-dimensional array IPAR(20).
IPAR(1): Reserved for future extensions. Specify 0 for each, just in case.

IPAR(2): Input. Specify percentage(%) which is the ratio of nonzero elements of approximate inverse against that of the coefficient matrix $\boldsymbol{A}$ (≥ 0). It is used as upper limit control for nonzero elements generations. For instance, if specify as IPAR(2)=50, approximate inverse matrix will be generated having total nonzero number which is about 50% of that of coefficient matrix as an upper limit. Default value is 100.
For more detail, see note 3) in (3), "Comments on use".

IPAR(3): Input. Specify incremental number which is number of adding new indices during computation of column vector of approximate inverse matrix ($n \geq$ IPAR(3) ≥ 0). For instance, if specify as IPAR(3)=2, the number of indices within each column of approximate inverse will be incremented by 2 indices which are the most effective indices in term of the norm minimization. Default value is 1.
For more detail, see note 4) in (3), "Comments on use".

IPAR(4): Input. Specify the order of shadow residual *s* of Induced Dimension Reduction method *IDRstab(s,l)* ($n \geq s \geq 0$). Default value is 4.

IPAR(5): Input. Specify the order of acceleration polynomial *l* of Induced Dimension Reduction method *IDRstab(s,l)* ($n \geq l \geq 0$). Default value is 1.

IPAR(6): Input. Specify the upper limit of iteration counts for *IDRstab(s,l)* method (≥ 0). Default value is 2000.

IPAR(7): Output. Actual iteration counts.

IPAR(8): Output. Actual evaluation counts of matrix-vector multiplications $\boldsymbol{Av}$ where $\boldsymbol{A}$ is the coefficient matrix and $\boldsymbol{v}$ is iterative vector in *IDRstab(s,l)* method.

IPAR(9): Output. Estimated size NWM for AM, NCOLM etc. For more detail, see note 1) in (3), "Comments on use".

IPAR(10:12): Reserved for future extensions. Specify 0 for each, just in case.

IPAR(13): Output. Actual size LMMAX used for VW2 and IVW2.

IPAR(14): Output. Actual size LNMAX used for VW2.

IPAR(15:20): Reserved for future extensions. Specify 0 for each, just in case.

RPAR ..... Control parameters having real values. Some parameters may be modified on output. When specify 0.0 for any parameter, it will be assumed to specify default value on it. If no convergence is met by using default parameters, it is recommended to try again by making parameters change.
One-dimensional array RPAR(20).

RPAR(1):Input. Specify convergence criteria *eps* with iterative computation for each column of approximate inverse matrix (≥ 0.0). Default value is 0.3. For more detail, see note a. "Approximate Inverse Matrix" in (4), "Method".

RPAR(2):Input. Specify convergence criteria *epst* for iterative solution of given a system of linear equations by *IDRstab(s,l)* method (≥ 0.0). Default value is 10^{-8}. For more detail, see note c. "Convergence Check" in (4), "Method".

RPAR(3): Output. Relative residual norm for residual vector of the solution.

RPAR(4:20): Reserved for future extensions. Specify 0.0 for each, just in case.

VW1..... Work area. One-dimensional array VW1(NWM).

IVW1 ..... Work area. One-dimensional array IVW1(NWM).

VW2..... Work area. Three-dimensional array VW2(LMMAX,LNMAX+3,NUMT).

IVW2 ..... Work area. Three-dimensional array IVW2(LMMAX,LNMAX+3,NUMT).

LMMAX.... Input. The first dimension of working array (≥ 1).
LMMAX is a certain value related to the number of nonzero elements of matrix $\boldsymbol{A}$. Lets see certain column of matrix $\boldsymbol{A}$, we defines the total number of nonzero elements in the column and another columns which are relatives of the nonzero elements of the column. Specify the maximum number of the total number between columns. In general, it is adequate to specify as LMMAX=1000. If no solution is met, it is recommended to try again by making parameters change. For more detail, see note 5) in (3), "Comments on use".

LNMAX..... Input. The second dimension of working array (≥ 1).
LNMAX is a certain value proportional to the maximum number of nonzero elements between columns of matrix ***A***. In general, specify the maximum number of nonzero elements for regular use with IPAR(2)=100. If no solution is met, it is recommended to try again by making parameters change.
For more detail, see note 5) in (3), "Comments on use".

NUMT..... Input. The third dimension of working array (≥ 1).
Specify maximum number of threads for parallel processing.

ICON ..... Output. Condition code.
See Table DM_VLSPAXCR2-1.

Table DM_VLSPAXCR2 Condition codes

Code	Meaning	Processing
0	No error.	–
11000	Matrix ***A*** may be near singular.	Processing is continued.
19000	Non diagonal element(s) is detected in matrix ***A***.	
20000	The iteration counts reached the upper limit.	Processing is discontinued. The already calculated approximate value is output to array X along with relative residual error.
25000	Array AM and NCOLM overflow due to too small value NWM.	Processing is discontinued. Estimated minimum size is output to IPAR(9).
26000	Work area VW2, IVW2 overflow due to too small value LMMAX.	Processing is discontinued.
27000	Work area VW2 overflow due to too small value LNMAX.	
29000	Matrix ***A*** is singular.	
30000	Parameter error(s). N<1, NZ<1, NZ≠NFRFZ(N+1)-1, ISW<1, ISW>3, NWM<1N, NZM<0, IPAR(1)<0, IPAR(2)<0, IPAR(3)<0, IPAR(3)<0, IPAR(4)<0, *n*<IPAR(4), IPAR(5)<0, *n*<IPAR(5), IPAR(6)<0, LMMAX<1, LNMAX<1, NUMT<1, RPAR(1)<0.0, RPAR(2)<0.0.	
30011	Parameter error(s) related to matrix ***A***. Some parameter value show following relation. NFRNZ(*k*)>NFRNZ(*k*+1), *k*=1,...,*n*	
30012	Parameter error(s) related to matrix ***A***. Some parameter value show following relation. NCOL(*l*)>NCOL(*l*+1), *l*=NFRNZ(*k*),...,NFRNZ(*k*+1), *k*=1,...,*n*	

Code	Meaning	Processing
30021	Parameter error(s) related to matrix $\boldsymbol{M_0}$. Some parameter value show following relation. NFRNZM(k)>NFRNZM(k+1), k=1,…,n.	Processing is discontinued.
30022	Parameter error(s) related to matrix $\boldsymbol{M_0}$. Some parameter value show following relation. NCOLM(l)>NCOLM(l+1), l=NFRNZM(k),...,NFRNZM(k+1), k=1,...,n.	

(3) Comments on use

a. Notes

1) About the size of arrays for approximate inverse matrix

The size *nzm* of approximate inverse matrix $\boldsymbol{M}$ is calculated by the formula below where nz_k is number of nonzero elements in the k-th column of matrix $\boldsymbol{A}$.

$$nzm = \sum_{k=1}^{n} \max(1, nz_k \times IPAR(2)/100)$$

Then the size of array NWM is specified as follows;

$$NWM = \max(nzm, nz).$$

In general, if you use default value for IPAR(2), that is IPAR(2)=0, which specifies upper limit of percentage of nonzero elements generations, it is adequate to specify as NWM=NZ. When it is difficult to calculate NWM by above formula, it is recommended to specify enough big size such as NWM=2×NZ. As a result of operation of this subroutine, the suggested size is output on IPAR(9). This resultant value gives good suggestion for subsequent call to solve a system with a similar sparse matrix. If you solve another system having the same sparse structure and the equivalent nonzero percentage of approximate inverse, you can take IPAR(9) as a suggestion. On the other hand, if you solve another system having much more nonzero elements than previous, or increasing percentage of nonzero elements in approximate inverse, you can take IPAR(9) multiplied by each increasing ratio as a suggestion.

2) About the initial approximate inverse matrix

If you have a good approximate inverse matrix $\boldsymbol{M_0}$, you can specify it as an initial value on relevant parameters. You can specify total nonzero number of the matrix $\boldsymbol{M_0}$ on NZM, and specify the initial approximate inverse matrix on AM,NCOLM and NFRNZM respectively.

Such usage is recommended for user who would process following type of problems in efficient manner.

#1 to solve multiple set of equations with the same sparse structure and different coefficient matrix $\boldsymbol{A}$ and constant vector $\boldsymbol{b}$.

#2 to solve multiple set of equations with similar sparse structure.

Process is controlled along with parameter ISW. In these cases, change only the value of A and/or related parameters and B, X, and do not change other parameters such as AM and work areas in which previous results are stored.

In this case, it is possible to increase the upper limit by making parameter IPAR(2) change.

3) About total nonzero number of approximate inverse matrix $\boldsymbol{M}$

This subroutine solves a system of linear equations with preconditioning based on approximate inverse matrix,

$\boldsymbol{AMy} = \boldsymbol{b}$, $\boldsymbol{x} = \boldsymbol{My}$.

Approximate inverse matrix $\boldsymbol{M}$ is computed so as to be satisfied $\boldsymbol{AM} \fallingdotseq \boldsymbol{I}$. The

total number of nonzero elements of $\boldsymbol{M}$ affects not only accuracy of inverse but also performance of matrix vector multiplication which is appeared frequently during iterations. In this subroutine, it is able to control the total number of nonzero elements of matrix $\boldsymbol{M}$ via parameter IPAR(2). In general, it is recommended the nonzero number take the same order with that of matrix $\boldsymbol{A}$. That is, IPAR (2)=100 is recommended.
This subroutine computes inverse matrix $\boldsymbol{M}$ column by column, $\boldsymbol{m}_k$, k=1, ...,n.
The iterate $\boldsymbol{m}_k$ of inverse matrix $\boldsymbol{M}$ is accepted as a minimum solution if

$$\left\|\boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k\right\|_2 \le eps$$

is satisfied even if nonzero number in $\boldsymbol{m}_k$ does not reach upper limit nz_k×IPAR (2)/100.
Where nz_k is number of nonzero elements in k-th column of matrix $\boldsymbol{A}$.

4) About incremental number during computation of column vector of inverse
This subroutine computes column vector $\boldsymbol{m}_k$ of matrix $\boldsymbol{M}$ by solving least squares problems as follows;

$$\min_{m_k}\left\|\boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k\right\|_2, k = 1,...,n \ .$$

Where $\boldsymbol{e}_k$ is unit vector. Residual vector based on the solution above may lead candidates of new nonzeros in next step $\boldsymbol{m}_k$. This subroutine selects new indices automatically from candidates in terms of the most profitable one which minimizes coming residual vector. Key point of this algorithm lies in determining a good sparsity structure of the column of approximate inverse. In order to increase nonzero elements gradually, it is recommended to specify as IPAR(3)=1 which is number of adding new indices during computation of column vector.

5) About work area VW2,IVW2
Work area VW2 and IVW2 are three dimensional array respectively. These areas are used for solving least squares problems in order to compute column vector $\boldsymbol{m}_k$ of approximate inverse matrix $\boldsymbol{M}$. In general, column vector $\boldsymbol{m}_k$ is sparse vector and its density of nonzero elements is varied during computation. The least squares problems are defined corresponding to the formula of previous section 4). The residual vector $\boldsymbol{A}\boldsymbol{m}_k$ - $\boldsymbol{e}_k$ can be formulated only by nonzero elements of $\boldsymbol{m}_k$ and certain columns of $\boldsymbol{A}$ related with nonzero elements of $\boldsymbol{m}_k$. From such point of view, rectangular system which is constructed by nonzero elements is derived. You can specify LMMAX and LNMAX as maximum number of rectangular matrix and allocate array VW2 and IVW2. Actual number of rectangular matrix desired in this subroutine depend on characteristics of matrix A and value of parameters such as IPAR(2). Therefore you can try to call this subroutine by using suggested manner below. If no solution is met, it is recommended to try again by making parameters change.
LMMAX is a certain value related to the number of nonzero elements of matrix $\boldsymbol{A}$. Lets see k-th column of matrix $\boldsymbol{A}$, we defines the total number of nonzero elements in k-th column and another columns which are relatives of the nonzero elements of k-th column. You can specify the maximum number of the total number between columns. In general, it is adequate to specify as LMMAX=1000. In case that density of nonzero elements is rather high or relation between elements tend to be strong or certain columns have more nonzero elements than others, it is recommended to increase LMMAX.
LNMAX is a certain value proportional to the maximum number of nonzero elements between columns of matrix $\boldsymbol{A}$. The maximum number of nonzero is calculated by the formula below where nz_k is number of nonzero elements in the k-th column of matrix $\boldsymbol{A}$.

$$\max_k \left[\max(\ 1, nz_k \times IPAR\ (2)/100\ )\right]$$

You can specify LNMAX as this maximum number multiplied by 1.2.

After computation, this subroutine output the actual size in IPAR(13) and IPAR(14) corresponding to LMMAX and LNMAX respectively.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

$-\Delta \boldsymbol{u}+\boldsymbol{a}\nabla \boldsymbol{u}+\boldsymbol{u}=\boldsymbol{f}$

with zero boundary conditions on a cube and the coefficient $\boldsymbol{a}=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine INIT_MAT_DIAG. Then it is converted into the storage scheme in compressed storage. The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60)
      PARAMETER (NX=NORD,NY=NORD,NZ=NORD)
      PARAMETER (N=NX*NY*NZ)
      PARAMETER (K=N+1,NDIAG=7,L=4)
      PARAMETER (LMMAX=1000,LNMAX=200,NUMT=4)
      DIMENSION  NOFST(NDIAG),DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION  A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
     +           W(K*NDIAG),IW(2,K*NDIAG)
      DIMENSION  X(N),B(N),SOLEX(N),Y(N),IVW(N)
      DIMENSION  VW2(LMMAX,LNMAX+3,NUMT),IVW2(LMMAX,3,NUMT)
      DIMENSION  IPAR(20),RPAR(20)
      DIMENSION  NFRNZ(N+1),NFRNZM(N+1)
C
      REAL*8,ALLOCATABLE  :: AA(:),AM(:),VW1(:)
      INTEGER,ALLOCATABLE :: NCOL(:),JVWB(:),
     +                       NCOLM(:),IVW1(:)
C
      PRINT *,' *** SPARSE LINEAR EQUATIONS BY IDR METHOD',
     +        ' WITH PRECONDITIONING'
      PRINT *,' *** COMPRESSED ROW STORAGE.'
      PRINT *
      SOLEX(1:N)=1.0D0
      PRINT *,' *** EXPECTED SOLUTIONS'
      PRINT *,' X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *
      VA1 = 3.0D0
      VA2 = 1.0D0/3D0
      VA3 = 5.0D0
      VC = 1.0D0
      XL = 1.0D0
      YL = 1.0D0
      ZL = 1.0D0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     & ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
      DO I=1,NDIAG
C
```

```
      IF(NOFST(I).LT.0)THEN
         NBASE=-NOFST(I)
         LENGTH=N-NBASE
         DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
         NBASE=NOFST(I)
         LENGTH=N-NBASE
         DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
      DO J=1,N
         NTOPCFG=1
         DO I=NDIAG,1,-1
            IF(DIAG2(J,I).NE.0.0D0)THEN
               NCOLL=J-NOFST(I)
               A(NUMNZ)=DIAG2(J,I)
               NROW(NUMNZ)=NCOLL
               IF(NTOPCFG.EQ.1)THEN
                  NFCNZ(J)=NUMNZ
                  NTOPCFG=0
               ENDIF
               NUMNZ=NUMNZ+1
            ENDIF
         ENDDO
      ENDDO
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1
      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,SOLEX,B,W,IW,ICON)
      ERR1 = ERRNRM(SOLEX,X,N)
C
      X(1:N)=0.0D0
      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,X,Y,W,IW,ICON)
      ERR2 = ERRNRM(Y,B,N)
C
      ALLOCATE (AA(NNZ),NCOL(NNZ),AM(NNZ),NCOLM(NNZ)
     +         ,VW1(NNZ),IVW1(NNZ))
      ISW=1
      DO I=1,20
         IPAR(I)=0
         RPAR(I)=0.0D0
      END DO
      NWM=NNZ
      NZM=0
C
      CALL CONVGCR(A,N,NFCNZ,NROW,AA,NFRNZ,NCOL,IVW)
      CALL DM_VLSPAXCR2(AA,NNZ,NCOL,NFRNZ,N,B,ISW,X
     +    ,AM,NZM,NCOLM,NFRNZM,NWM,IPAR,RPAR
     +    ,VW1,IVW1,VW2,IVW2,LMMAX,LNMAX,NUMT,ICON)
C
      EPS=RPAR(2)
```

```
      ITMAX=2000
      ERR3 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,X,Y,W,IW,ICONT)
      ERR4 = ERRNRM(Y,B,N)
      PRINT *,' *** COMPUTED SOLUTIONS'
      PRINT *,' X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,' DM_VLSPAXCR2 ICON = ',ICON
      PRINT *
      PRINT *,' N        = ',N
      PRINT *,'       NX = ',NX
      PRINT *,'       NY = ',NY
      PRINT *,'       NZ = ',NZ
      PRINT *,' ITER MAX = ',ITMAX
      PRINT *,' ITER     = ',IPAR(7)
      PRINT *,' ICMAV    = ',IPAR(8)
      PRINT *
      PRINT *,' EPS      = ',RPAR(2)
      PRINT *
      PRINT *,' INITIAL ERROR          = ',ERR1
      PRINT *,' INITIAL RESIDUAL ERROR  = ',ERR2
      PRINT *,' CRITERIA RESIDUAL ERROR = ',ERR2*EPS
      PRINT *
      PRINT *,' ERROR                  = ',ERR3
      PRINT *,' RESIDUAL ERROR         = ',ERR4
      PRINT *
      PRINT *
      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
      WRITE(*,*)' ********** OK **********'
      ELSE
      WRITE(*,*)' ********** NG **********'
      ENDIF
      STOP
      END
C =======================================
C INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     & ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
         WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
         WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
      RETURN
      ENDIF
!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+ SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+ ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
C NDIAG CANNOT BE GREATER THAN 7
      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)
!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
         D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO
      NXY = NX*NY
C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
         OFFSET(L) = -NXY
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
         OFFSET(L) = -NX
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
         OFFSET(L) = -1
         L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
         OFFSET(L) = 1
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
         OFFSET(L) = NX
         L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
         OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE
C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
         JS = J
C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
         K0 = (JS-1)/NXY+1
         IF (K0 .GT. NZ) THEN
            PRINT*,'ERROR; K0.GH.NZ '
            GOTO 100
         ENDIF
         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
  100 CONTINUE
!$OMP ENDDO
!$OMP END PARALLEL
      RETURN
      END
C =====================================
C     ABSOLUTE ERROR : | X1 - X2 |
C =====================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
      S = 0D0
      DO 100 I = 1,LEN
         SS = X1(I) - X2(I)
         S = S + SS * SS
  100 CONTINUE
      ERRNRM = SQRT( S )
      RETURN
      END
C=============================================================
```

```
C     MODE CONV UNSYM MATRIX FROM COMPRESSED COLUMN TO ROW.
C============================================================
      SUBROUTINE CONVGCR(AC,N,IC,JC,AR,IR,JR,IW)
      IMPLICIT   REAL*8 (A-H,O-Z)
      DIMENSION  AC(*),IC(N+1),JC(*),AR(*),IR(N+1),JR(*),IW(N)
      NZ=IC(N+1)-1
      DO I=1,N+1
         IR(I)=0
      END DO
      DO J=1,NZ
         IR(JC(J)+1)=IR(JC(J)+1)+1
      END DO
      IR(1)=1
      DO I=2,N+1
         IR(I)=IR(I)+IR(I-1)
      END DO
      DO I=1,N
         IW(I)=IR(I)
      END DO
      ICOL=1
      DO J=1,NZ
         IF(J.EQ.IC(ICOL+1)) ICOL=ICOL+1
         JR(IW(JC(J)))=ICOL
         AR(IW(JC(J)))=AC(J)
         IW(JC(J))=IW(JC(J))+1
      END DO
      RETURN
      END
```

(4) Method

This subroutine solves a system of linear equations with unsymmetric real sparse matrices as coefficient matrices using Induced Dimension Reduction method with stabilization, *IDRstab(s,l)* , with preconditioning by sparse approximate inverse.

a. Approximate inverse matrix

In general, the convergence of iteration method is not guaranteed or may be extremely slow. Preconditioning method makes the system to more tractable form and reduces total iteration counts. On such point of view, incomplete decomposition method, e.g. ILU method, is well known. The ILU method is very simple algorithm and having very effective performance for well conditioned matrices. However, it tends to be poor for parallel processing because of its recurrence nature in triangular solvers which arise frequently during iterations.

This subroutine employs a preconditioning on approximate inverse matrix method which has more suitable characteristics for parallel processing rather than incomplete decomposition method. This subroutine applies matrix $\boldsymbol{M}$ to the right preconditioned system as follows;

$\boldsymbol{AMy = b,\ x = My}$.

Where $\boldsymbol{M}$ is an approximate inverse matrix. In order to compute matrix $\boldsymbol{M}$, this subroutine applies the Frobenius norm to minimize $|| \boldsymbol{AM - I} ||$. This choice leads to inherent parallelism, that is, the columns $\boldsymbol{m}_k$ of $\boldsymbol{M}$ can be computed independently of one another.

Since

$\|\boldsymbol{AM} - \boldsymbol{I}\|_F^2 = \sum_{k=1}^{n} \|(\boldsymbol{AM} - \boldsymbol{I})\boldsymbol{e}_k\|_2^2$,

n set of independent least squares problems can be derived as follows;

$\min_{m_k} \|\boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k\|_2, k = 1,...,n$.

The unit vector is employed for initial value of $\boldsymbol{m}_k$ as a default. This subroutine solves the least squares minimization problem by using QR method.
Residual vector based on the minimum solution above may lead candidates of new nonzeros in next step $\boldsymbol{m}_k$. This subroutine selects new indices automatically from candidates in terms of the most profitable one which minimizes coming residual vector.
The iterate $\boldsymbol{m}_k$ of inverse matrix $\boldsymbol{M}$ is accepted as a minimum solution if it satisfies convergence criteria by RPAR(1), that is

$\|\boldsymbol{A}\boldsymbol{m}_k - \boldsymbol{e}_k\|_2 \le eps$,

or if number of nonzero elements in the column reaches upper limit based on IPAR(2).

b. Induced Dimension Reduction method *IDRstab(s,l)*

Induced Dimension Reduction method is the one of the Krylov subspace method. This subroutine employs *IDRstab(s,l)* method which is revised by exploiting *BICGstab(l)* strategies to original *IDR(s)* method. Where the parameter *s* is the order of shadow residual and *l* is the order of acceleration polynomial. One of the key feature of *IDRstab(s,l)* method is that you are able to specify higher order of acceleration polynomial compared with original *IDR(s)* method.
You can select arbitrary parameters *s* and *l*. When *l*=1, this subroutine select another method, *BIDR(s)* method, which is stabilized further more by taking bi-orthogonalization technique when *s* is large case.

c. Convergence test

The iterate $\boldsymbol{x}_k$ is accepted as a solution of the system if the residual satisfies

$\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|_2 \le epst\|\boldsymbol{b}\|_2$.

Where *epst* is a convergence criteria specified in RPAR(2). Default value of *epst* is 10^{-8}.
The final relative residual norm

$\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|_2 / \|\boldsymbol{b}\|_2$

is stored in RPAR(3), even if in the case that the residual does not satisfy convergence test. The residual vector $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$ is computed by using recurrence in the iteration formula.

For details of the algorithms, see [29], [31] and [73] in Appendix A, "References".

DM_VLSX

A system of linear equations with symmetric positive definite matrices (blocked modified Cholesky decomposition method)
CALL DM_VLSX(A, K,N, B,EPSZ,ISW,ICON)

(1) Function

This subroutine decomposes the coefficient matrix ***A*** of a system of a real coefficient linear equation (1. 1) as shown in (1. 2) using the blocked modified Cholesky decomposition of outer products. It then solves the system of equations, where ***A*** is a symmetric positive definite matrix ($n \times n$), ***b*** is an n-dimensional real constant vector, ***x*** is an n-dimensional solution vector, ***L*** is a unit lower triangular matrix, and ***D*** is a diagonal matrix. It is assumed that $n \geq 1$.

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1.1}$$

$$\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{T}} \tag{1.2}$$

(2) Parameter

A ..... Input. Coefficient matrix ***A***.

The lower triangular part $\{a_{ij} \mid i \geq j\}$ of ***A*** is stored in the lower triangular part $\{A(i,j) \mid i \geq j\}$ of A(1:N,1:N) for input.

Output. Decomposed matrix.

After the first set of equations has been solved, the lower triangular part of A(i,j) contains l_{ij} ($i > j$) and reciprocals of d_{ii} ($i = j$). The upper triangular part $\{A(i,j) \mid i < j\}$ is altered.

(See Figure DM_VLSX-1.)

This is a double precision real two-dimensional array A(K,N).

K ..... Input. The size of the first dimension of array A.

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

B ..... Input. Constant vector ***b***

Output. Solution vector ***x***.

A double precision real one-dimensional array of size n.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed.

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

ISW ..... Input. Control information.

When solving several sets of equations that have an identical coefficient matrix, specify as follows.

Specify ISW = 1 for the first set of equations.

Specify ISW = 2 for the second and subsequent sets of equations.

When specifying ISW = 2, change only the value of array B into a new constant vector ***b***. Do not change any other parameters.

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

ICON ..... Output. Condition code.

(See Table DM_VLSX-1.)

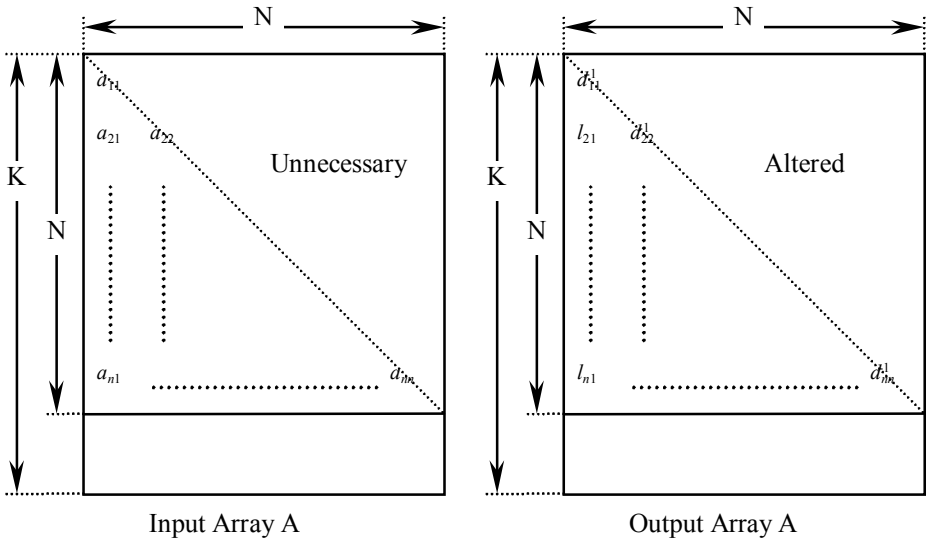


Figure DM_VLSX-1 Storing the data for the Cholesky decomposition method

The diagonal elements and lower triangular part (a_{ij}) of the LDL^T-decomposed positive definite matrix are stored in array A(i,j), $i = j, ... n, j = 1, ..., n$.

After LDL^T decomposition, the matrix $\boldsymbol{D}^{-1}$ is stored in the diagonal part and $\boldsymbol{L}$ (except for the diagonal elements) are stored in the lower triangular part respectively.

Table DM_VLSX-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	The pivot becomes negative. The coefficient matrix is not positive definite.	Processing is continued
20000	The pivot became relatively zero. The coefficient matrix A may be singular.	Processing is discontinued.
30000	N < 1, EPSZ < 0, K < N, or ISW ≠ 1, 2.	

(3) Comments on use

a. Notes

1) If a value is set for the judgment of relative zero, it has the following meaning: if the absolute value of the selected pivot is less than EPSZ during LDL^T decomposition by the modified Cholesky decomposition, the pivot is assumed to be relatively zero and decomposition is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$.
When the computation is to be continued even if the pivot becomes small, assign the minimum value to EPSZ. In this case, however the result is not assured.

2) When several sets of linear equations having an identical coefficient matrix are solved, the value of ISW should be 2 from the second time on. This reduces the execution time because LDL^T decomposition for coefficient matrix $\boldsymbol{A}$ is bypassed.

3) If the pivotal value becomes negative during decomposition, the coefficient matrix is no longer positive definite. Processing is continued with ICON = 10000. However, the accuracy of the result may not be maintained because no pivoting is performed.

4) After the calculation has been completed, the determinant of the coefficient matrix is computed by multiplying all the *n* diagonal elements of the array A and taking the reciprocal of the result.

5) This subroutine calls DM_VSLDL and DM_VLDLX internally. Therefore, instead of calling this function in a parallel region with specifying the number of threads by run-time library OMP_SET_NUM_THREADS(), call DM_VSLDL and DM_VLDLX directly with specifying the number of threads with OMP_SET_NUM_THREADS() just before the each of them.

b. Example

A system of linear equations with a 4000 × 4000 coefficient matrix is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (K=4001,N=4000)
      REAL*8    A(K,N),B(N)
C
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(A,B)
!$OMP DO
      DO J=1,N
      DO I=J,N
      A(I,J)=MIN(I,J)
      ENDDO
      ENDDO
!$OMP END DO

!$OMP DO
      DO I=1,N
      B(I)=I*(I+1)/2+I*(N-I)
      ENDDO
!$OMP END DO

!$OMP END PARALLEL

      ISW=1
      CALL DM_VLSX(A,K,N,B,1.D-13,ISW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 100
      WRITE(6,620) (B(I),I=1,10)
C
      S=1.0D0
```

```
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A)
!$OMP+          REDUCTION(*:S)
      DO I=1,N
      S=S*A(I,I)
      ENDDO
!$OMP END PARALLEL DO

      DET=1.0D0
      DET=1.D0/DET
      WRITE(6,630) DET
  100 STOP
  600 FORMAT(1H1/10X,6HORDER=,I5)
  610 FORMAT(1H0,10X,5HICON=,I5)
  620 FORMAT(11X,15HSOLUTION VECTOR
     */(10X,3D23.16))
  630 FORMAT(1H0,10X
     *,34HDETERMINANT OF COEFFICIENT MATRIX=
     *,D23.16)
      END
```

(4) Method

See [30], [54], and [70] in Appendix A, "References," for details of the blocked modified Cholesky decomposition method of outer product type.

DM_VLUX

A system of linear equations with LU-decomposed real matrices
CALL DM_VLUX(B,FA,KFA,N,IP,ICON)

(1) Function

This subroutine solves a system of linear equations having LU-decomposed real coefficient matrices.

$$\boldsymbol{LUx} = \boldsymbol{Pb}$$

where, $\boldsymbol{L}$ and $\boldsymbol{U}$ are respectively a unit lower triangular matrix and a unit upper triangular $n \times n$ matrix, $\boldsymbol{P}$ is a permutation matrix (interchanging rows of the coefficient matrix for partial pivoting in LU-decomposition), $\boldsymbol{b}$ is an n-dimensional real constant vector, and $\boldsymbol{x}$ is an n-dimensional solution vector ($n \geq 1$).

(2) Parameters

B ..... Input. Constant vector $\boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

A double precision one-dimensional array of size n.

FA ..... Input. Matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are stored into FA(1:N,1:N).

See Figure DM_VLUX-1.

This is a double precision real two-dimensional array FA(KFA,N).

KFA ..... Input. The size of the first dimension of the array for storage FA (≥ N).

N ..... Input. Order n of matrices $\boldsymbol{L}$ and $\boldsymbol{U}$.

IP ..... Input. The transposition vector recording the process of row interchange in partial pivoting.

A one-dimensional array of size n.

(See note 2) in (3), "Comments on use" for subroutine DM_VALU.)

ICON ..... Output. Condition code.

See Table DM_VLUX-1.

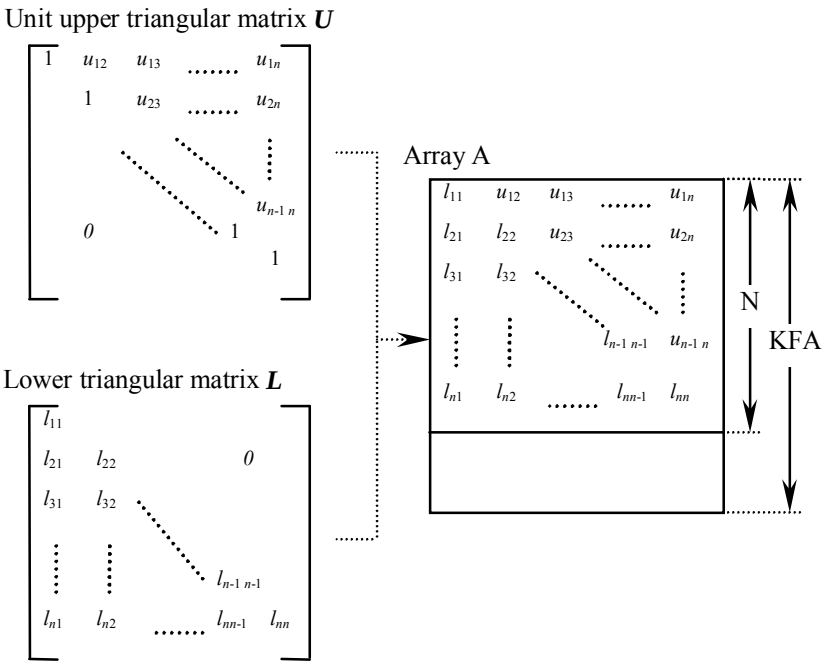


Figure DM_VLUX-1 Storing L and U in array FA

After LU decomposition is executed, the upper triangular part of ***U*** (except for the diagonal elements) and the lower part of ***L*** are stored in array FA(1:N,1:N).

Table DM_VLUX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The coefficient matrix is singular.	Processing is discontinued.
30000	KFA < N, N < 1. The contents of IP are incorrect, ISW ≠ 1, 2.	

(3) Comments on use

a. Notes

1) Although a system of linear equations with a coefficient matrix can be solved by calling this subroutine after calling subroutine DM_VALU, the subroutine DM_VLAX should be usually used to solve a system of linear equations in one step.

b. Example

A system of linear equations is solved by LU-decomposing the coefficient 4000 × 4000 matrix.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (IPN=4)
      DIMENSION  A(4001,4000)
      DIMENSION  B(4000),IP(4000)
C
      N=4000

!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(A,B,N)
!$OMP DO
      DO J=1,N
      DO I=1,N
      A(I,J)=MIN(I,J)
      ENDDO
      ENDDO
!$OMP END DO

!$OMP DO
      DO I=1,N
      B(I)=I*(I+1)/2+I*(N-I)
      ENDDO
!$OMP END DO
!$OMP END PARALLEL
C
      KFA=4001
      CALL  DM_VALU(A,KFA,N,0.0D0,IP,IS,ICON)
      WRITE(6,610)ICON
      IF(ICON.GE.20000)STOP
      CALL  DM_VLUX(B,A,KFA,N,IP,ICON)
      WRITE(6,620)ICON
      WRITE(6,630)(I,B(I),I=1,10)

  610 FORMAT(1H0,10X,26HCONDITION CODE (DM_VALU) =,I5)
  620 FORMAT(1H0,10X,26HCONDITION CODE (DM_VLUX) =,I5)
  630 FORMAT(1H0,10X,17HSOLUTION VECTOR =,
     */(10X,5(1H(,I3,1H),D23.16)))
      END
```

DM_VMGGM

Matrix multiplication (real matrix)
CALL DM_VMGGM(A,KA,B,KB,C,KC,M,N,L,ICON)

(1) Function

This subroutine obtains product $\boldsymbol{C}$ by multiplying a real matrix $\boldsymbol{A}$ ($m \times n$) by a real matrix $\boldsymbol{B}$ ($n \times l$).

$$\boldsymbol{C} = \boldsymbol{AB}$$

where $\boldsymbol{C}$ is a real matrix ($m \times l$), where $m, n, l \geq 1$.

(2) Parameters

A ..... Input. Matrix $\boldsymbol{A}$.
Data must be stored in A(1:M,1:N).
A double precision real two-dimensional array A(KA,N).

KA ..... Input. The size of the first dimension of the arrays A (≥ M).

B ..... Input. Matrix $\boldsymbol{B}$.
The data must be stored in B(1:N,1:L).
The double precision real two-dimensional array B(KB,L).

KB ..... Input. The size of the first dimension of array B (≥ N).

C ..... Output. Matrix $\boldsymbol{C}$.
The data is stored in C(1:M,1:L).
The double precision real two-dimensional array C(KC,L).

KC ..... Input. The size of the fist dimension of array C, (≥ M).

M ..... Input. The number of rows m in matrices $\boldsymbol{A}$ and $\boldsymbol{C}$.

N ..... Input. The number of columns n in matrix $\boldsymbol{A}$ and number of rows n in matrix $\boldsymbol{B}$.

L ..... Input. The number of columns l in matrices $\boldsymbol{B}$ and $\boldsymbol{C}$.

ICON ..... Output. Condition code.
See Table DM_VMGGM-1.

Table DM_VMGGM-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	M < 1, N < 1, L < 1, KA < M, KB < N, KC < M.	Processing is discontinued.

(3) Comments on use

a. Example

A product is obtained for real matrices ***A*** and ***B***.

Subroutine PGM in this example is for printing a real matrix.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     ** EXAMPLE **
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (KK=4001,M=4000,N=M,L=M)
      PARAMETER (KA=KK,KB=KK,KC=KK)
      REAL*8    A(KA,N),B(KB,L),C(KC,L)

C
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(A,B)
!$OMP DO
      DO J=1,M
      DO I=1,N
      IF(J.GT.I)THEN
      A(I,J)=0.0d0
      ELSE
      A(I,J)=1.0d0
      ENDIF
      ENDDO
      ENDDO
!$OMP END DO

!$OMP DO
      DO J=1,M
      DO I=1,N
      IF(J.GE.I)THEN
      B(I,J)=1.0d0
      ELSE
      B(I,J)=0.0d0
      ENDIF
      ENDDO
      ENDDO
!$OMP END DO
!$OMP END PARALLEL

      CALL DM_VMGGM(A,KA,B,KB,C,KC,M,N,L,ICON)
      IF(ICON.NE.0) GOTO 10
      CALL PGM(A,KA,N)
      CALL PGM(B,KB,L)
      CALL PGM(C,KC,L)
      GOTO 10

  150 FORMAT(1H1///10X,27H** MATRIX MULTIPLICATION **)
  10  STOP
      END
```

```
C     ** MATRIX PRINT(REAL NON-SYMMETRIC) **
      SUBROUTINE PGM(A,KA,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(KA,N)
      DO 10 I=1,5
      WRITE(6,610) I,(J,A(I,J),J=1,5)
   10 CONTINUE
      RETURN
  610 FORMAT(/5X,I3,3(4X,I3,D23.16),(/8X,3(4X,I3,D23.16)))
      END
```

(4) Method

This subroutine uses the method of blocked matrix multiplication. For details, see [30] in Appendix A, "References."

DM_VMINV

Inverse of real matrix (blocked Gauss-Jordan method)
CALL DM_VMINV(A,K,N,EPSZ,ICON)

(1) Function

This subroutine obtains the inverse $\boldsymbol{A}^{-1}$ of the $n \times n$ non-singular real matrix $\boldsymbol{A}$ using the Gauss-Jordan method.

(2) Parameters

A ..... Input. Matrix $\boldsymbol{A}$ is stored in A(1:N,1:N).

Output. Matrix $\boldsymbol{A}^{-1}$ is stored in A(1:N,1:N).

The double precision real two-dimensional array A(K,N).

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

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

EPSZ ..... Input. Judgment of relative zero of the pivot. (≥ 0.0)

When EPSZ is 0.0, the standard value is assumed.

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

ICON ..... Output. Condition code.

See Table DM_VMINV-1.

Table DM_VMINV-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	All row elements in matrix $\boldsymbol{A}$ are zero or the pivot becomes a relatively zero. Matrix $\boldsymbol{A}$ may be singular.	Processing is discontinued.
30000	N < 1, K < N, or EPSZ < 0.0.	

(3) Comments on use

a. Notes

1) When the pivot element selected by partial pivoting is 0.0 or the absolute value is less than EPSZ, it is assumed to be relatively zero. In this case, processing is discontinued with ICON=20000. When unit round off is u, the standard value of EPSZ is $16u$. If the minimum value is assigned to EPSZ, processing is continued, but the result is not assured.

b. Example

The inverse of a matrix is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4

when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N=2000,K=N+1)
C
      REAL*8    A(K,N),AS(K,N)
C

      C=SQRT(2.0D0/DBLE(1+N))
      T=DATAN(1.0D0)*4.0D0/(1+N)
C
      DO 100 J=1,N
      DO 100 I=1,N
      A(I,J)=C*SIN(T*I*J)
      AS(I,J)=A(I,J)
 100  CONTINUE
C
      EPSZ=0.0D0
      CALL  DM_VMINV(A,K,N,EPSZ,ICON)
      PRINT*,'ICON=',ICON
C
      TMP=0.0D0
      DO I=1,N
      DO J=1,N
      TMP2=DABS(A(I,J)-AS(I,J))
      IF(TMP2.GT.TMP)TMP=TMP2
      ENDDO
      ENDDO
      PRINT*,'ORDER=',N,' ; ERROR = ',TMP
C
      STOP
      END
```

(4) Method

This subroutine solves an inverse of matrix using the blocked Gauss-Jordan method (see [30] in Appendix A, “References.”).

DM_VMLBIFE

System of linear equations with sparse matrices (Multilevel iteration method based on incomplete block factorization, ELLPACK format storage method)
CALL DM_VMLBIFE (A, K, IWIDT, N, ICOL, B, ISW, IGUSS, INFO, INFOEP, EPSOT, EPSIN, EPSEP, X, W, NW, IW, NIW, ICON)

(1) Function

This subroutine solves, using the iterative method, a system of linear equations with sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the ELLPACK format storage method. Vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ are n-dimensional vectors.

The solution method is ORTHOMIN if $\boldsymbol{A}$ is symmetric and GMRES if $\boldsymbol{A}$ is non-symmetric. The iteration (called outer iteration) is preconditioned by the multilevel incomplete block factorizations and stable. The iteration procedure is preconditioned by repeated elimination of certain sets of unknowns. The elimination procedure uses approximative inverses of the sub-matrices produced by the sets of eliminated unknowns. The elimination procedure is repeated until on the so-called coarsest level a smaller linear system is produced. For every step of the outer iteration this linear system is solved iteratively (called inner iteration).

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A(1:N,1:IWIDT).

Two-dimensional array A(K,IWIDT)

For an explanation of the ELLPACK format storage method, see Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the SSL II Extended Capability User's Guide II.

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

IWIDT ..... Input. Maximum number of row-vector-direction nonzero elements of coefficient matrix $\boldsymbol{A}$. Size of second-dimension of A and ICOL.

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

ICOL ..... Input. Column index used in ELLPACK format. Used to indicate to which column vector the corresponding element of A belongs.

Two-dimensional array ICOL(K,IWIDT)

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N) .

ISW ..... Input. Control information.

ISW=1 Initial calling.

ISW=2 Second or subsequent calling.

The values of A, IW and W must not be changed if the routine is called again with ISW=2.

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

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

INFO ..... Input. The control information of the iteration.

One dimensional array of INFO(14).

For example, for symmetric coefficient matrix ***A***, INFO is set as follows;

INFO(1)=10

INFO(2)=NTHRD×100

INFO(3)=0

INFO(5)=1

INFO(6)=2000

INFO(10)=1

INFO(11)=1000

For example, for unsymmetric coefficient matrix A, INFO is set as follows;

INFO(1)=10

INFO(2)=NTHRD×100

INFO(3)=0

INFO(5)=2

INFO(6)=2000

INFO(7)=5

INFO(8)=20

INFO(10)=2

INFO(11)=1000

INFO(12)=10

INFO(13)=0

Where NTHRD is the number of threads which are executed in parallel.

INFO(1)=MAXLVL

Input. Maximal number of levels in the algebraic multilevel iteration method.

MAXLVL<1 No preconditioner is applied.

MAXLVL>0 The coarser level than the specified depth is not used.

(See 5),8) in a, “Notes,” in (3), “Comments on use.”)

INFO(2)=MINUK

Input. Minimal number of unknowns for the smallest linear system in the deepest level in the inner iteration. It is recommendable to set MINUK very larger than the number of threads NTHRD and very smaller than N. For example, 100×NTHRD.

INFO(3)=NORM

Input. The type of normalization.

NORM<1 The matrix is normalized from the right and the left by the inverse of the square root of the main diagonal of A. This effects that the main diagonal of the normalized matrix A is equal to one and the matrix is symmetric if A is symmetric.

It is recommendable to use symmetrical normalization. However, in some cases the non-symmetrical normalization can produce faster convergence. Criterion value for judgment of convergency.

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

NORM≥1 The matrix is normalized from the left by the inverse of the absolute row sums of A multiplied with the sign of the main diagonal element. In general the normalized matrix will be non-symmetric even if the matrix A is symmetric.

(See 4) in a, “Notes,” in (3), “Comments on use.”)

INFO(4)

Output. Number of levels.

INFO(5)=METHOT

Input. The iterative method used in the outer iteration.

METHOT=1 Preconditioned ORTHOMIN is used. It should be used if the matrix ***A*** is symmetric and a symmetrical normalization is used.

METHOT≠1 Restarted and truncated GMRES is used. It should be used if the matrix ***A*** is non-symmetric or a non-symmetrical normalization is used.

INFO(6)=ITMXOT

Input. The maximal number of iteration steps in the outer iteration, for example 2000. If the maximum iteration number of outer iteration is reached the processing is terminated and the returned solution does not fulfill the stopping criterion.

INFO(7)=NRESOT

Input. The number of residuals in the orthogonalization procedure of the outer iteration, i.e. truncation after NRESOT residuals. For example , 5. Only used if GMRES is applied.

(See 4) in a, “Notes,” in (3), “Comments on use.”)

INFO(8)=NRSTOT

Input. Input. After NRSTOT iteration steps the outer iteration is restarted. For example , 20. NRSTOT ≥ NRESOT = INFO(7). If it is NRSTOT<1 there is no restart. Only used if GMRES is applied.

(See 4) in a, “Notes,” in (3), “Comments on use.”)

INFO(9)=ITEROT

Output. The number of iteration steps in the outer iteration procedure.

INFO(10)=METHIN

Input. The iterative method used in the inner iteration.

METHIN=1 Preconditioned ORTHOMIN is used. It should be used if the matrix ***A*** is symmetric and a symmetrical normalization is used.

METHIN≠1 Restarted and truncated GMRES is used. It should be used if the matrix ***A*** is non-symmetric or a non-symmetrical normalization is used.

INFO(11)=ITMXIN

Input. The maximal number of iteration steps in the inner iteration, for example 1000.

If ITMXIN is reached the processing is continued on the outer iteration.

INFO(12)=NRESIN

Input. The number of residuals in the orthogonalization procedure of the inner iteration, ie. truncation after NRESIN residuals. For example , 10. Only used if GMRES is applied.

(See 5) in a, “Notes,” in (3), “Comments on use.”)

INFO(13)=NRSTIN

Input. After NRSTIN iteration steps the inner iteration is restarted. NRSTIN ≥ NRESIN = INFO(12).

Only used if GMRES is applied. If it is NRSTIN<1 there is no restart.

(See 5) in a, “Notes,” in (3), “Comments on use.”)

INFO(14)

Output. The average number of the inner iteration.

INFOEP..... Input. The control information for the block matrix of the removed unknowns and the reduced matrix. One dimensional array of INFOEP(3).

For example, INFOEP is set as follows to specify the method for approximating the inverse matrix of a matrix block, which is used for calculating the Schur complement in each level:

(See 7) in a, “Notes,” in (3), “Comments on use.”)

1) in case of approximating the inverse matrix with a diagonal matrix

INFOEP(1)=1

INFOEP(2)=5

INFOEP(3)=2×NROW

where, NROW indicates the representative number of nonzero entries per row in the coefficient matrix ***A***.

2) in case of seeking an approximative inverse matrix with an iterative method

INFOEP(1)=NROW

INFOEP(2)=5

INFOEP(3)=2×NROW

INFOEP(1)=MAXNCV

Input. Maximal number of nonzero entries per row in the approximative inverse of the eliminated matrix block. Typically it is set MAXNCV=1 or MAXNCV=MAXNC. Notice that MAXNCV=1 effects that the matrix block is approximated by its main diagonal.

INFOEP(2)=MAXITV

Input. Maximal number of approximative inverse steps. MAXITV specifies the maximal number of iteration steps which are allowed to calculate the approximative inverse matrix with accuracy TAUV. If the number of iteration steps reaches MAXITV the procedure is terminated. Notice that in any case the approximation procedure will need less than $\frac{\log(\text{TAUV})}{\log(\text{LAMBDA})}$ steps. If MAXITV≤1 the matrix block is approximated by its main diagonal.

INFOEP(3)=MAXNC

Input. MAXNC limits the entries remaining in the reduced matrix as Schur complement in block decomposition. If MAXNC<2 small entries of the reduces system less than TAU are dropped. If MAXNC > 1 the number of non-zero entries per row is limited by MAXNC. In this case only the MAXNC largest entries in every row are kept. Other entries are dropped even if they are greater than TAU.

(See 8) in a, "Notes," in (3), "Comments on use.")

EPSOT ..... Input. The desired accuracy for the solution. The outer iteration is stopped in the *k*-th iteration step if the normalized $\hat{r}_k = \hat{A}x_k - \hat{b}_k$ residual of the current approximation $\boldsymbol{x_k}$ satisfies the condition

$$\|\hat{r}_k\| \le \text{EPSOT}\|\hat{b}\|$$

where $\|y\|^2 = y^{\mathrm{T}}y$ denotes the Euclidean norm $\hat{\boldsymbol{A}}$ and $\hat{b}$ and are the coefficient matrix and the right hand side of the normalized linear system.

EPSIN ..... Input. The tolerance for the inner iteration. Normally 10^{-3} is optimal.

EPSEP..... Input. The control information for the approximation of the reduced system and the inverse of the eliminated matrix block. One dimensional array of EPSEP(4).

For example, set as follows:

EPSEP(1)=1.0D-2

EPSEP(2)=1.0D-2

EPSEP(3)=0.2

EPSEP(4)=1.0D-3

EPSEP(1)=TAU

Input. The dropping tolerance. In the reduced systems as Schur complement in block decomposition, entries less than TAU are dropped to keep the sparsity. As larger TAU as faster is the iterative solver on the lowest level. But on the other hand there is a larger loss of information, which deteriorates the quality of the preconditioner. It has to be $0 \le \text{TAU} < 1$.

EPSEP(2)=TAUV

Input. The tolerance of the approximative inverse. A small value for TAUV will increase the time for the elimination procedure but improve the quality of the preconditioner. Normally EPSIN=TAUV is optimal.

EPSEP(3)=LAMBDA

Input. Diagonal threshold for the block matrix. The entries in the block matrix of the removed unknowns are selected such that the absolute sum per row is less than LAMBDA times the main diagonal entry. A larger value for LAMBDA will produce a smaller set of removed unknowns but will increase the costs for the calculation of the approximative inverse of the block. Recommendation: LAMBDA=0.2. It should be TAUV ≤ LAMBDA < 1 or LAMDA=0.

EPSEP(4)=RHO

Input. Unknowns with small entries in their main diagonal are not considered in the elimination procedure. A main diagonal entry is small if it is smaller than RHO times the absolute sum of the row entries.
Recommendation: RHO=1.0D-3. It has to be 0<RHO<1.

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X.

One-dimensional array X(N).

W ..... Work area. One-dimensional array W(NW) .

NW ..... Input. Size of the work array W. A rough upper bound is given by

NW ≤ max(2×MAXLVL+2,10)×NBAND×MAXT+(4×NC+6)×(N+MAXT)
+max(2×NC×(N+MAXT),LR0(N)
+max(LR0(Nf)+N+MAXT,6×(N+MAXT))).

In this formula MAXLVL denotes the number of levels of the incomplete block factorization, and NBAND denotes the bandwidth of the matrix, NC an upper bound for the number of non-zero entries per row (typically NC=MAXNC), and Nf the number of unknowns in the final level
(typically $Nf=2^{-MAXLVL}\times(N+MAXT)$) and MAXT is the maximum number of threads which are created in this routine.
Moreover it is

$$LR0(N) = \begin{cases} 4\times N & : \text{ ORTHOMIN method} \\ (2\times NRES+1)\times N & : \text{ GMRES method} \end{cases},$$

where NRES denotes the number of residuals used in GMRES. Normally the term LR0(Nf) can be neglected.

IW ..... Work area. One-dimensional array IW(NIW).

NIW ..... Input. Size of the work array IW. A rough upper bound is given by

NIW ≤ ((4×MAXLVL+10)×MAXT+12×NBAND)+3400)×MAXT
+ (6×NC+11)×(N+MAXT)
In this formula MAXLVL denotes the number of levels of the incomplete block factorization, and NBAND denotes the bandwidth of the matrix, NC an upper bound for the number of non-zero entries per row (typically NC=MAXNC), and MAXT is the maximum number of threads which are created in this routine.

ICON ..... Output. Condition code.

See Table DM_VMLBIFE-1.

Table DM_VMLBIFE-1 Condition codes

Code	Meaning	Processing
0	No error	–
10100	Inverse matrix could not be calculated with sufficient accuracy.	Processing is continued.
10800	Curable break down in GMRES.	
20001	Stopping criterion could not be reached within the given number of iteration steps.	Processing is discontinued. The approximate value obtained is output in array X, but the precision is not assured.
20003	Non-curable break down in GMRES.	Processing is discontinued.
20005	Non-curable break down in ORTHOMIN by $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{p}=0$ with $\boldsymbol{p}\neq 0$.	
20006	Non-curable break down in ORTHOMIN by $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{r}=0$.	
30000	N<1,N>K,IWIDT<1, ISW<1, ISW>2.	
30103	Incorrect entry in column list ICOL.	
30105	Main diagonal is missed.	
30210	Matrix condensation fails by non-positive value.	
30213	There is a row with only non-zero entries.	
30310	Too small integer work array.	
30320	Too small real work array.	

(3) Comments on use

a. Notes

1) When multiple linear equations with the same coefficient matrix but different right hand side vectors are solved set ISW=1 in the first call and ISW=2 in the second and all subsequent calls. Then the coarse level matrices assembled in the first call are reused.

2) Normally it is sufficient to set NC=IWIDT×1.5 in the formulas for the length for the work arrays. In general, if the work arrays are too small it is recommendable to increase NC. If the given matrix has a very large bandwidth it is recommendable to increase NBAND first.

3) It is always recommendable to use ORTHOMIN if possible. This requires that the matrix is symmetric. As this routine removes easily computable unknowns from the matrix before the iteration starts it can happen that the actual iteration matrix is symmetric even if the given matrix is not. Therefore it is recommendable to try ORTHOMIN with symmetrical normalization first if there is a chance that the iteration matrix is symmetric.

4) If the matrix is non-symmetric it is recommendable to use the non-symmetric normalization together with GMRES. Normally it is sufficient to truncate after NRESOT=5 residuals and to restart after 20 steps in the outer iteration. In the inner iteration it can be necessary to select a higher value for the truncation NRESIN and to restart after a larger number of iteration steps or even to forbid a restart. If NRESIN is increased it can happen that more real work space is required. Then it is necessary to increase NRES in the formula for the length workspace NW but, NRES can be set to a smaller value than NRESOT. In general the convergence of GMRES method becomes better as NRESIN and NRESOT are set to larger. But it requires longer computation time and larger amount of memory.

5) The elimination of unknowns is stopped if one of the following conditions is fulfilled:

   - the number of level is greater or equal MAXLVL
   - the coefficient matrix of the final level is a diagonal matrix
   - the number of eliminated unknowns is less than 10% of the number of unknowns in the final level.

6) When setting LAMBDA=0, RHO=0.99, TAU=0, MAXNC=IWIDT the routine is (similar to) the classical ILUM preconditioner with wavefront ordering. (See [65] in Appendix A, "References.") For LAMBDA=0, RHO<1, TAU>0 and MAXNC>>IWIDT the routine is the ILUM preconditioner with threshold. (See [64] in Appendix A, "References.")

7) It is emphasized that not every setting of the parameters produces necessarily an efficient preconditioner. So it can be necessary to test some values for the parameters till an optimal selection has been found.

8) The preconditioner bases on nested incomplete block factorizations using the Schur complement. The matrix A_n, n=1,...,MAXLVL−1 in each level can be blocked as follows choosing the appropriate sets of eliminated unknowns:

$$A_n = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}.$$

And define a matrix $S = A_{22} - A_{21}\, A_{11}^{-1}\, A_{12}$, which is called Schur complement. A_n can be factorized as follows:

$$A_n = \begin{bmatrix} A_{11} & 0 \\ A_{21} & I \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & S \end{bmatrix}.$$

The matrix $A_{n+1}$of next level n+1 can be regarded as a Schur complement matrix with approximating the A_{11}^{-1}. These incomplete factorization are used for preconditioning in this routine.

b. Example

The partial differential equation

$$-\left(\frac{\partial^2 u}{\partial^2 x_1}+\frac{\partial^2 u}{\partial^2 x_2}+\frac{\partial^2 u}{\partial^2 x_3}\right)+t\left((x_2-x_3)\frac{\partial u}{\partial x_1}+(x_3-x_1)\frac{\partial u}{\partial x_2}+(x_1-x_2)\frac{\partial u}{\partial x_3}\right)=f$$

is solved on the domain $[0,1]^2$. Dirichlet boundary condition u=0 is imposed and the value of t is set to 1.0.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE
      INTEGER   MAXT,N1,N2,N3,KA,NA,L1,L2,L3,LGRW,LGIW,
     &          NLBMAX,MAXNC

      PARAMETER(MAXT=2,N1=39,N2=N1,N3=N1,
     &    L1=N1,L2=N2,L3=N3,
     &    KA=N1*N2*N3,NA=7,NLBMAX=N1*N2,
     &    MAXNC=11,
     &    LGRW=(KA+MAXT)*(6*MAXNC+11)+(85*NLBMAX+100)*MAXT,
     &    LGIW=(KA+MAXT)*(6*MAXNC+11)+(13*NLBMAX+200
     &    +61*51+13)*MAXT)

      INTEGER          NDLT(NA),IW(LGIW),
     &                 ICOL(KA,NA)
      DOUBLE PRECISION X1(L1),X2(L2),X3(L3),
     &                 A1(L1,L2,L3),A2(L1,L2,L3),A3(L1,L2,L3),
     &                 B1(L1,L2,L3),B2(L1,L2,L3),B3(L1,L2,L3),
     &                 C(L1,L2,L3),F(L1,L2,L3),
     &                 RW(LGRW)
      REAL*8 EPSIN,EPSOT,EPSEP(10)
      INTEGER INFO(40),INFOEP(10),ISW,IGUSS,IS,NBAND

      DOUBLE PRECISION MAT(KA,NA),RHS(KA),V(KA),
     &                 SOL(3*KA),RHSX(KA),RHSC(KA),TMP

      INTEGER Z1,Z2,Z3,NDIAG,N,ICON,I,Z,NC
      DOUBLE PRECISION ONE,T,HR1,HR2,HR3,HR4,HR6,HR7,HR13
      PARAMETER (ONE=1.D0)
C
C-------------------------------------------------------------
C
C**** THESE ARE PARAMETERS OF THE TEST PDES. CHANGES OF THE
C     VALUES CAN PRODUCE DIVERGENCE IN THE ITERATIVE SOLVER.
C
      T=1
C
C****** CREATE NODE COORDINATES
C
        DO 11 Z1=1,N1
          X1(Z1)=DBLE(Z1-1)/DBLE(N1-1)
11      CONTINUE
        DO 12 Z2=1,N2
          X2(Z2)=DBLE(Z2-1)/DBLE(N2-1)
12      CONTINUE
        DO 13 Z3=1,N3
          X3(Z3)=DBLE(Z3-1)/DBLE(N3-1)
13      CONTINUE
```

```
C
C   -UX1X1-UX2X2-UX3X3+T*((X2-X3)*UX1+(X3-X1)*UX2+(X1-X2)*UX3)=F
C
C   REMARK: IF T IS TO LARGE THE PDE IS SINGULAR.
C
      DO 203 Z3=1,N3
      DO 203 Z2=1,N2
      DO 203 Z1=1,N1
      A1(Z1,Z2,Z3)=1
      A2(Z1,Z2,Z3)=1
      A3(Z1,Z2,Z3)=1
      B1(Z1,Z2,Z3)=T*(X2(Z2)-X3(Z3))
      B2(Z1,Z2,Z3)=T*(X3(Z3)-X1(Z1))
      B3(Z1,Z2,Z3)=T*(X1(Z1)-X2(Z2))
      C (Z1,Z2,Z3)=0
      HR1 = ONE-X2(Z2)
      HR2 = X2(Z2)*HR1
      HR3 = ONE-X3(Z3)
      HR4 = X3(Z3)*HR3
      HR6 = ONE-X1(Z1)
      HR7 = X1(Z1)*HR6
      HR13 = HR1*X3(Z3)*HR3
      F(Z1,Z2,Z3) = 2*HR2*HR4+2*HR7*HR4+2*HR7*HR2+
     &          T*((X2(Z2)-X3(Z3))*
     &          (HR6*X2(Z2)*HR13-X1(Z1)*X2(Z2)*HR13)+
     &          (X3(Z3)-X1(Z1))*
     &          (HR7*HR13-HR7*X2(Z2)*X3(Z3)*HR3)+
     &          (X1(Z1)-X2(Z2))*
     &          (HR7*HR2*HR3-HR7*HR2*X3(Z3)))
203       CONTINUE
C
C***** DIRICHLET CONDITIONS:
C
        DO 300  Z3=1,N3
        DO 300  Z2=1,N2
        C(1,Z2,Z3)=1
        B1(1,Z2,Z3)=0
        B2(1,Z2,Z3)=0
        B3(1,Z2,Z3)=0
        F(1,Z2,Z3)=0
        C(N1,Z2,Z3)=1
        B1(N1,Z2,Z3)=0
        B2(N1,Z2,Z3)=0
        B3(N1,Z2,Z3)=0
        F(N1,Z2,Z3)=0
        IF (Z2.EQ.1) THEN
         DO 325 Z1=1,N1
         C(Z1,1,Z3)=1
         B1(Z1,1,Z3)=0
         B2(Z1,1,Z3)=0
         B3(Z1,1,Z3)=0
         F(Z1,1,Z3)=0
325      CONTINUE
```

```
        ELSEIF (Z2.EQ.N2) THEN
         DO 326 Z1=1,N1
         C(Z1,N2,Z3)=1
         B1(Z1,N2,Z3)=0
         B2(Z1,N2,Z3)=0
         B3(Z1,N2,Z3)=0
         F(Z1,N2,Z3)=0
326      CONTINUE
        ENDIF
        IF (Z3.EQ.1) THEN
         DO 335 Z1=1,N1
         C(Z1,Z2,1)=1
         B1(Z1,Z2,1)=0
         B2(Z1,Z2,1)=0
         B3(Z1,Z2,1)=0
         F(Z1,Z2,1)=0
335      CONTINUE
        ELSEIF (Z3.EQ.N3) THEN
         DO 336 Z1=1,N1
         C(Z1,Z2,N3)=1
         B1(Z1,Z2,N3)=0
         B2(Z1,Z2,N3)=0
         B3(Z1,Z2,N3)=0
         F(Z1,Z2,N3)=0
336      CONTINUE
        ENDIF
300     CONTINUE
C
      N=N1*N2*N3
      CALL DM_VPDE3D(A1,L1,L2,N1,N2,N3,
     $               A2,A3,X1,X2,X3,B1,B2,
     $               B3,C,F,MAT,KA,NA,N,
     $               NDIAG,NDLT,RHS,ICON)
      PRINT*,'ICON OF DM_VPDE3D = ',ICON
      IF (ICON.GT.29999) STOP
C
C
      DO Z=1,N
      RHSX(Z)=RHS(Z)
      ENDDO
      NBAND=0
      DO I=1,NDIAG
      NBAND=MAX(NBAND,ABS(NDLT(I)))
      ENDDO
C
C
C**** CHANGE TO ELLPACK FORMAT:
C
      NC=NDIAG
      DO I=1,NC
      DO Z=1,KA
      IS=Z+NDLT(I)
      ICOL(Z,I)=IS
```

```
      ENDDO
      ENDDO
C
C****** CALL THE ITERATIVE SOLVER:
C
      ISW=1
      IGUSS=0
      EPSOT=1.D-6
      EPSIN=1.D-3
      INFO(1)=10
      INFO(2)=MAXT*100
      INFO(3)=1
      INFO(5)=2
      INFO(6)=5000
      INFO(7)=5
      INFO(8)=20
      INFO(11)=5000
      INFO(10)=2
      INFO(12)=20
      INFO(13)=0
      INFOEP(1)=1
      INFOEP(2)=5
      INFOEP(3)=14
      EPSEP(1)=1.D-2
      EPSEP(2)=EPSEP(1)
      EPSEP(3)=0.2
      EPSEP(4)=1.D-3
      CALL DM_VMLBIFE(MAT,KA,NC,N,ICOL,
     &              RHS,ISW,IGUSS,INFO,INFOEP,EPSOT,EPSIN,
     &              EPSEP,V,RW,LGRW,IW,LGIW,ICON)
      PRINT*,'ICON OF DM_VEBIFE = ',ICON
      IF (ICON.GT.29999) STOP
C
      DO I=1,NBAND
      SOL(I)=0.0D0
      SOL(NBAND+N+I)=0.0D0
      ENDDO
      DO Z=1,N
      SOL(NBAND+Z)=V(Z)
      ENDDO
      CALL DM_VMVSD(MAT,KA,NDIAG,N,NDLT,NBAND,SOL,RHSC,ICON)
      TMP=0
      DO Z=1,N
      TMP=MAX(TMP,ABS((RHSX(Z)-RHSC(Z))/(RHSX(Z)+1.0)))
      ENDDO
C
      PRINT*,' ERROR = ',TMP
C
      STOP
      END
```

(4) Method

The calculation stars by removing rows containing only zero entries outside the main diagonal (typically arising from Dirichlet conditions). This can effect that the matrix becomes symmetric. The linear system is normalized to achieve that the row sums are in the order of one and the main diagonal contains only non-negative entries. The normalized system is solved by the ORTHOMIN or GMRES method. The preconditioner bases on a nested incomplete block factorizations using (approximative) Schur complements. The set of simultaneously eliminated unknowns are defined by searching a maximal independent set in the undirected graph created by the large entries in the matrix. In the Schur complement the small entries are dropped to keep the sparsity of the matrices. The linear system on the final level is normalized and iteratively solved by ORTHOMIN or GMRES.

DM_VMVSCC

Multiplication of a real sparse matrix and a real vector (compressed column storage method)
CALL DM_VMVSCC(A, NZ, NROW, NFCNZ, N, X, Y, W, IW, ICON)

(1) Function

This subroutine obtains a product by multiplying an $n \times n$ sparse matrix by a vector.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

The sparse matrix $\boldsymbol{A}$ is stored by the compressed column storage method.

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

(2) Parameters

A ..... Input. The non-zero elements of a coefficient matrix are stored.

The non-zero elements of a sparse matrix are stored in A(1:NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1.

One-dimensional array A(NZ).

NZ ..... Input. The total number of the nonzero elements belong to a coefficient matrix $\boldsymbol{A}$.

NROW ..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ ..... Input. The position of the first nonzero element stored in an array A by the compressed column storage method which stores the nonzero elements column by column. NFCNZ(N+1) = NZ + 1.

One-dimensional array NFCNZ(N+1).

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

X ..... Input. Vector $\boldsymbol{x}$ is stored in X(1:N).

One-dimensional array X(N).

Y ..... Output. The product of a matrix and vector is stored in Y(1:N).

A one-dimensional array Y(N).

W ..... Work area. One-dimensional array W(NZ).

IW ..... Work area. Two-dimensional array IW(2, NZ).

ICON ..... Output. Condition code

See Table DM_VMVSCC-1.

Table DM_VMVSCC-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ + 1.	Processing is discontinued.

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 0 & 6 \\ 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 \end{bmatrix}$$

$$\downarrow$$

$$\mathrm{NFCNZ} = \begin{bmatrix} 1 \\ 3 \\ 6 \\ 9 \\ 12 \end{bmatrix}, \quad \mathrm{CC} = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 7 \\ 3 \\ 8 \\ 10 \\ 6 \\ 9 \\ 11 \end{bmatrix}, \quad \mathrm{NROW} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \\ 3 \\ 1 \\ 3 \\ 4 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Figure DM_VMVSCC-1 Storing a coefficient matrix *A* in compressed column storage method

The way how to store a coefficient matrix ***A*** in compressed column storage method is explained.

The nonzero elements of each column vector of a matrix ***A*** are stored in compressed mode into a one-dimensional array CC column by column. The position in the array CC where the first nonzero element in the *i*-th column vector is stored is set into NFCNZ(*i*).

The value of NFCNZ(N+1) is set to NZ+1, where N is an order of the matrix ***A*** and NZ is the total number of the nonzero elements in this matrix.

The row number of the nonzero element of the matrix ***A*** stored in the *i*-th array element CC(*i*) is set into NROW(*i*).

(3) Comments on use

a. Example

A product is obtained by multiplying the sparse matrix by a vector.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)

      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG)
      DIMENSION A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
     $          W(K*NDIAG),IW(2,K*NDIAG)
      DIMENSION X(N),B(N),Y(N)

      X(1:N)=1.0D0

      NOFST(1)=-NX*NY
      NOFST(2)=-NX
      NOFST(3)=-1
      NOFST(4)=0
      NOFST(5)=1
      NOFST(6)=NX
      NOFST(7)=NX*NY

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG(1:LENGTH,I)=DBLE(I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG(NBASE+1:N,I)=DBLE(I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
      DO J=1,N
      NTOPCFG=1
      DO I=NDIAG,1,-1
C
      IF(DIAG(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      A(NUMNZ)=DIAG(J,I)
      NROW(NUMNZ)=NCOL
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
```

```
      NUMNZ=NUMNZ+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1

      CALL DM_VMVSCC(A,NNZ,NROW,NFCNZ,N,X,
     $             Y,W,IW,ICON)
C
      B(1:N)=0.0D0
      DO I=1,N
      NS=NFCNZ(I)
      NE=NFCNZ(I+1)-1
      DO J=NS,NE
      II=NROW(J)
      B(II)=B(II)+A(J)*X(I)
      ENDDO
      ENDDO
C
      S=0.0D0
      DO I=1,N
      S=MAX(S,ABS(Y(I)-B(I)))
      ENDDO
C
      PRINT*,'ERROR=',S

      STOP
      END
```

DM_VMVSCCC

Multiplication of a complex sparse matrix and a complex vector (compressed column storage method)
CALL DM_VMVSCCC(ZA, NZ, NROW, NFCNZ, N, ZX, ZY, ZW, IW, ICON)

(1) Function

This subroutine obtains a product by multiplying an $n \times n$ complex sparse matrix by a complex vector.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

The sparse matrix $\boldsymbol{A}$ is stored by the compressed column storage method.

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

(2) Parameters

ZA ..... Input. The non-zero elements of a coefficient matrix are stored.

The non-zero elements of a sparse matrix are stored in ZA(1:NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1. For a complex matrix , the real array CC in this Figure is replaced with complex array.

A double precision complex one-dimensional array ZA(NZ).

NZ ..... Input. The total number of the nonzero elements belong to a coefficient matrix $\boldsymbol{A}$.

NROW ..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array ZA.

One-dimensional array NROW(NZ).

NFCNZ ..... Input. The position of the first nonzero element stored in an array A by the compressed column storage method which stores the nonzero elements column by column. NFCNZ(N+1) = NZ + 1.

One-dimensional array NFCNZ(N+1).

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

ZX ..... Input. Vector $\boldsymbol{x}$ is stored in ZX(1:N).

A double precision complex one-dimensional array ZX(N).

ZY ..... Output. The product of a matrix and vector is stored in ZY(1:N).

A double precision complex one-dimensional array ZY(N).

ZW ..... Work area. A double precision complex one-dimensional array ZW(NZ).

IW ..... Work area. Two-dimensional array IW(2, NZ).

ICON ..... Output. Condition code

See Table DM_VMVSCCC-1.

Table DM_VMVSCCC-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ + 1.	Processing is discontinued.

(3) Comments on use

a. Example

A product is obtained by multiplying the complex sparse matrix by a complex vector.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)

      DIMENSION NOFST(NDIAG)
      COMPLEX*16 ZDIAG(K,NDIAG),ZA(K*NDIAG),ZW(K*NDIAG)
      DIMENSION NROW(K*NDIAG),NFCNZ(N+1),
     $          IW(2,K*NDIAG)
      COMPLEX*16 ZX(N),ZB(N),ZY(N)

      ZX(1:N)=(1.0D0,0.0D0)

      NOFST(1)=-NX*NY
      NOFST(2)=-NX
      NOFST(3)=-1
      NOFST(4)=0
      NOFST(5)=1
      NOFST(6)=NX
      NOFST(7)=NX*NY

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      ZDIAG(1:LENGTH,I)=DCMPLX(DBLE(I),0.0D0)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      ZDIAG(NBASE+1:N,I)=DCMPLX(DBLE(I),0.0D0)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
      DO J=1,N
      NTOPCFG=1
      DO I=NDIAG,1,-1
```

```
C
      IF(ZDIAG(J,I).NE.(0.0D0,0.0D0))THEN
C
      NCOL=J-NOFST(I)
      ZA(NUMNZ)=ZDIAG(J,I)
      NROW(NUMNZ)=NCOL
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      NUMNZ=NUMNZ+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1

      CALL DM_VMVSCCC(ZA,NNZ,NROW,NFCNZ,N,ZX,
     $             ZY,ZW,IW,ICON)
C
      ZB(1:N)=(0.0D0,0.0D0)
      DO I=1,N
      NS=NFCNZ(I)
      NE=NFCNZ(I+1)-1
      DO J=NS,NE
      II=NROW(J)
      ZB(II)=ZB(II)+ZA(J)*ZX(I)
      ENDDO
      ENDDO
C
      S=0.0D0
      DO I=1,N
      S=MAX(S,CDABS(ZY(I)-ZB(I)))
      ENDDO
C
      PRINT*,'ERROR=',S

      STOP
      END
```

DM_VMVSD

Multiplication of a real sparse matrix and a real vector (diagonal format storage method)
CALL DM_VMVSD(A,K, NDIAG,N, NOFST,NLB,X,Y,ICON)

(1) Function

This subroutine obtains a product by multiplying an $n \times n$ sparse matrix by a vector.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

The sparse matrix $\boldsymbol{A}$ is stored by the diagonal format storage method.

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

(2) Parameters

A ..... Input. The non-zero elements of a coefficient matrix are stored.

The non-zero elements of a sparse matrix are stored in A(1:N,1:NDIAG).

For the diagonal format storage method, refer to Item b of Section 3.2.1.1 of Part I of the SSL II Extended Capabilities User's Guide.

Two-dimensional array A(K,NDIAG).

K ..... Input. The size of the first dimension of array A($\geq n$).

NDIAG ..... Input. The total number of diagonal vectors including non-zero elements of a coefficient matrix to be stored in array A.

The size of the second dimension of array A.

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

NOFST ..... Input. The distance from the main diagonal vector corresponding to the diagonal vector to be stored in array A is stored. The upper diagonal vector matrix is indicated by a positive value and the lower diagonal vector matrix is indicated by a negative value.

One-dimensional array NOFST(NDIAG).

NLB ..... Input. The lower bandwidth of matrix $\boldsymbol{A}$.

X ..... Input. Vector $\boldsymbol{x}$ is stored in X(NLB+1:NLB+N).

One-dimensional array X(n+nlb+nub), where nlb is the lower band width and nub is the upper band width.

Y ..... Output. The product of a matrix and vector is stored in Y(1:N).

A one-dimensional array Y(N).

ICON ..... Output. Condition code

See Table DM_VMVSD-1.

Table DM_VMVSD-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 1, NDIAG < 1, K < N, NLB ≠ Max(-NOFST(I)) or |NOFST(I) | > N-1.	Processing is discontinued.

(3) Comments on use

a. Notes

1) Notes an using the diagonal format

Zeros need to be set for those elements of the diagonal vectors outside the coefficient matrix ***A***.

There are no special restrictions on the storage order of diagonal vector columns in array A.

The merit of this method is that the computation is possible without using an indirect index with matrix vector multiplication. But its demerit is that matrices having no diagonal structure cannot be stored efficiently.

b. Example

A product is obtained by multiplying the sparse matrix by a vector.

The sparse matrix is generated by init_mat_diag. (Refer to the example program of DM_VBCSD.)

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER(NVW=3*K)
      DIMENSION NOFST(NDIAG)
      DIMENSION A(K,NDIAG)
      DIMENSION Y(N),B(N)
      DIMENSION X(NVW)


      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)
```

```
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A,B)
      DO I=1,N
      B(I)=0.0D0
      DO J=1,NDIAG
      B(I)=B(I)+A(I,J)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO


      NBANDL=0
      NBANDR=0
      DO I=1,NDIAG
      IF(NOFST(I).LT.0)THEN
      NBANDL=MAX(NBANDL,-NOFST(I))
      ELSE
      NBANDR=MAX(NBANDR,NOFST(I))
      ENDIF
      ENDDO

      X(1+NBANDL:N+NBANDL) = 1.0D0
      CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,X,Y,ICON)

      ERROR=0.0D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(Y,B)
!$OMP+         REDUCTION(MAX:ERROR)
      DO I=1,N
      ERROR=MAX(ERROR,DABS(Y(I)-B(I)))
      ENDDO
!$OMP END PARALLEL DO

      PRINT*,'ERROR = ',ERROR

      STOP
      END
```

DM_VMVSE

Multiplication of a real sparse matrix and a real vector (ELLPACK format storage method)
CALL DM_VMVSE(A,K,NW,N,ICOL,X,Y,ICON)

(1) Function

This subroutine obtains a product by multiplying an $n \times n$ sparse matrix by a vector.

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

The coefficient matrix ($n \times n$) is stored by the ELLPACK format storage method using two arrays.

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

(2) Parameters

A ..... Input. Non-zero elements of a coefficient matrix are stored in A(1:N,1:NW).

For the ELLPACK format storage method, refer to Item b of Section 3.2.1.1 of Part I of the SSLII Extended Capabilities User's Guide.

A two-dimensional array A(K,NW).

K ..... Input. The size of the first dimension of an array A(≥ n).

NW ..... Input. The size of the second dimension of array A and ICOL. The maximum number of non-zero elements in each row of matrix $\boldsymbol{A}$ to be stored in array A.

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

ICOL ..... Input. The column index used in the ELLPACK format storage method that indicates the column vector to which the element to be stored in A belongs.

The two-dimensional array, ICOL(K,NW).

X ..... Input. The vector $\boldsymbol{x}$ is stored in X(1:N).

A one-dimensional array X(N).

Y ..... Output. The product of a matrix and vector is stored in Y(1:N).

A one-dimensional array Y(N).

ICON ..... Output. Condition code.

See Table DM_VMVSE-1.

Table DM_VMVSE-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 1, NW < 1, K < N.	Processing is discontinued.

(3) Comments on use

a. Note

1) When using the ELLPACK storage format
It is recommended that array A is initialized with zero and ICOL with a row vector number.

b. Example

A product is obtained by multiplying the sparse matrix by a vector.

The sparse matrix is generated by init_mat_ell. (Refer to the example program of DM_VBCSE.)

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=60,NX =NORD ,NY = NORD,NZ = NORD,
     &           N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (IWIDT = 7)
      DIMENSION ICOL(K,IWIDT)
      DIMENSION A(K,IWIDT)
      DIMENSION X(N),B(N),Y(N)

      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL
     &          ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)

!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A,B)
      DO I=1,N
      B(I)=0.0D0
      DO J=1,IWIDT
      B(I)=B(I)+A(I,J)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO

      Y(1:N)=1.0D0
      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,Y,X,ICON2)

      ERROR=0.0D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X,B)
!$OMP+         REDUCTION(MAX:ERROR)
      DO I=1,N
      ERROR=MAX(ERROR,DABS(X(I)-B(I)))
```

```
      ENDDO
!$OMP END PARALLEL DO

      PRINT*,'ERROR = ',ERROR

      STOP
      END
```

DM_VPDE2D

Generation of System of linear equations with sparse matrices by the finite difference discretization of a two dimensional boundary value problem for second order partial differential equation
CALL DM_VPDE2D (A1, L1, N1, N2, A2, X1, X2, B1, B2, C, F, A, K, NA, N, NDIAG, NOFST, R, ICON)

(1) Function

This subroutine assembles the system of linear equations by the finite difference discretization of the linear, two dimensional boundary value problem on the rectangular domain B:

The partial differential equation (1) on the domain B with the boundary conditions (2) on the boundary of the domain B is satisfied.

$$-\left(\frac{\partial}{\partial x_1} a_1 \frac{\partial u}{\partial x_1} + \frac{\partial}{\partial x_2} a_2 \frac{\partial u}{\partial x_2}\right) + b_1 \frac{\partial u}{\partial x_1} + b_2 \frac{\partial u}{\partial x_2} + cu = f \qquad (1)$$

$$\beta_1 \frac{\partial u}{\partial x_1} + \beta_2 \frac{\partial u}{\partial x_2} + \gamma u = \phi \qquad (2)$$

a_1, a_2, b_1, b_2, c and f are given functions on the domain and β_1, β_2, γ and ϕ are given functions on the boundary of the domain.

The N1 × N2 grid is defined by X(i,j)=(X1(i),X2(j))

i=1,...,N1, j=1,...,N2 with

B:=[X1(1),X1(N1)] × [X2(1),X2(N2)] ;

The functions involved in the partial differential equation and the boundary conditions are defined by their values at the grid points.

The returned coefficient matrix is stored by the diagonal format storage method, see section 3.2.1.2 in the *SSL II Extended Capabilities User's Guide II*.

(2) Parameters

A1 ..... Input. The coefficients of $a_1(x_{ij})$ are stored in A1(1:N1, 1:N2).
Two-dimensional array A1(L1,N2).

L1 ..... Input. Size of first-dimension of array A1, A2, B1, B2, C and F (L1≥N1).

N1 ..... Input. Number of grid points in the x_1-direction (N1>2).

N2 ..... Input. Number of grid points in the x_2-direction (N2>2).

A2 ..... Input. The coefficients of $a_2(x_{ij})$ are stored in A2(1:N1, 1:N2).
Two-dimensional array A2(L1,N2).

X1 ..... Input. The x_1-coordinates of the grid points are stored in X1(1:N1). The coordinates of the grid points have to be increasing:
X1(i)<X1(i+1) , i=1,...,N1-1
One-dimensional array of X1(N1).

X2 ..... Input. The x_2-coordinates of the grid points are stored in X2(1:N2). The coordinates of the grid points have to be increasing:

X2(i)<X2(i+1) , i=1,...,N2-1

One-dimensional array X2(N2).

B1 ..... Input. The coefficients of $b_1(\boldsymbol{x}_{ij})$ and the boundary condition β_1 are stored in B1(1:N1, 1:N2).

$$\mathrm{B1}(i,j)=\begin{cases}\beta_1(x_{1,j}) & i=1;\\ \beta_1(x_{\mathrm{N1},j}) & i=\mathrm{N1};\\ \beta_1(x_{i,1}) & j=1;\\ \beta_1(x_{i,\mathrm{N2}}) & j=\mathrm{N2};\\ b_1(x_{i,j}) & \text{else;}\end{cases}$$

Two-dimensional array B1(L1,N2).

B2 ..... Input. The coefficients of $b_2(x_{ij})$ and the boundary condition β_2 are stored in B2(1:N1, 1:N2).

$$\mathrm{B2}(i,j)=\begin{cases}\beta_2(x_{1,j}) & i=1;\\ \beta_2(x_{\mathrm{N1},j}) & i=\mathrm{N1};\\ \beta_2(x_{i,1}) & j=1;\\ \beta_2(x_{i,\mathrm{N2}}) & j=\mathrm{N2};\\ b_2(x_{i,j}) & \text{else;}\end{cases}$$

Two-dimensional array B2(L1,N2).

C ..... Input. The coefficients of $c(x_{ij})$ and the boundary condition γ are stored in C(1:N1, 1:N2).

$$\mathrm{C}(i,j)=\begin{cases}\gamma(x_{1,j}) & i=1;\\ \gamma(x_{\mathrm{N1},j}) & i=\mathrm{N1};\\ \gamma(x_{i,1}) & j=1;\\ \gamma(x_{i,\mathrm{N2}}) & j=\mathrm{N2};\\ c(x_{i,j}) & \text{else;}\end{cases}$$

Two-dimensional array C(L1,N2).

F ..... Input. The coefficients of $f(x_{ij})$ and the boundary condition ϕ are stored in F(1:N1, 1:N2).

$$\mathrm{F}(i,j)=\begin{cases}\phi(x_{1,j}) & i=1;\\ \phi(x_{\mathrm{N1},j}) & i=\mathrm{N1};\\ \phi(x_{i,1}) & j=1;\\ \phi(x_{i,\mathrm{N2}}) & j=\mathrm{N2};\\ f(x_{i,j}) & \text{else;}\end{cases}$$

Two-dimensional array F(L1,N2).

A ..... Output. The nonzero elements of a coefficient matrix are stored in A.

The coefficient matrix is stored in A(1:N,1:NDIAG).

Two-dimensional array A(K,NA).

For an explanation of the diagonal format storage method, see b, "Diagonal format storage method of general sparse matrices," in Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the *SSL II Extended Capability User's Guide II*.

K ..... Input. Size of first-dimension of array A (≥ N=N1×N2).

NA ..... Input. Size of second-dimension of array A. (≥ NDIAG=5).

N ..... Input. Order *n* of matrix A (=N1×N2).

NDIAG ..... Output. Number of columns in array A and size of array NOFST (=5).

NOFST ..... Output. Offsets of diagonals of $\boldsymbol{A}$ stored A. Main diagonal has offset 0, subdiagonals have negative offsets, and superdiagonals have positive offsets.

One-dimensional array NOFST(NDIAG)

R ..... Output. The right-side constant vectors of a system of linear equations are stored in R(1:N).

One-dimensional array R(K) .

ICON ..... Output. Condition code.

See Table DM_VPDE2D-1.

Table DM_VPDE2D-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	L1 < N1, N1 < 3 , N2 < 3 ,NA < 5 , or K < N1 × N2.	Processing is discontinued.
30001	The coordinates of the grid points is not increasing.	

(3) Comments on use

a. Notes

1) The quality of the value of the solution at the grid points delivered by the solver of the linear system or an eigenvalue problem solver depends strictly on the number and the location of the grid points.

2) The changes of the distances of the grid points to their nearest neighbor should be moderate. For instance in $\boldsymbol{x}_1$-direction the condition

$$0.5 \le \frac{\mathrm{X1}(i)-\mathrm{X1}(i-1)}{\mathrm{X1}(i+1)-\mathrm{X1}(i)} \le 2 \quad ,i=2,...,\mathrm{N1}-1$$

should be met (for the $\boldsymbol{x}_2$-direction analogously).

If this condition is not fulfilled the coefficient matrix can become ill--posed. Keep in mind that the condition number of the coefficient matrix is not only determined by the grid but also by the coefficient functions.

b. Example

The domain is the box $[-1,1]^2$. The partial differential equation is

$$-\left(\frac{\partial^2 u}{\partial^2 x_1}+\frac{\partial^2 u}{\partial^2 x_2}\right)+v_1\frac{\partial u}{\partial x_1}+v_2\frac{\partial u}{\partial x_2}=0$$

modeling a diffusion of the quantity u through the cannel driven by the rotating velocity field

$$\boldsymbol{v}=(v_1,v_2)=v_0\cdot\left(\frac{x_2}{\sqrt{x_1^2+x_2^2}},\frac{-x_1}{\sqrt{x_1^2+x_2^2}}\right)$$

where v_0 is real constant (e.g. v_0=1). The boundary conditions are set as follows:

$u=0 \qquad x_2=-1$

$u=1 \qquad x_2=1$

$\frac{\partial u}{\partial n}=0 \qquad else$

where *n* denotes the outer normal field at the boundary of the box.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE

      INTEGER   N1,N2,KA,NA,L1,L2

      PARAMETER (N1=49,N2=49, L1=N1,L2=N2, KA=N1*N2,NA=5)

      INTEGER   NOFST(NA)
      DOUBLE PRECISION V0,X1(L1),X2(L2),
     &         A1(L1,L2),A2(L1,L2),B1(L1,L2),B2(L1,L2),
     &         C(L1,L2),F(L1,L2),A(KA,NA),R(KA)

      INTEGER Z1,Z2,ICON,I,J,N,NDIAG

      V0=1.
C
C create grid nodes nodes:
C
      DO 11 Z1=1,N1
        X1(Z1)=(2*DBLE(Z1-1)/DBLE(N1-1)-1.)
11    CONTINUE
      DO 13 Z2=1,N2
        X2(Z2)=(2*DBLE(Z2-1)/DBLE(N2-1)-1.)
13    CONTINUE
C
C coefficient functions:
C
      DO 2000 Z2=1,N2

       DO 20 Z1=1,N1
```

```
        A1(Z1,Z2)=1
        A2(Z1,Z2)=1
20     CONTINUE
       DO 21 Z1=2,N1-1
          B1(Z1,Z2)= V0*X2(Z2)/SQRT(X1(Z1)**2+X2(Z2)**2+1.D-10)
          B2(Z1,Z2)=-V0*X1(Z1)/SQRT(X1(Z1)**2+X2(Z2)**2+1.D-10)
          C (Z1,Z2)=0
          F (Z1,Z2)=0
21    CONTINUE
C
C boundary conditions at faces X1=-1 and X1=1:
C
      B1(1,Z2)=-1
      B2(1,Z2)=0
      C (1,Z2)=0
      F (1,Z2)=0

      B1(N1,Z2)=1
      B2(N1,Z2)=0
      C (N1,Z2)=0
      F (N1,Z2)=0
C
C boundary conditions at faces X2=-1 and X2=1:
C
      IF (Z2.EQ.1) THEN
        DO 103 Z1=1,N1
          B1(Z1,1)=0
          B2(Z1,1)=0
          C (Z1,1)=1
          F (Z1,1)=0
103     CONTINUE
      ELSE IF (Z2.EQ.N2) THEN
        DO 113 Z1=1,N1
          B1(Z1,N2)=0
          B2(Z1,N2)=0
          C (Z1,N2)=1
          F (Z1,N2)=1
113     CONTINUE
      END IF
2000  CONTINUE

C
C build the linear system:
C
      N=N1*N2
      CALL DM_VPDE2D(A1,L1,N1,N2,A2,X1,X2, B1,B2,C,F,A,KA,NA,N,
     &               NDIAG,NOFST,R,ICON)
      PRINT*,'ICON of DM_VPDE2D =',ICON
      IF (ICON.GT.29999) GOTO 9999
C
C write the matrix to a file:
C
      WRITE (6,'(3D23.16)') ((A(I,J),I=1,N,100),J=1,NDIAG)
```

```
      WRITE (6,'(3I10)') (NOFST(J),J=1,NDIAG)
      WRITE (6,'(3D23.16)') (R(I),I=1,N,100)
 9999 CONTINUE
      END
```

(4) Method

The diffusion term $-\nabla a \nabla$ is approximated by the product scheme of centered finite difference schemes of order two for the x_1- and x_2- direction. The convective term $b\nabla$ is approximated by an upwind scheme of order one. More details are presented in [75] in Appendix A, “References.”

DM_VPDE3D

Generation of System of linear equations with sparse matrices by the finite difference discretization of a three dimensional boundary value problem for second order partial differential equation
CALL DM_VPDE3D (A1, L1, L2, N1, N2, N3, A2, A3, X1, X2, X3, B1, B2, B3, C, F, A, KA, NA, N, NDIAG, NOFST, R, ICON)

(1) Function

This subroutine assembles the system of linear equations by the finite difference discretization of the linear, three dimensional boundary value problem on the rectangular domain B:

The partial differential equation (1) on the domain B with the boundary conditions (2) on the boundary of the domain B is satisfied.

$$-\left(\frac{\partial}{\partial x_1}a_1\frac{\partial u}{\partial x_1}+\frac{\partial}{\partial x_2}a_2\frac{\partial u}{\partial x_2}+\frac{\partial}{\partial x_3}a_3\frac{\partial u}{\partial x_3}\right)+b_1\frac{\partial u}{\partial x_1}+b_2\frac{\partial u}{\partial x_2}+b_3\frac{\partial u}{\partial x_3}+cu=f \quad (1)$$

$$\beta_1\frac{\partial u}{\partial x_1}+\beta_2\frac{\partial u}{\partial x_2}+\beta_3\frac{\partial u}{\partial x_3}+\gamma u=\phi \quad (2)$$

a_1, a_2, a_3, b_1, b_2, b_3, c and f are given functions on the domain and β_1, β_2, β_3, γ and ϕ are given functions on the boundary of the domain.

The N1 × N2 × N3 grid is defined by X(i,j,k)=(X1(i),X2(j),X3(k))

for i=1,...,N1, j=1,...,N2 and k=1,...,N3 with

B:=[X1(1),X1(N1)] × [X2(1),X2(N2)] × [X3(1),X3(N3)];

The functions involved in the partial differential equation and the boundary conditions are defined by their values at the grid points.

The returned coefficient matrix is stored by the diagonal format storage method, see Section 3.2.1.2 in the *SSL II Extended Capabilities User's Guide II*.

(2) Parameters

A1 ..... Input. The coefficients of $a_1(x_{ijk})$ are stored in A1(1:N1, 1:N2, 1:N3).
Three-dimensional array A1(L1,L2,N3).

L1 ..... Input. Size of first-dimension of array A1, A2, A3, B1, B2, B3, C and F.

L2 ..... Input. Size of second-dimension of array A1, A2, A3, B1, B2, B3, C and F.

N1 ..... Input. Number of grid points in the x_1-direction. (N1>2)

N2 ..... Input. Number of grid points in the x_2-direction. (N2>2)

N3 ..... Input. Number of grid points in the x_3-direction. (N3>2)

A2 ..... Input. The coefficients of $a_2(x_{ijk})$ are stored in A2(1:N1, 1:N2, 1:N3).
Three-dimensional array A2(L1,L2,N3).

A3 ..... Input. The coefficients of $a_3(x_{ijk})$ are stored in A3(1:N1, 1:N2, 1:N3).
Three-dimensional array A3(L1,L2,N3).

X1 ..... Input. The x_1-coordinates of the grid points are stored in X1(1:N1). The coordinates of the grid points have to be increasing:
X1(i)<X1(i+1) , i=1,...,N1-1
One-dimensional array of X1(N1).

X2 ..... Input. The x_2-coordinates of the grid points are stored in X2(1:N3). The coordinates of the grid points have to be increasing:
X2(i)<X2(i+1) , i=1,...,N2-1
One-dimensional array of X2(N2).

X3 ..... Input. The x_3-coordinates of the grid points are stored in X3(1:N3). The coordinates of the grid points have to be increasing:
X3(i)<X3(i+1) , i=1,...,N3-1
One-dimensional array X3(N3).

B1 ..... Input. The coefficients of $b_1(x_{ijk})$ and the boundary condition β_1 are stored in B1(1:N1, 1:N2, 1:N3) as follows.

$$\mathrm{B1}(i,j,k)=\begin{cases}\beta_1(x_{1,j,k}) & i=1;\\ \beta_1(x_{\mathrm{N1},j,k}) & i=\mathrm{N1};\\ \beta_1(x_{i,1,k}) & j=1;\\ \beta_1(x_{i,\mathrm{N2},k}) & j=\mathrm{N2};\\ \beta_1(x_{i,j,1}) & k=1;\\ \beta_1(x_{i,j,\mathrm{N3}}) & k=\mathrm{N3};\\ b_1(x_{i,j,k}) & \text{else;}\end{cases}$$

Three-dimensional array B1(L1,L2,N3).

B2 ..... Input. The coefficients of $b_2(x_{ijk})$ and the boundary condition β_2 are stored in B2(1:N1, 1:N2, 1:N3) as follows.

$$\mathrm{B2}(i,j,k)=\begin{cases}\beta_2(x_{1,j,k}) & i=1;\\ \beta_2(x_{\mathrm{N1},j,k}) & i=\mathrm{N1};\\ \beta_2(x_{i,1,k}) & j=1;\\ \beta_2(x_{i,\mathrm{N2},k}) & j=\mathrm{N2};\\ \beta_2(x_{i,j,1}) & k=1;\\ \beta_2(x_{i,j,\mathrm{N3}}) & k=\mathrm{N3};\\ b_2(x_{i,j,k}) & \text{else;}\end{cases}$$

Three-dimensional array B2(L1,L2,N3).

B3 ..... Input. The coefficients of $\boldsymbol{b}_3(\boldsymbol{x}_{ijk})$ and the boundary condition β_3 are stored in B3(1:N1, 1:N2, 1:N3) as follows.

$$\mathrm{B3}(i,j,k)=\begin{cases}\beta_3(x_{1,j,k}) & i=1;\\ \beta_3(x_{\mathrm{N1},j,k}) & i=\mathrm{N1};\\ \beta_3(x_{i,1,k}) & j=1;\\ \beta_3(x_{i,\mathrm{N2},k}) & j=\mathrm{N2};\\ \beta_3(x_{i,j,1}) & k=1;\\ \beta_3(x_{i,j,\mathrm{N3}}) & k=\mathrm{N3};\\ b_3(x_{i,j,k}) & \text{else};\end{cases}$$

Three-dimensional array B3(L1,L2,N3).

C ..... Input. The coefficients of $c(x_{ijk})$ and the boundary condition γ are stored in C(1:N1, 1:N2, 1:N3) as follows.

$$\mathrm{C}(i,j,k)=\begin{cases}\gamma(x_{1,j,k}) & i=1;\\ \gamma(x_{\mathrm{N1},j,k}) & i=\mathrm{N1};\\ \gamma(x_{i,1,k}) & j=1;\\ \gamma(x_{i,\mathrm{N2},k}) & j=\mathrm{N2};\\ \gamma(x_{i,j,1}) & k=1;\\ \gamma(x_{i,j,\mathrm{N3}}) & k=\mathrm{N3};\\ c(x_{i,j,k}) & \text{else};\end{cases}$$

Three-dimensional array C(L1,L2,N3).

F ..... Input. The coefficients of $f(x_{ijk})$ and the boundary condition ϕ are stored in F(1:N1, 1:N2, 1:N3) as follows.

$$\mathrm{F}(i,j,k)=\begin{cases}\phi(x_{1,j,k}) & i=1;\\ \phi(x_{\mathrm{N1},j,k}) & i=\mathrm{N1};\\ \phi(x_{i,1,k}) & j=1;\\ \phi(x_{i,\mathrm{N2},k}) & j=\mathrm{N2};\\ \phi(x_{i,j,1}) & k=1;\\ \phi(x_{i,j,\mathrm{N3}}) & k=\mathrm{N3};\\ f(x_{i,j,k}) & \text{else};\end{cases}$$

Three-dimensional array F(L1,L2,N3).

A ..... Output. The nonzero elements of a coefficient matrix are stored in A(1:N,1:NDIAG).

Two-dimensional array A(KA,NA).

For an explanation of the diagonal format storage method, see b, “Diagonal format storage method of general sparse matrices,” in Section 3.2.1.1, “Storing the general sparse matrices,” in Part I, “Outline,” in the *SSL II Extended Capability User’s Guide II*.

KA ..... Input. Size of first-dimension of array A (KA ≥ N1 × N2 ×N3).

NA ..... Input. Size of second-dimension of array A. (NA ≥ NDIAG=7).

N ..... Input. Order n of matrix $\boldsymbol{A}$ (N=N1 × N2 ×N3).

NDIAG ..... Output. Number of columns in array A and size of array NOFST. (=7)

NOFST ..... Output. Offsets of diagonals of ***A*** stored A. Main diagonal has offset 0, subdiagonals have negative offsets, and superdiagonals have positive offsets.

One-dimensional array NOFST(NDIAG)

R ..... Output. The right-side constant vectors of a system of linear equations are stored in R(1:N).

One-dimensional array R(N).

ICON ..... Output. Condition code.

See Table DM_VPDE3D-1.

Table DM_VPDE3D-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	L1 < N1, L2 < N2 , N1 < 3 , N2 < 3 , N3 < 3 , NA < 7 , KA < N1 × N2 × N3.	Processing is discontinued.
30001	The coordinates of the grid points is not increasing.	

(3) Comments on use

a. Notes

1) The quality of the value of the solution at the grid points delivered by the solver of the linear system or an eigenvalue problem solver depends strictly on the number and the location of the grid points.

2) The changes of the distances of the grid points to their nearest neighbor should be moderate. For instance in $\boldsymbol{x}_1$-direction the condition

$$0.5 \le \frac{\mathrm{X1}(i) - \mathrm{X1}(i-1)}{\mathrm{X1}(i+1) - \mathrm{X1}(i)} \le 2 \quad ,i = 2,..., \mathrm{N1}-1$$

should be met (for the $\boldsymbol{x}_2$-direction and $\boldsymbol{x}_3$-direction analogously).

If this condition is not fulfilled the coefficient matrix can become ill-posed. Keep in mind that the condition number of the coefficient matrix is not only determined by the grid but also by the coefficient functions.

b. Example

The domain is the channel $[-1,1]^2 \times [0,5]$. The partial differential equation is

$$-\left(\frac{\partial^2 u}{\partial^2 x_1} + \frac{\partial^2 u}{\partial^2 x_2} + \frac{\partial^2 u}{\partial^2 x_3}\right) + v_1 \frac{\partial u}{\partial x_1} + v_2 \frac{\partial u}{\partial x_2} = 0$$

modeling a diffusion of the quantity u through the cannel driven by the rotating velocity field

$$\boldsymbol{v} = (v_1, v_2, v_3) = v_0 \cdot \left(\frac{x_2}{\sqrt{x_1^2 + x_2^2}}, \frac{-x_1}{\sqrt{x_1^2 + x_2^2}}, 0\right)$$

where v_0 is real constant (e.g. v_0=1). The boundary conditions

are set as follows:

$$u = 0 \qquad x_3 = 0$$
$$u = 1 \qquad x_3 = 5$$
$$\frac{\partial u}{\partial n} = 0 \qquad else$$

where n denotes the outer normal field at the boundary of the channel.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT NONE

      INTEGER   N1,N2,N3,KA,NA,L1,L2,L3

      PARAMETER(N1=49,N2=49,N3=25,L1=N1,L2=N2,L3=N3,
     &          KA=N1*N2*N3,NA=7)

      INTEGER  NOFST(NA)
      DOUBLE PRECISION V0,X1(L1),X2(L2),X3(L3),
     &         A1(L1,L2,L3),A2(L1,L2,L3),A3(L1,L2,L3),
     &         B1(L1,L2,L3),B2(L1,L2,L3),B3(L1,L2,L3),
     &         C(L1,L2,L3),F(L1,L2,L3),A(KA,NA),R(KA)

      INTEGER  Z1,Z2,Z3,ICON,I,J,N,NDIAG

      V0=1.
C
C create grid nodes nodes:
C
      DO 11 Z1=1,N1
        X1(Z1)=(2*DBLE(Z1-1)/DBLE(N1-1)-1.)
11    CONTINUE
      DO 12 Z2=1,N2
        X2(Z2)=(2*DBLE(Z2-1)/DBLE(N2-1)-1.)
12    CONTINUE
      DO 13 Z3=1,N3
        X3(Z3)=DBLE(Z3-1)/DBLE(N3-1)
13    CONTINUE
C
C coefficient functions:
C
      DO 2000 Z3=1,N3

       DO 20 Z2=1,N2
        DO 20 Z1=1,N1
          A1(Z1,Z2,Z3)=1
          A2(Z1,Z2,Z3)=1
```

```
          A3(Z1,Z2,Z3)=1
20     CONTINUE
       DO 21 Z2=2,N2-1
        DO 21 Z1=2,N1-1
          B1(Z1,Z2,Z3)= V0*X2(Z2)/SQRT(X1(Z1)**2+X2(Z2)**
     &                  2+1.D-10)
          B2(Z1,Z2,Z3)=-V0*X1(Z1)/SQRT(X1(Z1)**2+X2(Z2)**
     &                  2+1.D-10)
          B3(Z1,Z2,Z3)=0
          C (Z1,Z2,Z3)=0
          F (Z1,Z2,Z3)=0
21    CONTINUE
C
C boundary conditions at faces X1=-1 and X1=1:
C
      DO 101 Z2=1,N2
        B1(1,Z2,Z3)=-1
        B2(1,Z2,Z3)=0
        B3(1,Z2,Z3)=0
        C (1,Z2,Z3)=0
        F (1,Z2,Z3)=0

        B1(N1,Z2,Z3)=1
        B2(N1,Z2,Z3)=0
        B3(N1,Z2,Z3)=0
        C (N1,Z2,Z3)=0
        F (N1,Z2,Z3)=0
101   CONTINUE
C
C boundary conditions at faces X2=-1 and X2=1:
C
      DO 102 Z1=1,N1
        B1(Z1,1,Z3)=0
        B2(Z1,1,Z3)=-1
        B3(Z1,1,Z3)=0
        C (Z1,1,Z3)=0
        F (Z1,1,Z3)=0

        B1(Z1,N2,Z3)=0
        B2(Z1,N2,Z3)=1
        B3(Z1,N2,Z3)=0
        C (Z1,N2,Z3)=0
        F (Z1,N2,Z3)=0
102   CONTINUE
C
C boundary conditions at faces X3=0 and X3=5:
C
      IF (Z3.EQ.1) THEN
        DO 103 Z1=1,N1
          DO 103 Z2=1,N2
            B1(Z1,Z2,1)=0
            B2(Z1,Z2,1)=0
            B3(Z1,Z2,1)=0
```

```
            C (Z1,Z2,1)=1
            F (Z1,Z2,1)=0
103     CONTINUE
      ELSE IF (Z3.EQ.N3) THEN
        DO 113 Z1=1,N1
          DO 113 Z2=1,N2
            B1(Z1,Z2,N3)=0
            B2(Z1,Z2,N3)=0
            B3(Z1,Z2,N3)=0
            C (Z1,Z2,N3)=1
            F (Z1,Z2,N3)=1
113     CONTINUE
      END IF

2000  CONTINUE
C
C build the linear system:
C
      N=N1*N2*N3
      CALL DM_VPDE3D (A1,L1,L2,N1,N2,N3,A2,A3,X1,X2,X3,B1,B2,B3,
     &                C,F,A,KA,NA,N,NDIAG,NOFST,R,ICON)
      PRINT*,'DM_VPDE3D : ICON=',ICON
      IF (ICON.GT.29999) GOTO 9999
C
C write the matrix to a file:
C
      WRITE (6,'(3D23.16)') ((A(I,J),I=1,N,1000),J=1,NDIAG)
      WRITE (6,'(3I10)') (NOFST(J),J=1,NDIAG)
      WRITE (6,'(3D23.16)') (R(I),I=1,N,1000)
9999  CONTINUE
      STOP
      END
```

(4) Method

The diffusion term $-\nabla a \nabla$ is approximated by the product scheme of centered finite difference schemes of order two for the x_1-, x_2- and x_3-direction. The convective term $b\nabla$ is approximated by an upwind scheme of order one. More details are presented in [75] in Appendix A, “References.”

DM_VRADAU5

System of stiff ordinary differential equations or differential-algebraic equations (Implicit Runge-Kutta method)
CALL DM_VRADAU5 (N,FCN,X,Y,XEND,H,RTOL,ATOL,ITOL,JAC,IJAC,MLJAC, MUJAC,MAS,IMAS,MLMAS,MUMAS,SOLOUT,IOUT, WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,ICON)

(1) Function

This subroutine solves a system of stiff ordinary differential equations or differential-algebraic equations of the following form:

$$\boldsymbol{M}\boldsymbol{y}' = \boldsymbol{f}(x,\boldsymbol{y}) \qquad \boldsymbol{y}(x_0) = \boldsymbol{y}_0 \tag{1.1}$$

, where ***M*** is a constant *n*-by-*n* matrix (called mass-matrix) , ***y*** is the solution vector of size *n* (with components $y_1, y_2, ..., y_n$), ***f(x,y)*** is function vector of size *n* (with components $f_1, f_2, ..., f_n$) and $\boldsymbol{y}_0$ is the initial value at $x = x_0$ (with components $y_{01}, y_{02}, ..., y_{0n}$) .
When ***M*** is a non-singular matrix other than identity matrix, the system becomes an implicit system of ordinary differential equations. When ***M*** is a singular matrix, the system becomes a system of differential-algebraic equations.
This subroutine returns to the caller program when a numerical solution at $x_{end} (\neq x_0)$ is obtained. When integrating the system from x_0 toward x_{end} , a numerical solution after each successful step can be provided to a user's subroutine (its subroutine name is given as parameter SOLOUT).

This subroutine is based on RADAU5, a free software developed by E. Haier and G. Wanner (Universite de Geneve , as of March 2011) and provided under license agreement which copy is listed in Appendix B.

(2) Parameters

N ..... Input. Dimension of the system(N ≥ 1).

FCN .....Input. Name(EXTERNAL) of subroutine computing the value of ***f***(*x*,***y***):

```
SUBROUTINE FCN(N,X,Y,F,RPAR,IPAR)
REAL*8 X,Y(N),F(N)
F(1)=...  etc.
```

RPAR, IPAR (see below)

X ..... Input. Initial *x*-value x_0.

Output. *x*-value for which the solution has been computed(after successful return X=XEND).

Y ..... Input. Initial values for ***y***: Y(1)= y_{01} , Y(2)= y_{02} ,…, Y(N)= y_{0n} .
One-dimensional array of size *n*.
Output. Numerical solution at X (=XEND on successful return).

XEND .....Input. Final *x*-value x_{end} ($x_{end} - x_0$ may be positive or negative)

H ..... Input. Initial step size guess;
For stiff equations with initial transient, H=1.0/(norm of $\boldsymbol{f}'(x,\boldsymbol{y})$), usually 1.D-3 or 1.D-5, is good. This choice is not very important, the step size is quickly adapted (if H=0.D0, the code puts H=1.D-6).
Output. Predicted step size of the last accepted step.

RTOL,ATOL. Input. Relative and absolute error tolerances. They can be both scalars (must be variables) or else both vectors of length N. ATOL (or ATOL(I))>0 and RTOL (or RTOL(I))> 10u, where u is the round off unit.

ITOL……. Input. Switch for RTOL and ATOL:
ITOL=0:Both RTOL and ATOL are scalars. The code keeps, roughly, the local error of Y(I) below RTOL*ABS(Y(I))+ATOL.
ITOL $\neq$ 0: Both RTOL and ATOL are vectors. The code keeps, roughly, the local error of Y(I) below RTOL(I)*ABS(Y(I))+ATOL(I).

JAC…….. Input. Name(EXTERNAL) of the subroutine which computes the partial derivatives of $\boldsymbol{f}(x,\boldsymbol{y})$ with respect to $\boldsymbol{y}$ (This subroutine is only called if IJAC $\neq$ 0; Supply a dummy subroutine in the case IJAC=0).
For IJAC $\neq$ 0, this subroutine must have the form

```
SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
REAL*8 X,Y(N),DFY(LDFY,N)
DFY(1,1)= ...
```

LDFY, the column-length of the array, is furnished by the calling program.
If MLJAC=N the Jacobian is supposed to be full and the partial derivatives are stored in DFY as

$$\text{DFY(I,J)} = \frac{\partial f_i}{\partial y_j}$$

else, the Jacobian is taken as banded and the partial derivatives are stored diagonal-wise as

$$\text{DFY(I - J+MUJAC+1, J)} = \frac{\partial f_i}{\partial y_j}$$

Fig. DM_VRADAU5-1 shows how a banded Jacobian is stored in DFY in the case of N=6, MLJAC=3, and MUJAC=1, where $a_{ij} = \partial f_i / \partial y_j$ The elements marked *are not used.

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & \\ & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ & & a_{63} & a_{64} & a_{65} & a_{66} \end{pmatrix} \Rightarrow \begin{array}{|c|c|c|c|c|c|} \hline * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ \hline a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ \hline a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ \hline a_{31} & a_{42} & a_{53} & a_{64} & * & * \\ \hline a_{41} & a_{52} & a_{63} & * & * & * \\ \hline \end{array}$$

Fig. DM_VRADAU5-1

IJAC ..... Input. Switch for the computation of the Jacobian:

IJAC=0: Jacobian is computed internally by finite differences , subroutine "JAC" is never called.
IJAC $\neq$ 0: Jacobian is supplied by subroutine JAC.

MLJAC……. Input. Switch for the banded structure of the Jacobian:

MLJAC=N: Jacobian is a full matrix. The linear algebra is done by full-matrix Gauss-elimination.
$0 \le$ MLJAC<N: MLJAC is the lower bandwidth of Jacobian matrix ($\ge$ number of non-zero diagonals below the main diagonal).

MUJAC…… Input. Upper bandwidth of Jacobian matrix ($\ge$ number of non-zero diagonals above the main diagonal). Need not be defined if MLJAC=N.

MAS……. Input. Name (EXTERNAL) of subroutine computing the mass-matrix $\boldsymbol{M}$.

If IMAS=0, the matrix is assumed to be the identity matrix and needs not to be defined; Supply a dummy subroutine in this case.
If IMAS $\ne 0$, the subroutine MAS is of the form

```
SUBROUTINE MAS(N,AM,LMAS,RPAR,IPAR)
REAL*8 AM(LMAS,N)
AM(1,1)= ....
```

If MLMAS=N the mass-matrix is stored as full matrix like
AM(I,J) = $\boldsymbol{M}_{ij}$

else, the matrix is taken as banded and stored diagonal-wise as
AM(I-J+MUMAS+1,J) = $\boldsymbol{M}_{ij}$.

IMAS…… Input. Information on the mass-matrix;

IMAS=0: $\boldsymbol{M}$ is supposed to be the identity matrix, MAS is never called.
IMAS $\ne 0$: Mass-matrix is supplied.

MLMAS……Input. Switch for the banded structure of the mass-matrix:

MLMAS=N: the full matrix case. The linear algebra is done by full-matrix Gauss-elimination.
$0 \le$ MLMAS<N: MLMAS is the lower bandwidth of the matrix ($\ge$ number of non-zero diagonals below the main diagonals). MLMAS $\le$ MLJAC.

MUMAS….. Input. Upper bandwidth of mass-matrix ($\ge$ number of non-zero diagonals above the main diagonal). Need not be defined if MLMAS=N.MUMAS $\le$ MUJAC.

SOLOUT….. Input. Name (EXTERNAL) of subroutine providing the numerical solution during integration.

If IOUT $\ne 0$, it is called after every successful step. Supply a dummy subroutine if IOUT=0.
It must have the form

```
SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,
                   RPAR,IPAR,IRTRN,WORK2,IWORK2)
REAL*8 X,Y(N),CONT(LRC)
....
```

SOLOUT furnishes the solution "Y" at the NR-th grid-point "X" (thereby the initial value is the first grid-point with NR=1 and XEND is the final grid-point). "XOLD" is the preceding grid-point. "IRTRN" serves to interrupt the integration. If IRTRN is set <0, DM_VRADAU5 returns to the calling program.

----- CONTINUOUS OUTPUT: -----

During calls to "SOLOUT", a continuous solution for the interval [XOLD,X] is available through the function of type REAL*8:

>>> DM_VCONTR5(I,S,CONT,LRC,WORK2,IWORK2) <<<
which provides an approximation to the I-th component of the solution

($1 \le I \le N$) at the point S. The value S should lie in the interval [XOLD,X]. Do not change the entries of CONT(LRC),WORK2(*),and IWORK2(*).

IOUT…… Input. Switch for calling the subroutine SOLOUT:

IOUT=0: Subroutine is never called
IOUT $\neq$ 0: Subroutine is available for output.

WORK…….. Work area. One-dimensional array of size LWORK.

WORK(1), WORK(2),.., WORK(20) serve as parameters for the code. For standard use of the code WORK(1),..,WORK(20) must be set to zero before calling. See below for a more sophisticated use.
WORK(21),..,WORK(LWORK) serve as working space for all vectors and matrices.
"LWORK" must be at least
N*(LJAC+LMAS+3*LE+12)+20
where
LJAC=N if MLJAC=N (full Jacobian)
LJAC=MLJAC+MUJAC+1 if MLJAC<N (banded JAC.)
and
LMAS=0 if IMAS=0
LMAS=N if IMAS $\neq$ 0 and MLMAS=N (full)
LMAS=MLMAS+MUMAS+1 if MLMAS<N (banded mass-***M***.)
and
LE=N if MLJAC=N (full Jacobian)
LE=2*MLJAC+MUJAC+1 if MLJAC<N (banded JAC.)

In the usual case where the Jacobian is full and the mass-matrix is the identity (IMAS=0), the minimum storage requirement is
LWORK = 4*N*N+12*N+20.
If IWORK(9)=M1>0 then "LWORK" must be at least
N*(LJAC+12)+(N - M1)*(LMAS+3*LE)+20
where in the definitions of LJAC, LMAS and LE the number N can be replaced by N - M1.

LWORK……Input. Declared length of array “WORK”.

IWORK…… Work area. One-dimensional integer array of size LIWORK.

IWORK(1),IWORK(2),...,IWORK(20) serve as parameters for the code. For standard use, set IWORK(1),.., IWORK(20) to zero before calling.
IWORK(21),...,IWORK(LIWORK) serve as working space.
"LIWORK" must be at least 3*N+20.

Output. IWORK(14) through IWORK(20) contain statistics at completion of integration up to XEND.
IWORK(14) NFCN Number of function evaluations(those for numerical evaluation of the Jacobian are not counted)
IWORK(15) NJAC Number of Jacobian evaluations (either analytically or numerically)
IWORK(16) NSTEP Number of computed steps
IWORK(17) NACCPT Number of accepted steps
IWORK(18) NREJCT Number of rejected steps(due to error test) ,(step rejections in the first step are not counted)
IWORK(19) NDEC Number of LU-decompositions of both matrices
IWORK(20) NSOL Number of forward-backward substitutions, of both systems; The NSTEP forward-backward substitutions,

needed for step size selection, are not counted

LIWORK….. Input. Declared length of array “IWORK”.

RPAR,IPAR.. Real and integer parameters (or parameter arrays) which can be used for communication between your calling program and subroutines FCN, JAC, MAS, and SOLOUT.

ICON…… Output. Condition code. See Table DM_VRADAU5-1 Condition codes.

Sophisticated Setting of Parameters:

Several parameters of the code are tuned to make it work well. They may be defined by setting WORK(1),... as well as IWORK(1),... different from zero. For zero input, the code chooses default values:

IWORK(1)… Input. If IWORK(1) $\neq 0$, the code transforms the Jacobian matrix to Hessenberg form. This is particularly advantageous for large systems with full Jacobian. It does not work for banded Jacobian (MLJAC<N) and not for implicit systems (IMAS $\neq 0$).

IWORK(2)… Input. This is the maximal number of allowed steps. The default value (for IWORK(2)=0) is 100000.

IWORK(3)… Input. The maximum number of Newton iterations for the solution of the implicit system in each step. The default value (for IWORK(3)=0) is 7.

IWORK(4)… Input. If IWORK(4)=0 the extrapolated collocation solution is taken as starting value for Newton’s method. If IWORK(4) $\neq 0$ zero starting values are used. The latter is recommended if Newton’s method has difficulties with convergence (This is the case when NSTEP is larger than NACCPT + NREJCT; See output parameters). Default is IWORK(4)=0.

The following 3 parameters are important for differential-algebraic systems of index > 1. The function-subroutine should be written such that the index 1,2,3 variables appear in this order. In estimating the error the index 2 variables are multiplied by H, the index 3 variables by H**2. (In the cases where ***M*** is the identity matrix or non-singular, the system is just ordinary differential equations, so all variables are index 1 variables and it is sufficient to set 3 parameters to zero.)
If the user sets any of these 3 parameters different from 0 ,the sum of 3 parameters must be N.

IWORK(5)… Input. Dimension of the index 1 variables.

IWORK(6)… Input. Dimension of the index 2 variables. Default IWORK(6)=0.

IWORK(7)… Input. Dimension of the index 3 variables. Default IWORK(7)=0.

IWORK(8)… Input. Switch for step size strategy.
If IWORK(8) =1 modified predictive controller (Gustafsson)
If IWORK(8) >1 classical step size control
The default value (for IWORK(8)=0) is IWORK(8)=1. The choice IWORK(8)=1 seems to produce safer results. For simple problems, the choice IWORK(8)> 1 produces often slightly faster runs.

If the differential system has the special structure that
Y(I)' = Y(I+M2) for I=1,...,M1,
with M1 a multiple of M2, a substantial gain in computer time can be achieved by setting the parameters IWORK(9) and IWORK(10). For example, second order systems

$\boldsymbol{p}'' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{p}')$ can be rewritten as

$$\boldsymbol{p}' = \boldsymbol{v}$$
$$\boldsymbol{v}' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v})$$

, where $\boldsymbol{p}$ and $\boldsymbol{v}$ are vectors of dimension N/2. In this case one has to put M1=M2=N/2. For M1>0 some of the input parameters have different meanings:

JAC……Input. Only the elements of the non-trivial part of the Jacobian have to be stored. For example, with the above first order system reduced from the second order system, subroutine JAC has to store only

$$\begin{pmatrix} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{p}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{v}} \end{pmatrix}$$

, which is N/2 × N non-trivial matrix.
Suppose Y and F are solution vector and right hand side function vector , respectively, of resulting first order system.
If MLJAC=N-M1 the Jacobian is supposed to be full;

$$\mathrm{DFY(\ I,\ J\ )} = \frac{\partial \mathrm{F(I+M1)}}{\partial \mathrm{Y(J)}}\ ,\ \mathrm{I=1,\ldots,N-M1,\quad J=1,\ldots,N}$$

If $0 \le$ MLJAC<N-M1 the Jacobian is banded (M1 = M2 * MM);

$$\mathrm{DFY(I-J+MUJAC+1,J+K\times M2)} = \frac{\partial \mathrm{F(I+M1)}}{\partial \mathrm{Y(J+K\times M2)}}$$

I=1, ... ,N-M1, J=1, …, M2, K=0,..,MM
In the banded case, N=M1+ M2 has to be met.

MLJAC..Input.
MLJAC=N-M1 : if the non-trivial part of the Jacobian is full.
$0 \le$ MLJAC<N-M1: if the (MM+1) submatrices (M1= M2 * MM),

$$\frac{\partial \mathrm{F(I+M1)}}{\partial \mathrm{Y(J+K\times M2)}}\ ,\ \mathrm{I=1,\ldots,N-M1\ ,\ J=1,\ldots,M2,\ K=0,\ldots,MM}$$

are all banded , and MLJAC is the maximal lower bandwidth of these MM+1 submatrices.

MUJAC…Input.
Maximal upper bandwidth of these MM+1 submatrices. Need not be defined if MLJAC=N-M1.

MAS……Input.
If IMAS=0 this matrix is assumed to be the identity and need not be defined. Supply a dummy subroutine in this case.
If IMAS $\neq 0$ it is assumed that only the elements of right lower block of dimension N-M1 differ from that of the identity matrix and only the elements of right lower block of dimension N-M1 must be given in subroutine MAS. For example , consider the following system.

$$\boldsymbol{M}\boldsymbol{p}'' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{p}')$$

This can be rewritten as

$$\boldsymbol{p}' = \boldsymbol{v}$$
$$\boldsymbol{M}\boldsymbol{v}' = \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v})$$

and expressed in the following form.

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{pmatrix} \begin{pmatrix} \boldsymbol{p}' \\ \boldsymbol{v}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{g}(x, \boldsymbol{p}, \boldsymbol{v}) \end{pmatrix}$$

In this case the coefficient matrix of the left hand side corresponds to $\boldsymbol{M}$ in (1.1). Denoting by $\boldsymbol{M}$ the coefficient matrix of the left hand side, if MLMAS=N-M1

the right lower block is supposed to be full; the array AM in the subroutine MAS should be set as
AM(I,J)= M(I+M1, J+M1) , I=1,…,N - M1 , J=1,…, N - M1.
If MLMAS ≠ N - M1 the right low block is supposed to be banded:
AM(I - J+MUMAS+1, J)= M(I+M1, J+M1)

MLMAS…Input.
MLMAS=N - M1: If the non-trivial part of ***M*** is full.
0 ≤ MLMAS<N - M1: Lower bandwidth of the mass matrix. MLMAS ≤ MLJAC must be met.

MUMAS….Input.
Upper bandwidth of the mass matrix. MUMAS ≤ MUJAC must be met. Need not be defined if MLMAS=N - M1.

IWORK(9)… Input. The value of M1 (≥ 0). Default M1=0.

IWORK(10).. Input. The value of M2 (≥ 0). Default M2=M1.
If IWORK(9) >0 , IWORK(9)+IWORK(10) ≤ N must be met.

WORK(1)…. Input. The round off unit u. DMACH() ≤ WORK(1)<1.0D0 must be met.
Default u=DMACH().

WORK(2)…. Input. The safety factor in step size prediction.
0.001D0<WORK(2)<1.0D0 must be met. Default 0.9D0.

WORK(3)…. Input.
Decides whether the Jacobian should be recomputed; increase WORK(3), to 0.1 say, when Jacobian evaluations are costly. For small systems WORK(3) should be smaller (0.001D0, say). Negative WORK(3) forces the code to compute the Jacobian after every accepted step.
Default 0.001D0. WORK(3) < 1.0D0 must be met.

WORK(4)…. Input.
Stopping criterion for Newton's method, usually chosen <1. Smaller values of WORK(4) make the code slower , but safer.
DEFAULT MAX(10u/TOLST,MIN(0.03D0, $\sqrt{\text{TOLST}}$)) , where u is the round off unit, TOLST=0.1·RTOL**(2/3) , and RTOL=RTOL(1) when RTOL is vector. WORK(4) > u/TOLST must be met.

WORK(5),

WORK(6)…. Input.
If WORK(5) < HNEW/HOLD < WORK(6), then the step size is not changed. This saves, together with a large WORK(3), LU-decompositions and computing time for large systems. For smaller systems one may have WORK(5)=1.D0, WORK(6)=1.2D0, for large full systems WORK(5)=0.99D0, WORK(6)=2.D0 might be good.
DEFAULTS WORK(5)=1.D0, WORK(6)=1.2D0 .
WORK(5) ≤ 1.0D0 and WORK(6) ≥ 1.0D0 must be met.

WORK(7)…. Input. Maximal step size. Default $x_{end} - x_0$.

WORK(8),

WORK(9)…. Input. Parameters for step size selection.
The new step size is chosen subject to the restriction
WORK(8) ≤ HNEW/HOLD ≤ WORK(9)

Default values : WORK(8)=0.2D0,WORK(9)=8.D0.
WORK(8) ≤ 1.0D0 and WORK(9) ≥ 1.0D0 must be met.

Table DM_VRADAU5-1 Condition codes

Code	Meaning	Processing
0	No error	–
100	In subroutine SOLOUT, parameter IRTRN was set to be negative.	Processing is discontinued. Solutions obtained so far were correct.
10000	Number of steps exceeded the value specified in IWORK(2).	Processing is discontinued. Integration did not reach XEND. The user can try a larger value for IWORK(2).
21000	Step size became too small.	Processing is discontinued.
22000	Matrix was repeatedly singular.	
30000	There was an inconsistent input.	

(3) Comments on use

a. Notes

1) Role of SOLOUT
During integration from x_0 to x_{end} this subroutine provides numerical solutions after every accepted step to the subroutine SOLOUT when IOUT ≠ 0.
Namely, when $x_0 < x_{end}$, every accepted step results in a sequence of grid-point such as

$$x_0 < x_1 < x_2 < \ldots < x_{end}$$

and x_i and solutions at x_i are passed to SOLOUT (x_0 and x_{end} included). x_i is determined under step size control to meet required accuracies.
If the user requires solutions at intended grid-points, the function subprogram DM_VCONTR5 can be used for dense output. For instance, if solutions are required at equally spaced grid-points one can refer to Example 1 below.
Note that repeated calls to DM_VRADAU5 by incrementing XEND is inefficient way for that purpose.

2) Thread parallelization of user's subroutines
In any of user's subroutines FCN, JAC, MAS, and SOLOUT, the user can use OpenMP parallelization when necessary.

3) Index and initial values for differential-algebraic equations
In the model $\boldsymbol{M}\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$ if $\boldsymbol{M}$ is non-singular the system is just ordinary differential equations, and "index" of variables in $\boldsymbol{y}$ is 1. In this case IWORK(5) ~IWORK(7) should be set to 0.
If $\boldsymbol{M}$ is singular, the system becomes a differential-algebraic equations, and IWORK(5) ~IWORK(7) and initial values should be given carefully. Here is a brief guideline.
For singular $\boldsymbol{M}$, we can decompose the matrix (e.g., by Gaussian elimination with total pivoting) as

$$\boldsymbol{M} = \boldsymbol{S}\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}\boldsymbol{T}$$

where $\boldsymbol{S}$ and $\boldsymbol{T}$ are n-by-n non-singular matrices , and $\boldsymbol{I}$ is the identity matrix of smaller size. Inserting this into (1.1), multiplying by $\boldsymbol{S}^{-1}$, and using the transformed variables

$$\boldsymbol{Ty} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{w} \end{pmatrix}$$

gives

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}' \\ \boldsymbol{w}' \end{pmatrix} = \boldsymbol{S}^{-1} \boldsymbol{f}(x, \boldsymbol{T}^{-1} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{w} \end{pmatrix}) =: \begin{pmatrix} \boldsymbol{g}(x, \boldsymbol{u}, \boldsymbol{w}) \\ \boldsymbol{h}(x, \boldsymbol{u}, \boldsymbol{w}) \end{pmatrix}$$

or

$$\boldsymbol{u}' = \boldsymbol{g}(x, \boldsymbol{u}, \boldsymbol{w})$$
$$\boldsymbol{0} = \boldsymbol{h}(x, \boldsymbol{u}, \boldsymbol{w})$$

These are called Hessenberg form of the differential-algebraic equations, where the system is split into a smaller ordinary differential equations and a smaller algebraic equations. The Hessenberg forms are often encountered in practice, and can be said as differential equations with algebraic constraints. Below, we give some typical Hessenberg forms which illustrate index 1,2 and 3 variables. We omit, from now on, the independent variable in equations to simplify mathematical expressions.

a) System of index 1

Let us consider the following system

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}) \quad (3.1a)$$
$$\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) \quad (3.1b)$$

, where $\boldsymbol{y}$ and $\boldsymbol{z}$ are unknown function vectors, and sum of each size is n. The mass-matrix $\boldsymbol{M}$ here is

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$$

Differentiating (3.1b) and using (3.1a) we get

$$\boldsymbol{0} = \boldsymbol{g}_y(\boldsymbol{y}, \boldsymbol{z}) \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}) + \boldsymbol{g}_z(\boldsymbol{y}, \boldsymbol{z}) \boldsymbol{z}' \quad (3.1c)$$

, where $\boldsymbol{g}_y(\boldsymbol{y}, \boldsymbol{z})$ and $\boldsymbol{g}_z(\boldsymbol{y}, \boldsymbol{z})$ are $\partial \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) / \partial \boldsymbol{y}$ and $\partial \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) / \partial \boldsymbol{z}$ respectively. If $\boldsymbol{g}_z(\boldsymbol{y}, \boldsymbol{z})$, the coefficient of $\boldsymbol{z}'$, is non-singular in a neighborhood of the solution we get

$$\boldsymbol{z}' = -\boldsymbol{g}_z^{-1}(\boldsymbol{y}, \boldsymbol{z}) \boldsymbol{g}_y(\boldsymbol{y}, \boldsymbol{z}) \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z})$$

In this case, $\boldsymbol{y}$ and $\boldsymbol{z}$ are index 1 variables. Initial values $\boldsymbol{y}_0$ and $\boldsymbol{z}_0$ should be given to satisfy (3.1b).

b) System of index 2

Next, we consider the following

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}) \quad (3.2a)$$
$$\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{y}) \quad (3.2b)$$

, where $\boldsymbol{z}$ is absent in the algebraic constraint and $\boldsymbol{M}$ is as follows.

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$$

Differentiating (3.2b) gives

$$\boldsymbol{0} = \boldsymbol{g}_y(\boldsymbol{y}) \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{z}) \quad (3.2c)$$

Differentiating (3.2c) gives the coefficient of $\boldsymbol{z}'$ as

$$\boldsymbol{g}_y(\boldsymbol{y}) \boldsymbol{f}_z(\boldsymbol{y}, \boldsymbol{z}) \quad (3.2d)$$

If (3.2d) is non-singular in a neighborhood of the solution, $\boldsymbol{y}$ is index 1 variable and $\boldsymbol{z}$ is index 2 variable. Initial values $\boldsymbol{y}_0$ and $\boldsymbol{z}_0$ should be given to satisfy not

only (3.2b) but (3.2c).

c) System of index 3
Finally, we consider the following system.

$$\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y},\boldsymbol{z}) \quad (3.3a)$$

$$\boldsymbol{z}' = \boldsymbol{k}(\boldsymbol{y},\boldsymbol{z},\boldsymbol{u}) \quad (3.3b)$$

$$\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{y}) \quad (3.3c)$$

Here the sum of length of $\boldsymbol{y}$, $\boldsymbol{z}$, and $\boldsymbol{u}$ is n. $\boldsymbol{M}$ is written as

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}.$$

Differentiating (3.3c) and using (3.3a) we get

$$\boldsymbol{0} = \boldsymbol{g}_y \boldsymbol{f} \quad (3.3d)$$

Differentiating (3.3d) and using (3.3a,b) we get

$$\boldsymbol{0} = \boldsymbol{g}_{yy}(\boldsymbol{f},\boldsymbol{f}) + \boldsymbol{g}_y \boldsymbol{f}_y \boldsymbol{f} + \boldsymbol{g}_y \boldsymbol{f}_z \boldsymbol{k} \quad (3.3e)$$

, where the first term of the right hand side means matrix vector multiplication with the matrix $\boldsymbol{g}_{yy}$ obtained by differentiating matrix $\boldsymbol{g}_y$ and the vector $\boldsymbol{f}$.

Furthermore, differentiating (3.3e) brings about $\boldsymbol{u}'$. If its coefficient, written as $\boldsymbol{g}_y \boldsymbol{f}_z \boldsymbol{k}_u$, is non-singular in a neighborhood of the solution, $\boldsymbol{y}$ is index 1 variable, $\boldsymbol{z}$ is index 2 variable, and $\boldsymbol{u}$ is index 3 variable in the original system (3.3a,b,c). Initial values $\boldsymbol{y}_0$, $\boldsymbol{z}_0$ and $\boldsymbol{u}_0$ should be given to satisfy the three constraints (3.3 c,d,e).

b. Example

■ Example 1:Ordinary differential equations of the form $\boldsymbol{y}' = \boldsymbol{f}(x,\boldsymbol{y})$

Let us consider a simple system:

$$y_1' = y_2$$

$$y_2' = \frac{((1-y_1^2)y_2 - y_1)}{\varepsilon}, \quad \varepsilon = 10^{-6}$$

$$y_1(0) = 2, y_2(0) = 0$$

Suppose we want to find solutions at $x = 1,2,\ldots,11$ and print them out. In this problem, the Jacobian matrix $\partial \boldsymbol{f}/\partial \boldsymbol{y}$ is as follows.

$$\begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ (-2y_1y_2 - 1)/\varepsilon & (1-y_1^2)/\varepsilon \end{pmatrix}$$

We provide subroutine JVPOL as real argument of JAC.

```
IMPLICIT REAL*8 (A-H,O-Z)
PARAMETER (ND=2,LWORK=4*ND*ND+12*ND+20,LIWORK=3*ND+20)
DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
DIMENSION RPAR(2)

EXTERNAL FVPOL,JVPOL,SOLOUT
RPAR(1)=1.0D-6
RPAR(2)=0.2D0
N=ND
IJAC=1
```

```
      MLJAC=N
      IMAS=0
      IOUT=1
      X=0.0D0
      Y(1)=2.0D0
      Y(2)=-0.66D0
      XEND=11.0D0
      RTOL=1.0D-4
      ATOL=1.0D0*RTOL
      ITOL=0
      H=1.0D-6
      DO I=1,20
         IWORK(I)=0
         WORK(I)=0.D0
      END DO
      CALL DM_VRADAU5(N,FVPOL,X,Y,XEND,H,
     &                RTOL,ATOL,ITOL,
     &                JVPOL,IJAC,MLJAC,MUJAC,
     &                FVPOL,IMAS,MLMAS,MUMAS,
     &                SOLOUT,IOUT,
     &                WORK,LWORK,IWORK,LIWORK,
     &   RPAR,IPAR,ICON)
      WRITE(6,*) 'ICON=', ICON
      WRITE (6,99) X,Y(1),Y(2)
 99   FORMAT(1X,'X =',F5.2,'    Y =',2E18.10)
      STOP
      END
C
C
      SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN,
     &   WORK2,IWORK2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),CONT(LRC),RPAR(*)
      IF (NR.EQ.1) THEN
         WRITE (6,99) X,Y(1),Y(2),NR-1
      ELSE
 10      CONTINUE
         IF (X.GE.RPAR(2)) THEN
C --- CONTINUOUS OUTPUT FOR RADAU5
      WRITE (6,99) RPAR(2),DM_VCONTR5(1,RPAR(2),CONT,LRC,WORK2,
     &  IWORK2),DM_VCONTR5(2,RPAR(2),CONT,LRC,WORK2,IWORK2),
     &  NR-1
            RPAR(2)=RPAR(2)+0.2D0
            GOTO 10
         END IF
      END IF
 99   FORMAT(1X,'X =',F5.2,'    Y =',2E18.10,'    NSTEP =',I4)
      RETURN
      END
C
C
      SUBROUTINE FVPOL(N,X,Y,F,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
```

```
      DIMENSION Y(N),F(N),RPAR(*)
      F(1)=Y(2)
      F(2)=((1-Y(1)**2)*Y(2)-Y(1))/RPAR(1)
      RETURN
      END
C
C
      SUBROUTINE JVPOL(N,X,Y,DFY,LDFY,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),DFY(LDFY,N),RPAR(*)
      DFY(1,1)=0.0D0
      DFY(1,2)=1.0D0
      DFY(2,1)=(-2.0D0*Y(1)*Y(2)-1.0D0)/RPAR(1)
      DFY(2,2)=(1.0D0-Y(1)**2)/RPAR(1)
      RETURN
      END
```

■ Example 2: $\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$ with banded Jacobian.

Consider the following partial differential equations. "*t*" means time and "*x*" is scalar space variable.

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}$$

$0 \le x \le 1,\ A = 1,\ B = 3,\ \alpha = 1/50$

Boundary conditions : $u(0,t) = u(1,t) = 1, v(0,t) = v(1,t) = 3$

Initial values : $u(x,0) = 1 + \frac{1}{2}\sin(2\pi x), v(x,0) = 3$

We replace the second spatial derivatives by finite differences on a grid of N points, $x_i = i/(N+1)$ $(1 \le i \le \mathrm{N})$, $\Delta x = 1/(N+1)$ and then obtain a system of ordinary differential equations with independent variable "t" and 2N unknowns $u_i = u(t, x_i)$ and $v_i = v(t, x_i)$.

$$u_i' = 1 + u_i^2 v_i - 4u_i + \alpha/(\Delta x)^2 (u_{i-1} - 2u_i + u_{i+1})$$

$$v_i' = 3u_i - u_i^2 v_i + \alpha/(\Delta x)^2 (v_{i-1} - 2v_i + v_{i+1})$$

$$u_0(t) = u_{N+1}(t) = 1, v_0(t) = v_{N+1}(t) = 3$$

$$u_i(0) = 1 + \frac{1}{2}\sin(2\pi x_i), v_i(0) = 3, i = 1,2,\ldots,N$$

When using this subroutine we define $\boldsymbol{y}$ as $\boldsymbol{y} = (u_1, v_1, u_2, v_2, \ldots, u_N, v_N)^{\mathrm{T}}$. Then the Jacobian becomes a banded matrix with the upper and lower bandwidth 2. In the following example, we set N=500, XEND=10, and IOUT=0 and print some components of the solutions at XEND.

```
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (ND=1000,NL=2,NU=2)
      PARAMETER (LWORK=(7*NL+4*NU+16)*ND+20,LIWORK=3*ND+20)
      DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
      DIMENSION RPAR(2)
```

```
      EXTERNAL FBRUS,JBRUS,SOLOUT
      PI=3.14159265358979324D0
      N=500
      N2=2*N
      USDELQ=(DBLE(N+1))**2
      GAMMA=0.02D0*USDELQ
      GAMMA2=2.D0*GAMMA
      RPAR(1)=GAMMA
      RPAR(2)=GAMMA2
      X=0.D0
      XEND=10.D0
      ANP1=N+1
      DO 1 I=1,N
      XI=I/ANP1
      Y(2*I)=3.D0
   1  Y(2*I-1)=1.D0+0.5D0*DSIN(2.D0*PI*XI)
      IJAC=1
C   Jacobian is a banded matrix.
      MLJAC=NL
      MUJAC=NU
      IMAS=0
C   Output Routine is not used.
      IOUT=0
      RTOL=1.0D-6
      ATOL=RTOL
      ITOL=0
      H=1.0D-6
      DO I=1,20
         WORK(I)=0.D0
         IWORK(I)=0
      END DO
      CALL DM_VRADAU5(N2,FBRUS,X,Y,XEND,H,
     &                RTOL,ATOL,ITOL,
     &                JBRUS,IJAC,MLJAC,MUJAC,
     &                FBRUS,IMAS,MLMAS,MUMAS,
     &                SOLOUT,IOUT,
     &                WORK,LWORK,IWORK,LIWORK,
     &  RPAR,IPAR,ICON)
      WRITE(6,*) 'ICON=',ICON
      WRITE(6,99) Y(1),Y(2),Y(N2-1),Y(N2)
 99   FORMAT(1X,4F18.10)
      STOP
      END
C
      SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN,
     &  WORK2,IWORK2)
      RETURN
      END
C
      SUBROUTINE FBRUS(N2,X,Y,F,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N2),F(N2),RPAR(*)
      N=N2/2
```

```
      GAMMA=RPAR(1)
      GAMMA2=RPAR(2)
      I=1
      IU=2*I-1
      IV=2*I
      UI=Y(IU)
      VI=Y(IV)
          UIM=1.D0
          VIM=3.D0
          UIP=Y(IU+2)
          VIP=Y(IV+2)
      PROD=UI*UI*VI
      F(IU)=1.D0+PROD-4.D0*UI+GAMMA*(UIM-2.D0*UI+UIP)
      F(IV)=3.D0*UI-PROD+GAMMA*(VIM-2.D0*VI+VIP)
      DO 5 I=2,N-1
      IU=2*I-1
      IV=2*I
      UI=Y(IU)
      VI=Y(IV)
          UIM=Y(IU-2)
          VIM=Y(IV-2)
          UIP=Y(IU+2)
          VIP=Y(IV+2)
      PROD=UI*UI*VI
      F(IU)=1.D0+PROD-4.D0*UI+GAMMA*(UIM-2.D0*UI+UIP)
      F(IV)=3.D0*UI-PROD+GAMMA*(VIM-2.D0*VI+VIP)
    5 CONTINUE
      I=N
      IU=2*I-1
      IV=2*I
      UI=Y(IU)
      VI=Y(IV)
          UIM=Y(IU-2)
          VIM=Y(IV-2)
          UIP=1.D0
          VIP=3.D0
      PROD=UI*UI*VI
      F(IU)=1.D0+PROD-4.D0*UI+GAMMA*(UIM-2.D0*UI+UIP)
      F(IV)=3.D0*UI-PROD+GAMMA*(VIM-2.D0*VI+VIP)
      RETURN
      END
C
      SUBROUTINE JBRUS(N2,X,Y,DFY,LDFY,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N2),DFY(LDFY,N2),RPAR(*)
      N=N2/2
      GAMMA=RPAR(1)
      GAMMA2=RPAR(2)
      DO 1 I=1,N
      IU=2*I-1
      IV=2*I
      UI=Y(IU)
      VI=Y(IV)
```

```
      UIVI=UI*VI
      UI2=UI*UI
      DFY(3,IU)=2.D0*UIVI-4.D0-GAMMA2
      DFY(2,IV)=UI2
      DFY(4,IU)=3.D0-2.D0*UIVI
      DFY(3,IV)=-UI2-GAMMA2
      DFY(2,IU)=0.D0
      DFY(4,IV)=0.D0
1     CONTINUE
      DO 2 I=1,N2-2
      DFY(1,I+2)=GAMMA
      DFY(5,I)=GAMMA
2     CONTINUE
      RETURN
      END
```

■ Example 3:Second order system $\boldsymbol{y}'' = \boldsymbol{f}(x, \boldsymbol{y}, \boldsymbol{y}')$

Next , we consider a partial differential equations defined in rectangular plate $\Omega = \{(x, y); 0 \le x \le 2, 0 \le y \le 4/3\}$:

$$\frac{\partial^2 u}{\partial t^2} + \omega \frac{\partial u}{\partial t} + \sigma \Delta \Delta u = f(x, y, t), \text{ where } \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Boundary conditions: $u|_{\partial\Omega} = 0, \Delta u|_{\partial\Omega} = 0$

Initial conditions: $u(x, y, 0) = 0, \frac{\partial u}{\partial t}(x, y, 0) = 0$

The plate Ω is discretized on a grid 8×5 interior points $x_i = ih, y_j = jh, i = 1,2,\ldots,8, j = 1,2,\ldots,5$, $h = 2/9$.

We replace the special derivatives by finite differences, then setting $v_{ij} = u'_{ij}$ gives the following ordinary differential system.

$$u'_{ij} = v_{ij}$$

$$v'_{ij} = -\omega v_{ij} - \frac{\sigma}{h^4}(20u_{ij} - 8u_{i-1\,j} - 8u_{i+1\,j} + 2u_{i+1\,j+1} + 2u_{i+1\,j-1}$$
$$+ 2u_{i-1\,j-1} + 2u_{i-1\,j+1} + u_{i-2\,j} + u_{i+2\,j} + u_{i\,j-2} + u_{i\,j+2}) + f(x_i, y_j, t)$$

With mapping k=i+8(j - 1) from (i,j) , we set $y_k = u_{ij}$ and $y_{k+40} = v_{ij}$ .Then we obtain system with $(y_1, y_2, \ldots, y_{40}, y_{41}, \ldots, y_{80})^{\mathrm{T}}$ as unknown vector. In the following program we set IWORK(9)=40 and subroutine JPLATSB computes only non-trivial part of the Jacobian.

$$\omega = 1000, \sigma = 100$$

$$f(x, y, t) = \begin{cases} 2000(\mathbf{e}^{-5(t-x-2)^2} + \mathbf{e}^{-5(t-x-5)^2}) & \text{if } y = y_2 \text{ or } y_4 \\ 0 & \text{for all other } y \end{cases}$$

```
IMPLICIT REAL*8 (A-H,O-Z)
PARAMETER(MX=8,MY=5)
PARAMETER (ND=2*MX*MY,LWORK=4*ND*ND+12*ND+20,LIWORK=3*ND+20)
DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
EXTERNAL FPLATE,JPLATSB,SOLOUT
```

```
      DIMENSION RPAR(4),IPAR(7)

      NX=MX
      NY=MY
      NACHS1=2
      NACHS2=4
      NXM1=NX-1
      NYM1=NY-1
      NDEMI=NX*NY
      OMEGA=1000.D0
      STIFFN=100.D0
      WEIGHT=200.D0
      DENOM=NX+1
      DELX=2.D0/DENOM
      USH4=1.D0/(DELX**4)
      FAC=STIFFN*USH4
      N=ND
      IMAS=0
C --- OUTPUT ROUTINE IS USED DURING INTEGRATION
      IOUT=1
C --- INITIAL VALUES
      X=0.0D0
      DO  I=1,N
      Y(I)=0.D0
      END DO
C --- REQUIRED TOLERANCE
        RTOL=1.0D-6
        ATOL=RTOL*1.0D-3
        ITOL=0
C --- INITIAL STEP SIZE
        H=1.0D-2
C --- SET DEFAULT VALUES
        DO I=1,20
           WORK(I)=0.D0
           IWORK(I)=0
        END DO
C --- SECOND ORDER OPTION AND BANDED
           IJAC=1
           IWORK(9)=N/2
           MLJAC=2*MX
           MUJAC=2*MX
C --- ENDPOINT OF INTEGRATION
        XEND=7.D0
C --- COMMUNICATION VALUES
      IPAR(1)=NX
      IPAR(2)=NXM1
      IPAR(3)=NY
      IPAR(4)=NYM1
      IPAR(5)=NDEMI
      IPAR(6)=NACHS1
      IPAR(7)=NACHS2
      RPAR(1)=OMEGA
      RPAR(2)=DELX
```

```
      RPAR(3)=FAC
      RPAR(4)=WEIGHT

C --- CALL OF THE SUBROUTINE RADAU5
        CALL DM_VRADAU5(N,FPLATE,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JPLATSB,IJAC,MLJAC,MUJAC,
     &                  FPLATE,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,
     &   RPAR,IPAR,ICON)
      WRITE(6,*) 'ICON=',ICON
      DO  K=1,N
      WRITE (6,*) Y(K)
      END DO
      STOP
      END
C
      SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN,
     &  WORK2,IWORK2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),CONT(LRC)
      NHALF=N/2
      WRITE (6,991) X,NHALF,Y(1),Y(NHALF),NR-1
 991  FORMAT(1X,'X =',F9.5,' Y(1) and Y(',I3,')=',2F18.10,
     &  ' NSTEP =',I4)
      RETURN
      END
C
      SUBROUTINE FPLATE (N, X, Y, F, RPAR, IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N), F(N)
      DIMENSION RPAR(*),IPAR(*)
      NX=IPAR(1)
      NXM1=IPAR(2)
      NY=IPAR(3)
      NYM1=IPAR(4)
      NDEMI=IPAR(5)
      NACHS1=IPAR(6)
      NACHS2=IPAR(7)
      OMEGA=RPAR(1)
      DELX=RPAR(2)
      FAC=RPAR(3)
      WEIGHT=RPAR(4)

      DO 1 I=1,NX
      DO 1 J=1,NY
      K=I+NX*(J-1)
C -------- SECOND DERIVATIVE ----
      F(K)=Y(K+NDEMI)
C ------ CENTRAL POINT---
      UC=16.D0*Y(K)
      IF(I.GT.1)THEN
```

```
            UC=UC+Y(K)
            UC=UC-8.D0*Y(K-1)
         END IF
         IF(I.LT.NX)THEN
            UC=UC+Y(K)
            UC=UC-8.D0*Y(K+1)
         END IF
         IF(J.GT.1)THEN
            UC=UC+Y(K)
            UC=UC-8.D0*Y(K-NX)
         END IF
         IF(J.LT.NY)THEN
            UC=UC+Y(K)
            UC=UC-8.D0*Y(K+NX)
         END IF
         IF(I.GT.1 .AND.J.GT.1 )UC=UC+2.D0*Y(K-NX-1)
         IF(I.LT.NX.AND.J.GT.1 )UC=UC+2.D0*Y(K-NX+1)
         IF(I.GT.1 .AND.J.LT.NY)UC=UC+2.D0*Y(K+NX-1)
         IF(I.LT.NX.AND.J.LT.NY)UC=UC+2.D0*Y(K+NX+1)
         IF(I.GT.2)UC=UC+Y(K-2)
         IF(I.LT.NXM1)UC=UC+Y(K+2)
         IF(J.GT.2)UC=UC+Y(K-2*NX)
         IF(J.LT.NYM1)UC=UC+Y(K+2*NX)
         IF(J.EQ.NACHS1.OR.J.EQ.NACHS2)THEN
            XI=I*DELX
            FORCE=EXP(-5.D0*(X-XI-2.D0)**2)+EXP(-5.D0*(X-XI-5.D0)**2)
         ELSE
            FORCE=0.D0
         END IF
         F(K+NDEMI)=-OMEGA*Y(K+NDEMI)-FAC*UC+FORCE*WEIGHT
1     CONTINUE
      RETURN
      END

      SUBROUTINE JPLATSB(N,X,Y,DFY,LDFY,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),DFY(LDFY,N)
      DIMENSION RPAR(*),IPAR(*)
      NX=IPAR(1)
      NXM1=IPAR(2)
      NY=IPAR(3)
      NYM1=IPAR(4)
      NDEMI=IPAR(5)
      OMEGA=RPAR(1)
      FAC=RPAR(3)

      DO 1 I=1,LDFY
      DO 1 J=1,N
1     DFY(I,J)=0.D0
      MU=2*NX+1
      FAC2=FAC*2.0D0
      FAC8=FAC*8.0D0
      FAC16=FAC*16.0D0
```

```
      DO 2 I=1,NX
      DO 2 J=1,NY
      K=I+NX*(J-1)
      DFY(MU,K)=-FAC16
      IF(I.GT.1)THEN
         DFY(MU,K)=DFY(MU,K)-FAC
         DFY(MU+1,K-1)=FAC8
      END IF
      IF(I.LT.NX)THEN
         DFY(MU,K)=DFY(MU,K)-FAC
         DFY(MU-1,K+1)=FAC8
      END IF
      IF(J.GT.1)THEN
         DFY(MU,K)=DFY(MU,K)-FAC
         DFY(MU+NX,K-NX)=FAC8
      END IF
      IF(J.LT.NY)THEN
         DFY(MU,K)=DFY(MU,K)-FAC
         DFY(MU-NX,K+NX)=FAC8
      END IF
      IF(I.GT.1 .AND.J.GT.1 )DFY(MU+NX+1,K-NX-1)=-FAC2
      IF(I.LT.NX.AND.J.GT.1 )DFY(MU+NX-1,K-NX+1)=-FAC2
      IF(I.GT.1 .AND.J.LT.NY)DFY(MU-NX+1,K+NX-1)=-FAC2
      IF(I.LT.NX.AND.J.LT.NY)DFY(MU-NX-1,K+NX+1)=-FAC2
      IF(I.GT.2)DFY(MU+2,K-2)=-FAC
      IF(I.LT.NXM1)DFY(MU-2,K+2)=-FAC
      IF(J.GT.2)DFY(MU+2*NX,K-2*NX)=-FAC
      IF(J.LT.NYM1)DFY(MU-2*NX,K+2*NX)=-FAC
      DFY(MU,K+NDEMI)= -OMEGA
    2 CONTINUE
      RETURN
      END
```

■ Example 4:Differential-algebraic system $\boldsymbol{M}\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$.

Finally, we consider the following system with independent variable *t* and 8 unknowns $y_1, y_2, \ldots, y_8$.

$$C_5(y_2' - y_1') = y_1/R_9$$
$$-C_5(y_2' - y_1') = \alpha f(y_4 - y_3) - U_b/R_8 + y_2/R_8$$
$$-C_4 y_3' = y_3/R_7 - f(y_4 - y_3)$$
$$C_3(y_5' - y_4') = -U_b/R_6 + y_4(1/R_5 + 1/R_6) + (1-\alpha) f(y_4 - y_3)$$
$$-C_3(y_5' - y_4') = -U_b/R_4 + y_5/R_4 + \alpha f(y_7 - y_6)$$
$$-C_2 y_6' = y_6/R_3 - f(y_7 - y_6)$$
$$C_1(y_8' - y_7') = -U_b/R_2 + y_7(1/R_1 + 1/R_2) + (1-\alpha) f(y_7 - y_6)$$
$$C_1(y_7' - y_8') = y_8/R_0 - U_e(t)/R_0$$

where

$$C_k = k \cdot 10^{-6}, k = 1,2,\ldots,5$$
$$R_0 = 1000, R_k = 9000, k = 1,2,\ldots,9$$
$$f(y_i - y_j) = \beta(\mathbf{e}^{(y_i - y_j)/U_F} - 1)$$
$$U_F = 0.026, \alpha = 0.99, \beta = 10^{-6}, U_b = 6$$
$$U_e(t) = 0.1 \cdot \sin(200\pi t)$$

With $\boldsymbol{y} = (y_1, y_2, \ldots, y_8)^{\mathrm{T}}$ the left hand side of the above 8 equations can be written as **My'**, where **M** is a tridiagonal matrix.

$$\boldsymbol{M} = \begin{pmatrix} -C_5 & C_5 & & & & & & \\ C_5 & -C_5 & & & & & & \\ & & -C_4 & & & & & \\ & & & -C_3 & C_3 & & & \\ & & & C_3 & -C_3 & & & \\ & & & & & -C_2 & & \\ & & & & & & -C_1 & C_1 \\ & & & & & & C_1 & -C_1 \end{pmatrix}$$

Obviously, **M** is singular and its rank is 5. Because of this, the system is a differential-algebraic system. According to a detailed analysis this system is index 1 problem.
We integrate from t=0 through t=0.2. Initial values **y**(0) must be chosen so that the vector with 8 components from the right hand side of the above equations lies in the range of the matrix **M**. Such initial values are as follows.

$$y_1(0) = 0, y_2(0) = U_b - y_1(0) \cdot R_8 / R_9, y_3(0) = y_4(0) = U_b / (R_6 / R_5 + 1)$$
$$y_5(0) = U_b, y_6(0) = y_7(0) = U_b / (R_2 / R_1 + 1), y_8(0) = 0$$

The Jacobian matrix in this model becomes a banded matrix with upper bandwidth 2 and lower bandwidth 1. Additionally, all the unknown variables can be proved to be index 1.

```
IMPLICIT REAL*8 (A-H,O-Z)
PARAMETER (ND=8,LJAC=4,LMAS=3,LE=5)
PARAMETER (LWORK=ND*(LJAC+LMAS+3*LE+12)+20,LIWORK=3*ND+20)
DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK),RPAR(16)
EXTERNAL FAMPL,JBAMPL,BBAMPL,SOLOUT
UE=0.1D0
  RPAR(1)=UE
UB=6.0D0
  RPAR(2)=UB
UF=0.026D0
  RPAR(3)=UF
ALPHA=0.99D0
  RPAR(4)=ALPHA
BETA=1.0D-6
  RPAR(5)=BETA
R0=1000.0D0
  RPAR(6)=R0
R1=9000.0D0
  RPAR(7)=R1
R2=9000.0D0
```

```
        RPAR(8)=R2
       R3=9000.0D0
        RPAR(9)=R3
       R4=9000.0D0
        RPAR(10)=R4
       R5=9000.0D0
        RPAR(11)=R5
       R6=9000.0D0
        RPAR(12)=R6
       R7=9000.0D0
        RPAR(13)=R7
       R8=9000.0D0
        RPAR(14)=R8
       R9=9000.0D0
        RPAR(15)=R9
       RPAR(16)=0.0025D0
       N=8
       IJAC=1
       MLJAC=1
       MUJAC=2
       IMAS=1
       MLMAS=1
       MUMAS=1
       IOUT=1
       X=0.0D0
       Y(1)=0.D0
       Y(2)=UB-Y(1)*R8/R9
       Y(3)=UB/(R6/R5+1.D0)
       Y(4)=UB/(R6/R5+1.D0)
       Y(5)=UB
       Y(6)=UB/(R2/R1+1.D0)
       Y(7)=UB/(R2/R1+1.D0)
       Y(8)=0.D0
       XEND=0.2D0
       RTOL=1.0D-5
       ATOL=1.0D-6*RTOL
       ITOL=0
       H=1.0D-6
       DO 10 I=1,20
       IWORK(I)=0
 10    WORK(I)=0.D0
       CALL DM_VRADAU5(N,FAMPL,X,Y,XEND,H,
     &                 RTOL,ATOL,ITOL,
     &                 JBAMPL,IJAC,MLJAC,MUJAC,
     &                 BBAMPL,IMAS,MLMAS,MUMAS,
     &                 SOLOUT,IOUT,
     &                 WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,ICON)
       WRITE(6,*) 'ICON=',ICON
       WRITE (6,99) X,Y(1),Y(2)
 99    FORMAT(1X,'X =',F7.4,'    Y =',2E18.10)
       STOP
       END
C
```

```
C
      SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN,
     &  WORK2,IWORK2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),CONT(LRC),RPAR(*)
      IF (NR.EQ.1) THEN
         WRITE (6,99) X,Y(1),Y(2),NR-1
      ELSE
 10      CONTINUE
         IF (X.GE.RPAR(16)) THEN
         WRITE (6,99) RPAR(16),
     &     DM_VCONTR5(1,RPAR(16),CONT,LRC,WORK2,IWORK2),
     &     DM_VCONTR5(2,RPAR(16),CONT,LRC,WORK2,IWORK2),NR-1
         RPAR(16)=RPAR(16)+0.0025D0
         GOTO 10
         END IF
      END IF
 99   FORMAT(1X,'X =',F7.4,'    Y =',2E18.10,'    NSTEP =',I4)
      RETURN
      END
C
C
      SUBROUTINE FAMPL(N,X,Y,F,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 Y(N),F(N),RPAR(*)
      UE=RPAR(1)
      UB=RPAR(2)
      UF=RPAR(3)
      ALPHA=RPAR(4)
      BETA=RPAR(5)
      R0=RPAR(6)
      R1=RPAR(7)
      R2=RPAR(8)
      R3=RPAR(9)
      R4=RPAR(10)
      R5=RPAR(11)
      R6=RPAR(12)
      R7=RPAR(13)
      R8=RPAR(14)
      R9=RPAR(15)
      W=2.D0*3.141592654D0*100.D0
      UET=UE*DSIN(W*X)
      FAC1=BETA*(DEXP((Y(4)-Y(3))/UF)-1.D0)
      FAC2=BETA*(DEXP((Y(7)-Y(6))/UF)-1.D0)
      F(1)=Y(1)/R9
      F(2)=(Y(2)-UB)/R8+ALPHA*FAC1
      F(3)=Y(3)/R7-FAC1
      F(4)=Y(4)/R5+(Y(4)-UB)/R6+(1.D0-ALPHA)*FAC1
      F(5)=(Y(5)-UB)/R4+ALPHA*FAC2
      F(6)=Y(6)/R3-FAC2
      F(7)=Y(7)/R1+(Y(7)-UB)/R2+(1.D0-ALPHA)*FAC2
      F(8)=(Y(8)-UET)/R0
      RETURN
```

```
      END
C
C
      SUBROUTINE JBAMPL(N,X,Y,DFY,LDFY,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 Y(N),DFY(LDFY,N),RPAR(*)
      UE=RPAR(1)
      UB=RPAR(2)
      UF=RPAR(3)
      ALPHA=RPAR(4)
      BETA=RPAR(5)
      R0=RPAR(6)
      R1=RPAR(7)
      R2=RPAR(8)
      R3=RPAR(9)
      R4=RPAR(10)
      R5=RPAR(11)
      R6=RPAR(12)
      R7=RPAR(13)
      R8=RPAR(14)
      R9=RPAR(15)
      FAC14=BETA*DEXP((Y(4)-Y(3))/UF)/UF
      FAC27=BETA*DEXP((Y(7)-Y(6))/UF)/UF
      DO  J=1,8
       DFY(1,J)=0.D0
       DFY(2,J)=0.D0
       DFY(4,J)=0.D0
      END DO
      DFY(3,1)=1.D0/R9
      DFY(3,2)=1.D0/R8
      DFY(2,3)=-ALPHA*FAC14
      DFY(1,4)=ALPHA*FAC14
      DFY(3,3)=1.D0/R7+FAC14
      DFY(2,4)=-FAC14
      DFY(3,4)=1.D0/R5+1.D0/R6+(1.D0-ALPHA)*FAC14
      DFY(4,3)=-(1.D0-ALPHA)*FAC14
      DFY(3,5)=1.D0/R4
      DFY(2,6)=-ALPHA*FAC27
      DFY(1,7)=ALPHA*FAC27
      DFY(3,6)=1.D0/R3+FAC27
      DFY(2,7)=-FAC27
      DFY(3,7)=1.D0/R1+1.D0/R2+(1.D0-ALPHA)*FAC27
      DFY(4,6)=-(1.D0-ALPHA)*FAC27
      DFY(3,8)=1.D0/R0
      RETURN
      END
C
      SUBROUTINE BBAMPL(N,B,LB,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 B(LB,N),RPAR(*)
      DO  I=1,8
       B(1,I)=0.D0
       B(3,I)=0.D0
```

```
            END DO
            C1=1.D-6
            C2=2.D-6
            C3=3.D-6
            C4=4.D-6
            C5=5.D-6
      C
            B(2,1)=-C5
            B(1,2)=C5
            B(3,1)=C5
            B(2,2)=-C5
            B(2,3)=-C4
            B(2,4)=-C3
            B(1,5)=C3
            B(3,4)=C3
            B(2,5)=-C3
            B(2,6)=-C2
            B(2,7)=-C1
            B(1,8)=C1
            B(3,7)=C1
            B(2,8)=-C1
            RETURN
            END
```

(4) Method

This subroutine employs a 3-stage 5-th order implicit Runge-Kutta method (referred to as Radau IIA of order 5 in [33] and [34]) which is stable and efficient for stiff differential equations and differential-algebraic equations.
We first consider the case of ***M=I*** and think about the situation where the code advances one step from x_0 to x_1 with step size h. Here x_0 does not mean the initial value for x.
A 3-stage implicit Runge-Kutta method can be expressed as follows

$$\boldsymbol{g}_i = \boldsymbol{y}_0 + h\sum_{j=1}^{3} a_{ij}\,\boldsymbol{f}(x_0 + c_j h, \boldsymbol{g}_j) \quad i = 1,2,3 \tag{4.1a}$$

$$\boldsymbol{y}_1 = \boldsymbol{y}_0 + h\sum_{j=1}^{3} b_j\,\boldsymbol{f}(x_0 + c_j h, \boldsymbol{g}_j) \tag{4.1b}$$

, where a_{ij}, c_j and b_j are coefficients of Runge-Kutta formula and usually represented by the following table.

c_1	a_{11}	a_{12}	a_{13}
c_2	a_{21}	a_{22}	a_{23}
c_3	a_{31}	a_{32}	a_{33}
	b_1	b_2	b_3

The coefficients of 3-stage 5-th order Radau IIA formula, which this subroutine employs, are as follows.

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

In principle, the numerical solution $\boldsymbol{y}_1$ can be obtained by solving 3*n* dimensional non-linear equations (4.1a) for $\boldsymbol{g}_1, \boldsymbol{g}_2$ and $\boldsymbol{g}_3$ then substituting them into (4.1b). In real implementation, however, we use variables $\boldsymbol{z}_i = \boldsymbol{g}_i - \boldsymbol{y}_0$ for reducing computational errors, solve the non-linear system for $\boldsymbol{z}_i$ and use another form of (4.1b) to avoid evaluations of $\boldsymbol{f}$.
The non-linear algebraic system is solved by Newton iteration. The Jacobin is evaluated only once at $x_0, \boldsymbol{y}_0$ when solving for $\boldsymbol{z}_i$ and reused throughout iterations (Simplified Newton Iterations).
The step size *h* is controlled and chosen as large as possible on the condition that required accuracies are satisfied. For the step size control, error estimation of $\boldsymbol{y}_1$ is made and based on the difference $\hat{\boldsymbol{y}}_1$ - $\boldsymbol{y}_1$ where $\hat{\boldsymbol{y}}_1$ is another approximation computed by an embedded formula of order 4 (using the same $\boldsymbol{g}_1, \boldsymbol{g}_2$ and $\boldsymbol{g}_3$ but different $\hat{b}_i$, *j*=1,2,3).
When $\boldsymbol{M} \neq \boldsymbol{I}$ in (1.1), we can formally replace all $\boldsymbol{f}$ in Radau IIA formula by $\boldsymbol{M}^{-1}\boldsymbol{f}$ and multiply the resulting formula by $\boldsymbol{M}$, giving the method for (1.1).
For details, see [33] and [34] in Appendix A “References”. [33] presents a general discussion on methods for solving ordinary differential equations including Runge-Kutta methods and [34] treats stiff differential equations and differential-algebraic equations. [35] and [36] are Japanese translation of [33] and [34] , respectively.

DM_VRANN3

Generation of normal random numbers
CALL DM_VRANN3(DAM, DSD, IX, DA, K, N, DWORK, NWORK, ICON)

(1) Function

This subroutine generates normal random numbers from a normal-distribution density function (1.1) with 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 (> 0) of normal distribution.

Double-precision real type.

IX ..... Input. Starting point.

On the first call, the value of IX must be positive. On the second and later calls, return value 0 must be used. When a different starting point is specified for the initial call, a different random number sequence is created.

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

Output. 0.

DA ..... Output. N normal pseudorandom numbers generated by each thread.

Double precision two-dimensional array DA(K,NUMT), where, NUMT is the number of threads.

N pseudo random numbers generated by thread number *p* (which is from 0 to NUMT-1) are stored in DA(1:N,*p*+1).

K ..... Input. The size of the first dimension of the array DA (≥ N).

N ..... Input. Number of normally distributed pseudorandom numbers to be returned by each thread in DA.

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

DWORK ... Work area. A double precision two-dimensional array of DWORK(NWORK,NUMT).

When this subroutine is called repeatedly, the contents and NUMT must not be changed. DWORK contains all the current information required to restart this subroutine from its current point.

(See note 3) and 6) in (3), "Comments on use.")

NWORK ... Input. The size of array DWORK. NWORK ≥ 1156.

ICON ..... Output. Condition codes.

(See Table DM_VRANN3-1.)

Table DM_VRANN3-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	K < N or K < 1	Processing is discontinued.
30001	The value of NWORK is too small. IX < 0, DSD ≤ 0	
30002	An internal error occurred.	
30003 to 30009	The value of DWORK was changed. Or, IX was set to 0 at the first call.	
40000	The value of IX is too large.	

(3) Comments on use

a. Notes

1) Starting point IX
When a sequence of pseudo random numbers is to be generated by a deterministic program, there must be some random input. Thus, the user must give a starting point IX. This is often called a "seed". On the first call to this subroutine the seed IX should be a positive integer. (For exception, See note 5) in (3), "Comments on use.") On the subsequent call IX should be zero. This indicates that more pseudo random numbers from the same sequence are to be generated. To simplify programming, IX is returned as zero after the first call to this subroutine.
This subroutine appends the thread number +1, OMP_GET_THREAD_NUM() +1, to the seed, as in seed = seed * OMP_GET_NUM_THREADS() + OMP_GET_THREAD_NUM() +1. Thus the seeds used on different threads are assured to be distinct, and hence subsequences of length less than 10^{18} will not overlap. (See (4), "Method" below.)

2) Parameter N
This subroutine returns the next N pseudo random numbers from the infinite sequence defined by the initial seed IX. If N ≤ 0, no pseudo random numbers are returned.
For efficiency, N should be large (for example, N = 100,000). This reduces the overhead of subroutine calls. N may be different on successive calls to this routine, provided that K (the size of the first dimension of the array DA) is larger than the maximum value of N.

3) Work area DWORK
When this subroutine is to be called two or more times, DWORK is used as the work area for storing the information for the next call. While this subroutine is called, the contents of DWORK must not be changed by the called program.

4) Parameter NWORK
DWORK(1,:), ... and, DWORK(NWORK,:) are used by this subroutine. The value of NWORK must not be changed at any call of this subroutine. For efficient processing, NWORK must be set to 1,156 or higher. When this subroutine is to be used on a vector processor, the value of NWORK must be 100,000 or higher.

5) Regeneration of the same random numbers
When DWORK(1,:), ... and, DWORK(NWORK,:) are saved, the same random number sequence as that used during the saving can be regenerated by reusing the DWORK and by calling this subroutine with condition IX=0.

6) The number of the threads or NUMT, used with this subroutine can be assigned by user with an OpenMP environment variable "OMP_NUM_THREADS" or a run-time library routine "OMP_SET_NUM_THREADS( )". In case of specifying the number of threads with run-time library OMP_SET_NUM_THREADS(), assign the same number of threads as that of first calling immediately before the second or later calling also with OMP_SET_NUM_THREADS().

b. Example

10,000,000 × 4 normal pseudorandom numbers are generated, and their mean and standard deviation are calculated.

```
C     ** EXAMPLE **
      PARAMETER (NUMT=4)
      PARAMETER (NRAN=10000000)
      PARAMETER (NSEED=12345)
      PARAMETER (NWMAX=100000)
      PARAMETER (NBUF=120000,K=NBUF)
      DOUBLE PRECISION DA(K,NUMT)
      DOUBLE PRECISION DWORK(NWMAX,NUMT)
      DOUBLE PRECISION DSUM,DSUM2,DSSUM,DSSUM2
      DOUBLE PRECISION DMEAN,DSIG
      DOUBLE PRECISION DAM,DSD
      INTEGER NTOT
C     Initialize ix,n and nwork
      IX=NSEED
      WRITE (*,*)' Seed ',IX
      DAM=0.0D0
      DSD=1.0D0
      WRITE (*,*)' Mean ',DAM
      WRITE (*,*)' Standard deviation ',DSD
      N=NBUF
      NWORK=NWMAX
      DSUM=0.0D0
      DSSUM=0.0D0
C     ngen counts down to 0
      NGEN=NRAN
      NTOT=NRAN*NUMT
C     Generate ngen numbers
C     with maximum NBUF at a time.
      KRPT=(NRAN+NBUF-1)/NBUF
      WRITE (*,*)' Generating ',NTOT,' numbers'
      WRITE (*,*)' with ',KRPT,
     $   ' calls to dm-vrann3 on ',NUMT,' threads'
      CALL OMP_SET_NUM_THREADS(NUMT)
      DO 20 IZ=1,KRPT
      N=MIN0(NBUF,NGEN)
```

```
      CALL DM_VRANN3(DAM,DSD,IX,DA,K,N,DWORK,NWORK,ICON)
      IF (ICON.NE.0) WRITE (*,*)' ICON ',ICON
C     Accumulate sum of numbers
      DSUM2=0.0D0
      DO 30 J=1,NUMT
      DO 10 I=1,N
      DSUM2=DSUM2+DA(I,J)
  10  CONTINUE
  30  CONTINUE
C     Accumulate sum of numbers globally.
      DSSUM2=0.0D0
      DO 40 J=1,NUMT
      DO 50 I=1,N
       DSSUM2=DSSUM2+DA(I,J)*DA(I,J)
 50   CONTINUE
 40   CONTINUE
      DSUM=DSUM+DSUM2
      DSSUM=DSSUM+DSSUM2
C Count down numbers still to generate
C on each processor
      NGEN=NGEN-N
 20   CONTINUE
C Compute overall mean.
      DMEAN=DSUM/DFLOAT(NTOT)
      WRITE (*,*) ' Sample mean ',DMEAN
C Compute overall sample standard deviation.
      DSIG=DSSUM/DFLOAT(NTOT)
      WRITE (*,*)' Sample standard deviation ',DSIG
      STOP
      END
```

(4) Method

This routine uses the Polar method with fast elementary function calculation to generate normally distributed pseudorandom numbers. This method requires uniform pseudorandom numbers generated using the same technique as that of DVRAU4 (see *SSL II Extended Capability User's Guide II*).

For an explanation of the Polar method, see [46] in Appendix A, "References." For details of the actual processing and comparisons with other methods, see [11] in Appendix A, "References."

DM_VRANN4

Generation of normal random numbers (Wallace's method)
CALL DM_VRANN4 (DAM, DSD, IX, DA, K, N, DWORK, NWORK, ICON)

(1) Function

This subroutine generates normal random numbers from a normal-distribution density function (1.1) with 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 (> 0) of normal distribution.

Double-precision real type.

IX ..... Input. Starting point.

On the first call, the value of IX must be positive. On the second and later calls, return value 0 must be used. When a different starting point is specified for the initial call, a different random number sequence is created.

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

Output. 0.

DA ..... Output. N normal pseudorandom numbers generated by each thread.

Double precision two-dimensional array DA(K,NUMT), where, NUMT is the number of threads.

N pseudo random numbers generated by thread number *p* (which is from 0 to NUMT-1) are stored in DA(1:N,*p*+1).

K ..... Input. The size of the first dimension of the array DA (≥ N).

N ..... Input. Number of normally distributed pseudorandom numbers to be returned by each thread in DA.

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

DWORK ... Work area. A double precision two-dimensional array of DWORK(NWORK,NUMT).

When this subroutine is called repeatedly, the contents and NUMT must not be changed. DWORK contains all the current information required to restart this subroutine from its current point.

(See note 3) and 6) in (3), "Comments on use.")

NWORK .... Input. Size of array DWORK. NWORK ≥ 1350.

ICON ..... Output. Condition codes.

(See Table DM_VRANN4-1.)

Table DM_VRANN4-1 Condition codes

Code	Meaning	Processing
0	No error	
30000	K < N or K < 1	Processing is discontinued.
30001	The value of NWORK is too small. IX < 0, DSD ≤ 0	
30002	An internal error occurred.	
30003 to 30008	The value of DWORK was changed. Or, IX was set to 0 at the first call.	
30009	The value of IX is too large.	
40000 to 40002	The value of DWORK was changed. Or, IX was set to 0 at the first call.	

(3) Comments on use

a. Notes

1) Starting point IX
When a sequence of pseudo random numbers is to be generated by a deterministic program, there must be some random input. Thus, the user must give a starting point IX. This is often called a "seed". On the first call to this subroutine the seed IX should be a positive integer. (For exception, See note 5) in (3), "Comments on use.") On the subsequent call IX should be zero. This indicates that more pseudo random numbers from the same sequence are to be generated. To simplify programming, IX is returned as zero after the first call to this subroutine.

2) Parameter N
This subroutine returns the next N pseudo random numbers from the infinite sequence defined by the initial seed IX. If N ≤ 0, no pseudo random numbers are returned.
For efficiency, N should be large (for example, N = 100,000). This reduces the overhead of subroutine calls. N may be different on successive calls to this routine, provided that K (the size of the first dimension of the array DA) is larger than the maximum value of N.

3) Work area DWORK
When this subroutine is to be called two or more times, DWORK is used as the work area for storing the information for the next call. While this subroutine is called, the contents of DWORK must not be changed by the called program.

4) Parameter NWORK
DWORK(1,:), ... and, DWORK(NWORK,:) are used by this subroutine. The value of NWORK must not be changed at any call of this subroutine. For efficient processing, NWORK must be set to 1,350 or higher. When this subroutine is to be used on a vector processor, the value of NWORK must be 500,000 or higher.

5) Regeneration of the same random numbers
When DWORK(1,:), ... and, DWORK(NWORK,:) are saved, the same random

number sequence as that used during the saving can be regenerated by reusing the DWORK and by calling this subroutine with condition IX=0.

6) The number of the threads or NUMT, used with this subroutine can be assigned by user with an OpenMP environment variable "OMP_NUM_THREADS" or a run-time library routine "OMP_SET_NUM_THREADS( )". In case of specifying the number of threads with run-time library OMP_SET_NUM_THREADS(), assign the same number of threads as that of first calling immediately before the second or later calling also with OMP_SET_NUM_THREADS().

7) The implementation of Wallece's method in this routine is about three times faster than the implementation of the Polar method in DM_VRANN3.

b. Example

10,000,000 × 4 normal pseudorandom numbers are generated, and their mean and standard deviation are calculated.

```
C     ** EXAMPLE **
      PARAMETER (NUMT=4)
      PARAMETER (NRAN=10000000)
      PARAMETER (NSEED=12345)
      PARAMETER (NWMAX=100000)
      PARAMETER (NBUF=120000,K=NBUF)
      DOUBLE PRECISION DA(K,NUMT)
      DOUBLE PRECISION DWORK(NWMAX,NUMT)
      DOUBLE PRECISION DSUM,DSUM2,DSSUM,DSSUM2
      DOUBLE PRECISION DMEAN,DSIG
      DOUBLE PRECISION DAM,DSD
      INTEGER NTOT
C     Initialize ix,n and nwork
      IX=NSEED
      WRITE (*,*)' Seed ',IX
      DAM=0.0D0
      DSD=1.0D0
      WRITE (*,*)' Mean ',DAM
      WRITE (*,*)' Standard deviation ',DSD
      N=NBUF
      NWORK=NWMAX
      DSUM=0.0D0
      DSSUM=0.0D0
C     ngen counts down to 0
      NGEN=NRAN
      NTOT=NRAN*NUMT
C     Generate ngen numbers
C     with maximum NBUF at a time.
      KRPT=(NRAN+NBUF-1)/NBUF
      WRITE (*,*)' Generating ',NTOT,' numbers'
      WRITE (*,*)' with ',KRPT,
     $   ' calls to dm-vrann3 on ',NUMT,
     $   ' threads'
      CALL OMP_SET_NUM_THREADS(NUMT)
      DO 20 IZ=1,KRPT
```

```
      N=MIN0(NBUF,NGEN)
      CALL DM_VRANN4(DAM,DSD,IX,DA,K,N,DWORK,NWORK,ICON)
      IF (ICON.NE.0) WRITE (*,*)' ICON ',ICON
C     Accumulate sum of numbers
      DSUM2=0.0D0
      DO 30 J=1,NUMT
      DO 10 I=1,N
      DSUM2=DSUM2+DA(I,J)
  10  CONTINUE
  30  CONTINUE
C     Accumulate sum of numbers globally.
      DSSUM2=0.0D0
      DO 40 J=1,NUMT
      DO 50 I=1,N
       DSSUM2=DSSUM2+DA(I,J)*DA(I,J)
 50   CONTINUE
 40   CONTINUE
      DSUM=DSUM+DSUM2
      DSSUM=DSSUM+DSSUM2
C Count down numbers still to generate
C on each processor
      NGEN=NGEN-N
 20   CONTINUE
C Compute overall mean.
      DMEAN=DSUM/DFLOAT(NTOT)
      WRITE (*,*) ' Sample mean ',DMEAN
C Compute overall sample standard deviation.
      DSIG=DSSUM/DFLOAT(NTOT)
      WRITE (*,*)' Sample standard deviation ',DSIG
      STOP
      END
```

(4) Method

This routine uses a variant of Wallece's method to generate normally distributed pseudo-random numbers. This method requires uniform pseudorandom numbers generated using the same technique as that of DVRAU4 (see *SSL II Extended Capability User's Guide II*).

For Wallace's method, see [78] in Appendix A, "References."

For implementation details and comparisons with other methods, see [11] and [12] in Appendix A, "References."

DM_VRANU4

Generation of uniform random numbers [0,1)
CALL DM_VRANU4(IX,DA,K,N,DWORK,NWORK,ICON)

(1) Function

This subroutine generates different sequences of pseudo random numbers from a uniform distribution on [0,1) on each thread.

(2) Parameters

IX ..... Input. Starting point.

Output. Zero.

On the first call, IX should be positive. IX is returned as zero and should remain zero for subsequent calls. IX < 8000000

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

DA ..... Output. N uniform pseudo random numbers on [0,1) generated by each thread.

Double precision two-dimensional array DA(K,NUMT), where, NUMT is the number of threads.

N pseudo random numbers generated by thread number *p* (which is from 0 to NUMT-1) are stored in DA(1:N,*p*+1).

(See note 6) in (3), "Comments on use.")

K ..... Input. The size of the first dimension of the array DA (≥ N).

N ..... Input. The number of uniformly distributed pseudo random numbers on each processor to be returned in DA.

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

DWORK ... Work area. A double precision two-dimensional array of DWORK(NWORK,NUMT).

When this subroutine is called repeatedly, the contents and NUMT must not be changed. DWORK contains all the current information required to restart this subroutine from its current point.

(See note 3) and 6) in (3), "Comments on use.")

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

Refer to (4), "Method" for the relation between the size of work area and the period of the random number.

ICON ..... Output. Condition codes.

(See Table DM_VRANU4-1.)

Table DM_VRANU4-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	K < N or K < 1	Processing is discontinued.
30001	NWORK too small	
30002	The internal check failed	
30003 to 30008	DWORK overwritten or IX = 0 on first call	
30009	IX too large	

(3) Comments on use

a. Notes

1) Starting value IX

When a sequence of pseudo random numbers is to be generated by a deterministic program, there must be some random input. Thus, the user must give a starting point IX. This is often called a "seed". On the first call to this subroutine the seed IX should be a positive integer. (For exception, See note 5) in (3), "Comments on use.") On the subsequent call IX should be zero. This indicates that more pseudo random numbers from the same sequence are to be generated. To simplify programming, IX is returned as zero after the first call to this subroutine.

This subroutine appends the thread number +1, OMP_GET_THREAD_NUM() +1, to the seed, as in seed = seed * OMP_GET_NUM_THREADS() + OMP_GET_THREAD_NUM() +1. Thus the seeds used on different threads are assured to be distinct, and hence subsequences of length less than 10^{18} will not overlap. (See (4), "Method" below.)

2) Parameter N

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

For efficiency, N should be large (for example, N = 100,000). This reduces the overhead of subroutine calls. N may be different on successive calls to this routine, provided that K (the size of the first dimension of the array DA) is larger than the maximum value of N.

3) Work area DWORK

DWORK is used as a work area to store state information between calls to this subroutine. The calling program must not change the contents of the array DWORK between calls.

4) Parameter NWORK

DWORK(1,:), ..., DWORK(NWORK,:) are used by this subroutine. NWORK should be the same on each call to this subroutine. NWORK should be at least 388.

5) Checkpointing

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

6) The number of the threads or NUMT, used with this subroutine can be assigned by user with an OpenMP environment variable "OMP_NUM_THREADS" or a run-time library routine "OMP_SET_NUM_THREADS( )". In case of specifying the number of threads with run-time library OMP_SET_NUM_THREADS(), assign the same number of threads as that of first calling immediately before the second or later calling also with OMP_SET_NUM_THREADS().

b. Example

1,000,000 × 4 uniform pseudo random numbers are generated and their mean value is calculated. The starting point is 123.

```
C     **EXAMPLE**
      PARAMETER(NUMT=4)
      PARAMETER(NRAN=1000000)
      PARAMETER(NSEED=123)
      PARAMETER(NWMAX=5000)
      PARAMETER(NBUF=25000)
      DOUBLE PRECISION DA(NBUF,NUMT)
      DOUBLE PRECISION DWORK(NWMAX,NUMT)
      DOUBLE PRECISION DSUM,DSUM2
      DOUBLE PRECISION DMEAN,DSIG
      INTEGER TNO,NTOT

C     Initialize ix, n and nwork
      IX=NSEED
      PRINT *,' Seed ',IX
      N=NBUF
      NWORK=NWMAX
      DSUM=0.0D0
C ngen counts down to 0
      NGEN=NRAN
      NTOT=NRAN*NUMT
C Generate ngen numbers on each thread
C with maximum NBUF at a time
      KRPT=(NRAN+NBUF-1)/NBUF
      PRINT *,' Generating ',NTOT,
     $        ' numbers'
      PRINT *,' with ',KRPT,
     $        ' calls to dm_vranu4 on ',NUMT,
     $        ' threads'
      DO 20 J=1,KRPT
      N=MIN0(NBUF,NGEN)
      DSUM2=0.0D0
      CALL OMP_SET_NUM_THREADS(NUMT)
      CALL DM_VRANU4(IX,DA,NBUF,N,DWORK,NWORK,ICON)
      IF(ICON.NE.0) PRINT *,
     $        ' Error return,',
     $        ' ICON ',ICON
      DO 30 TNO=1,NUMT
C Accumulate sum of numbers locally
      DO 10 I=1,N
```

```
      DSUM2=DSUM2+DA(I,TNO)
   10 CONTINUE
   30 CONTINUE
C Accumulate sum of numbers globally
      DSUM=DSUM+DSUM2
C Count down numbers still to generate
C on each processor
      NGEN=NGEN-N
   20 CONTINUE
C Compute overall mean
      DMEAN=DSUM/DFLOAT(NTOT)
      PRINT *,' Mean ',DMEAN
C Compute deviation from 0.5 normalized
C by expected value 1/sqrt(12*ntot).
c This should be (approximately) normally
C distributed with mean 0, variance 1.
      DSIG=(DMEAN-0.5D0)*DSQRT(12.0D0*NTOT)
      PRINT *,' Normalized 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)= α * X(J-*r*) + β * X(J-*s*) (modulo 1)

where J > *r* > *s*. Here, *r* and *s* are fixed positive integers (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 largest *r* is chosen, subject to the constraint that N and NWORK are large enough. Thus, the user can select a suitable generator as shown below.

- A good generator with a moderately long period, low initialization overhead and small storage requirements (e.g., by setting NWORK = 1000).
- A very good generator with extremely long period, high initialization overhead and high storage requirement (e.g., by setting NWORK = 133000).
- Some intermediate compromise, which does not require knowing the precise details of how to choose pairs (*r*, *s*). The pairs (*r*, *s*) used by this subroutine are given in Table DM_VRANU4-2. Tables of primitive trinomials may be found in [41] in Appendix A, "References."

Table DM_VRANU4-2 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 $(\alpha,\beta) = (7, 9)$ if $r \leq 1000$, and $(\alpha,\beta) = (1,15)$ if $r > 1000$. The rationale is that performance on statistical tests is likely to be improved if $\alpha > 1$, but this improvements is only significant for the smaller values of r. For the larger values of r the performance on statistical tests is very good even if $\alpha = 1$, and this value increases the speed of random number generation.

The period of the random number sequence is $W(2^r -1)$ where r is in the range 127 (for small NWORK) to 132,049 (for $N \geq 264{,}098$ and $NWORK \geq 132{,}056$). The factor W depends on the word length ($W = 2^{48}$ on the Fujitsu VPP series and PRIMEPOWER series (SPARC architecture), and the period is at least 10^{52} or more).

The initialization ensures that sequences of pseudo random numbers returned for different initial seeds IX are separated by a distance of at least $2^{60} > 10^{18}$ in the full periodic sequence. Thus, for all practical purposes, different initial seeds IX ensure different sequences of pseudo random numbers. This subroutine appends the thread number+1 to the seed and thus assure different seeds are used on different threads.

The method and implementation details are described in more detail in [9] and [10] in Appendix A, "References." For further information and comparisons with other methods, see [4], [24], [42], and [53] in Appendix A, "References."

(5) Tests for uniform random numbers

Table DM_VRANU4-3 shows the results of testing of statistical hypotheses on the pseudo random numbers generated by DM_VRANU4 with NWORK = 44504 ($r = 44497$, $s = 23463$).

In this table the number of degrees of freedom, f, for the chi-squared tests is very large - in the millions. In this case the expression $\sqrt{2\chi^2} - \sqrt{2f-1}$ should be approximated extremely well as a normal deviate with unit variance.

Table DM_VRANU4-3 Chi-squared tests (Uniform distribution in the n- dimensional unit hypercube)

dim(*1)	Size(*2)	*resl*(*3)	*resv*(*4)	dens(*5)	thrd1(*6)	thrd2(*6)	thrd3(*6)	thrd4(*6)
1	10^9	5×10^7	50000000	20.00	1.21	1.37	-0.24	0.90
1	0.8×10^9	1.25×10^7	12500000	64.00	-0.67	0.79	0.39	-1.04
2	10^9	7071	49999041	10.00	-0.10	0.42	0.30	-0.65
2	2×10^9	3535	12496225	80.02	-0.37	-0.25	1.44	-0.07
3	2×10^9	368	49836032	13.38	1.40	-0.21	-1.92	-0.47
3	2×10^9	232	12487168	53.39	-0.96	-0.63	0.46	-0.22
4	2×10^9	84	49787136	10.04	0.76	1.51	1.10	-1.45
4	2×10^9	59	12117361	41.26	-0.38	0.08	0.32	0.16

*1 Dimension of unit hypercube
*2 Number of pseudo randoms generated
*3 Number of equal subintervals partitioning [0,1) in each dimension
*4 Number of equal hypercubes partitioning the unit hypercube
*5 Average number of random points per small hypercube
*6 For each thread, the variable $\sqrt{2\chi^2} - \sqrt{2f-1}$

DM_VRANU5

Generation of uniform random numbers [0,1) (MRG8)
CALL DM_VRANU5(IX,DA,N,J,DWORK,ICON)

(1) Function

This subroutine generates sequence of pseudo random numbers from a uniform distribution on [0,1) by Multiple Recursive Generator with 8th-order full primitive polynomials (MRG8).

This subroutine generates same sequence of random number in any thread numbers. When the reproducibility is needed, use this subroutine instead of DM_VRANU4 . The interface of this subroutine is different from the interface of DM_VRANU4.

This subroutine supports jumping-ahead method, which jumps J steps in a sequence of pseudo random numbers. This is useful to generate distinct sub sequence in parallel execution.

The performance of DM_VRANU4 is better than this subroutine.

Both this subroutine and DM_VRANU4 passed the bigCrush test of TESTU01 which is the statistical testing program of uniform random number generators.

(2) Parameters

IX ..... Input. Starting point.

Output. Zero.

On the first call, IX should be positive. IX is returned as zero and should remain zero for subsequent calls.

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

DA ..... Output. N uniform pseudo random numbers on [0,1).

Double precision array DA(N).

N ..... Input. The number of uniformly distributed pseudo random numbers to be returned in DA.

J ..... Input. Number of jumping steps in the sequence of pseudo random numbers.

8 byte integer.

0_8 (zero of 8-byte integer type) is to be set to generate pseudo random numbers just after the sequence.

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

DWORK ... Work area. A double precision array of DWORK(8).

When this subroutine is called repeatedly, the contents must not be changed. DWORK contains all the current information required to restart this subroutine from its current point.

(See note 3) in (3), "Comments on use.")

ICON ..... Output. Condition codes.

(See Table DM_VRANU5-1.)

Table DM_VRANU5-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	IX<0, N<1 or J<0	Processing is discontinued.

(3) Comments on use

a. Notes

1) Starting value IX
When a sequence of pseudo random numbers is to be generated by a deterministic program, there must be some random input. Thus, the user must give a starting point IX. This is often called a "seed". On the first call to this subroutine the seed IX should be a positive integer. (For exception, See note 4) in (3), "Comments on use.")
On the subsequent call IX should be zero. This indicates that more pseudo random numbers from the same sequence are to be generated. To simplify programming, IX is returned as zero after the first call to this subroutine.

2) Parameter J
This subroutine supports jumping-ahead method, which jumps J steps in a sequence of pseudo random numbers by setting J≥0.
This subroutine generates distinct sub sequence of pseudo random numbers in each process by setting same IX and different J in parallel execution. (See Example 2 and 3 in (3), "Comments on use")

3) Work area DWORK
DWORK is used as a work area to store state information between calls to this subroutine. The calling program must not change the contents of the array DWORK between calls.

4) Checkpointing
If DWORK are saved, the same sequence of random numbers can be generated again (from the point where DWORK was saved) by restoring DWORK and calling this subroutine with argument IX = 0.

b. Example

Example 1.

1,000,000 uniform pseudo random numbers are generated and their mean value is calculated. The starting point is 123.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE 1**
      INTEGER NRAN,NSEED,NBUF
      PARAMETER(NRAN=10000000)
      PARAMETER(NSEED=123)
      PARAMETER(NBUF=25000)
```

```
      DOUBLE PRECISION DA(NBUF)
      DOUBLE PRECISION DWORK(8)
      DOUBLE PRECISION DSUM,DSUM2
      DOUBLE PRECISION DMEAN
      INTEGER IX,N,ICON
      INTEGER I,J
C
C Generate NRAN numbers with maximum NBUF at a time
      IX=NSEED
      PRINT *,' Seed ',IX
      PRINT *,' Generating ',NRAN,' numbers'
C
      DSUM=0.0D0
      DO J=1,NRAN,NBUF
        N=MIN0(NBUF,NRAN-J+1)
        CALL DM_VRANU5(IX,DA,N,0_8,DWORK,ICON)
        IF(ICON.NE.0) THEN
          PRINT *,' Error return ICON ',ICON
        END IF
        DSUM2=0.0D0
        DO I=1,N
          DSUM2=DSUM2+DA(I)
        END DO
        DSUM=DSUM+DSUM2
      END DO
C Compute mean
      DMEAN=DSUM/DFLOAT(NRAN)
      PRINT *,' Mean ',DMEAN

      STOP
      END
```

Example 2.

Distinct 100,000 uniform pseudo random numbers are generated in each MPI processes and their mean value is calculated. The starting point is 123.

In this program, J is set to 2^{31}-1. As far as the length of each sub sequences is smaller than 2^{31}-1 they are not overlapping.

```
C     **EXAMPLE 2**
      INTEGER,PARAMETER::N=10000
      INTEGER(8),PARAMETER::JUMP=2147483647_8   ! =2**31-1
      REAL(8)::X(N)
      REAL(8)::DNALL
      INTEGER::IRANK,NP,IERROR
      INTEGER::IX,ICON
      INTEGER::I
      INTEGER(8)::J
      REAL(8)::WORK(8)
      REAL(8)::DSUM,DSUMALL,DMEAN
```

```
      INCLUDE 'mpif.h'
C
      CALL MPI_INIT(IERROR)
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, IRANK, IERROR )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, NP, IERROR )
C
      IX=123
      J=IRANK*JUMP
      CALL DM_VRANU5(IX,X,N,J,WORK,ICON)
      IF(ICON.NE.0) THEN
        WRITE(6,*) 'DM_VRANU5 ERROR ICON= ',ICON
      END IF
C
      DSUM=0.0D0
      DO I=1,N
        DSUM=DSUM+X(I)
      END DO
      CALL MPI_REDUCE(DSUM,DSUMALL,1,MPI_REAL8,MPI_SUM,0,
     -                MPI_COMM_WORLD,IERROR)
C Compute overall mean
      DNALL=DFLOAT(N)*DFLOAT(NP)
      IF(IRANK.EQ.0) THEN
        DMEAN=DSUMALL/DNALL
        WRITE(6,*) 'Mean    ',DMEAN
      END IF
C
      CALL MPI_FINALIZE(IERROR)
      END
```

Example 3.

Two uniform pseudo random number sequences X and Y are generated by four MPI process and their mean values are calculated. The total number of each vector is 1,000,000 and the starting point is 123.

In this program, 1,000,000 pseudo random numbers are split into NP blocks, where NP is the number of processes, and each of the sequences is generated by each of the processes. Even if NP is changed, the whole sequence of pseudo random numbers is the same.

```
C     **EXAMPLE 3**
      INTEGER::NX,NY,NP
      PARAMETER(NX=100000)
      PARAMETER(NY=100000)
      PARAMETER(NP=4)        ! NUMBER OF PROCESS
      REAL(8)::X((NX+NP-1)/NP),Y((NY+NP-1)/NP)
      INTEGER::IRANK,NSIZE,IERROR
      INTEGER::IX,NL,ICON,JUMP
      INTEGER::I
      INTEGER(8)::J0,J
      REAL(8)::WORK(8)
```

```
      REAL(8)::DSUM,DSUMALL,DMEAN
      INCLUDE 'mpif.h'
C
      CALL MPI_INIT(IERROR)
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, IRANK, IERROR )
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, NSIZE, IERROR )
      IF(NP.NE.NSIZE) THEN
        CALL MPI_FINALIZE(IERROR)
        STOP
      END IF
C
      IX=123
      JUMP=(NX+NP-1)/NP
      J=MIN(IRANK*JUMP,NX)
      NL=MIN(JUMP,NX-J)
      IF(NL.GE.1) THEN
        CALL DM_VRANU5(IX,X,NL,J,WORK,ICON)
        IF(ICON.NE.0) THEN
          WRITE(6,*) 'DM_VRANU5 ERROR ICON= ',ICON
        END IF
        J0=NX-(J+NL)
      ELSE
        J0=NX
      END IF
C
      DSUM=0.0D0
      DO I=1,NL
        DSUM=DSUM+X(I)
      END DO
      CALL MPI_REDUCE(DSUM,DSUMALL,1,MPI_REAL8,MPI_SUM,0,
     *                MPI_COMM_WORLD,IERROR)
C Compute overall mean of X
      IF(IRANK.EQ.0) THEN
        DMEAN=DSUMALL/DFLOAT(NX)
        WRITE(6,*) 'Mean of X   ',DMEAN
      END IF
C
      JUMP=(NY+NP-1)/NP
      J=MIN(IRANK*JUMP,NY)
      NL=MIN(JUMP,NY-J)
      J=J+J0
      IF(NL.GE.1) THEN
        CALL DM_VRANU5(IX,Y,NL,J,WORK,ICON)
        IF(ICON.NE.0) THEN
          WRITE(6,*) 'DM_VRANU5 ERROR ICON= ',ICON
        END IF
      END IF
C
      DSUM=0.0D0
      DO I=1,NL
        DSUM=DSUM+Y(I)
      END DO
      CALL MPI_REDUCE(DSUM,DSUMALL,1,MPI_REAL8,MPI_SUM,0,
```

```
     *                MPI_COMM_WORLD,IERROR)
C Compute overall mean of Y
      IF(IRANK.EQ.0) THEN
        DMEAN=DSUMALL/DFLOAT(NY)
        WRITE(6,*) 'Mean of Y   ',DMEAN
      END IF
C
      CALL MPI_FINALIZE(IERROR)
      END
```

(4) Method

This subroutine uses the Multiple Recursive Generator with 8th-order full primitive polynomials (MRG8). The sequence of pseudo random numbers x_1, x_2, ... is generated by the following formula.

$x_i = (a_1x_{i-1} + a_2x_{i-2} + a_3x_{i-3} + a_4x_{i-4} + a_5x_{i-5} + a_6x_{i-6} + a_7x_{i-7} + a_8x_{i-8}) \bmod p$
$p=2^{31}-1$,
$a_1= 1089656042$, $a_2= 1906537547$, $a_3= 1764115693$, $a_4= 1304127872$,
$a_5= 189748160$, $a_6= 1984088114$, $a_7= 626062218$, $a_8= 1927846343$.
$DA(i)= x_i*(1/p)$

The period of the random number sequence is $(2^{32}-1)^8-1$ (about $4.5*10^{74}$).

The method and implementation details are described in [82] in Appendix A, "References.".

MRG8 give the good result in Monte Carlo Simulations, see [83] in Appendix A, "References."

(5) Tests for uniform random numbers

This subroutine passed bigCrush test of TESTU01 which is the statistical testing program of uniform random number generators. See [84] for the details of TESU01.

DM_VSCHOL

LDL^T decomposition of a symmetric positive definite sparse matrix (Left-looking Cholesky decomposition method)
CALL DM_VSCHOL(A, NZ, NROW, NFCNZ, N, IORDERING, NPERM, ISW, EPSZ, NASSIGN, NSUPNUM, NFCNZFACTOR, PANELFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NDIM, NPOSTO, W, IW1, IW2, IW3, ICON)

(1) Function

This subroutine executes LDL^T decomposition for an $n \times n$ symmetric positive definite sparse matrix using modified Cholesky decomposition method, so that

$$\boldsymbol{QPAP}^T\boldsymbol{Q}^T = \boldsymbol{LDL}^T, \tag{1.1}$$

where $\boldsymbol{P}$ is a permutation matrix of ordering and $\boldsymbol{Q}$ is a permutation matrix of post ordering. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices, L is a unit lower triangular matrix, and $\boldsymbol{D}$ is a diagonal matrix.

(2) Parameter

A ..... Input. The non-zero elements of the lower triangular part $\{a_{ij} \mid i \geq j\}$ of a symmetric sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ ..... Input. The total number of the nonzero elements belong to the lower triangular part of a symmetric sparse matrix A.

NROW ..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IORDERING ..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PAP}^T$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

Specify IORDERING=1 for the decomposition of the matrix $\boldsymbol{PAP}^T$.

Specify the other value for the decomposition of the matrix $\boldsymbol{A}$ as it is.

NPERM ..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information .

1) Specify ISW=1 for the first call.

2) Specify ISW=2 for the subsequent call if the previous call has failed with ICON=31000, that means the size of PANELFACTOR or NPANELINDEX were not enough. In this case, the PANELFACTOR or NPANELINDEX must be reallocated with the necessary sizes which are returned in the NSIZEFACTOR or NSIZEINDEX at the precedent call.

Besides, the values of A, NZ, NROW, NFCNZ, N, IORDERING, NPERM, NASSIGN, NSUPNUM, NFCNZFACTOR, NFCNZINDEX, NPANELINDEX, NPOSTO, NDIM, W, IW1, IW2, and IW3 must be unchanged after the first call.

3) Specify ISW=3 for the second and subsequent calls when solving another system of equations which have the same non-zero pattern of the matrix ***A*** but the values of its elements are different. In this case, the information obtained in symbolic decomposition and the array PANELFACTOR and NPANELINDEX of the same size required in previous call can be reused. Then numerical LDL^T decomposition will proceed with that information and the new linear equations can be solved efficiently. Store the values of the matrix elements in the array A, or store in another array B and let it be as the parameter A.

Besides, the values of NZ, NROW, NFCNZ, N, IORDERING, NPERM, NASSIGN, NSUPNUM, NFCNZFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NPOSTO, NDIM, W, IW1, IW2, and IW3 must be unchanged as the previous call.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed.

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

NASSIGN ..... Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position, where this panel is allocated as a part of the one-dimensional array PANELFACTOR. When *j*=NASSIGN(*i*), the *i*-th supernode is allocated at *j*-th position.

Input. The values of the first call are reused when ISW $\neq$ 1 specified.

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

One-dimensional array NASSIGN(N).

(See note 3) in (3), "Comments on use.")

NSUPNUM ..... Output. The total number of supernodes.

Input. The values of the first call are reused when ISW $\neq$ 1 specified. ($\leq$ n)

NFCNZFACTOR.. Output. Each supernode consists of multiple column vectors, and the factorized matrix of supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element panel(1,1) of the *i*-th panel, where this panel is allocated as a part of the one-dimensional array PANELFACTOR.

One-dimensional 8-byte integer array NFCNZFACTOR(N+1).

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTOR.. Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These panels are stored in this matrix.

The positions of the panel corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZFACTOR(*j*). The decomposed result is stored in each panel.

The size of the i-th panel can be considered to be two-dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding part where the lower triangular unit matrix except the diagonal part is stored in panel(*s*, *t*), *s* > *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*) of the *i*-th panel. The corresponding part of the diagonal matrix ***D*** is stored in panel(*t*, *t*).

One-dimensional array PANELFACTOR(NSIZEFACTOR).

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

(See note 4) in (3), "Comments on use.")

NSIZEFACTOR... Input. The size of the array PANELFACTOR. 8-byte integer.

Output. The necessary size for the array PANELFACTOR is returned.

(See note 4) in (3), "Comments on use.")

NFCNZINDEX..... Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element of the *i*-th row indices vector, where this panel is allocated as a part of the one-dimensional array NPANELINDEX.

One-dimensional 8-byte integer array NFCNZINDEX(N+1).

Input. The values set by the first call are reused when ISW ≠ 1 specified.

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

NPANELINDEX.. Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These row indices vectors are stored in this matrix. The positions of the row pointer vector corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZINDEX(j). The row indices vector is stored by each panel. This row indices are the row indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the matrix A is permuted by post ordering.

One-dimensional array NPANELFACTOR(NSIZEINDEX).

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

(See note 4) in (3), "Comments on use.")

NSIZEINDEX..... Input. The size of the array PANELINDEX. 8-byte integer.

Output. The necessary size is returned.

(See note 4) in (3), "Comments on use.")

NDIM ..... Output. The size of first and second dimension of the *i*-th panel are stored in NDIM(1,*i*) and NDIM(2,*i*) respectively.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

Two-dimensional array NDIM(2,N).

For the storage method of the decomposed results, refer to Figure DM_VSCHOL-1.

NPOSTO ..... Output. The one dimensional vector is stored which indicates what column index of A the *i*-th node in post ordering corresponds to.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array NPOSTO(N).

(See note 5) in (3), "Comments on use.")

W ..... Work area.

Output/Input.

When IORDERING=1, one-dimensional array of size NZ.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

When IORDERING≠1, one-dimensional array of size 1.

IW1 ..... Work area.

Output/Input.

When IORDERING=1, one-dimensional array of size NZ+N+1.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

When IORDERING≠1, one-dimensional array of size 1.

IW2 ..... Work area.

Output/Input. One-dimensional array of size NZ+N+1.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

IW3 ..... Work area.

Output/Input. One-dimensional array of size N×35+35.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

ICON ..... Output. Condition code.

(See Table DM_VSCHOL-1.)

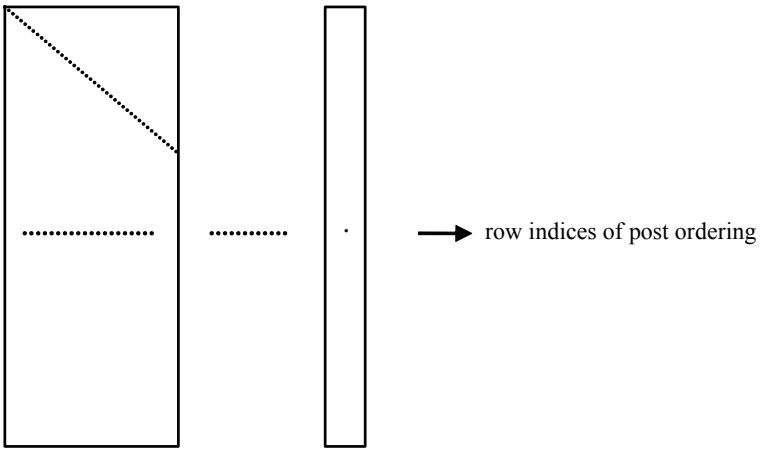


Figure DM_VSCHOL-1 concept of storing the data for decomposed result

j = NASSIGN(i) → The i-th supernode is stored at the j-th position.

p = NFCNZFACTOR(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTOR.

q = NFCNZINDEX(j) → The row pointer vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of PANELINDEX.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular unit matrix $\boldsymbol{L}$ except the diagonal part is stored in

panel(s, t), $s > t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The corresponding part of the diagonal matrix D is stored in panel(t, t).

The row pointers indicate the column indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the node of the matrix A is permuted by post ordering.

Table DM_VSCHOL-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	The coefficient matrix is not positive definite.	Processing is continued.
20000	The pivot became relatively zero. The coefficient matrix A may be singular.	Processing is discontinued.
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTOR < 1, NSIZEINDEX < 1, EPSZ < 0.0, ISW < 1, or ISW > 3.	

Code	Meaning	Processing
30100	The permutation matrix specified in NPREM is not correct.	Processing is discontinued.
30200	The row pointer k stored in NROW(j) is $k < i$ or $k >$ N.	
30300	The number of row indices belong to i-th column is NFCNZ(i+1)-NFCNZ(i) > n - i+1.	
30400	There is a column without a diagonal element.	
31000	The value of NSIZEFACTOR is not enough as the size of PANELFACTOR, or the value of NSIZEINDEX is not enough as the size of NPANELINDEX.	Reallocate the PANELFACTOR or NPANELINDEX with the necessary size which are returned in the NSIZEFACTOR or NSIZEINDEX, and call this subroutine again with ISW=2.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix $\boldsymbol{P}$, set NPERM(i)=j.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LDL^T decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
When the pivot becomes negative during the decomposition, the coefficient matrix is not a positive definite. In this case, processing is continued as ICON=10000, but the numerical error may be large because of no pivoting.

3) The linear equations $\boldsymbol{LDL^TPQx} = \boldsymbol{PQb}$ which is a derived form from $\boldsymbol{Ax} = \boldsymbol{b}$ can be solved by calling subroutine DM_VSCHOLX following this subroutine with the decomposed result data such as NASSIGN, NSUPNUM, NFCNZFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NPOSTO, NDIM, IW3 unchanged.

4) The necessary sizes for the array PANELFACTOR and NPANELINDEX that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in

NSIZEFACTOR and NSIZEINDEX. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTOR and NSIZEINDEX are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

5) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2, a_3 and c are zero constants, that means the operator is Laplacian. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine init_mat_diag, and transferred into compressed column storage format.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7,NDIAGH=4)

      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAGH*N),NROW(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),NASSIGN(N),W(NDIAGH*N),
     $          NPOSTO(N),NDIM(2,N),
     $          IW1(NDIAGH*N+N+1),
     $          IW2(NDIAGH*N+N+1),
     $          IW3(35*N+35)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTOR
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEX
      REAL*8 DUMMYF
      INTEGER*4 NDUMMYI
      INTEGER*8 NSIZEFACTOR,NSIZEINDEX,
```

```
     $          NFCNZFACTOR(N+1),
     $          NFCNZINDEX(N+1)
      DIMENSION X(N),B(N),SOLEX(N)


      PRINT *,'    LEFT-LOOKING MODIFIED CHOLESKY METHOD'
      PRINT *,'    FOR SPARSE POSITIVE DEFINITE MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 0.0D0
      VA2 = 0.0D0
      VA3 = 0.0D0
      VC =  0.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
      NUMNZ=1
      DO J=1,N
      NTOPCFGC=1
      NTOPCFG=1
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NCOL.GE.J)THEN
```

```
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
      ENDIF

C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(NCOL.GE.J)THEN
      NUMNZ=NUMNZ+1
      ENDIF
C
      NUMNZC=NUMNZC+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1
C

      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
     $             B,WC,IWC,ICON)
C
      X=B
      IORDERING=0
      ISW=1
      EPSZ=0.0D0
      NSIZEFACTOR=1
      NSIZEINDEX=1

      CALL DM_VSCHOL(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,DUMMYF,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NDUMMYI,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      PRINT *
      PRINT *,'    ICON = ',ICON,' NSIZEFACTOR = ',NSIZEFACTOR,
     $       'NSIZEINDEX = ',NSIZEINDEX
      PRINT *
C
C     ALLOCATE STORAGES IN RETURNED SIZES
```

```
C
      ALLOCATE( PANELFACTOR(NSIZEFACTOR) )
      ALLOCATE( NPANELINDEX(NSIZEINDEX) )

      ISW=2

      CALL DM_VSCHOL(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,PANELFACTOR,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NPANELINDEX,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      CALL DM_VSCHOLX(N,IORDERING,
     $              NPERM,X,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,PANELFACTOR,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NPANELINDEX,NSIZEINDEX,NDIM,NPOSTO,
     $              IW3,ICON)

      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTOR,NPANELINDEX )

      STOP
      END

C =====================================
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &               ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
```

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

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
    PRINT*,'ERROR; K0.GH.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
```

```
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C ======================================
* SOLUTE ERROR
* | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

Through the symbolic decomposition process, this routine analyze the data dependence among columns and the structure of the non-zero elements of matrix ***L*** which is a factor matrix of modified Cholesky LDL^T decomposition. Based on this analysis, the supernodes that bundles certain columns are detected. The columns which have similar non-zero pattern are merged as a supernode together. This means that some rows include additional zero elements and that the number of columns composing a supernode increases. Then data during the numerical decomposition on cache is reused efficiently.

A union set of the row indices that indicate the row indices of the nonzero element of the result of the modified Cholesky decomposition is computed on the columns that compose a supernode. The result of the modified Cholesky decomposition of supernodes is stored compressing it into the two-dimensional panel of which size of the first dimension becomes the number of elements of this set of row indices. The set of row indices is represented as a vector.

The left-looking modified Cholesky decomposition method is used.

For general information on this topic, refer to [19] in Appendix A, “References.”.

DM_VSCHOLX

A system of linear equations with LDL^T-decomposed symmetric positive definite sparse matrices
CALL DM_VSCHOLX(N, IORDERING, NPERM, B, NASSIGN, NSUPNUM, NFCNZFACTOR, PANELFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NDIM, NPOSTO, IW3, ICON)

(1) Function

This subroutine solves a system of equations with a LDL^T-decomposed symmetric positive definite sparse coefficient $n \times n$ matrix.

$$\boldsymbol{LDL^TQPx} = \boldsymbol{QPb}, \qquad (1.1)$$

where $\boldsymbol{P}$ is a permutation matrix of ordering and $\boldsymbol{Q}$ is a permutation matrix of post ordering. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices, L is a unit lower triangular matrix, $\boldsymbol{D}$ is a diagonal matrix, $\boldsymbol{b}$ is a constant vector, and $\boldsymbol{x}$ is a solution vector.

(2) Parameter

N ..... Input. Order n of matrix.

IORDERING ..... Input. Control information whether the coefficient matrix was permuted into $\boldsymbol{PAP}^T$ by the permutation matrix $\boldsymbol{P}$ before decomposition.

Specify IORDERING=1 for the LDL^T decomposed from $\boldsymbol{PAP}^T$.

Specify the other value for the LDL^T decomposed matrix from $\boldsymbol{A}$ as it is.

NPERM ..... Input. The permutation matrix $\boldsymbol{P}$ is specified as a vector when IORDERING=1.

One-dimensional array NPERM(N).

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

B ..... Input. The right-hand side constant vector $\boldsymbol{b}$ of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

One-dimensional array B(N).

NASSIGN ..... Input. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position, where this panel is allocated as a part of the one-dimensional array PANELFACTOR. When j=NASSIGN(i), the i-th supernode is allocated at j-th position.

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

One-dimensional array NASSIGN(N).

NSUPNUM ..... Input. The total number of supernodes.

NFCNZFACTOR.. Input. Each supernode consists of multiple column vectors, and the factorized matrix of supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element panel(1,1) of the i-th panel,

where this panel is allocated as a part of the one-dimensional array PANELFACTOR.

One-dimensional 8-byte integer array NFCNZFACTOR(N+1).

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

(See note 3) in (3), "Comments on use.")

PANELFACTOR.. Input. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These panels are stored in this matrix.

The positions of the panel corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZFACTOR(*j*). The decomposed result is stored in each panel.

The size of the i-th panel can be considered to be two-dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding part where the lower triangular unit matrix except the diagonal part is stored in panel(*s*, *t*), *s* > *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*) of the *i*-th panel. The corresponding part of the diagonal matrix ***D*** is stored in panel(*t*, *t*).

One-dimensional array PANELFACTOR(NSIZEFACTOR).

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

NSIZEFACTOR.. Input. The size of the array PANELFACTOR. 8-byte integer.

NFCNZINDEX... Input. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element of the *i*-th row indices vector, where this panel is allocated as a part of the one-dimensional array NPANELINDEX.

One-dimensional 8-byte integer array NFCNZINDEX(N+1).

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

NPANELINDEX.. Input. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These row pointer vectors are stored in this matrix. The positions of the row pointer vector corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZINDEX(j). The row indices vector is stored by each panel. This row indices are the row indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the matrix A is permuted by post ordering.

One-dimensional array NPANELFACTOR(NSIZEINDEX).

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

NSIZEINDEX..... Input. The size of the array PANELINDEX. 8-byte integer.

NDIM ..... Input. The size of first and second dimension of the *i*-th panel are stored in NDIM(1,*i*) and NDIM(2,*i*) respectively.

Two-dimensional array NDIM(2,N).

For the storage method of the decomposed results, refer to Figure DM_VSCHOLX-1.

NPOSTO ..... Input. The one dimensional vector is stored which indicates what column index of A the *i*-th node in post ordering corresponds to.

One-dimensional array NPOSTO(N).

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

IW3 ..... Input. Specify the IW3 which is used by DM_VSCHOL before calling this routine. The contents must not be changed.

One-dimensional array IW3(N×35+35).

ICON ..... Output. Condition code.

(See Table DM_VSCHOLX-1.)

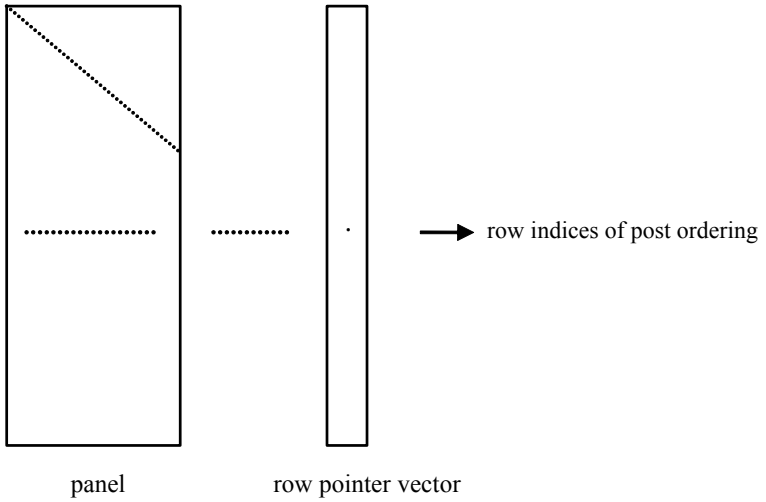


Figure DM_VSCHOLX-1 concept of storing the data for decomposed result

j = NASSIGN(i) → The i-th supernode is stored at the j-th position.

p = NFCNZFACTOR(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTOR.

q = NFCNZINDEX(j) → The row pointer vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of PANELINDEX.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular unit matrix $\boldsymbol{L}$ except the diagonal part is stored in

panel(s, t), $s > t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The corresponding part of the diagonal matrix D is stored in panel(t, t).

The row pointers indicate the column indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the node of the matrix A is permuted by post ordering.

Table DM_VSCHOLX-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 1, NSIZEFACTOR < 1, NSIZEINDEX < 1, or NSUPNUM < 1.	Processing is discontinued.
30100	The permutation matrix specified in NPREM is not correct.	

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

2) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix ***A*** into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

3) The linear system of equations can be solved by calling this subroutine with specifying the $\mathrm{LDL}^{\mathrm{T}}$-decomposed results which are calculated by DM_VSCHOL subroutine.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where ***A*** results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2, a_3 and c are zero constants, that means the operator is Laplacian. The matrix ***A*** in Diagonal format is generated by the subroutine init_mat_diag, and transferred into compressed column storage format.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
```

```
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7,NDIAGH=4)

      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAGH*N),NROW(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),NASSIGN(N),W(NDIAGH*N),
     $          NPOSTO(N),NDIM(2,N),
     $          IW1(NDIAGH*N+N+1),
     $          IW2(NDIAGH*N+N+1),
     $          IW3(35*N+35)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTOR
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEX
      REAL*8 DUMMYF
      INTEGER*4 NDUMMYI
      INTEGER*8 NSIZEFACTOR,NSIZEINDEX,
     $          NFCNZFACTOR(N+1),
     $          NFCNZINDEX(N+1)
      DIMENSION X(N),B(N),SOLEX(N)


      PRINT *,'    LEFT-LOOKING MODIFIED CHOLESKY METHOD'
      PRINT *,'    FOR SPARSE POSITIVE DEFINITE MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 0.0D0
      VA2 = 0.0D0
      VA3 = 0.0D0
      VC =  0.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
```

```
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
      NUMNZ=1
      DO J=1,N
      NTOPCFGC=1
      NTOPCFG=1
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NCOL.GE.J)THEN
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
      ENDIF

C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(NCOL.GE.J)THEN
      NUMNZ=NUMNZ+1
      ENDIF
C
      NUMNZC=NUMNZC+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1
C

      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
     $             B,WC,IWC,ICON)
C
```

```
      X=B
      IORDERING=0
      ISW=1
      EPSZ=0.0D0
      NSIZEFACTOR=1
      NSIZEINDEX=1

      CALL DM_VSCHOL(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,DUMMYF,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NDUMMYI,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      PRINT *
      PRINT *,'    ICON = ',ICON,' NSIZEFACTOR = ',NSIZEFACTOR,
     $       'NSIZEINDEX = ',NSIZEINDEX
      PRINT *
C
C     ALLOCATE STORAGES IN RETURNED SIZES
C
      ALLOCATE( PANELFACTOR(NSIZEFACTOR) )
      ALLOCATE( NPANELINDEX(NSIZEINDEX) )

      ISW=2

      CALL DM_VSCHOL(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,PANELFACTOR,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NPANELINDEX,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      CALL DM_VSCHOLX(N,IORDERING,
     $              NPERM,X,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,PANELFACTOR,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NPANELINDEX,NSIZEINDEX,NDIM,NPOSTO,
     $              IW3,ICON)

      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *
      PRINT *
```

```
      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTOR,NPANELINDEX )

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &               ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
    PRINT*,'ERROR; K0.GH.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
* SOLUTE ERROR
* | X1 - X2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

DM_VSCLU

LU decomposition of an unsymmetric complex sparse matrix
CALL DM_VSCLU(ZA, NZ, NROW, NFCNZ, N, IPLEDSM, MZ, ISCLITERMAX, IORDERING, NPERM, ISW, NROWSYM, NFCNZSYM, NASSIGN, NSUPNUM, NFCNZFACTORL, ZPANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, ZPANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, SCLROW,SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, NFCNZPIVOT, NPIVOTP, NPIVOTQ, ZW, W, IW1, IW2, ICON)

(1) Function

The large entries of an $n \times n$ unsymmetric complex sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. And LU decomposition is performed, in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.

The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for the permutation of elements, scaling and Pivot.

The unsymmetric complex sparse matrix is transformed as below.

$$\boldsymbol{A_1=D_rAP_cD_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2=QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM=A_1+A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate.
If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.

(2) Parameter

ZA..... Input. The nonzero elements of an unsymmetric sparse matrix $\boldsymbol{A}$ are stored in ZA(1:NZ).

A double precision complex one-dimensional array ZA(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector). For a complex matrix , a real array CC in this Figure is replaced with a complex array.

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric complex sparse matrix $\boldsymbol{A}$.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array ZA.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array ZA in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IPLEDSM..... Input. Control information whether to permute the large entries to the diagonal of a matrix $\boldsymbol{A}$.
When IPLEDSM=1 is specified, a matrix $\boldsymbol{A}$ is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Output. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(i)=j indicates that the j-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to i-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix $\boldsymbol{A}$.

When ISCLITERMAX ≤ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX ≥ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

When IORDERING=10 is specified, calling this routine with ISW=1 produces the informations which is needed to generate an ordering regarding $\boldsymbol{A_1}$ and they are set in NROWSYM and NFCNZSYM.

When IORDERING 11 is specified, it is indicated that after an ordering is set in NPERM, the computation is resumed.
Using the informations obtained in NROWSYM and NFCNZSYM after calling this routines with ISW=1 and IORDERING=10, an ordering is determined.
After specifying this ordering in NPERM, this routine is called again with ISW=1and IORDERING=11 and the computation is resumed.
LU decomposition of the matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is continued.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

Output. IORDERING is set to 11 after this routine is called with IORDERING=10 and ISW=1. Therefore after an ordering is set in NPERM the computation is resumed in the subsequent call without IORDERING=11 being specified explicitly.

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

NPERM..... Input. The permutation matrix ***P*** is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
After symmetrization of a matrix and symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.
Call with IORDERING=10 produces the informations needed for seeking an ordering in NROWSYM and NFCNZSYN. Using these informations an ordering for ***SYM*** is determined. After an ordering is set in NPERM, calling this routine with IORDERING=11 and also ISW=1 again resumes the computation.
When IORDERING is neither 10 nor 11, no ordering is specified.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of ZPANELFACTORL or ZPANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed.
Before calling again with ISW=2, the ZPANELFACTORL or ZPANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEXL or NSIZEINDEXU at the precedent call and specified in corresponding arguments.
Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

NROWSYM..... Output. When it is called with IORDERING=10, the row indices of nonzero pattern of the lower triangular part of $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1}^T$ in the compressed column storage method are generated.

One-dimensional array NROWSYM(NZ+N).

NFCNZSYM..... Output. When it is called with IORDERING=10, the position of the first row index of each column stored in array NROWSYM in the compressed column storage method which stores the nonzero pattern of the lower part of a matrix ***SYM*** column by column.

NFCNZSYM(N+1)=NSYMZ+1 where NSYMZ is the total nonzero elements in the lower triangular part.

One-dimensional array NFCNZ(N+1).

NASSIGN..... Output. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in ZPANELFACTORL and ZPANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively.
Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSCLU-1. One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of an unsymmetric complex sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

ZPANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array ZPANELFACTORL. 8-byte integer.

Output. The necessary size for the array ZPANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric complex sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

ZPANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array ZPANELFACTORU. 8-byte integer.

Output. The necessary size for the array ZPANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of a complex number with the absolute value of SPEPSZ.
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified.
The following conditions must hold.
THEPSZ≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

NFCNZPIVOT.... Output. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the *i*-th supernode is assigned is known by *j*=NASSIGN(*i*). The position of the first element of that section is stored in NFCNZPIVOT(*j*). The information of exchange rows and columns within the *i*-th supernode is stored in the elements of is=NFCNZPIVOT(*j*),…, ie=NFCNZPIVOT(*j*)+NDIM(2,*j*)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Output. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Output. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

ZW.....Work area.

Output/Input.
A double precision complex one-dimensional array of size 2*NZ.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

W..... Work area.

Output/Input.
One-dimensional array of size 4*NZ+6*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW1..... Work area.

Output/Input.
One-dimensional array of size 2*NZ+2*(N+1)+16*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW2..... Work area.

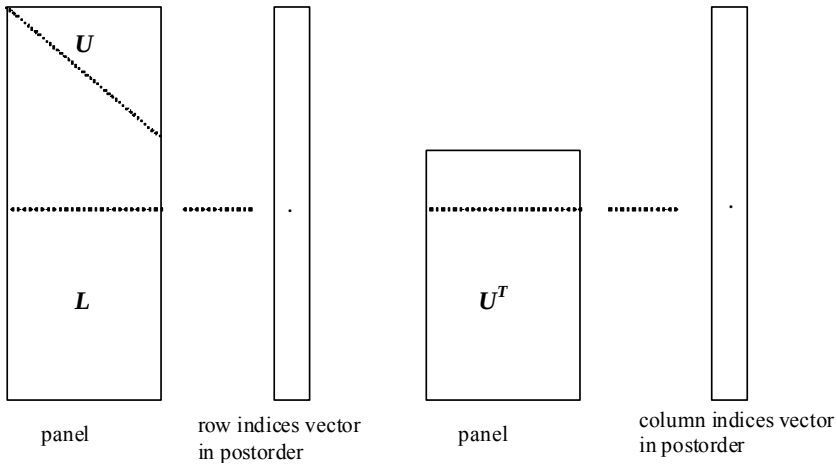
Output/Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSCLU-1.)

Figure DM_VSCLU-1 Conceptual scheme for storing decomposed results



j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of ZPANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular matrix $\boldsymbol{L}$ of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix $\boldsymbol{U}$ of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,DIM(2, j),

t = 1,...,DIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (DIM(3, j)-DIM(2,j))×DIM(2, j) from the u-th element of ZPANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length DIM(3,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, j)-DIM(2, j))×DIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U^T}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix $\boldsymbol{A}$ is permuted in post ordering.

Table DM_VSCLU-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	When ISTATIC=1 is specified, Static pivot which replaces the pivot candidate with too small value with SPEPSZ is made.	–
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
20100	When IPLEDSM is specified, maximum matching with the length N is sought in order to permute large entries to the diagonal but can not be found. The coefficient matrix ***A*** may be singular.	
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix ***A***. The coefficient matrix ***A*** may be singular.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ISW < 1, or ISW > 2	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index *k* stored in NROW(*j*) is *k* < 1 or *k* >*n*.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n*.	
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or SPEPSZ>THEPSZ	
31000	The value of NSIZEFACTORL is not enough as the size of ZPANELFACTORL, or the value of NSIZEINDEXL is not enough as the size of NPANELINDEXL, or the value of NSIZEFACTORU is not enough as the size of ZPANELFACTORU, or the value of NSIZEINDEXU is not enough as the size of NPANELINDEXU.	Reallocate the ZPANELFACTORL or NPANELINDEXL or ZPANELFACTORU or NPANELINDEXU with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEXL or NSIZEFACTORU or NSIZEINDEXU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix $\boldsymbol{P}$, set NPERM(i)=j.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, "References." in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for Pivot.
When the computation is to be continued even if the absolute value of diagonal element is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with a complex number with the absolute value of SPEPSZ.

3) The necessary sizes for the array ZPANELFACTORL, NPANELINDEXL, ZPANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the i-th node in post order is corresponding. It means j-th position when j = NPOSTO(i).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) A system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be solved by calling DM_VSCLUX subsequently in use of the results of LU decomposition obtained by this routine. The following arguments used in this routine are specified.
See example in (3), "Comments on use.".

```
ZA, NZ, NROW, NFCNZ, N,
IPLEDSM, MZ, IORDERING, NPERM,
NASSIGN, NSUPNUM,
NFCNZFACTORL, ZPANELFACTORL,
```

NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL,
NSIZEINDEXL, NDIM,
NFCNZFACTORU, ZPANELFACTORU, NSIZEFACTORU,
NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO,
SCLROW,SCLCOL,
NFCNZPIVOT,
NPIVOTP, NPIVOTQ, IW2

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix $\boldsymbol{A}$ built in this way is stored into a complex sparse array and is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
     $      N = KX*KY*KZ)
      PARAMETER (NBORDER=N+1,NOFFDIAG=6)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      INTEGER*4 WL,ZWL
      PARAMETER (NALL=NDIAG*N,
C
     $   ZWL =2*NALL,
     $   WL  =4*NALL+6*N,
     $   IW1L=2*NALL+2*(N+1)+16*N,
     $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      COMPLEX*16 ZA(K*NDIAG),ZWC(K*NDIAG),
     $           ZW(ZWL),ZONE
      PARAMETER(ZONE=(1.0D0,0.0D0))
      DIMENSION NROW(K*NDIAG),NFCNZ(N+1),
     $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
     $          IWC(2,K*NDIAG)
      DIMENSION NPERM(N),W(WL),
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
      COMPLEX*16, DIMENSION(:), ALLOCATABLE ::
     $        ZPANELFACTORL,ZPANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      COMPLEX*8 ZDUMMYFL,ZDUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
```

```
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      COMPLEX*16 ZB(N),ZSOLEX(N)
      REAL*8 THEPSZ,EPSZ,SPEPSZ,
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      ZSOLEX(I)= ZONE
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',ZSOLEX(1),' X(N) = ',ZSOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
```

```
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      ZA(NUMNZ)=DCMPLX(DIAG2(J,I),0.0D0)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCCC(ZA,NZ,NROW,NFCNZ,N,ZSOLEX,
     $             ZB,ZWC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
      NSIZEINDEXU=1
      EPSZ=1.0D-16
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSCLU(ZA,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZDUMMYFL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZDUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NDUMMYIU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
```

```
     $              ZW,W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
      ALLOCATE( ZPANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( ZPANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSCLU(ZA,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZPANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZPANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              ZW,W,IW1,IW2,ICON)
C
      CALL DM_VSCLUX(N,
     $              IORDERING,
     $              NPERM,
     $              ZB,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZPANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZPANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              IPLEDSM,MZ,
     $              SCLROW,SCLCOL,
     $              NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              ZA,NZ,NROW,NFCNZ,
     $              IW2,ICON)
C
      ERR = ERRNRM(ZSOLEX,ZB,N)
```

```
      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',ZB(1),' X(N) = ',ZB(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF
C
      DEALLOCATE( ZPANELFACTORL,ZPANELFACTORU,
     $            NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =====================================
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &             ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY
```

```
C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END
C ======================================
* SOLUTE ERROR
* | Z1 - Z2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(Z1,Z2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      COMPLEX*16 Z1(*),Z2(*),SS
C
      S = 0D0
      DO 100 I = 1,LEN
        SS  = Z1(I) - Z2(I)
        S = S + DREAL(SS * DCONJG(SS))
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The permutation which moves large entries to the diagonal is performed. And the permutated matrix is scaled in order to equilibrate both rows and columns norms.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for the permutation of elements, scaling and Pivot.
The LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.
Refer to references in Appendix A, “References.” in detail.
Refer to [23], [57] on the method how the elements of large absolute value are permuted to diagonal, to [13] on the application algorithms of matching, to [17] on Fibonacci Heaps, to [19], [2], [22], [48], [68] on the base of the LU decomposition of unsymmetric complex sparse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VSCLUX

A system of linear equations with LU-decomposed unsymmetric complex sparse matrices
CALL DM_VSCLUX(N, IORDERING, NPERM, ZB, NASSIGN, NSUPNUM, NFCNZFACTORL, ZPANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, ZPANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, IPLEDSM, MZ, SCLROW,SCLCOL, NFCNZPIVOT, NPIVOTP, NPIVOTQ, IREFINE, EPSR, ITERMAX, ITER, ZA, NZ, NROW, NFCNZ, IW2, ICON)

(1) Function

An $n \times n$ unsymmetric complex sparse matrix $\boldsymbol{A}$ of which LU decomposition is made as below is given. In this decomposition the large entries of an $n \times n$ unsymmetric complex sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. Subsequently LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode is performed and results in the following form. This routine solves the following linear equation in use of these results of LU decomposition.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for the permutation of elements, scaling and Pivot.

$$\boldsymbol{Ax=b}$$

A matrix $\boldsymbol{A}$ is decomposed into as below.

$$\boldsymbol{P_{rs}QPD_rAP_cD_cP^TQ^TP_{cs}=LU}$$

The unsymmetric complex sparse matrix $\boldsymbol{A}$ is transformed as below.

$$\boldsymbol{A_1=D_rAP_cD_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2=QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
$\boldsymbol{P_{rs}}$ and $\boldsymbol{P_{cs}}$ represent row and column exchanges in orthogonal matrices respectively. The actual exchanges are restricted to the reduced part of the matrix belonging to each supernode.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM=A_1+A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

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

IORDERING..... Input. When IORDERING 11 is specified, it is indicated that LU decomposition is performed with an ordering specified in NPERM.
The matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is decomposed into LU decomposition.

Otherwise. No ordering is specified.

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

NPERM..... Input. When IORDEING=11 is specified, a vector presenting the permutation matrix ***P*** used is stored.

One-dimensional array NPERM(N).

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

ZB..... Input. The right-hand side constant vector ***b*** of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector ***x***.

A double precision complex one-dimensional array ZB(N).

NASSIGN..... Input. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in ZPANELFACTORL and ZPANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Regarding the storage methods of decomposed matrices, refer to Figure DM_VSCLUX-1.
One-dimensional array NASSING(N).

NSUPNUM..... Input. The total number of supernodes.(≤ n)

NFCNZFACTORL..Input. The decomposed matrices ***L*** and ***U*** of an unsymmetric complex sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

ZPANELFACTORL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NSIZEFACTORL.. Input. The size of the array ZPANELFACTORL. 8-byte integer.

NFCNZINDEXL... Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NPANELINDEXL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

NDIM..... Input. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NFCNZFACTORU..Input. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric complex sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

ZPANELFACTORU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NSIZEFACTORU.. Input. The size of the array ZPANELFACTORU. 8-byte integer.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NPANELINDEXU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCLUX-1.

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

NPOSTO..... Input. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

One-dimensional array NPOSTO(N).

(See note 3) in (3), "Comments on use.")

IPLEDSM..... Input. Information indicating whether for LU decomposition it is specified to permute the large entries to the diagonal of a matrix ***A***.
When IPLEDSM=1 is specified, a matrix ***A*** is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Input. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(i)=j indicates that the j-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to i-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

SCLROW..... Input. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

One-dimensional array SCLROW (N).

SCLCOL..... Input. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

One-dimensional array SCLCOL(N).

NFCNZPIVOT.... Input. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the i-th supernode is assigned is known by j=NASSIGN(i). The position of the first element of that section is stored in NFCNZPIVOT(j). The information of exchange rows and columns within the i-th supernode is stored in the elements of is=NFCNZPIVOT(j),…, ie=NFCNZPIVOT(j)+NDIM(2,j)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Input. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Input. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector $\boldsymbol{b}$-$\boldsymbol{Ax}$ is sufficiently smaller compared with the absolute value of $\boldsymbol{b}$.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

ZA..... Input. The nonzero elements of an unsymmetric complex sparse matrix $\boldsymbol{A}$ are stored in ZA(1:NZ).

A double precision complex one-dimensional array ZA(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector). For a complex matrix , a real array CC in this Figure is replaced with a complex array.

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric complex sparse matrix ***A***.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array ZA.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array ZA in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

IW2..... Work area.

Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

The data derived from calling DM_VSCLU of LU decomposition of an unsymmetric complex sparse matrix is transferred in this work area. The contents must not be changed among calls.

ICON..... Output. Condition code.

(See Table DM_VSCLUX-1.)

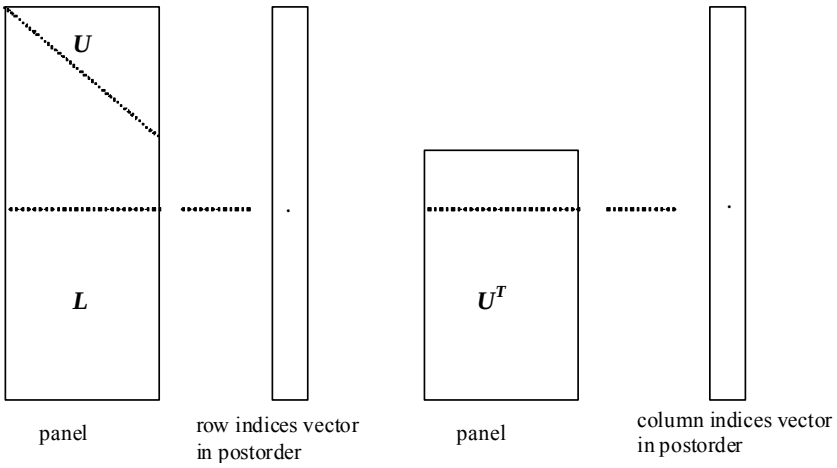


Figure DM_VSCLUX-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of ZPANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular matrix $\boldsymbol{L}$ of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix $\boldsymbol{U}$ of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,DIM(2, j),

t = 1,...,DIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (DIM(3, j)-DIM(2,j))×DIM(2, j) from the u-th element of ZPANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length DIM(3,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, j)-DIM(2, j))×DIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U^T}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix $\boldsymbol{A}$ is permuted in post ordering.

Table DM_VSCLUX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	Processing is discontinued.
20500	The norm of residual vector for the solution vector is greater than that of $\boldsymbol{b}$ multiplied by EPSR, which is the right term constant vector in $\boldsymbol{Ax}$=$\boldsymbol{b}$. The coefficient matrix $\boldsymbol{A}$ may be close to a singular matrix.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ITERMAX<1 when IREFINE=1.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index k stored in NROW(j) is k < 1 or k > n.	

Code	Meaning	Processing
30300	The number of row indices belong to i-th column is NFCNZ(i+1)-NFCNZ(i) > n.	Processing is discontinued.

(3) Comments on use

a. Notes

1) The results of LU decomposition obtained by DM_VSCLU is used. See note 5) (3), "Comments on use." of DM_VSCLU and example in (3), "Comments on use." of DM_VSCLUX.

2) When the element p_{ij}=1 of the permutation matrix $\boldsymbol{P}$, set NPERM(i)=j. The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

3) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the i-th node in post order is corresponding. It means j-th position when j = NPOSTO(i).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 2) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient a=(a_1,a_2,a_3).
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix $\boldsymbol{A}$ built in this way is stored into a complex sparse matrix and is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
     $      N = KX*KY*KZ)
      PARAMETER (NBORDER=N+1,NOFFDIAG=6)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
```

```
      INTEGER*4 WL,ZWL
      PARAMETER (NALL=NDIAG*N,
C
     $   ZWL =2*NALL,
     $   WL  =4*NALL+6*N,
     $   IW1L=2*NALL+2*(N+1)+16*N,
     $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      COMPLEX*16 ZA(K*NDIAG),ZWC(K*NDIAG),
     $           ZW(ZWL),ZONE
      PARAMETER(ZONE=(1.0D0,0.0D0))
      DIMENSION NROW(K*NDIAG),NFCNZ(N+1),
     $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
     $          IWC(2,K*NDIAG)
      DIMENSION NPERM(N),W(WL),
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
      COMPLEX*16, DIMENSION(:), ALLOCATABLE ::
     $        ZPANELFACTORL,ZPANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      COMPLEX*8 ZDUMMYFL,ZDUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      COMPLEX*16 ZB(N),ZSOLEX(N)
      REAL*8 THEPSZ,EPSZ,SPEPSZ,
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      ZSOLEX(I)= ZONE
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',ZSOLEX(1),' X(N) = ',ZSOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
```

```
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      ZA(NUMNZ)=DCMPLX(DIAG2(J,I),0.0D0)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCCC(ZA,NZ,NROW,NFCNZ,N,ZSOLEX,
     $             ZB,ZWC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
      NSIZEINDEXU=1
```

```
      EPSZ=1.0D-16
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSCLU(ZA,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZDUMMYFL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZDUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NDUMMYIU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              ZW,W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
      ALLOCATE( ZPANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( ZPANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSCLU(ZA,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZPANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZPANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
```

```
     $              NPIVOTP,NPIVOTQ,
     $              ZW,W,IW1,IW2,ICON)
C
      CALL DM_VSCLUX(N,
     $              IORDERING,
     $              NPERM,
     $              ZB,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,ZPANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,ZPANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              IPLEDSM,MZ,
     $              SCLROW,SCLCOL,
     $              NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              ZA,NZ,NROW,NFCNZ,
     $              IW2,ICON)
C
      ERR = ERRNRM(ZSOLEX,ZB,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',ZB(1),' X(N) = ',ZB(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF
C
      DEALLOCATE( ZPANELFACTORL,ZPANELFACTORU,
     $            NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &            ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
```

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

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J
```

```
C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END
C =======================================
* SOLUTE ERROR
* | Z1 - Z2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(Z1,Z2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      COMPLEX*16 Z1(*),Z2(*),SS
C
      S = 0D0
      DO 100 I = 1,LEN
        SS  = Z1(I) - Z2(I)
        S = S + DREAL(SS * DCONJG(SS))
 100  CONTINUE
C
```

```
      ERRNRM = SQRT( S )
      RETURN
      END
```

DM_VSCS

A system of linear equations with unsymmetric complex sparse matrices (LU decomposition method)
CALL DM_VSCS(ZA, NZ, NROW, NFCNZ, N, IPLEDSM, MZ, ISCLITERMAX, IORDERING, NPERM, ISW, NROWSYM, NFCNZSYM, ZB, NASSIGN, NSUPNUM, NFCNZFACTORL, ZPANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, ZPANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, SCLROW, SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, NFCNZPIVOT, NPIVOTP, NPIVOTQ, IREFINE, EPSR, ITERMAX, ITER, ZW, W, IW1, IW2, ICON)

(1) Function

The large entries of an $n \times n$ unsymmetric complex sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. Subsequently this subroutine solves a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ in use of LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for the permutation of elements, scaling and Pivot.

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

The unsymmetric complex sparse matrix is transformed as below.

$$\boldsymbol{A_1}=\boldsymbol{D_r}\boldsymbol{A}\boldsymbol{P_c}\boldsymbol{D_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2}=\boldsymbol{QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate. If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.
The solution is computed using LU decomposition.
It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

ZA..... Input. The nonzero elements of an unsymmetric complex sparse matrix ***A*** are stored in ZA(1:NZ).

A double precision complex one-dimensional array ZA(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector). For a complex matrix, a real array CC in this Figure is replaced with a complex array.

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric complex sparse matrix ***A***.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array ZA.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array ZA in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IPLEDSM..... Input. Control information whether to permute the large entries to the diagonal of a matrix ***A***.
When IPLEDSM=1 is specified, a matrix ***A*** is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Output. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(*i*)=*j* indicates that the *j*-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to *i*-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix ***A***.

When ISCLITERMAX ≤ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX ≥ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix ***P*** of ordering or to decompose the matrix ***A***.

When IORDERING=10 is specified, calling this routine with ISW=1 produces the informations which is needed to generate an ordering regarding $\boldsymbol{A_1}$ and they are set in NROWSYM and NFCNZSYM.

When IORDERING 11 is specified, it is indicated that after an ordering is set in NPERM, the computation is resumed.
Using the informations obtained in NROWSYM and NFCNZSYM after calling this routines with ISW=1 and IORDERING=10, an ordering is determined. After specifying this ordering in NPERM, this routine is called again with ISW=1and

IORDERING=11 and the computation is resumed.
LU decomposition of the matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is continued.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

Output. IORDERING is set to 11 after this routine is called with IORDERING=10 and ISW=1. Therefore after an ordering is set in NPERM the computation is resumed in the subsequent call without IORDERING=11 being specified explicitly.

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

NPERM..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
After symmetrization of a matrix and symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.
Call with IORDERING=10 produces the informations needed for seeking an ordering in NROWSYM and NFCNZSYN. Using these informations an ordering for $\boldsymbol{SYM}$ is determined. After an ordering is set in NPERM, calling this routine with IORDERING=11 and also ISW=1 again resumes the computation.
When IORDERING is neither 10 nor 11, no ordering is specified.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of ZPANELFACTORL or ZPANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed.
Before calling again with ISW=2, the ZPANELFACTORL or ZPANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEXL or NSIZEINDEXU at the precedent call and specified in corresponding arguments.
Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

3) When ISW=3 is specified.
The subsequent call with ISW=3 solves another system of equations of which the coefficient matrix is as same as previous call but the right-hand side vector $\boldsymbol{b}$ is changed. In this case, the information obtained by the previous LU decomposition can be reused.
Besides, except ISW as control information and B for storing the new right-hand side $\boldsymbol{b}$, the values in the other arguments must not be changed between the previous and following calls.

NROWSYM..... Output. When it is called with IORDERING=10, the row indices of nonzero pattern of the lower triangular part of $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1}^{\boldsymbol{T}}$ in the compressed column storage method are generated.

One-dimensional array NROWSYM(NZ+N).

NFCNZSYM..... Output. When it is called with IORDERING=10, the position of the first row index of each column stored in array NROWSYM in the compressed column storage method which stores the nonzero pattern of the lower part of a matrix $\boldsymbol{SYM}$ column by column.

NFCNZSYM(N+1)=NSYMZ+1 where NSYMZ is the total nonzero elements in the lower triangular part.

One-dimensional array NFCNZ(N+1).

ZB..... Input. The right-hand side constant vector ***b*** of a system of linear equations ***Ax*** = ***b***.

Output. Solution vector ***x***.

A double precision complex one-dimensional array ZB(N).

NASSIGN..... Output. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in ZPANELFACTORL and ZPANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSCS-1.
One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of an unsymmetric complex sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

ZPANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array ZPANELFACTORL. 8-byte integer.

Output. The necessary size for the array ZPANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric complex sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into ZPANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

ZPANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

A double precision complex one-dimensional array ZPANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array ZPANELFACTORU. 8-byte integer.

Output. The necessary size for the array ZPANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSCS-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of $\boldsymbol{A}$ the i-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of a complex number with the absolute value of

SPEPSZ.
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.
d) IREFINE=1 must be specified for the iterative refinement of the solution.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified. The following conditions must hold.
1.0D-8≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

NFCNZPIVOT.... Output. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the *i*-th supernode is assigned is known by *j*=NASSIGN(*i*). The position of the first element of that section is stored in NFCNZPIVOT(*j*). The information of exchange rows and columns within the *i*-th supernode is stored in the elements of is=NFCNZPIVOT(*j*),…, ie=NFCNZPIVOT(*j*)+NDIM(2,*j*)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Output. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Output. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

When ISTATIC=1 is specified, IREFINE=1 must be specified.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector ***b-Ax*** is sufficiently smaller compared with the absolute value of ***b***.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

ZW.....Work area.

Output/Input.
A double precision complex one-dimensional array of size 2*NZ.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

W..... Work area.

Output/Input.
One-dimensional array of size 4*NZ+6*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW1..... Work area.

Output/Input.
One-dimensional array of size 2*NZ+2*(N+1)+16*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW2..... Work area.

Output/Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

When this subroutine is called repeatedly with ISW=1, 2, 3 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSCS-1.)

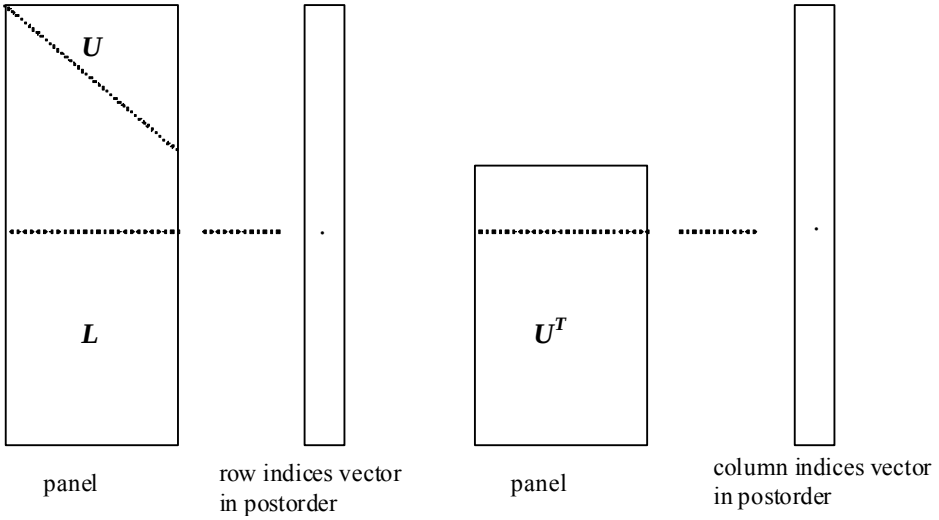


Figure DM_VSCS-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of ZPANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, *j*)×DIM(2, *j*).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(*s*, *t*), $s \geq t$, s = 1,...,DIM(1, *j*),

t = 1,...,DIM(2, *j*).

The block diagonal portion except diagonals of the unit upper triangular matrix ***U*** of decomposed results is stored in

panel(*s*, *t*), $s < t$, s = 1,...,DIM(2, *j*),

t = 1,...,DIM(2, *j*).

u = NFCNZFACTORU(*j*) → The *j*-th panel occupies the area with a length (DIM(3, *j*)-DIM(2,*j*))×DIM(2, *j*) from the *u*-th element of ZPANELFACTORU.

v = NFCNZINDEXU(*j*) → The column indices vector of the *j*-th panel occupies the area with a length DIM(3,*j*) from the *v*-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, *j*)-DIM(2, *j*))×DIM(2, *j*).

The transposed unit upper triangular matrix $\boldsymbol{U^T}$ except its block diagonal portion of decomposed results is stored in

panel(*x*, *y*), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, *j*).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix ***A*** is permuted in post ordering.

Table DM_VSCS-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
20100	When IPLEDSM is specified, maximum matching with the length N is sought in order to permute large entries to the diagonal but can not be found. The coefficient matrix ***A*** may be singular.	
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix ***A***. The coefficient matrix ***A*** may be singular.	
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	
20500	The norm of residual vector for the solution vector is greater than that of ***b*** multiplied by EPSR, which is the right term constant vector in $\boldsymbol{Ax}=\boldsymbol{b}$. The coefficient matrix ***A*** may be close to a singular matrix.	

Code	Meaning	Processing
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ISW < 1, or ISW > 3, ITERMAX<1 when IREFINE=1.	Processing is discontinued.
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index *k* stored in NROW(*j*) is *k* < 1 or *k* > *n*.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n*.	
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or IREFINE≠1 or SPEPSZ>THEPSZ or SPEPSZ>1.0D-8	
31000	The value of NSIZEFACTORL is not enough as the size of ZPANELFACTORL, or the value of NSIZEINDEXL is not enough as the size of NPANELINDEXL, or the value of NSIZEFACTORU is not enough as the size of ZPANELFACTORU, or the value of NSIZEINDEXU is not enough as the size of NPANELINDEXU.	Reallocate the ZPANELFACTORL or NPANELINDEXL or ZPANELFACTORU or NPANELINDEXU with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEXL or NSIZEFACTORU or NSIZEINDEXU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is 16 × *u*.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for Pivot. When the computation is to be continued even if the absolute value of diagonal element is small, assign the

minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with SPEPSZ. It is required to specify to do iterative refinement.

3) The necessary sizes for the array ZPANELFACTORL, NPANELINDEXL, ZPANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) Instead of this routine, a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be solved by calling both DM_VSCLU to perform LU decomposition of an unsymmetric complex sparse matrix $\boldsymbol{A}$ and DM_VSCLUX to solve the linear equation in use of decomposed results.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix $\boldsymbol{A}$ built in this way is stored into a complex sparse matrix and is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
     $      N = KX*KY*KZ)
      PARAMETER (NBORDER=N+1,NOFFDIAG=6)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      INTEGER*4 WL,ZWL
```

```
      PARAMETER (NALL=NDIAG*N,
C
     $   ZWL =2*NALL,
     $   WL  =4*NALL+6*N,
     $   IW1L=2*NALL+2*(N+1)+16*N,
     $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      COMPLEX*16 ZA(K*NDIAG),ZWC(K*NDIAG),
     $           ZW(ZWL),ZONE
      PARAMETER(ZONE=(1.0D0,0.0D0))
      DIMENSION NROW(K*NDIAG),NFCNZ(N+1),
     $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
     $          IWC(2,K*NDIAG)
      DIMENSION NPERM(N),W(WL),
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
      COMPLEX*16, DIMENSION(:), ALLOCATABLE ::
     $        ZPANELFACTORL,ZPANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      COMPLEX*16 ZDUMMYFL,ZDUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      COMPLEX*16 ZB(N),ZSOLEX(N)
      REAL*8 EPSZ,THEPSZ,SPEPSZ,
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC COMPLEX MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      ZSOLEX(I)=ZONE
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',ZSOLEX(1),' X(N) = ',ZSOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
```

```
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      ZA(NUMNZ)=DCMPLX(DIAG2(J,I),0.0D0)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCCC(ZA,NZ,NROW,NFCNZ,N,ZSOLEX,
     $             ZB,ZWC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      EPSZ=1.0D-16
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
```

```
      NSIZEINDEXU=1
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSCS(ZA,NZ,NROW,NFCNZ,N,
     $             IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $             NPERM,ISW,
     $             NROWSYM,NFCNZSYM,
     $             ZB,
     $             NASSIGN,
     $             NSUPNUM,
     $             NFCNZFACTORL,ZDUMMYFL,
     $             NSIZEFACTORL,
     $             NFCNZINDEXL,
     $             NDUMMYIL,NSIZEINDEXL,
     $             NDIM,
     $             NFCNZFACTORU,ZDUMMYFU,
     $             NSIZEFACTORU,
     $             NFCNZINDEXU,
     $             NDUMMYIU,NSIZEINDEXU,
     $             NPOSTO,
     $             SCLROW,SCLCOL,
     $             EPSZ,THEPSZ,
     $             IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $             NPIVOTP,NPIVOTQ,
     $             IREFINE,EPSR,ITERMAX,ITER,
     $             ZW,W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
      ALLOCATE( ZPANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( ZPANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSCS(ZA,NZ,NROW,NFCNZ,N,
     $             IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $             NPERM,ISW,
     $             NROWSYM,NFCNZSYM,
     $             ZB,
     $             NASSIGN,
     $             NSUPNUM,
     $             NFCNZFACTORL,ZPANELFACTORL,
     $             NSIZEFACTORL,
     $             NFCNZINDEXL,
     $             NPANELINDEXL,NSIZEINDEXL,
     $             NDIM,
     $             NFCNZFACTORU,ZPANELFACTORU,
     $             NSIZEFACTORU,
     $             NFCNZINDEXU,
     $             NPANELINDEXU,NSIZEINDEXU,
     $             NPOSTO,
```

```
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              ZW,W,IW1,IW2,ICON)
C
      ERR = ERRNRM(ZSOLEX,ZB,N)
C
      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',ZB(1),' X(N) = ',ZB(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *
C
      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF
C
      DEALLOCATE( ZPANELFACTORL,ZPANELFACTORU,
     $            NPANELINDEXL,
     $            NPANELINDEXU )
C
      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &           ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
```

```
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
* SOLUTE ERROR
* | Z1 - Z2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(Z1,Z2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      COMPLEX*16 Z1(*),Z2(*),SS
C
      S = 0D0
      DO 100 I = 1,LEN
        SS  = Z1(I) - Z2(I)
        S = S + DREAL(SS * DCONJG(SS))
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The permutation which moves large entries to the diagonal is performed. And the permutated matrix is scaled in order to equilibrate both rows and columns norms.
The absolute value of a complex number is approximated as a sum of the absolute value of both its real part ant its imaginary part for the permutation of elements, scaling and Pivot. Subsequently the LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.

Refer to references in Appendix A, “References.” in detail.
Refer to [23], [57] on the method how the elements of large absolute value are permuted to diagonal, to [13] on the application algorithms of matching, to [17] on Fibonacci Heaps, to [19], [2], [22], [48], [68] on the base of the LU decomposition of unsymmetric complex parse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VSEVPH

Eigenvalues and eigenvectors of real symmetric matrices (tridiagonalization, multisection method, and inverse iteration)
CALL DM_VSEVPH (A, K, N, NF, NL, IVEC, ETOL, CTOL, NEV, E, MAXNE, M, EV, 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.

$$\boldsymbol{Ax} = \lambda \boldsymbol{x} \tag{1.1}$$

where, ***A*** is an $n \times n$ real symmetric matrix.

(2) Parameters

A ..... Input. The lower triangular part $\{a_{ij} \mid i \geq j\}$ of real symmetric matrix ***A*** is stored in the lower triangular part $\{A(i,j) \mid i \geq j\}$ of A(1:N,1:N).

After calculation, the value of A is not assured.

Two-dimensional double-precision real 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.

When the value of IVEC is 1, the eigenvalues and corresponding eigenvectors are calculated.

When the value of IVEC is not 1, only the eigenvalues are calculated.

ETOL ..... Input. Criterion value for checking whether the eigenvalues are numerically different from each other or are multiple. When ETOL is less than 3.0D-16, this value is used as the standard value.

CTOL ..... Input. Criterion value for checking whether the adjacent eigenvalues can be considered to be approximately equal to each other. This check uses formula (3.1). This value is used to assure the linear independence of the eigenvector corresponding to the eigenvalue belonging to approximately multiple eigenvalues (clusters).

The value of CTOL should be generally 5.0D-12. For a very large cluster, a large CTOL value is required.

1.0D-6 ≥ CTOL ≥ ETOL

When condition CTOL > 1.0D-6 occurs, CTOL is set to 1.0D-6.

When condition CTOL < ETOL occurs, CTOL = 10 × ETOL is set as the standard value.

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

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

One-dimensional array NEV(5).

The detail information is as follows:

NEV(1) indicates the number of different eigenvalues calculated.

NEV(2) indicates the number of approximately multiple, different eigenvalues (clusters) calculated.

NEV(3) indicates the total number of eigenvalues (including multiple eigenvalues) calculated.

NEV(4) indicates the number representing the first of the eigenvalues calculated.

NEV(5) indicates the number representing the last of the eigenvalues calculated.

E ..... Output. Eigenvalues are stored in E.

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

One-dimensional array E(MAXNE).

MAXNE .... Input. Maximum number of eigenvalues that can be calculated.

When it can be considered that there are two or more eigenvalues with multiplicity m, MAXNE must be set to a larger value than NL−NF+1 + 2×m that is bounded by n. Size of the dimension of array E.

When condition NEV(3) > MAXNE occurs, the eigenvectors cannot be calculated.

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

M ..... Output. Information about multiplicity of eigenvalues calculated.

M(i,l) indicates the multiplicity of the i-th eigenvalue λ_i. M(i,2) indicates the multiplicity of the i-th cluster when the adjacent eigenvalues are regarded as clusters.

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

Two-dimensional array M(MAXNE,2).

EV ..... Output. When IVEC = 1, the eigenvectors corresponding to the eigenvalues are stored in EV.

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

Two-dimensional array EV(K,MAXNE).

ICON ..... Output. Condition code.

See Table DM_VSEVPH-1.

Table DM_VSEVPH-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. A suitable value for MAXNE to allow calculation to proceed is returned in NEV(3). (See 2) in a, "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.

(3) Comments on use

a. Notes

1) This routine calculates eigenvalues independently from each other by dividing them into nonoverlapping, sequenced sets (parallel processing).

When ε = ETOL, the following condition is satisfied for consecutive eigenvalues λ_j ($j = s - 1, s, ..., s + k$ (k ≥ 0)):

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

If formula (3.1) is satisfied for i when $i = s, s + 1, ..., s + k$ but not satisfied when $i = s-1$ and $i = s + k + 1$, it is assumed that the eigenvalues λ_j ($j = s - 1, s, ..., s + k$) are numerically multiple.

The standard value of ETOL is 3.0D-16 (about the unit round off). With this value, the eigenvalues are refined up to the maximum machine precision.

If formula (3.1) is not satisfied when ε = ETOL, it can be considered that λ_{i-1} and λ_i are distinct eigenvalues.

When ε = ETOL, assume that consecutive eigenvalues λ_m ($m = t - 1, t, ..., t + k$ (k ≥ 0)) are different eigenvalues. Also, when ε = CTOL, assume that formula (3.1) is satisfied for i when $i = t, t + 1, ..., t + k$ but not satisfied when $i = t - 1$ and $i = t + k + 1$. In this case, it is assumed that their different eigenvalues λ_m ($m = t - 1, t, ..., t + k$) are approximately multiple (i.e. form a cluster). In this case, independent starting vectors are generated for inverse iteration, and eigenvectors corresponding to λ_m ($m = t - 1, t, \ldots, t + k$) are reorthogonalized.

2) The maximum number of eigenvalues that can be calculated is specified in MAXNE. When the value of CTOL is increased, the cluster size also increases. Therefore, the total number of eigenvalues calculated might exceed the value of MAXNE. In this case, decrease the value of CTOL or increase the value of MAXNE.

If the total number of eigenvalues calculated exceeds the value of MAXNE, ICON = 20000 is returned. In this case, the eigenvectors cannot be calculated even if eigenvector calculation is specified. Eigenvalues are calculated, but are not stored repeatedly according to the multiplicity.

The calculated different eigenvalues are stored in E(1:NEV(1)). The information about the multiplicity of the corresponding eigenvalues is stored in M(1:NEV(1),1).

When all the eigenvalues are different from each other and there are no approximately multiple eigenvalues, MAXNE can be set to NT (NT=NL-NF+1). However, when there are multiple eigenvalues and the multiplicity can be assumed to be m, then MAXNE must be set to at least NT + 2 × m.

If the total number of eigenvalues to be calculated exceeds the value of MAXNE, the value required to continue the calculation is returned to NEV(3). The calculation can be continued by allocating the area by using this returned value and by calling the routine again.

b. Example

This example calculates the specified eigenvalues and eigenvectors of a real symmetric matrix whose eigenvalues and eigenvectors are known.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N=2000,K=N)
      PARAMETER (NE=N,MAX_NEV=NE)
      DIMENSION A(K,N),B(K,N),C(K,N),D(K,N),AC(K,N)
      DIMENSION NEV(5),MULT(MAX_NEV,2)
      DIMENSION EVAL(MAX_NEV),EVEC(K,MAX_NEV)
CC
      PAI=4.0D0*DATAN(1.0D0)
      COEF=DSQRT(2.0D0/(N+1))
      DO J=1,N
      DO I=1,N
      D(I,J)=COEF*DSIN(PAI/(N+1)*I*J)
      ENDDO
      ENDDO
CC
      DO J=1,N
      DO I=1,N
      IF(I.EQ.J)THEN
      C(I,J)=I
      ELSE
      C(I,J)=0.0D0
      ENDIF
      ENDDO
      ENDDO
CC
CC    d x c -> b
CC
```

```
      CALL DM_VMGGM(D,K,C,K,B,K,N,N,N,ICON)
CC
CC    b x d -> a
CC
      CALL DM_VMGGM(B,K,D,K,A,K,N,N,N,ICON)
CC
      DO I=1,N
      DO J=I,N
      AC(J,I)=A(J,I)
      ENDDO
      ENDDO
      NF=1
      NL=NE
      IVEC=1
      EVAL_TOL=1.0D-15
      CLUS_TOL=1.0D-10
      CALL DM_VSEVPH( AC,K,N,NF,NL,IVEC,EVAL_TOL,CLUS_TOL,NEV,
     &                EVAL,MAX_NEV,MULT,EVEC,ICON )
      DO I=1,NE,N/20
      PRINT*,'EIGEN VALUE IN EVAL(',I,') = ',EVAL(I)
      ENDDO
C
      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 a parallel version of the Householder reduction to tridiagonal form. (See [30] in Appendix A, “References.”)

The eigenvalue problem of a tridiagonal matrix is calculated using multisectioning to find the eigenvalues and inverse iteration for the eigenvectors. For details, see “DM_VTDEVC” and see [61] in Appendix A, “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.

DM_VSLDL

LDL^T decomposition of a symmetric positive definite matrix (blocked modified Cholesky decomposition method)
CALL DM_VSLDL(A,K,N,EPSZ,ICON)

(1) Function

This subroutine executes LDL^T decomposition for an $n \times n$ positive definite matrix $\boldsymbol{A}$ using the blocked modified Cholesky decomposition method of outer product type, so that

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$$

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

(2) Parameters

A ..... Input. The coefficient matrix $\boldsymbol{A}$

Output. Matrices $\boldsymbol{L}$ and $\boldsymbol{D}^{-1}$.

For input, the lower triangular part of $\boldsymbol{A}$ $\{a_{ij} \mid i \geq j\}$ is stored in the lower triangular part $\{A(i, j) \mid i \geq j\}$ of A(1:N,1:N).

For output, the contents of A(i,j) is

l_{ij} ($i > j$),

A reciprocal of d_{ii} ($i = j$).

altered ($i < j$),

(See Figure DM_VSLDL-1.)

This is a double precision real two-dimensional array A(K,N).

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

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

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed.

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

ICON ..... Output. Condition code.

(See Table DM_VSLDL-1.)

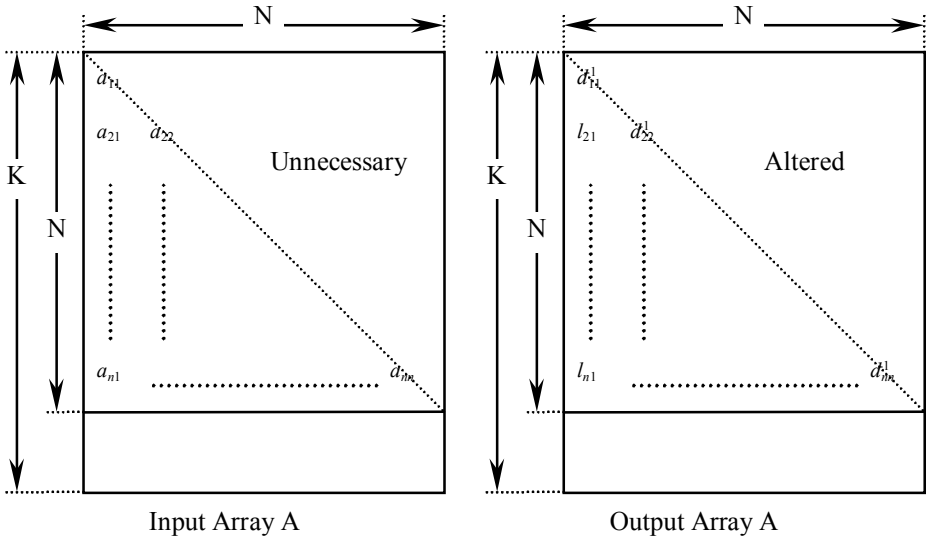


Figure DM_VSLDL-1 Storing data by Cholesky decomposition

The diagonal elements and lower triangular part aij of the positive definite matrix for which LDL^T decomposition is performed is stored in array A(i, j), $i = j,..., n, j = 1,..., n$.

After LDL^T decomposition, the matrix $\boldsymbol{D}^{-1}$ is stored in diagonal elements and $\boldsymbol{L}$ (except the diagonal elements) are stored in the lower triangular part respectively.

Table DM_VSLDL-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	The pivot became negative. The coefficient matrix is not positive definite.	Processing is continued.
20000	The pivot became relatively zero. The coefficient matrix may be singular.	Processing is discontinued.
30000	N < 1, EPSZ < 0, K < N	

(3) Comments on use

a. Notes

1) If a value is set for EPSZ, the value has the following meaning: if the absolute value of the selected pivot is less than the value for EPSZ during LDL^T decomposition by the modified Cholesky decomposition, the value of the pivot is assumed to be relatively zero and processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$.
When the computation is to be continued even if the value of the pivot becomes small, assign the minimum value to EPSZ. In this case, however the result is not assured.

2) If the pivotal value becomes negative during decomposition, the coefficient matrix is no longer positive definite. Processing continues with ICON = 10000. However, the accuracy of the result may not be maintained because no pivoting is performed.

3) After the calculation has been completed, the determinant of the coefficient matrix is computed by multiplying all the *n* diagonal elements of the array A and taking the reciprocal of the result.

b. Example

LDL^T decomposition is executed for a 4000×4000 matrix.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (K=4000,N=4000)
      REAL*8    A(K,N)
C
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A)
      DO J=1,N
      DO I=J,N
      A(I,J)=MIN(I,J)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO

      CALL DM_VSLDL(A,K,N,1.D-13,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 10
C
      S=1.D0
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(A)
!$OMP+         REDUCTION(*:S)
      DO I=1,N
      S=S*A(I,I)
      ENDDO
!$OMP END PARALLEL DO

      DET=S
      DET=1.D0/DET
      WRITE(6,620) DET
      WRITE(6,640)
      DO J=1,5
      WRITE(6,600) J,(A(I,J),I=J,5)
      ENDDO
      GO TO 10
  600 FORMAT(1H,I5/(10X,3D23.16))
  610 FORMAT(/10X,5HICON=,I5)
  620 FORMAT(//10X
     *,22HDETERMINANT OF MATRIX=,D23.16)
  640 FORMAT(/10X,17HDECOMPOSED MATRIX)
  10  stop
      END
```

(4) Method

See [30], [54], and [70] in Appendix A, "References" for details of blocked modified Cholesky decomposition method of the outer product type.

DM_VSRLU

LU decomposition of an unsymmetric real sparse matrix
CALL DM_VSRLU(A, NZ, NROW, NFCNZ, N, IPLEDSM, MZ, ISCLITERMAX, IORDERING, NPERM, ISW, NROWSYM, NFCNZSYM, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, SCLROW,SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, NFCNZPIVOT, NPIVOTP, NPIVOTQ, W, IW1, IW2, ICON)

(1) Function

The large entries of an $n \times n$ unsymmetric real sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. And LU decomposition is performed, in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.

The unsymmetric real sparse matrix is transformed as below.

$$\boldsymbol{A_1=D_rAP_cD_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2=QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM=A_1+A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate.
If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.

(2) Parameter

A..... Input. The nonzero elements of an unsymmetric real sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric real sparse matrix $\boldsymbol{A}$.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IPLEDSM..... Input. Control information whether to permute the large entries to the diagonal of a matrix $\boldsymbol{A}$.
When IPLEDSM=1 is specified, a matrix $\boldsymbol{A}$ is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Output. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(i)=j indicates that the j-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to i-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix $\boldsymbol{A}$.

When ISCLITERMAX ≤ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX ≥ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

When IORDERING=10 is specified, calling this routine with ISW=1 produces the informations which is needed to generate an ordering regarding $\boldsymbol{A_1}$ and they are set in NROWSYM and NFCNZSYM.

When IORDERING 11 is specified, it is indicated that after an ordering is set in NPERM, the computation is resumed.
Using the informations obtained in NROWSYM and NFCNZSYM after calling this routines with ISW=1 and IORDERING=10, an ordering is determined. After specifying this ordering in NPERM, this routine is called again with ISW=1and IORDERING=11 and the computation is resumed.
LU decomposition of the matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is continued.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

Output. IORDERING is set to 11 after this routine is called with IORDERING=10 and ISW=1. Therefore after an ordering is set in NPERM the

computation is resumed in the subsequent call without IORDERING=11 being specified explicitly.

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

NPERM..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
After symmetrization of a matrix and symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.
Call with IORDERING=10 produces the informations needed for seeking an ordering in NROWSYM and NFCNZSYN. Using these informations an ordering for $\boldsymbol{SYM}$ is determined. After an ordering is set in NPERM, calling this routine with IORDERING=11 and also ISW=1 again resumes the computation.
When IORDERING is neither 10 nor 11, no ordering is specified.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed. Before calling again with ISW=2, the PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEXL or NSIZEINDEXU at the precedent call and specified in corresponding arguments.
Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

NROWSYM..... Output. When it is called with IORDERING=10, the row indices of nonzero pattern of the lower triangular part of $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1^T}$ in the compressed column storage method are generated.

One-dimensional array NROWSYM(NZ+N).

NFCNZSYM..... Output. When it is called with IORDERING=10, the position of the first row index of each column stored in array NROWSYM in the compressed column storage method which stores the nonzero pattern of the lower part of a matrix $\boldsymbol{SYM}$ column by column.

NFCNZSYM(N+1)=NSYMZ+1 where NSYMZ is the total nonzero elements in the lower triangular part.

One-dimensional array NFCNZ(N+1).

NASSIGN..... Output. $\boldsymbol{L}$ and $\boldsymbol{U}$ belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the i-th supernode is stored into the j-th block of a subarray, where j=NASSIN(i).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSRLU-1.
One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of an unsymmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,DIM(2,*i*).

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

Output. The necessary size for the array PANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into

NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

PANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

Output. The necessary size for the array PANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of DSIGN(SPEPSZ,PIVOT).
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified.
The following conditions must hold.
THEPSZ≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

NFCNZPIVOT.... Output. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the *i*-th supernode is assigned is known by *j*=NASSIGN(*i*). The position of the first element of that

section is stored in NFCNZPIVOT(*j*). The information of exchange rows and columns within the *i*-th supernode is stored in the elements of is=NFCNZPIVOT(*j*),…, ie=NFCNZPIVOT(*j*)+NDIM(2,*j*)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Output. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Output. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

W..... Work area.

Output/Input.
One-dimensional array of size 4*NZ+6*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW1..... Work area.

Output/Input.
One-dimensional array of size 2*NZ+2*(N+1)+16*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW2..... Work area.

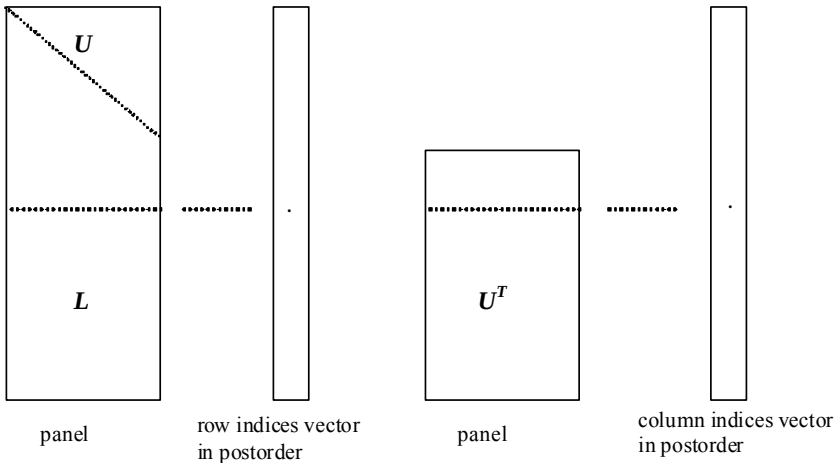
Output/Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSRLU-1.)

Figure DM_VSRLU-1 Conceptual scheme for storing decomposed results



j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular matrix $\boldsymbol{L}$ of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix $\boldsymbol{U}$ of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,DIM(2, j),

t = 1,...,DIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (DIM(3, j)-DIM(2,j))×DIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length DIM(3,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, j)-DIM(2, j))×DIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U^T}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix $\boldsymbol{A}$ is permuted in post ordering.

Table DM_VSRLU-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	When ISTATIC=1 is specified, Static pivot which replaces the pivot candidate with too small value with SPEPSZ is made.	–
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
20100	When IPLEDSM is specified, maximum matching with the length N is sought in order to permute large entries to the diagonal but can not be found. The coefficient matrix ***A*** may be singular.	
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix ***A***. The coefficient matrix ***A*** may be singular.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ISW < 1, or ISW > 2	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index *k* stored in NROW(*j*) is *k* < 1 or *k* >*n*.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n*.	
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or SPEPSZ>THEPSZ	
31000	The value of NSIZEFACTORL is not enough as the size of PANELFACTORL, or the value of NSIZEINDEXL is not enough as the size of NPANELINDEXL, or the value of NSIZEFACTORU is not enough as the size of PANELFACTORU, or the value of NSIZEINDEXU is not enough as the size of NPANELINDEXU.	Reallocate the PANELFACTORL or NPANELINDEXL or PANELFACTORU or NPANELINDEXU with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEXL or NSIZEFACTORU or NSIZEINDEXU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the absolute value of diagonal element is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with SPEPSZ.

3) The necessary sizes for the array PANELFACTORL, NPANELINDEXL, PANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix ***A*** into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) A system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be solved by calling DM_VSRLUX subsequently in use of the results of LU decomposition obtained by this routine. The following arguments used in this routine are specified.
See example in (3), "Comments on use.".

```
A, NZ, NROW, NFCNZ, N,
IPLEDSM, MZ, IORDERING, NPERM,
NASSIGN, NSUPNUM,
NFCNZFACTORL, PANELFACTORL,
NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL,
NSIZEINDEXL, NDIM,
NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU,
```

NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO,
SCLROW,SCLCOL,
NFCNZPIVOT,
NPIVOTP, NPIVOTQ, IW2

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix $\boldsymbol{A}$ built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
     $      N = KX*KY*KZ)
      PARAMETER (NBORDER=N+1,NOFFDIAG=6)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      INTEGER*4 WL
      PARAMETER (NALL=NDIAG*N,
C
     $   WL  =4*NALL+6*N,
     $   IW1L=2*NALL+2*(N+1)+16*N,
     $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
     $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
     $
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION NPERM(N),W(WL),
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION B(N),SOLEX(N)
      REAL*8 THEPSZ,EPSZ,SPEPSZ,
```

```
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      SOLEX(I)=DBLE(1)
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
```

```
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCC(A,NZ,NROW,NFCNZ,N,SOLEX,
     $             B,WC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0            !
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
      NSIZEINDEXU=1
      EPSZ=1.0D-16
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSRLU(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,DUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NDUMMYIU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
```

```
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSRLU(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              W,IW1,IW2,ICON)
C
      CALL DM_VSRLUX(N,
     $              IORDERING,
     $              NPERM,
     $              B,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              IPLEDSM,MZ,
     $              SCLROW,SCLCOL,
     $              NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              A,NZ,NROW,NFCNZ,
     $              IW2,ICON)
C
      ERR = ERRNRM(SOLEX,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',B(1),' X(N) = ',B(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
```

```
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF
C

      DEALLOCATE( PANELFACTORL,PANELFACTORU,
     $            NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =====================================
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &            ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
* SOLUTE ERROR
* | X1 - X2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The permutation which moves large entries to the diagonal is performed. And the permutated matrix is scaled in order to equilibrate both rows and columns norms. The LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.
Refer to references in Appendix A, “References.” in detail.
Refer to [23], [57] on the method how the elements of large absolute value are permuted to diagonal, to [13] on the application algorithms of matching, to [17] on Fibonacci Heaps, to [19], [2], [22], [48], [68] on the LU decomposition of unsymmetric real sparse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VSRLUX

A system of linear equations with LU-decomposed unsymmetric real sparse matrices
CALL DM_VSRLUX(N, IORDERING, NPERM, B, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, IPLEDSM, MZ, SCLROW,SCLCOL, NFCNZPIVOT, NPIVOTP, NPIVOTQ, IREFINE, EPSR, ITERMAX, ITER, A, NZ, NROW, NFCNZ, IW2, ICON)

(1) Function

An $n \times n$ unsymmetric real sparse matrix $\boldsymbol{A}$ of which LU decomposition is made as below is given. In this decomposition the large entries of an $n \times n$ unsymmetric real sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. Subsequently LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode is performed and results in the following form. This routine solves the following linear equation in use of these results of LU decomposition.

$$\boldsymbol{Ax=b}$$

A matrix $\boldsymbol{A}$ is decomposed into as below.

$$\boldsymbol{P_{rs}QPD_rAP_cD_cP^TQ^TP_{cs}=LU}$$

The unsymmetric real sparse matrix $\boldsymbol{A}$ is transformed as below.

$$\boldsymbol{A_1=D_rAP_cD_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2=QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
$\boldsymbol{P_{rs}}$ and $\boldsymbol{P_{cs}}$ represent row and column exchanges in orthogonal matrices respectively. The actual exchanges are restricted to the reduced part of the matrix belonging to each supernode.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM=A_1+A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

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

IORDERING..... Input. When IORDERING 11 is specified, it is indicated that LU decomposition is performed with an ordering specified in NPERM.
The matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is decomposed into LU decomposition.

Otherwise. No ordering is specified.

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

NPERM..... Input. When IORDEING=11 is specified, a vector presenting the permutation matrix ***P*** used is stored.

One-dimensional array NPERM(N).

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

B..... Input. The right-hand side constant vector ***b*** of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector ***x***.

One-dimensional array B(N).

NASSIGN..... Input. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Regarding the storage methods of decomposed matrices, refer to Figure DM_VSRLUX-1.
One-dimensional array NASSING(N).

NSUPNUM..... Input. The total number of supernodes.(≤ n)

NFCNZFACTORL..Input. The decomposed matrices ***L*** and ***U*** of an unsymmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

PANELFACTORL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), $s \geq t$, $s = 1,\ldots,\mathrm{DIM}(1, i)$, $t=1,\ldots,\mathrm{DIM}(2,i)$. The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, $t=1,\ldots,\mathrm{DIM}(2,i)$.

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

NFCNZINDEXL... Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NPANELINDEXL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

NDIM..... Input. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NFCNZFACTORU..Input. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

PANELFACTORU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is

assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NPANELINDEXU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRLUX-1.

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

NPOSTO..... Input. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

One-dimensional array NPOSTO(N).

(See note 3) in (3), "Comments on use.")

IPLEDSM..... Input. Information indicating whether for LU decomposition it is specified to permute the large entries to the diagonal of a matrix ***A***.
When IPLEDSM=1 is specified, a matrix ***A*** is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Input. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(*i*)=*j* indicates that the *j*-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to *i*-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

SCLROW..... Input. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

One-dimensional array SCLROW (N).

SCLCOL..... Input. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

One-dimensional array SCLCOL(N).

NFCNZPIVOT.... Input. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the *i*-th supernode is assigned is known by *j*=NASSIGN(*i*). The position of the first element of that section is stored in NFCNZPIVOT(*j*). The information of exchange rows and columns within the *i*-th supernode is stored in the elements of is=NFCNZPIVOT(*j*),…, ie=NFCNZPIVOT(*j*)+NDIM(2,*j*)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Input. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Input. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector $\boldsymbol{b}-\boldsymbol{Ax}$ is sufficiently smaller compared with the absolute value of $\boldsymbol{b}$.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

A..... Input. The nonzero elements of an unsymmetric real sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric real sparse matrix $\boldsymbol{A}$.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

IW2..... Work area.

Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

The data derived from calling DM_VSRLU of LU decomposition of an unsymmetric real sparse matrix is transferred in this work area. The contents must not be changed among calls.

ICON..... Output. Condition code.

(See Table DM_VSRLUX-1.)

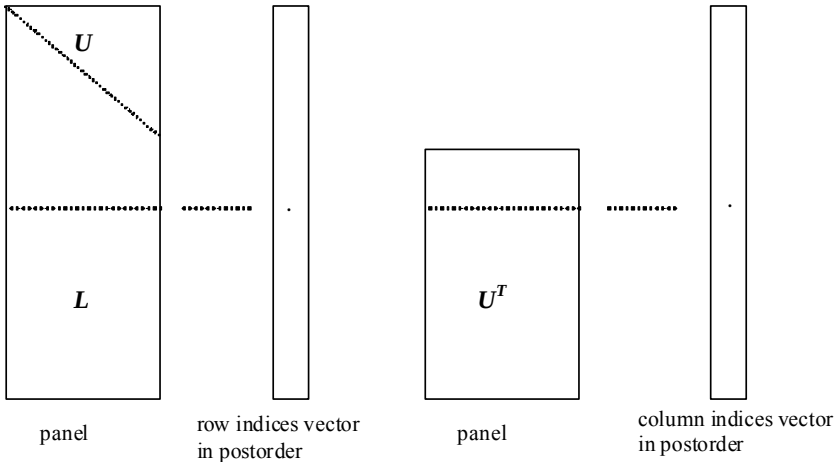


Figure DM_VSRLUX-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix ***U*** of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,DIM(2, j),

t = 1,...,DIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (DIM(3, j)-DIM(2,j))×DIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length DIM(3,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, j)-DIM(2, j))×DIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U}^{\boldsymbol{T}}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix ***A*** is permuted in post ordering.

Table DM_VSRLUX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	Processing is discontinued.
20500	The norm of residual vector for the solution vector is greater than that of ***b*** multiplied by EPSR, which is the right term constant vector in ***Ax***=***b***. The coefficient matrix ***A*** may be close to a singular matrix.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ITERMAX<1 when IREFINE=1.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index k stored in NROW(j) is k < 1 or k > n.	
30300	The number of row indices belong to i-th column is NFCNZ(i+1)-NFCNZ(i) > n.	

(3) Comments on use

a. Notes

1) The results of LU decomposition obtained by DM_VSRLU is used. See note 5) (3), "Comments on use." of DM_VSRLU and example in (3), "Comments on use." of DM_VSRLUX.

2) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

3) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as well as note 2) above, and corresponds to permute the matrix ***A*** into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix ***A*** built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C      **EXAMPLE**
       IMPLICIT  REAL*8 (A-H,O-Z)
       PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
      $      N = KX*KY*KZ)
       PARAMETER (NBORDER=N+1,NOFFDIAG=6)
       PARAMETER (K = N+1)
       PARAMETER (NDIAG = 7)
       INTEGER*4 WL
       PARAMETER (NALL=NDIAG*N,
C
      $   WL  =4*NALL+6*N,
      $   IW1L=2*NALL+2*(N+1)+16*N,
      $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
       DIMENSION NOFST(NDIAG)
       DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
       DIMENSION A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
      $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
      $
      $          WC(K*NDIAG),IWC(2,K*NDIAG)
       DIMENSION NPERM(N),W(WL),
```

```
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION B(N),SOLEX(N)
      REAL*8 THEPSZ,EPSZ,SPEPSZ,
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      SOLEX(I)=DBLE(1)
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
```

```
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCC(A,NZ,NROW,NFCNZ,N,SOLEX,
     $             B,WC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0            !
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
      NSIZEINDEXU=1
      EPSZ=1.0D-16
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSRLU(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
```

```
     $              NDUMMYIL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,DUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NDUMMYIU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSRLU(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              W,IW1,IW2,ICON)
C
      CALL DM_VSRLUX(N,
     $              IORDERING,
     $              NPERM,
     $              B,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
```

```
     $              NPOSTO,
     $              IPLEDSM,MZ,
     $              SCLROW,SCLCOL,
     $              NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              A,NZ,NROW,NFCNZ,
     $              IW2,ICON)
C
      ERR = ERRNRM(SOLEX,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',B(1),' X(N) = ',B(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF
C

      DEALLOCATE( PANELFACTORL,PANELFACTORU,
     $            NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =====================================
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &            ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
* SOLUTE ERROR
* | X1 - X2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

DM_VSRS

A system of linear equations with unsymmetric real sparse matrices (LU decomposition method)
CALL DM_VSRS(A, NZ, NROW, NFCNZ, N, IPLEDSM, MZ, ISCLITERMAX, IORDERING, NPERM, ISW, NROWSYM, NFCNZSYM, B, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEXL, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NSIZEINDEXU, NPOSTO, SCLROW, SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, NFCNZPIVOT, NPIVOTP, NPIVOTQ, IREFINE, EPSR, ITERMAX, ITER, W, IW1, IW2, ICON)

(1) Function

The large entries of an $n \times n$ unsymmetric real sparse matrix $\boldsymbol{A}$ are permutated to the diagonal and then it is scaled in order to equilibrate both rows and columns norms. Subsequently this subroutine solves a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ in use of LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.

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

The unsymmetric real sparse matrix is transformed as below.

$$\boldsymbol{A_1}=\boldsymbol{D_r}\boldsymbol{A}\boldsymbol{P_c}\boldsymbol{D_c}$$

where $\boldsymbol{P_c}$ is an orthogonal matrix for column permutation, $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2}=\boldsymbol{QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1^T}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder for $\boldsymbol{SYM}$. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate. If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.
The solution is computed using LU decomposition.
It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

A..... Input. The nonzero elements of an unsymmetric real sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements belong to an unsymmetric real sparse matrix $\boldsymbol{A}$.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IPLEDSM..... Input. Control information whether to permute the large entries to the diagonal of a matrix $\boldsymbol{A}$.
When IPLEDSM=1 is specified, a matrix $\boldsymbol{A}$ is transformed internally permuting large entries to the diagonal.

Otherwise no permutation is performed.

MZ..... Output. When IPLEDSM=1 is specified, it indicates a permutation of columns. MZ(i)=j indicates that the j-th column which the element of $\boldsymbol{a_{ij}}$ belongs to is permutated to i-th column. The element of $\boldsymbol{a_{ij}}$ is the large entry to be permuted to the diagonal.
One-dimensional array MZ(N).

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix $\boldsymbol{A}$.

When ISCLITERMAX $\leq$ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX $\geq$ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

When IORDERING=10 is specified, calling this routine with ISW=1 produces the informations which is needed to generate an ordering regarding $\boldsymbol{A_1}$ and they are set in NROWSYM and NFCNZSYM.

When IORDERING 11 is specified, it is indicated that after an ordering is set in NPERM, the computation is resumed.
Using the informations obtained in NROWSYM and NFCNZSYM after calling this routines with ISW=1 and IORDERING=10, an ordering is determined. After specifying this ordering in NPERM, this routine is called again with ISW=1and IORDERING=11 and the computation is resumed.
LU decomposition of the matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is continued.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

Output. IORDERING is set to 11 after this routine is called with IORDERING=10 and ISW=1. Therefore after an ordering is set in NPERM the computation is resumed in the subsequent call without IORDERING=11 being specified explicitly.

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

NPERM..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
After symmetrization of a matrix and symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.
Call with IORDERING=10 produces the informations needed for seeking an ordering in NROWSYM and NFCNZSYN. Using these informations an ordering for $\boldsymbol{SYM}$ is determined. After an ordering is set in NPERM, calling this routine with IORDERING=11 and also ISW=1 again resumes the computation.
When IORDERING is neither 10 nor 11, no ordering is specified.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed.
Before calling again with ISW=2, the PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEXL or NSIZEINDEXU at the precedent call and specified in corresponding arguments.
Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

3) When ISW=3 is specified.
The subsequent call with ISW=3 solves another system of equations of which the coefficient matrix is as same as previous call but the right-hand side vector $\boldsymbol{b}$ is changed. In this case, the information obtained by the previous LU decomposition can be reused.
Besides, except ISW as control information and B for storing the new right-hand side $\boldsymbol{b}$, the values in the other arguments must not be changed between the previous and following calls.

NROWSYM..... Output. When it is called with IORDERING=10, the row indices of nonzero pattern of the lower triangular part of $\boldsymbol{SYM}=\boldsymbol{A_1}+\boldsymbol{A_1}^{\boldsymbol{T}}$ in the compressed column storage method are generated.

One-dimensional array NROWSYM(NZ+N).

NFCNZSYM..... Output. When it is called with IORDERING=10, the position of the first row index of each column stored in array NROWSYM in the compressed column storage method which stores the nonzero pattern of the lower part of a matrix $\boldsymbol{SYM}$ column by column.

NFCNZSYM(N+1)=NSYMZ+1 where NSYMZ is the total nonzero elements in the lower triangular part.

One-dimensional array NFCNZ(N+1).

B..... Input. The right-hand side constant vector $\boldsymbol{b}$ of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector ***x***.

One-dimensional array B(N).

NASSIGN..... Output. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSRS-1.
One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of an unsymmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,DIM(2,*i*).

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

Output. The necessary size for the array PANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its

block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEXL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXL.... Input. The size of the array NPANELINDEXL. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(3,*i*) indicates the total amount of the size of the first dimension of the panel where a matrix ***U*** is transposed and stored and the size of its block diagonal portion.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(3,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of an unsymmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

PANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the

section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {DIM(3,*i*)-DIM(2,*i*)} × DIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,DIM(3, *i*)-DIM(2,i), *t*=1,...,DIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

Output. The necessary size for the array PANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix into which ***SYM*** is permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEXU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSRS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEXU.... Input. The size of the array NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of DSIGN(SPEPSZ,PIVOT).
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.
d) IREFINE=1 must be specified for the iterative refinement of the solution.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified. The following conditions must hold.
1.0D-10≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

NFCNZPIVOT.... Output. The location for the storage where the history of relative row and column exchanges for pivoting within each supernode is stored.

The block number of the section where the information on the *i*-th supernode is assigned is known by *j*=NASSIGN(*i*). The position of the first element of that section is stored in NFCNZPIVOT(*j*). The information of exchange rows and columns within the *i*-th supernode is stored in the elements of is=NFCNZPIVOT(*j*),…, ie=NFCNZPIVOT(*j*)+NDIM(2,*j*)-1 in NPIVOTP and NPIVOTQ respectively.

One-dimensional array NFCNZPIVOT(NSUPNUM+1).

NPIVOTP..... Output. The information on exchanges of rows within each supernode is stored.

One-dimensional array NPIVOTP(N).

NPIVORQ..... Output. The information on exchanges of columns within each supernode is stored.

One-dimensional array NPIVOTQ(N).

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

When ISTATIC=1 is specified, IREFINE=1 must be specified.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector ***b-Ax*** is sufficiently smaller compared with the absolute value of ***b***.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

W..... Work area.

Output/Input.
One-dimensional array of size 4*NZ+6*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW1..... Work area.

Output/Input.
One-dimensional array of size 2*NZ+2*(N+1)+16*N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW2..... Work area.

Output/Input.
One-dimensional array of size 47*N+47+NZ+4*(N+1)+2*(NZ+N).

When this subroutine is called repeatedly with ISW=1, 2, 3 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSRS-1.)

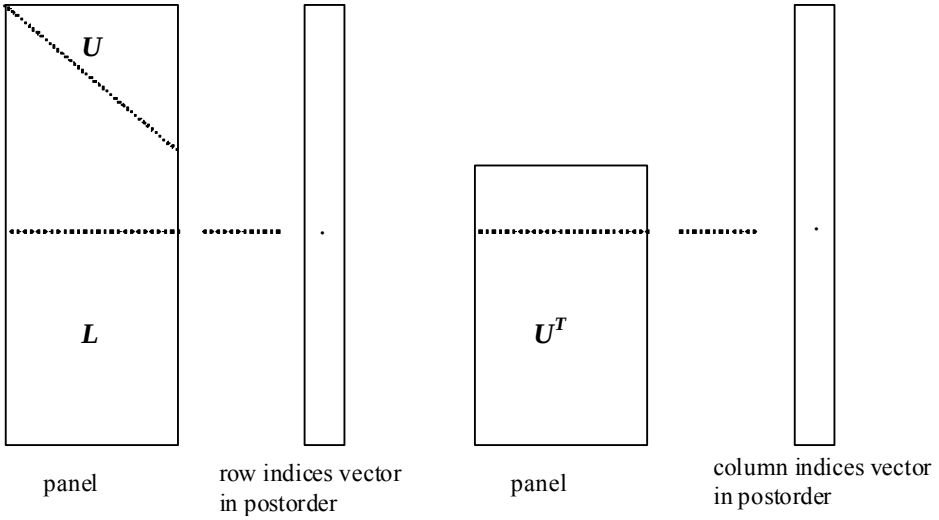


Figure DM_VSRS-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,DIM(1, j),
t = 1,...,DIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix ***U*** of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,DIM(2, j),
t = 1,...,DIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (DIM(3, j)-DIM(2,j))×DIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length DIM(3,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (DIM(3, *j*)-DIM(2, *j*))×DIM(2, *j*).

The transposed unit upper triangular matrix $\boldsymbol{U}^{\boldsymbol{T}}$ except its block diagonal portion of decomposed results is stored in

panel(*x*, *y*), $x = 1,...,$ DIM(3, j)-DIM(2, j), $y = 1,...,$DIM(2, *j*).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix $\boldsymbol{A}$ is permuted in post ordering.

Table DM_VSRS-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
20100	When IPLEDSM is specified, maximum matching with the length N is sought in order to permute large entries to the diagonal but can not be found. The coefficient matrix ***A*** may be singular.	
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix ***A***. The coefficient matrix ***A*** may be singular.	
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	
20500	The norm of residual vector for the solution vector is greater than that of ***b*** multiplied by EPSR, which is the right term constant vector in $\boldsymbol{Ax}=\boldsymbol{b}$. The coefficient matrix ***A*** may be close to a singular matrix.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEXL < 1, NSIZEFACTORU < 1, NSIZEINDEXU < 1, ISW < 1, or ISW > 3, ITERMAX<1 when IREFINE=1.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index *k* stored in NROW(*j*) is $k < 1$ or $k > n$.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n*.	

Code	Meaning	Processing
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or IREFINE≠1 or SPEPSZ>THEPSZ or SPEPSZ>1.0D-10	Processing is discontinued.
31000	The value of NSIZEFACTORL is not enough as the size of PANELFACTORL, or the value of NSIZEINDEXL is not enough as the size of NPANELINDEXL, or the value of NSIZEFACTORU is not enough as the size of PANELFACTORU, or the value of NSIZEINDEXU is not enough as the size of NPANELINDEXU.	Reallocate the PANELFACTORL or NPANELINDEXL or PANELFACTORU or NPANELINDEXU with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEXL or NSIZEFACTORU or NSIZEINDEXU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(i)=j.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is u, the standard value of EPSZ is $16 \times u$.
When the computation is to be continued even if the absolute value of diagonal element is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with SPEPSZ. It is required to specify to do iterative refinement.

3) The necessary sizes for the array PANELFACTORL, NPANELINDEXL, PANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEXL, NSIZEFACTORU and NSIZEINDEXU are

returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix ***A*** into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) Instead of this routine, a system of equations ***Ax***=***b*** can be solved by calling both DM_VSRLU to perform LU decomposition of an unsymmetric real sparse matrix ***A*** and DM_VSRLUX to solve the linear equation in use of decomposed results.

b. Example

The linear system of equations ***Ax***=***f*** is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and the portion in only its six lower diagonals are converted in compressed column storage format. The linear system of equations with an unsymmetric real sparse matrix ***A*** built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=40,KX = NORD,KY =NORD ,KZ = NORD,
     $      N = KX*KY*KZ)
      PARAMETER (NBORDER=N+1,NOFFDIAG=6)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      INTEGER*4 WL
      PARAMETER (NALL=NDIAG*N,
C
     $   WL  =4*NALL+6*N,
     $   IW1L=2*NALL+2*(N+1)+16*N,
     $   IW2L=47*N+47+4*(N+1)+NALL+2*(NALL+N))
C
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION A(K*NDIAG),NROW(K*NDIAG),NFCNZ(N+1),
     $          NROWSYM(K*NDIAG+N),NFCNZSYM(N+1),
     $
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION NPERM(N),W(WL),
     $          NPOSTO(N),NDIM(3,N),
     $          NASSIGN(N),
     $          MZ(N),
     $          IW1(IW1L),IW2(IW2L)
```

```
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,
     $          NDUMMYIU
      INTEGER*8 NSIZEFACTORL,
     $          NSIZEINDEXL,
     $          NSIZEINDEXU,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION B(N),SOLEX(N)
      REAL*8 EPSZ,THEPSZ,SPEPSZ,
     $       SCLROW(N),SCLCOL(N)
C
      INTEGER*4     IPIVOT,ISTATIC,NFCNZPIVOT(N+1),
     $              NPIVOTP(N),NPIVOTQ(N),
     $              IREFINE,ITERMAX,ITER,IPLEDSM
C
      PRINT *,'    LU DECOMPOSITION METHOD'
      PRINT *,'    FOR SPARSE UNSYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *
C
      DO I=1,N
      SOLEX(I)=DBLE(1)
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *
C
      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,KX,KY,KZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZ=1
C
      DO J=1,N
      NTOPCFG=1
```

```
C
      DO I=NDIAG,1,-1
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(J.LT.NBORDER.AND.I.GT.NOFFDIAG)THEN
      CONTINUE
      ELSE
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
C
      NUMNZ=NUMNZ+1
C
      ENDIF
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZ(N+1)=NUMNZ
      NZ=NUMNZ-1
C
      CALL DM_VMVSCC(A,NZ,NROW,NFCNZ,N,SOLEX,
     $             B,WC,IWC,ICON)
C
C     INITIAL CALL WITH IORDER=1
C
      IORDERING= 0            !
      IPLEDSM=1
      ISCLITERMAX=10
      ISW=1
      EPSZ=1.0D-16
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEXL=1
      NSIZEINDEXU=1
      THEPSZ=1.0D-2
      SPEPSZ=0.0D0
      IPIVOT=40
      ISTATIC=0
      IREFINE=1
      EPSR=0.0D0
      ITERMAX=10
C
      CALL DM_VSRS(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              B,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,DUMMYFU,
```

```
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NDUMMYIU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              W,IW1,IW2,ICON)
C
      PRINT*,'ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEXL=',NSIZEINDEXL,
     $       'NSIZEINDEXU=',NSIZEINDEXU,
     $       'NSUPNUM=',NSUPNUM
C
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEXL) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEXU) )
C
      ISW=2
C
      CALL DM_VSRS(A,NZ,NROW,NFCNZ,N,
     $              IPLEDSM,MZ,ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NROWSYM,NFCNZSYM,
     $              B,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,
     $              NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEXL,
     $              NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,
     $              NPANELINDEXU,NSIZEINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,THEPSZ,
     $              IPIVOT,ISTATIC,SPEPSZ,NFCNZPIVOT,
     $              NPIVOTP,NPIVOTQ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              W,IW1,IW2,ICON)
C
      ERR = ERRNRM(SOLEX,B,N)
C
      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',B(1),' X(N) = ',B(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *
C
      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
```

```
         WRITE(*,*)'********** OK **********'
       ELSE
         WRITE(*,*)'********** NG **********'
       ENDIF
C
       DEALLOCATE( PANELFACTORL,PANELFACTORU,
      $            NPANELINDEXL,
      $            NPANELINDEXU )
C
       STOP
       END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
       SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
      &           ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
       IMPLICIT  REAL*8(A-H,O-Z)
       DIMENSION D_L(NDIVP,NDIAG)
       INTEGER   OFFSET(NDIAG)
C
       IF (NDIAG .LT. 1) THEN
         WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
         WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
         RETURN
       ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
       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)

!$OMP DO
       DO I = 1,NDIVP
       DO J = 1,NDIAG
       D_L(I,J) = 0.0
       ENDDO
       ENDDO
!$OMP ENDDO

       NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
       L = 1
       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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
          PRINT*,'ERROR; K0.GH.NZ '
          GOTO 100
        ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C ======================================
* SOLUTE ERROR
* | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The permutation which moves large entries to the diagonal is performed. And the permutated matrix is scaled in order to equilibrate both rows and columns norms. Subsequently the LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.
Refer to references in Appendix A, “References.” in detail.
Refer to [23], [57] on the method how the elements of large absolute value are permuted to diagonal, to [13] on the application algorithms of matching, to [17] on Fibonacci Heaps, to [19], [2], [22], [48], [68] on the LU decomposition of unsymmetric real sparse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VSSPS

A system of linear equations with symmetric positive definite sparse matrices (Left-looking LDL^T decomposition method)
CALL DM_VSSPS(A, NZ, NROW, NFCNZ, N, IORDERING, NPERM, ISW, EPSZ, B, NASSIGN, NSUPNUM, NFCNZFACTOR, PANELFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NDIM, NPOSTO, W, IW1, IW2, IW3, ICON)

(1) Function

This subroutine solves a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ using modified Cholesky LDL^T decomposition, where $\boldsymbol{A}$ is a symmetric positive definite sparse matrix ($n \times n$).

The positive definite sparse matrix is decomposed as

$$\boldsymbol{QPAP}^T\boldsymbol{Q}^T = \boldsymbol{LDL}^T, \quad (1.1)$$

where $\boldsymbol{P}$ is a permutation matrix of ordering and $\boldsymbol{Q}$ is a permutation matrix of post ordering. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices, L is a unit lower triangular matrix, and $\boldsymbol{D}$ is a diagonal matrix.

(2) Parameter

A ..... Input. The non-zero elements of the lower triangular part $\{a_{ij} \mid i \geq j\}$ of a symmetric sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ ..... Input. The total number of the nonzero elements belong to the lower triangular part of a symmetric sparse matrix A.

NROW ..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

IORDERING ..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PAP}^T$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

Specify IORDERING=1 for the decomposition of the matrix $\boldsymbol{PAP}^T$.

Specify the other value for the decomposition of the matrix $\boldsymbol{A}$ as it is.

NPERM ..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW ..... Input. Control information .

1) Specify ISW=1 for the first call.

2) Specify ISW=2 for the subsequent call if the previous call has failed with ICON=31000, that means the size of PANELFACTOR or NPANELINDEX were not enough. In this case, the PANELFACTOR or NPANELINDEX must be reallocated with the necessary sizes which are returned in the NSIZEFACTOR or NSIZEINDEX at the precedent call.

Besides, the values of A, NZ, NROW, NFCNZ, N, IORDERING, NPERM, NASSIGN, NSUPNUM, NFCNZFACTOR, NFCNZINDEX, NPANELINDEX, NPOSTO, NDIM, W, IW1, IW2, and IW3 must be unchanged after the first call.

3) Specify ISW=3 for the second and subsequent calls when solving another system of equations which have the same non-zero pattern of the matrix $\boldsymbol{A}$ but the values of its elements are different. In this case, the information obtained in symbolic decomposition and the array PANELFACTOR and NPANELINDEX of the same size required in previous call can be reused. Then numerical LDL^T decomposition will proceed with that information and the new linear equations can be solved efficiently. Store the values of the matrix elements in the array A, or store in another array C and let it be as the parameter A. The value of NROW must be unchanged in both cases.

Besides, the values of NZ, NROW, NFCNZ, N, IORDERING, NPERM, NASSIGN, NSUPNUM, NFCNZFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NPOSTO, NDIM, W, IW1, IW2, and IW3 must be unchanged also as the previous call.

4) Specify ISW=4 for the second and subsequent calls when solving another system of equations of which the coefficient matrix is as same as previous call but the right-hand side vector $\boldsymbol{b}$ is changed. In this case, the information obtained by the previous LDL^T decomposition can be reused.

Besides the values of N, IORDERING, NPERM, NASSIGN, NSUPNUM, NFCNZFACTOR, NSIZEFACTOR, NFCNZINDEX, NPANELINDEX, NSIZEINDEX, NPOSTO, NDIM, and IW3 must be unchanged as the previous call.

EPSZ ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

When EPSZ is 0.0, the standard value is assumed.

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

B ..... Input. The right-hand side constant vector $\boldsymbol{b}$ of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector $\boldsymbol{x}$.

One-dimensional array B(N).

NASSIGN ..... Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position, where this panel is allocated as a part of the one-dimensional array PANELFACTOR. When j=NASSIGN(i), the i-th supernode is allocated at j-th position.

Input. The values in the first call are reused when ISW $\neq$ 1 specified.

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

One-dimensional array NASSIGN(N).

(See note 3) in (3), "Comments on use.")

NSUPNUM ..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTOR.. Output. Each supernode consists of multiple column vectors, and the factorized matrix of supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element panel(1,1) of the *i*-th panel, where this panel is allocated as a part of the one-dimensional array PANELFACTOR.

One-dimensional 8-byte integer array NFCNZFACTOR(N+1).

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTOR.. Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These panels are stored in this matrix.

The positions of the panel corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZFACTOR(*j*). The decomposed result is stored in each panel.

The size of the i-th panel can be considered to be two-dimensional array of DIM(1,*i*) × DIM(2,*i*). The corresponding part where the lower triangular unit matrix except the diagonal part is stored in panel(*s*, *t*), *s* > *t*, *s* = 1,...,DIM(1, *i*), *t*=1,...,DIM(2,*i*) of the *i*-th panel. The corresponding part of the diagonal matrix ***D*** is stored in panel(*t*, *t*).

One-dimensional array PANELFACTOR(NSIZEFACTOR).

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTOR.. Input. The size of the array PANELFACTOR. 8-byte integer.

Output. The necessary size for the array PANELFACTOR is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEX... Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. The elements of this array indicate the position of the first element of the *i*-th row indices vector, where this panel is allocated as a part of the one-dimensional array NPANELINDEX.

One-dimensional 8-byte integer array NFCNZINDEX(N+1).

Input. The values set by the first call are reused when ISW ≠ 1 specified.

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

NPANELINDEX.. Output. Each supernode consists of multiple column vectors, and the supernodes are stored in two-dimensional panel by compressing rows containing nonzero elements with a common row indices vector. These row indices vectors are stored in this matrix. The positions of the row pointer vector corresponding to the *i*-th supernode are indicated as *j*=NASSIGN(*i*). The first position is stored in NFCNZINDEX(j). The row indices vector is stored by each panel. This row indices are the row indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the matrix A is permuted by post ordering.

One-dimensional array NPANELFACTOR(NSIZEINDEX).

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEX..... Input. The size of the array PANELINDEX. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM ..... Output. The size of first and second dimension of the *i*-th panel are stored in NDIM(1,*i*) and NDIM(2,*i*) respectively.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

Two-dimensional array NDIM(2,N).

For the storage method of the decomposed results, refer to Figure DM_VSSPS-1.

NPOSTO ..... Output. The one dimensional vector is stored which indicates what column index of $\boldsymbol{A}$ the *i*-th node in post ordering corresponds to.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

W ..... Work area.

Output/Input.

When IORDERING=1, one-dimensional array of size NZ.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

When IORDERING≠1, one-dimensional array of size 1.

IW1 ..... Work area.

Output/Input.

When IORDERING=1, one-dimensional array of size NZ+N+1.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

When IORDERING≠1, one-dimensional array of size 1.

IW2 ..... Work area.

Output/Input. One-dimensional array of size NZ+N+1.

When this subroutine is called repeatedly with ISW=1,2,3, This work area is used for preserving information among calls. The contents must not be changed.

IW3 ..... Work area.

Output/Input. One-dimensional array of size N×35+35.

When this subroutine is called repeatedly with ISW=1,2,3,4, This work area is used for preserving information among calls. The contents must not be changed.

ICON ..... Output. Condition code.

(See Table DM_VSSPS-1.)

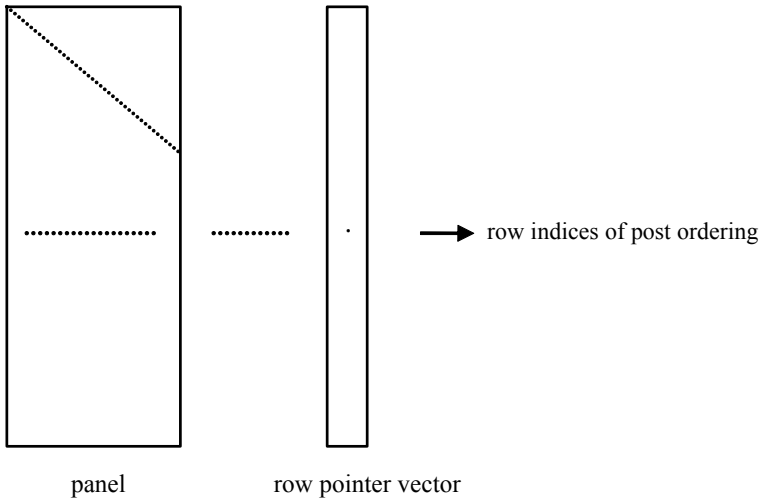


Figure DM_VSSPS-1 concept of storing the data for decomposed result

j = NASSIGN(i) → The i-th supernode is stored at the j-th position.

p = NFCNZFACTOR(j) → The j-th panel occupies the area with a length DIM(1, j)×DIM(2, j) from the p-th element of PANELFACTOR.

q = NFCNZINDEX(j) → The row pointer vector of the j-th panel occupies the area with a length DIM(1,j) from the q-th element of PANELINDEX.

A panel is regarded as an array of the size DIM(1, j)×DIM(2, j).

The lower triangular unit matrix $\boldsymbol{L}$ except the diagonal part is stored in

panel(s, t), $s > t$, s = 1,...,DIM(1, j),

t = 1,...,DIM(2, j).

The corresponding part of the diagonal matrix D is stored in panel(t, t).

The row pointers indicate the column indices of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the node of the matrix A is permuted by post ordering.

Table DM_VSSPS-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	The coefficient matrix is not positive definite.	Processing is continued.
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTOR < 1, NSIZEINDEX < 1, EPSZ < 0, ISW < 1, or ISW > 4.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row pointer *k* stored in NROW(*j*) is $k < i$ or $k >$ N.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n* - *i*+1.	
30400	There is a column without a diagonal element.	
31000	The value of NSIZEFACTOR is not enough as the size of PANELFACTOR, or the value of NSIZEINDEX is not enough as the size of NPANELINDEX.	Reallocate the PANELFACTOR or NPANELINDEX with the necessary size which are returned in the NSIZEFACTOR or NSIZEINDEX, and call this subroutine again.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, "References." in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LDL^T decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the pivot is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
When the pivot becomes negative during the decomposition, the coefficient matrix is not a positive definite. In this case, processing is continued as ICON=10000, but the numerical error may be large because of no pivoting.

3) The necessary sizes for the array PANELFACTOR and NPANELINDEX that store the decomposed results can not be determined beforehand. It is suggested

to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call. For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTOR and NSIZEINDEX. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTOR and NSIZEINDEX are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2, a_3 and c are zero constants, that means the operator is Laplacian. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine init_mat_diag, and transferred into compressed column storage format.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7,NDIAGH=4)

      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAGH*N),NROW(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),NASSIGN(N),W(NDIAGH*N),
     $          NPOSTO(N),NDIM(2,N),
     $          IW1(NDIAGH*N+N+1),
     $          IW2(NDIAGH*N+N+1),
     $          IW3(35*N+35)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTOR
```

```
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEX
      REAL*8 DUMMYF
      INTEGER*4 NDUMMYI
      INTEGER*8 NSIZEFACTOR,NSIZEINDEX,
     $          NFCNZFACTOR(N+1),
     $          NFCNZINDEX(N+1)
      DIMENSION X(N),B(N),SOLEX(N)


      PRINT *,'    LEFT-LOOKING MODIFIED CHOLESKY METHOD'
      PRINT *,'    FOR SPARSE POSITIVE DEFINITE MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 0.0D0
      VA2 = 0.0D0
      VA3 = 0.0D0
      VC =  0.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)

      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
      NUMNZ=1
      DO J=1,N
      NTOPCFGC=1
      NTOPCFG=1
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
```

```
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NCOL.GE.J)THEN
      A(NUMNZ)=DIAG2(J,I)
      NROW(NUMNZ)=NCOL
      ENDIF

C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
C
      IF(NTOPCFG.EQ.1)THEN
      NFCNZ(J)=NUMNZ
      NTOPCFG=0
      ENDIF
C
      IF(NCOL.GE.J)THEN
      NUMNZ=NUMNZ+1
      ENDIF
C
      NUMNZC=NUMNZC+1
      ENDIF
C
      ENDDO
      ENDDO
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
      NFCNZ(N+1)=NUMNZ
      NNZ=NUMNZ-1
C

      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
     $             B,WC,IWC,ICON)
C
      X=B
      IORDERING=0
      ISW=1
      EPSZ=0.0D0
      NSIZEFACTOR=1
      NSIZEINDEX=1

      CALL DM_VSSPS(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,X,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,DUMMYF,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NDUMMYI,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      PRINT *
      PRINT *,'    ICON = ',ICON,' NSIZEFACTOR = ',NSIZEFACTOR,
```

```
     $       'NSIZEINDEX = ',NSIZEINDEX
      PRINT *
C
C     ALLOCATE STORAGES IN RETURNED SIZES
C
      ALLOCATE( PANELFACTOR(NSIZEFACTOR) )
      ALLOCATE( NPANELINDEX(NSIZEINDEX) )

      ISW=2

      CALL DM_VSSPS(A,NNZ,NROW,NFCNZ,N,IORDERING,
     $              NPERM,ISW,EPSZ,X,NASSIGN,NSUPNUM,
     $              NFCNZFACTOR,PANELFACTOR,
     $              NSIZEFACTOR,NFCNZINDEX,
     $              NPANELINDEX,NSIZEINDEX,NDIM,NPOSTO,
     $              W,IW1,IW2,IW3,ICON)

      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTOR,NPANELINDEX )

      STOP
      END

C ======================================
C     INITIALIZE COEFFICIENT MATRIX
C ======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &              ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
```

```
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
    PRINT*,'ERROR; K0.GH.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
!$OMP ENDDO
```

```
!$OMP END PARALLEL

      RETURN
      END

C ======================================
* SOLUTE ERROR
* | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

Through the symbolic decomposition process, this routine analyze the data dependence among columns and the structure of the non-zero elements of matrix ***L*** which is a factor matrix of modified Cholesky LDL^T decomposition. Based on this analysis, the supernodes that bundles certain columns are detected. The columns which have similar non-zero pattern are merged as a supernode together. This means that some rows include additional zero elements and that the number of columns composing a supernode increases. Then data during the numerical decomposition on cache is reused efficiently.

A union set of the row indices that indicate the row indices of the nonzero element of the result of the modified Cholesky decomposition is computed on the columns that compose a supernode. The result of the modified Cholesky decomposition of supernodes is stored compressing it into the two-dimensional panel of which size of the first dimension becomes the number of elements of this set of row indices. The set of row indices is represented as a vector.

The left-looking modified Cholesky decomposition method is used.

For general information on this topic, refer to [19] in Appendix A, “References.”.

DM_VSSSLU

LU decomposition of a structurally symmetric real sparse matrix
CALL DM_VSSSLU(A, NZ, NROW, NFCNZ, N, ISCLITERMAX, IORDERING, NPERM, ISW, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEX, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NPOSTO, SCLROW,SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, W, IW, ICON)

(1) Function

An $n \times n$ structurally symmetric real sparse matrix $\boldsymbol{A}$ is scaled in order to equilibrate both rows and columns norms. And LU decomposition is performed, in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.
(Each nonzero element of a structurally symmetric real sparse matrix has the nonzero elements in its symmetric position. But the values of elements in a symmetric position are not necessarily same.)

The structurally symmetric real sparse matrix is transformed as below.

$$\boldsymbol{A_1 = D_r A D_c}$$

where $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2 = Q P A_1 P^T Q^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of elements for $\boldsymbol{A}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices.
Due to its structural symmetry each pattern of nonzero elements in the decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ respectively is also symmetric to each other. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate.
If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.

(2) Parameter

A..... Input. The nonzero elements of a structurally symmetric real sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements belong to a structurally symmetric real sparse matrix ***A***.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix ***A***.

When ISCLITERMAX ≤ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX ≥ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix ***P*** of ordering or to decompose the matrix ***A***.

When IORDERING 1 is specified, the matrix PA_1P^{T} is decomposed into LU.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

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

NPERM..... Input. The permutation matrix ***P*** is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
A first call. After symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed. Before calling again with ISW=2, the PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEX at the precedent call and specified in corresponding arguments.
Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

NASSIGN..... Output. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSSSLU -1. One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of a structurally symmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU -1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of NDIM(1,*i*) × NDIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,NDIM(1, *i*), *t*=1,...,NDIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU -1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

Output. The necessary size for the array PANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one

dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEX.... Input. The size of the arrays NPANELINDEXL and NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(1,*i*)-NDIM(2,*i*) and NDIM(2,*i*) indicates the total amount of the size of the first dimension and second dimension of the panel where a matrix ***U*** is transposed and stored.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(2,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of a structurally symmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

PANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {NDIM(1,*i*)-NDIM(2,*i*)} × NDIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,NDIM(1, *i*)-NDIM(2,i), *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

Output. The necessary size for the array PANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLU-1.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of DSIGN(SPEPSZ,PIVOT).
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified.
The following conditions must hold.
THEPSZ≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

W..... Work area.

Output/Input.
One-dimensional array of size NZ+N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW..... Work area.

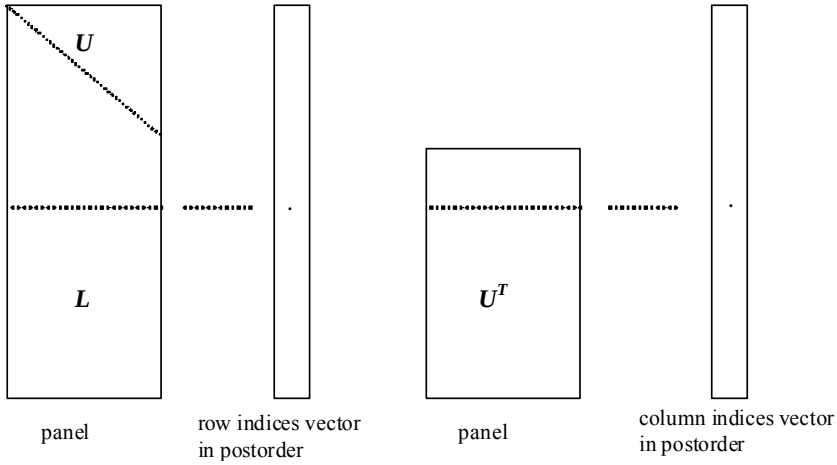
Output/Input.
One-dimensional array of size 36*N+36+2*NZ+3*(N+1).

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSSSLU-1.)

Figure DM_VSSSLU-1 Conceptual scheme for storing decomposed results



j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length NDIM(1, j)×NDIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size NDIM(1, j)×NDIM(2, j).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,NDIM(1, j),

t = 1,...,NDIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix $\boldsymbol{U}$ of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,NDIM(2, j),

t = 1,...,NDIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (NDIM(1, j)-NDIM(2,j))×NDIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (NDIM(1, j)-NDIM(2, j))×NDIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U}^{\boldsymbol{T}}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., NDIM(1, j)-NDIM(2, j), y = 1,...,NDIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix $\boldsymbol{A}$ is permuted in post ordering.

Table DM_VSSSLU-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	When ISTATIC=1 is specified, Static pivot which replaces the pivot candidate with too small value with SPEPSZ is made.	–
20000	The pivot became relatively zero. The coefficient matrix A may be singular.	Processing is discontinued.
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix $\boldsymbol{A}$. The coefficient matrix $\boldsymbol{A}$ may be singular.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEX < 1, NSIZEFACTORU < 1, ISW < 1, or ISW > 2	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index k stored in NROW(j) is k < 1 or k >n.	
30300	The number of row indices belong to i-th column is NFCNZ(i+1)-NFCNZ(i) > n.	

Code	Meaning	Processing
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or SPEPSZ>THEPSZ	Processing is discontinued.
30700	The matrix A is not structurally symmetric.	
31000	The value of NSIZEFACTORL is not enough as the size of PANELFACTORL, or the value of NSIZEINDEX is not enough as the size of NPANELINDEXL and NPANELINDEXU, or the value of NSIZEFACTORU is not enough as the size of PANELFACTORU.	Reallocate the PANELFACTORL or NPANELINDEXL and NPANELINDEXU or PANELFACTORU or with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEX or NSIZEFACTORU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is 16 × *u*. When the computation is to be continued even if the absolute value of diagonal element is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with SPEPSZ.

3) The necessary sizes for the array PANELFACTORL, NPANELINDEXL, PANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEX and NSIZEFACTORU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEX and NSIZEFACTORU are returned. Then the suspended process

can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix $\boldsymbol{Q}$ which is an orthogonal matrix also as well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) A system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be solved by calling DM_VSSSLUX subsequently in use of the results of LU decomposition obtained by this routine. The following arguments used in this routine are specified.
See example in (3), "Comments on use.".

```
A, NZ, NROW, NFCNZ, N,
IORDERING, NPERM,
NASSIGN, NSUPNUM,
NFCNZFACTORL, PANELFACTORL,
NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL,
NSIZEINDEX, NDIM,
NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU,
NFCNZINDEXU, NPANELINDEXU, NPOSTO,
SCLROW,SCLCOL,
IW
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and then it is converted in compressed column storage format. The linear system of equations with a structurally symmetric real sparse matrix $\boldsymbol{A}$ built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ,NXY=NX*NY)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER (NALL=NDIAG*N,
     $   IWL=36*N+36+2*NALL+3*(N+1))
      PARAMETER(IPRINT=0)
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
```

```
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAG*N),NCOLUMN(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),W(NDIAG*N+N),
     $          NPOSTO(N),NDIM(2,N),
     $          NASSIGN(N),
     $          IW(IWL)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,
     $                                        NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,NDUMMYIU
      INTEGER*8 NSIZEFACTORL,NSIZEINDEX,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION X(N),B(N),SOLEX(N),NPERM1(N)
C
      REAL*8 THEPSZ,
     $       EPSR,
     $       SEPSZ,
     $       SCLROW(N),SCLCOL(N)

      INTEGER*4     IPIVOT,ISTATIC,
     $              ISCLITERMAX,
     $              IREFINE,ITERMAX,ITER



      PRINT *,'    DIRECT METHOD'
      PRINT *,'    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *

      DO I=1,N
      SOLEX(I)=1.0D0
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
```

```
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
C
      DO J=1,N
      NTOPCFGC=1
C
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
C
      NUMNZC=NUMNZC+1
C
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
C
      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
     $             B,WC,IWC,ICON)
C
C
      X=B
      IORDERING=0
      ISCLITERMAX=10
      ISW=1
      EPSZ=1.0D-16
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEX=1
      THEPSZ=1.0D-2
      EPSR=1.0D-8
      SEPSZ=1.0D-10
      IPIVOT=40
      ISTATIC=1
      IREFINE=1
      ITERMAX=10

      CALL DM_VSSSLU(C,NNZC,NROWC,NFCNZC,N,
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,DUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NDUMMYIU,
```

```
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
     $              IPIVOT,ISTATIC,SEPSZ,
     $              W,IW,ICON)
      PRINT*,'    ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEX=',NSIZEINDEX
      PRINT*,'    NSUPNUM=',NSUPNUM
      PRINT *
C
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEX) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEX) )
C
      ISW=2
      CALL DM_VSSSLU(C,NNZC,NROWC,NFCNZC,N,
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NPANELINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
     $              IPIVOT,ISTATIC,SEPSZ,
     $              W,IW,ICON)
      CALL GETTOD(T3)
C
      CALL DM_VSSSLUX(N,
     $              IORDERING,
     $              NPERM,
     $              X,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NPANELINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              C,NNZC,NROWC,NFCNZC,
     $              IW,
     $              ICON)

C
      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
```

```
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTORL,PANELFACTORU,NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &            ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
 PRINT*,'ERROR; K0.GH.NZ '
 GOTO 100
 ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C ======================================
* SOLUTE ERROR
* | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The permutation which moves large entries to the diagonal is performed. And the permutated matrix is scaled in order to equilibrate both rows and columns norms. The LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.
Refer to references in Appendix A, “References.” in detail.
Refer to [19], [2], [22], [48], [68] on the LU decomposition of real sparse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VSSSLUX

A system of linear equations with LU-decomposed structurally symmetric real sparse matrices
CALL DM_VSSSLUX(N, IORDERING, NPERM, B, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEX, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NPOSTO, SCLROW,SCLCOL, IREFINE, EPSR, ITERMAX, ITER, A, NZ, NROW, NFCNZ, IW, ICON)

(1) Function

An $n \times n$ structurally symmetric real sparse matrix $\boldsymbol{A}$ of which LU decomposition is made as below is given. In this decomposition an $n \times n$ structurally symmetric real sparse matrix $\boldsymbol{A}$ is scaled in order to equilibrate both rows and columns norms. Subsequently LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode is performed and results in the following form. This routine solves the following linear equation in use of these results of LU decomposition.

$$\boldsymbol{Ax=b}$$

A matrix $\boldsymbol{A}$ is decomposed into as below.

$$\boldsymbol{P_{rs}QPD_rAD_cP^TQ^TP_{cs}=LU}$$

The structurally symmetric real sparse matrix $\boldsymbol{A}$ is transformed as below.

$$\boldsymbol{A_1= D_rAD_c}$$

Where $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2=QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
$\boldsymbol{P_{rs}}$ and $\boldsymbol{P_{cs}}$ represent row and column exchanges in orthogonal matrices respectively. The actual exchanges are restricted to the reduced part of the matrix belonging to each supernode.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of nonzero elements for $\boldsymbol{A}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix. It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

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

IORDERING..... Input. When IORDERING 1 is specified, it is indicated that LU decomposition is performed with an ordering specified in NPERM.
The matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is decomposed into LU decomposition.

Otherwise. No ordering is specified.

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

NPERM..... Input. When IORDEING=1 is specified, a vector presenting the permutation matrix ***P*** used is stored.

One-dimensional array NPERM(N).

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

B..... Input. The right-hand side constant vector ***b*** of a system of linear equations ***Ax*** = ***b***.

Output. Solution vector ***x***.

One-dimensional array B(N).

NASSIGN..... Input. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Regarding the storage methods of decomposed matrices, refer to Figure DM_VSSSLUX-1.
One-dimensional array NASSING(N).

NSUPNUM..... Input. The total number of supernodes.(≤ n)

NFCNZFACTORL..Input. The decomposed matrices ***L*** and ***U*** of a structurally symmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

PANELFACTORL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of NDIM(1,*i*) × NDIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,NDIM(1, *i*), *t*=1,...,NDIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

NFCNZINDEXL... Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NPANELINDEXL..Input. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NSIZEINDEX.... Input. The size of the arrays NPANELINDEXL and NPANELINDEXU. 8-byte integer.

NDIM..... Input. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(1,*i*) -NDIM(2,*i*) and NDIM(2,*i*) indicates the total amount of the size of the first dimension and second dimension of the panel where a matrix ***U*** is transposed and stored.

Two-dimensional array NDIM(2,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NFCNZFACTORU..Input. Regarding a matrix ***U*** derived from LU decomposition of a structurally symmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

PANELFACTORU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional

array of {NDIM(1,*i*)-NDIM(2,*i*)} × NDIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,NDIM(1, *i*)-NDIM(2,i), *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NPANELINDEXU..Input. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSLUX-1.

NPOSTO..... Input. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

One-dimensional array NPOSTO(N).

(See note 3) in (3), "Comments on use.")

SCLROW..... Input. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

One-dimensional array SCLROW (N).

SCLCOL..... Input. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

One-dimensional array SCLCOL(N).

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their

corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector ***b-Ax*** is sufficiently smaller compared with the absolute value of ***b***.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

A..... Input. The nonzero elements of a structurally symmetric real sparse matrix ***A*** are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements to belong to a structurally symmetric real sparse matrix ***A***.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element to stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

IW..... Work area.

Input.
One-dimensional array of size 36*N+36+2*NZ+3*(N+1).

The data derived from calling DM_VSSSLU of LU decomposition of a structurally symmetric real sparse matrix is transferred in this work area. The contents must not be changed among calls.

ICON..... Output. Condition code.

(See Table DM_VSSSLUX-1.)

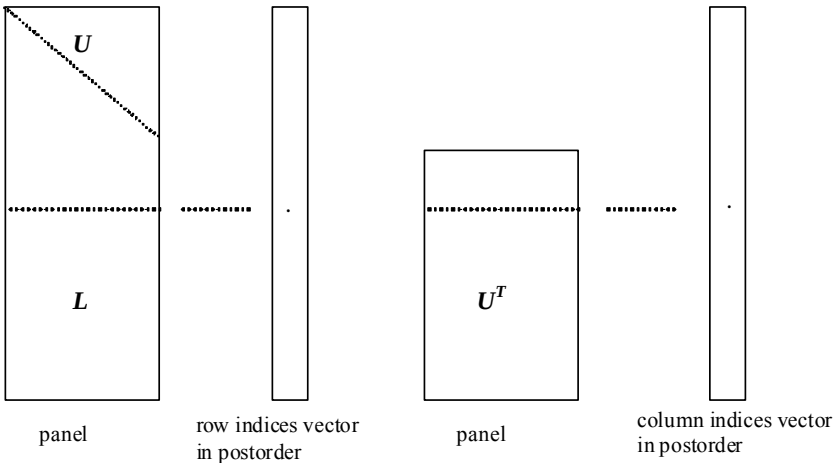


Figure DM_VSSSLUX-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length NDIM(1, j)×NDIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size NDIM(1, j)×NDIM(2, j).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,..., NDIM(1, j),

t = 1,..., NDIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix ***U*** of decomposed results is stored in

panel(s, t), $s < t$, s = 1,..., NDIM(2, j),

t = 1,..., NDIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (NDIM(1, j)-NDIM(2,j))×NDIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (NDIM(1, j)-NDIM(2, j))×NDIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U}^T$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., NDIM(1, j)-NDIM(2, j), y = 1,...,NDIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix ***A*** is permuted in post ordering.

Table DM_VSSSLUX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	Processing is discontinued.
20500	The norm of residual vector for the solution vector is greater than that of ***b*** multiplied by EPSR, which is the right term constant vector in ***Ax***=***b***. The coefficient matrix ***A*** may be close to a singular matrix.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEX < 1, NSIZEFACTORU < 1, ITERMAX<1 when IREFINE=1.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index k stored in NROW(j) is $k < 1$ or $k > n$.	
30300	The number of row indices belong to i-th column is NFCNZ(i+1)-NFCNZ(i) > n.	

(3) Comments on use

a. Notes

1) The results of LU decomposition obtained by DM_VSSSLU is used. See note 5) (3), "Comments on use." of DM_VSSSLU and example in (3), "Comments on use." of DM_VSSSLUX.

2) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(i)=j. The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

3) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the i-th node in post order is corresponding. It means j-th position when j = NPOSTO(i).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as well as note 2) above, and corresponds to permute the matrix ***A*** into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and then it is converted in compressed column storage format. The linear system of equations with a structurally symmetric real sparse matrix $\boldsymbol{A}$ built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ,NXY=NX*NY)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER (NALL=NDIAG*N,
     $   IWL=36*N+36+2*NALL+3*(N+1))
      PARAMETER(IPRINT=0)
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAG*N),NCOLUMN(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),W(NDIAG*N+N),
     $          NPOSTO(N),NDIM(2,N),
     $          NASSIGN(N),
     $          IW(IWL)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,
     $                                        NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,NDUMMYIU
      INTEGER*8 NSIZEFACTORL,NSIZEINDEX,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION X(N),B(N),SOLEX(N),NPERM1(N)
C
      REAL*8 THEPSZ,
     $       EPSR,
     $       SEPSZ,
     $       SCLROW(N),SCLCOL(N)

      INTEGER*4     IPIVOT,ISTATIC,
     $              ISCLITERMAX,
     $              IREFINE,ITERMAX,ITER



      PRINT *,'    DIRECT METHOD'
      PRINT *,'    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
```

```
      PRINT *

      DO I=1,N
      SOLEX(I)=1.0D0
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
C
      DO J=1,N
      NTOPCFGC=1
C
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
C
      NUMNZC=NUMNZC+1
C
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
C
      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
```

```
     $             B,WC,IWC,ICON)
C
C
      X=B
      IORDERING=0
      ISCLITERMAX=10
      ISW=1
      EPSZ=1.0D-16
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEX=1
      THEPSZ=1.0D-2
      EPSR=1.0D-8
      SEPSZ=1.0D-10
      IPIVOT=40
      ISTATIC=1
      IREFINE=1
      ITERMAX=10

      CALL DM_VSSSLU(C,NNZC,NROWC,NFCNZC,N,
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,DUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NDUMMYIU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
     $              IPIVOT,ISTATIC,SEPSZ,
     $              W,IW,ICON)
      PRINT*,'    ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       ' NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEX=',NSIZEINDEX
      PRINT*,'    NSUPNUM=',NSUPNUM
      PRINT *
C
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEX) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEX) )
C
      ISW=2
      CALL DM_VSSSLU(C,NNZC,NROWC,NFCNZC,N,
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NPANELINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
```

```
     $              IPIVOT,ISTATIC,SEPSZ,
     $              W,IW,ICON)
      CALL GETTOD(T3)
C
      CALL DM_VSSSLUX(N,
     $              IORDERING,
     $              NPERM,
     $              X,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NPANELINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              C,NNZC,NROWC,NFCNZC,
     $              IW,
     $              ICON)

C
      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTORL,PANELFACTORU,NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &            ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
```

```
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

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

```
 PRINT*,'ERROR; K0.GH.NZ '
 GOTO 100
 ENDIF
       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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =======================================
* SOLUTE ERROR
* | X1 - X2 |
C =======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

DM_VSSSS

A system of linear equations with structurally symmetric real sparse matrices (LU decomposition method)
CALL DM_VSSSS(A, NZ, NROW, NFCNZ, N, ISCLITERMAX, IORDERING, NPERM, ISW, B, NASSIGN, NSUPNUM, NFCNZFACTORL, PANELFACTORL, NSIZEFACTORL, NFCNZINDEXL, NPANELINDEXL, NSIZEINDEX, NDIM, NFCNZFACTORU, PANELFACTORU, NSIZEFACTORU, NFCNZINDEXU, NPANELINDEXU, NPOSTO, SCLROW, SCLCOL, EPSZ, THEPSZ, IPIVOT, ISTATIC, SPEPSZ, IREFINE, EPSR, ITERMAX, ITER, W, IW, ICON)

(1) Function

An $n \times n$ structurally symmetric real sparse matrix $\boldsymbol{A}$ is scaled in order to equilibrate both rows and columns norms. Subsequently this subroutine solves a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ in use of LU decomposition in which the pivot is taken as specified within the block diagonal portion belonging to each supernode.
(Each nonzero element of a structurally symmetric real sparse matrix has the nonzero element in its symmetric position. But the values of elements in a symmetric position are not necessarily same.)

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

The structurally symmetric real sparse matrix is transformed as below.

$$\boldsymbol{A_1}=\boldsymbol{D_r A D_c}$$

where $\boldsymbol{D_r}$ is a diagonal matrix for scaling rows and $\boldsymbol{D_c}$ is also a diagonal matrix for scaling columns.

$$\boldsymbol{A_2}=\boldsymbol{QPA_1P^TQ^T}$$

$\boldsymbol{A_2}$ is decomposed into $\boldsymbol{LU}$ decomposition permuting rows and columns within the block diagonal portion of each supernode according to specified pivoting.
In the right term $\boldsymbol{P}$ is a permutation matrix of ordering which is sought for a pattern of elements for $\boldsymbol{A}$ and $\boldsymbol{Q}$ is a permutation matrix of postorder. $\boldsymbol{P}$ and $\boldsymbol{Q}$ are orthogonal matrices.
Due to its structural symmetry each pattern of nonzero elements in the decomposed matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ respectively is also symmetric to each other. $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is a unit upper triangular matrix.
When in pivoting process a candidate matrix element whose absolute value is larger than or equal to the threshold specified in THEPSZ can not be found, the element with the largest absolute value which in the block diagonal portion of a supernode is regarded as a candidate. If the absolute value of the candidate element is too small, the matrix can be approximately decomposed into LU specifying an appropriate small value as a static pivot in place of the candidate sought.
The solution is computed using LU decomposition.
It can be specified to improve the precision of the solution by iterative refinement.

(2) Parameter

A..... Input. The nonzero elements of a structurally symmetric real sparse matrix $\boldsymbol{A}$ are stored in A(1:NZ).

One-dimensional array A(NZ).

For the compressed column storage method, refer to Figure DM_VMVSCC-1 in the description for DM_VMVSCC routine (multiplication of a real sparse matrix and a real vector).

NZ..... Input. The total number of the nonzero elements belong to a structurally symmetric real sparse matrix $\boldsymbol{A}$.

NROW..... Input. The row indices used in the compressed column storage method, which indicate the row number of each nonzero element stored in an array A.

One-dimensional array NROW(NZ).

NFCNZ..... Input. The position of the first nonzero element of each column stored in an array A in the compressed column storage method which stores the nonzero elements column by column.

NFCNZ(N+1)=NZ+1.

One-dimensional array NFCNZ(N+1).

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

ISCLITERMAX... Input. The upper limit for the number of iteration to seek scaling matrices of $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ to equilibrate both rows and columns of matrix $\boldsymbol{A}$.

When ISCLITERMAX ≤ 0 is specified no scaling is done. In this case $\boldsymbol{D_r}$ and $\boldsymbol{D_c}$ are assumed as unit matrices.

When ISCLITERMAX ≥ 10 is specified, the upper limit for the number of iteration is considered as 10.

IORDERING..... Input. Control information whether to decompose the reordered matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ permuted by the matrix $\boldsymbol{P}$ of ordering or to decompose the matrix $\boldsymbol{A}$.

When IORDERING=1 is specified, the matrix $\boldsymbol{PA_1P}^{\mathrm{T}}$ is decomposed into LU.

Otherwise. Without any ordering, the matrix $\boldsymbol{A_1}$ is decomposed into LU.

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

NPERM..... Input. The permutation matrix $\boldsymbol{P}$ is stored as a vector.

One-dimensional array NPERM(N).

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

ISW..... Input. Control information.

1)When ISW=1 is specified.
A first call. Symbolic decomposition, checking whether the sufficient amount of memory for storing data are allocated the computation is performed.

2) When ISW=2 specified.
After the previous call ends with ICON=31000, that means that the sizes of PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU were not enough, the suspended computation is resumed. Before calling again with ISW=2, the PANELFACTORL or PANELFACTORU or NPANELINDEXL or NPANELINDEXU must be reallocated with the necessary sizes which are returned in the NSIZEFACTORL NSIZEFACTORU or NSIZEINDEX at the precedent call and specified in corresponding arguments.

Besides, except these arguments and ISW as control information, the values in the other augments must not be changed between the previous and following calls.

3) When ISW=3 is specified.
The subsequent call with ISW=3 solves another system of equations of which the coefficient matrix is as same as previous call but the right-hand side vector ***b*** is changed. In this case, the information obtained by the previous LU decomposition can be reused.
Besides, except ISW as control information and B for storing the new right-hand side ***b***, the values in the other arguments must not be changed between the previous and following calls.

B..... Input. The right-hand side constant vector ***b*** of a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.

Output. Solution vector ***x***.

One-dimensional array B(N).

NASSIGN..... Output. ***L*** and ***U*** belonging to each supernode are compressed and stored in two dimensional panels respectively. These panels are stored in PANELFACTORL and PANELFACTORU as one dimensional subarray consecutively and its block number is stored. The corresponding indices vectors are similarly stored NPANELINDEXL and NPANELINDEXU respectively. Data of the *i*-th supernode is stored into the *j*-th block of a subarray, where *j*=NASSIN(*i*).

Input. When ISW≠1, the values stored in the first call are reused. Regarding the storage methods of decomposed matrices, refer to Figure DM_VSSSS-1.
One-dimensional array NASSING(N).

NSUPNUM..... Output. The total number of supernodes.

Input. The values in the first call are reused when ISW ≠ 1 specified. (≤ n)

NFCNZFACTORL..Output. The decomposed matrices ***L*** and ***U*** of a structurally symmetric real sparse matrix are computed for each supernode respectively. The columns of ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORL consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORL(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

PANELFACTORL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORL(*j*).

The size of the panel in the *i*-th block can be considered to be two dimensional array of NDIM(1,*i*) × NDIM(2,*i*). The corresponding parts of the lower triangular matrix ***L*** are store in this panel(*s*, *t*), *s*≥ *t*, *s* = 1,...,NDIM(1, *i*), *t*=1,...,NDIM(2,*i*). The corresponding block diagonal portion of the unit upper triangular matrix ***U*** except its diagonals is stored in the panel(s,t), s<t, *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORL(NSIZEFACTORL).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORL.. Input. The size of the array PANELFACTORL. 8-byte integer.

Output. The necessary size for the array PANELFACTORL is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXL... Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored in a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th row indices vector is mapped into NPANELINDEXL consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXL(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

NPANELINDEXL..Output. The columns of the decomposed matrix ***L*** belonging to each supernode are compressed to have the common row indices vector and stored into a two dimensional panel with the corresponding parts of the decomposed matrix ***U*** in its block diagonal portion. This column indices vector is mapped into NPANELINDEXL consecutively. The block number of the section where the row indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXL(*j*). This row indices are the row numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXL(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

(See note 3) in (3), "Comments on use.")

NSIZEINDEX.... Input. The size of the arrays NPANELINDEXL and NPANELINDEXU. 8-byte integer.

Output. The necessary size is returned.

(See note 3) in (3), "Comments on use.")

NDIM..... Output. NDIM(1,*i*) and NDIM(2,*i*) indicate the sizes of the first dimension and second dimension of the panel to store a matrix ***L*** respectively, which is allocated in the *i*-th location.
NDIM(1,*i*)-NDIM(2,*i*) and NDIM(2,*i*) indicates the total amount of the size of the first dimension and second dimension of the panel where a matrix ***U*** is transposed and stored.

Input. When ISW≠1, the values set by the first call are reused.

Two-dimensional array NDIM(2,N).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

NFCNZFACTORU..Output. Regarding a matrix ***U*** derived from LU decomposition of a structurally symmetric real sparse matrix, the rows of ***U*** except the of block diagonal portion belonging to each supernode are compressed to have the common column indices vector and stored into a two dimensional panel. The index number of the top array element of the one dimensional subarray where the *i*-th panel is mapped into PANELFACTORU consecutively or the location of panel(1,1) is stored.

One-dimensional 8-byte integer array NFCNZFACTORU(N+1).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

Input. When ISW ≠ 1, the values set by the first call are reused.

PANELFACTORU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The block number of the section where the panel corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray including the portion of decomposed matrices is stored in NFCNZFACTORU(*j*). The size of the panel in the *i*-th block can be considered to be two dimensional array of {NDIM(1,*i*)-NDIM(2,*i*)} × NDIM(2,*i*). The rows of the unit upper triangular matrix ***U*** except the block diagonal portion are compressed, transposed and stored in this panel(*s*, *t*), *s* = 1,...,NDIM(1, *i*)-NDIM(2,i), *t*=1,...,NDIM(2,*i*).

One-dimensional array PANELFACTORU(NSIZEFACTORU).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

(See note 3) in (3), "Comments on use.")

NSIZEFACTORU.. Input. The size of the array PANELFACTORU. 8-byte integer.

Output. The necessary size for the array PANELFACTORU is returned.

(See note 3) in (3), "Comments on use.")

NFCNZINDEXU...Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed to have the common column indices vector, transposed and stored in a two dimensional panel without its block diagonal portion. The index number of the top array element of the one dimensional subarray where the *i*-th column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively is stored.

One-dimensional 8-byte integer array NFCNZINDEXU(N+1).

Input. When ISW ≠ 1, the values set by the first call are reused.

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

NPANELINDEXU..Output. The rows of the decomposed matrix ***U*** belonging to each supernode are compressed, transposed and stored in a two dimensional panel without its block diagonal portion. The column indices vector including indices of the block diagonal portion is mapped into NPANELINDEXU consecutively. The block number of the section where the column indices vector corresponding to the *i*-th supernode is assigned is known from *j*=NASSIGN(*i*). The location of its top of subarray is stored in NFCNZINDEXU(*j*). These column indices are the column numbers of the matrix permuted in its post order.

One-dimensional array NPANELINDEXU(NSIZEINDEX).

Regarding the storage method of the decomposed results, refer to Figure DM_VSSSS-1.

(See note 3) in (3), "Comments on use.")

NPOSTO..... Output. The information about what column number of ***A*** the *i*-th node in post order corresponds to is stored.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array NPOSTO(N).

(See note 4) in (3), "Comments on use.")

SCLROW..... Output. The diagonal elements of $\boldsymbol{D_r}$ or a diagonal matrix for scaling rows are stored in one dimensional array.

Input. When ISW ≠ 1, the values set by the first call are reused.

One-dimensional array SCLROW (N).

SCLCOL..... Output. The diagonal elements of $\boldsymbol{D_c}$ or a diagonal matrix for scaling columns are stored in one dimensional array.

Input. The values set by the first call are reused when ISW ≠ 1 specified.

One-dimensional array SCLCOL(N).

EPSZ..... Input. Judgment of relative zero of the pivot (≥ 0.0).

Output. When EPSZ ≤ 0.0, it is set to the standard value.

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

THEPSZ..... Input. Threshold used in judgement for a pivot. Immediately after a candidate in pivot search is considered to have the value greater than or equal to the threshold specified, it is accepted as a pivot and the search of a pivot is broken off.
For example, 1.0D-2.

Output. When THEPSZ≤0.0D0, 1.0D-2 is set.
When EPSZ≥THEPSZ>0.0, it is set to the value of EPSZ.

IPIVOT..... Input. Control information on pivoting which indicates whether a pivot is searched and what kind of pivoting is chosen if any.
For example, 40 for complete pivoting.

IPIVOT<10 or IPIVOT≥ 50, no pivoting.

10≤IPIVOT<20, partial pivoting

20≤IPIVOT<30, diagonal pivoting

21 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting.

22 : When within a supernode diagonal pivoting fails, it is changed to Rook pivoting. If Rook pivoting fails, it is changed to complete pivoting.

30≤IPIVOT<40, Rook pivoting

32 : When within a supernode Rook pivoting fails, it is changed to complete pivoting.

40≤IPIVOT<50, complete pivoting

ISTATIC..... Input. Control information indicating whether Static pivoting is taken.

1) When ISTATIC=1 is specified.
When the pivot searched within a supernode is not greater than SPEPSZ, it is replaced with its approximate value of DSIGN(SPEPSZ,PIVOT).
If its value is 0.0D0, SPEPSZ is used as an approximation value.

The following conditions must be satisfied.
a) EPSZ must be less than or equal to the standard value of EPSZ.
b) Scaling must be performed with ISCLITERMAX=10.
c) THEPSZ≥SPEPSZ must hold.
d) IREFINE=1 must be specified for the iterative refinement of the solution.

2) When ISTATIC≠1 is specified.
No static pivot is performed.

SPEPSZ..... Input. The approximate value used in Static pivoting when ISTATIC=1 is specified. The following conditions must hold.
1.0D-10≥SPEPSZ≥EPSZ

Output. When SPEPSZ<EPSZ, it is set to 1.0D-10.

IREFINE..... Input. Control information indicating whether iterative refinement is performed when the solution is computed in use of results of LU decomposition. A residual vector is computed in quadruple precision.

When IREFINE=1 is specified.
The iterative refinement is performed. It is iterated until in the sequences of the solutions obtained in refinement the difference of the absolute values of their corresponding residual vectors become larger than a fourth of that of immediately previous ones.

When IREFINE≠1is specified.
No iterative refinement is performed.

When ISTATIC=1 is specified, IREFINE=1 must be specified.

EPSR..... Input. Criterion value to judge if the absolute value of the residual vector $\boldsymbol{b}-\boldsymbol{Ax}$ is sufficiently smaller compared with the absolute value of $\boldsymbol{b}$.

When EPSR ≤ 0.0, it is set to 1.0D-6.

ITERMAX..... Input. Upper limit of iterative count for refinement (≥ 1).

ITER..... Output. Actual iterative count for refinement.

W..... Work area.

Output/Input.
One-dimensional array of size NZ+N.

When this subroutine is called repeatedly with ISW=1, 2 this work area is used for preserving information among calls. The contents must not be changed.

IW..... Work area.

Output/Input.
One-dimensional array of size 36*N+36+2*NZ+3*(N+1).

When this subroutine is called repeatedly with ISW=1, 2, 3 this work area is used for preserving information among calls. The contents must not be changed.

ICON..... Output. Condition code.

(See Table DM_VSSSS-1.)

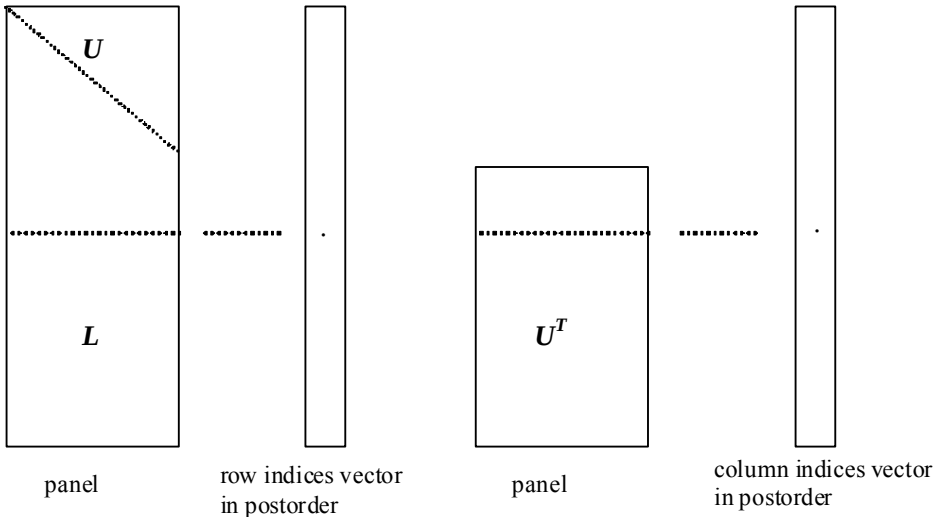


Figure DM_VSSSS-1 Conceptual scheme for storing decomposed results

j = NASSIGN(i) → The i-th supernode is stored at the j-th section.

p = NFCNZFACTORL(j) → The j-th panel occupies the area with a length NDIM(1, j)×NDIM(2, j) from the p-th element of PANELFACTORL.

q = NFCNZINDEXL(j) → The row indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the q-th element of NPANELINDEXL.

A panel is regarded as an array of the size NDIM(1, j)×NDIM(2, j).

The lower triangular matrix ***L*** of decomposed results is stored in

panel(s, t), $s \geq t$, s = 1,...,NDIM(1, j),

t = 1,...,NDIM(2, j).

The block diagonal portion except diagonals of the unit upper triangular matrix ***U*** of decomposed results is stored in

panel(s, t), $s < t$, s = 1,...,NDIM(2, j),

t = 1,...,NDIM(2, j).

u = NFCNZFACTORU(j) → The j-th panel occupies the area with a length (NDIM(1, j)-NDIM(2,j))×NDIM(2, j) from the u-th element of PANELFACTORU.

v = NFCNZINDEXU(j) → The column indices vector of the j-th panel occupies the area with a length NDIM(1,j) from the v-th element of NPANELINDEXU.

A panel is regarded as an array of the size (NDIM(1, j)-NDIM(2, j))×NDIM(2, j).

The transposed unit upper triangular matrix $\boldsymbol{U^T}$ except its block diagonal portion of decomposed results is stored in

panel(x, y), x = 1,..., DIM(3, j)-DIM(2, j), y = 1,...,DIM(2, j).

The indices indicate the column numbers of the matrix $\boldsymbol{QAQ}^{\mathrm{T}}$ to which the nodes of the matrix ***A*** is permuted in post ordering.

Table DM_VSSSS-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	The pivot became relatively zero. The coefficient matrix *A* may be singular.	Processing is discontinued.
20200	When seeking diagonal matrices for equilibrating both rows and columns, there is a zero vector in either rows or columns of the matrix ***A***. The coefficient matrix ***A*** may be singular.	
20400	There is a zero element in diagonal of resultant matrices of LU decomposition.	
20500	The norm of residual vector for the solution vector is greater than that of ***b*** multiplied by EPSR, which is the right term constant vector in ***Ax***=***b***. The coefficient matrix ***A*** may be close to a singular matrix.	
30000	N < 1, NZ < 0, NFCNZ(N+1) ≠ NZ+1, NSIZEFACTORL < 1, NSIZEINDEX < 1, NSIZEFACTORU < 1, ISW < 1, or ISW > 3, ITERMAX<1 when IREFINE=1.	
30100	The permutation matrix specified in NPREM is not correct.	
30200	The row index *k* stored in NROW(*j*) is $k < 1$ or $k > n$.	
30300	The number of row indices belong to *i*-th column is NFCNZ(*i*+1)-NFCNZ(*i*) > *n*.	
30500	When ISTATIC=1 is specified, the required conditions are not satisfied. EPSZ is greater than 16***u*** of the standard value or ISCLITERMAX<10 or IREFINE≠1 or SPEPSZ>THEPSZ or SPEPSZ>1.0D-10	
30700	The matrix A is not structurally symmetric.	

Code	Meaning	Processing
31000	The value of NSIZEFACTORL is not enough as the size of PANELFACTORL, or the value of NSIZEINDEX is not enough as the size of NPANELINDEXL and NPANELINDEXU, or the value of NSIZEFACTORU is not enough as the size of PANELFACTORU.	Reallocate the PANELFACTORL or NPANELINDEXL and NPANELINDEXU or PANELFACTORU or with the necessary size which are returned in the NSIZEFACTORL or NSIZEINDEX or NSIZEFACTORU respectively and call this subroutine again with ISW=2 specified.

(3) Comments on use

a. Notes

1) When the element p_{ij}=1 of the permutation matrix ***P***, set NPERM(*i*)=*j*.
The inverse of the matrix can be obtained as follows:

```
DO i = 1,n
j = NPERM(i)
NPERMINV(j) = i
ENDDO
```

Fill-reduction Orderings are obtained in use of METIS and so on.
Refer to [43], [44] in Appendix A, “References.” in detail.

2) If EPSZ is set, the pivot is assumed to be relatively zero when it is less than EPSZ in the process of LU decomposition. In this case, processing is discontinued with ICON = 20000. When unit round off is *u*, the standard value of EPSZ is $16 \times u$. When the computation is to be continued even if the absolute value of diagonal element is small, assign the minimum value to EPSZ. In this case, however, the result is not assured.
If Static pivot is specified to be performed, when the diagonal element is smaller than SPEPSZ, LU decomposition is approximately continued replacing it with SPEPSZ. It is required to specify to do iterative refinement.

3) The necessary sizes for the array PANELFACTORL, NPANELINDEXL, PANELFACTORU and NPANELINDEXU that store the decomposed results can not be determined beforehand. It is suggested to reallocate them by using the result of the symbolic decomposition analysis after the first call of this routine, or allocate large enough arrays at first call.
For instance, allocate the small one-dimensional arrays of size one at first. And call this routine with the small values such as one in the size specifying in NSIZEFACTORL, NSIZEINDEX and NSIZEFACTORU with ISW=1. This routine ends with ICON=31000, and the necessary sizes for NSIZEFACTORL, NSIZEINDEX and NSIZEFACTORU are returned. Then the suspended process can be resumed by calling it with ISW=2 after reallocating the arrays with the necessary sizes.

4) Nodes corresponding to column number is considered. The node number permuted in post order is stored in NPOSTO. This array indicates what node number in original node number the *i*-th node in post order is corresponding. It means *j*-th position when *j* = NPOSTO(*i*).
This array represents a permutation matrix ***Q*** which is an orthogonal matrix also as

well as note 1) above, and corresponds to permute the matrix $\boldsymbol{A}$ into $\boldsymbol{QAQ}^{\mathrm{T}}$.
The inverse matrix $\boldsymbol{Q}^{\mathrm{T}}$ can be obtained as follows:

```
DO i = 1,n
j = NPOSTO(i)
NPOSTOINV(j) = i
ENDDO
```

5) Instead of this routine, a system of equations $\boldsymbol{Ax}=\boldsymbol{b}$ can be solved by calling both DM_VSSSLU to perform LU decomposition of a structurally symmetric real sparse matrix $\boldsymbol{A}$ and DM_VSSSLUX to solve the linear equation in use of decomposed results.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where a matrix is built using results from the finite difference method applied to the elliptic equation

$$-\Delta u + a\nabla u + cu = f$$

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$.
The matrix in diagonal storage format is generated by the subroutine init_mat_diag and then it is converted in compressed column storage format. The linear system of equations with a structurally symmetric real sparse matrix $\boldsymbol{A}$ built in this way is solved.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (NORD=39,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ,NXY=NX*NY)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER (NALL=NDIAG*N,
     $   IWL=36*N+36+2*NALL+3*(N+1))
      PARAMETER(IPRINT=0)
      DIMENSION NOFST(NDIAG)
      DIMENSION DIAG(K,NDIAG),DIAG2(K,NDIAG)
      DIMENSION C(K*NDIAG),NROWC(K*NDIAG),NFCNZC(N+1),
     $          WC(K*NDIAG),IWC(2,K*NDIAG)
      DIMENSION A(NDIAG*N),NCOLUMN(K*NDIAG),NFCNZ(N+1),
     $          NPERM(N),W(NDIAG*N+N),
     $          NPOSTO(N),NDIM(2,N),
     $          NASSIGN(N),
     $          IW(IWL)
      REAL*8, DIMENSION(:), ALLOCATABLE ::  PANELFACTORL,PANELFACTORU
      INTEGER*4, DIMENSION(:), ALLOCATABLE :: NPANELINDEXL,
     $                                        NPANELINDEXU
      REAL*8 DUMMYFL,DUMMYFU
      INTEGER*4 NDUMMYIL,NDUMMYIU
      INTEGER*8 NSIZEFACTORL,NSIZEINDEX,
     $          NSIZEFACTORU,
     $          NFCNZFACTORL(N+1),
     $          NFCNZFACTORU(N+1),
     $          NFCNZINDEXL(N+1),
     $          NFCNZINDEXU(N+1)
      DIMENSION X(N),B(N),SOLEX(N),NPERM1(N)
C
      REAL*8 THEPSZ,
     $       EPSR,
     $       SEPSZ,
```

```
     $       SCLROW(N),SCLCOL(N)

      INTEGER*4     IPIVOT,ISTATIC,
     $              ISCLITERMAX,
     $              IREFINE,ITERMAX,ITER



      PRINT *,'    DIRECT METHOD'
      PRINT *,'    FOR SPARSE STRUCTURALLY SYMMETRIC REAL MATRICES'
      PRINT *,'    IN COMPRESSED COLUMN STORAGE'
      PRINT *

      DO I=1,N
      SOLEX(I)=1.0D0
      ENDDO
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 1.0D0
      VA2 = 2.0D0
      VA3 = 3.0D0
      VC =  4.0D0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_DIAG(VA1,VA2,VA3,VC,DIAG,NOFST
     &          ,NX,NY,NZ,XL,YL,ZL,NDIAG,N,K)
C
      DIAG2=0
C
      DO I=1,NDIAG
C
      IF(NOFST(I).LT.0)THEN
      NBASE=-NOFST(I)
      LENGTH=N-NBASE
      DIAG2(1:LENGTH,I)=DIAG(NBASE+1:N,I)
      ELSE
      NBASE=NOFST(I)
      LENGTH=N-NBASE
      DIAG2(NBASE+1:N,I)=DIAG(1:LENGTH,I)
      ENDIF
C
      ENDDO
C
      NUMNZC=1
C
      DO J=1,N
      NTOPCFGC=1
C
      DO I=NDIAG,1,-1
C
      IF(DIAG2(J,I).NE.0.0D0)THEN
C
      NCOL=J-NOFST(I)
      C(NUMNZC)=DIAG2(J,I)
      NROWC(NUMNZC)=NCOL
C
      IF(NTOPCFGC.EQ.1)THEN
      NFCNZC(J)=NUMNZC
      NTOPCFGC=0
      ENDIF
```

```
C
      NUMNZC=NUMNZC+1
C
      ENDIF
      ENDDO
      ENDDO
C
      NFCNZC(N+1)=NUMNZC
      NNZC=NUMNZC-1
C
      CALL DM_VMVSCC(C,NNZC,NROWC,NFCNZC,N,SOLEX,
     $             B,WC,IWC,ICON)
C
C
      X=B
      IORDERING=0
      ISCLITERMAX=10
      ISW=1
      EPSZ=1.0D-16
      NSIZEFACTORL=1
      NSIZEFACTORU=1
      NSIZEINDEX=1
      THEPSZ=1.0D-2
      EPSR=1.0D-8
      SEPSZ=1.0D-10
      IPIVOT=40
      ISTATIC=1
      IREFINE=1
      ITERMAX=10
C
      CALL DM_VSSSS(C,NNZC,NROWC,NFCNZC,N,
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              X,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,DUMMYFL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NDUMMYIL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,DUMMYFU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NDUMMYIU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
     $              IPIVOT,ISTATIC,SEPSZ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              W,IW,ICON)
C
      PRINT*,'    ICON=',ICON,' NSIZEFACTORL=',NSIZEFACTORL,
     $       'NSIZEFACTORU=',NSIZEFACTORU,
     $       'NSIZEINDEX=',NSIZEINDEX
      PRINT*,'    NSUPNUM=',NSUPNUM
      PRINT *
C
      ALLOCATE( PANELFACTORL(NSIZEFACTORL) )
      ALLOCATE( PANELFACTORU(NSIZEFACTORU) )
      ALLOCATE( NPANELINDEXL(NSIZEINDEX) )
      ALLOCATE( NPANELINDEXU(NSIZEINDEX) )
C
      ISW=2
      CALL DM_VSSSS(C,NNZC,NROWC,NFCNZC,N,
```

```
     $              ISCLITERMAX,IORDERING,
     $              NPERM,ISW,
     $              X,
     $              NASSIGN,
     $              NSUPNUM,
     $              NFCNZFACTORL,PANELFACTORL,
     $              NSIZEFACTORL,NFCNZINDEXL,
     $              NPANELINDEXL,NSIZEINDEX,NDIM,
     $              NFCNZFACTORU,PANELFACTORU,
     $              NSIZEFACTORU,
     $              NFCNZINDEXU,NPANELINDEXU,
     $              NPOSTO,
     $              SCLROW,SCLCOL,
     $              EPSZ,
     $              THEPSZ,
     $              IPIVOT,ISTATIC,SEPSZ,
     $              IREFINE,EPSR,ITERMAX,ITER,
     $              W,IW,ICON)

C
      ERR = ERRNRM(SOLEX,X,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *
      PRINT *,'    ERROR = ',ERR
      PRINT *,'    ITER=',ITER
      PRINT *
      PRINT *

      IF(ERR.LT.1.0D-8.AND.ICON.EQ.0)THEN
         WRITE(*,*)'    ********** OK **********'
      ELSE
         WRITE(*,*)'    ********** NG **********'
      ENDIF

      DEALLOCATE( PANELFACTORL,PANELFACTORU,NPANELINDEXL,
     $            NPANELINDEXU )

      STOP
      END

C =======================================
C     INITIALIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &           ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
```

```
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      IF (NDIAG_LOC .GE. 7) THEN
        OFFSET(L) = -NXY
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 5) THEN
        OFFSET(L) = -NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 3) THEN
        OFFSET(L) = -1
        L = L+1
      ENDIF
      OFFSET(L) = 0
      L = L+1
      IF (NDIAG_LOC .GE. 2) THEN
        OFFSET(L) = 1
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 4) THEN
        OFFSET(L) = NX
        L = L+1
      ENDIF
      IF (NDIAG_LOC .GE. 6) THEN
        OFFSET(L) = NXY
      ENDIF
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
       PRINT*,'ERROR; K0.GH.NZ '
       GOTO 100
       ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =====================================
* SOLUTE ERROR
* | X1 - X2 |
C =====================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

The matrix is scaled in order to equilibrate both rows and columns norms. Subsequently the LU decomposition of this matrix is made. Nonzero elements belonging to each supernode is stored in two-dimensional panel respectively. The pivot for numerical stabilization is sought with in its block diagonal portion. The threshold for pivot search can be specified so that immediately after a pivot candidate with the absolute value greater than it is encountered in pivot search it is accepted as a pivot. In addition the static pivoting can be specified so that even if the pivot obtained after pivot search is considered as too small, it is replaced with the value of SPEPSZ and LU decomposition can be approximately performed.
Refer to references in Appendix A, “References.” in detail.
Refer to [19], [2], [22], [48], [68] on the LU decomposition of real sparse matrices and to [63], [69] on equilibration of matrices and pivoting.

DM_VTDEVC

Eigenvalues and eigenvectors of real tridiagonal matrices
CALL DM_VTDEVC (D, SL, SU, N, NF, NL, IVEC, ETOL, CTOL, NEV, E, MAXNE, EV, K, M, ICON)

(1) Function

This subroutine calculates specified eigenvalues and, optionally, eigenvectors of a real tridiagonal matrix.

$$\boldsymbol{Tx} = \lambda \boldsymbol{x} \tag{1.1}$$

where, $\boldsymbol{T}$ is an n-dimensional real tridiagonal matrix.

Tridiagonal matrix T must satisfy the following condition:

$$l_i \, u_{i\text{-}1} > 0, \text{ where, } i = 2, ..., n \tag{1.2}$$

When the element of tridiagonal matrix $\boldsymbol{T}$ is t_{ij}, d_i indicates a tridiagonal element, and $l_i = t_{i,i\text{-}1}$ and $u_i = t_{i,i+1}$ indicate subdiagonal elements, where, $l_1 = u_n = 0$.

$$(\boldsymbol{Tv})_i = l_i \, v_{i\text{-}1} + d_i \, v_i + u_i \, v_{i+1}, \quad i = 1,2,...,n \tag{1.3}$$

(2) Parameters

D ..... Input. The diagonal elements d_i are stored in real double-precision one-dimensional array D(N).

SL ..... Input. The subdiagonal elements l_i are stored in SL(2:N) of real double-precision one-dimensional array SL(N). SL(1) = 0.

SU ..... Input. The super-diagonal elements u_i are stored in SU(1:N-1) of real double-precision one-dimensional array SU(N). SU(N) = 0.

N ..... Input. Order n of tridiagonal matrix

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.

When the IVEC value is 1, the eigenvalues and corresponding eigenvectors are calculated.

When the IVEC value is not 1, only the eigenvalues are calculated.

ETOL ..... Input. Criterion value for checking whether the eigenvalues are numerically different from each other or are multiple. This check uses formula (3.4). When ETOL is less than 3.0D-16, this value is used as the standard value.

(See 2) in a, “Notes,” in (3), “Comments on use.”)

CTOL ..... Input. The CTOL value is used to check whether the adjacent eigenvalues are approximately equal to each other. This check uses formula (3.4).

1.0D-6 ≥ CTOL ≥ ETOL

When condition CTOL > 1.0D-6 occurs, CTOL is set to 1.0D-6.

When condition CTOL < ETOL occurs, CTOL = 10 × ETOL is set as the standard value.

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

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

Details are given below.

NEV(1) indicates the number of different eigenvalues calculated.

NEV(2) indicates the number of approximately multiple, different eigenvalues (clusters) calculated.

NEV(3) indicates the total number of eigenvalues (including multiple eigenvalues) calculated.

NEV(4) indicates the number representing the first of the eigenvalues calculated.

NEV(5) indicates the number representing the last of the eigenvalues calculated.

One-dimensional array NEV(5).

E ..... Output. Eigenvalues are stored in E.

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

One-dimensional array E(MAXNE).

MAXNE ..... Input. Maximum number of eigenvalues that can be calculated.

When it can be considered that there are two or more eigenvalues with multiplicity *m*, MAXNE must be set to a larger value than NL - NF + 1 + 2 × m that is bounded by *n*. Size of the dimension of array E.

When condition NEV(3) > MAXNE occurs, the eigenvectors cannot be calculated.

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

EV ..... Output. When IVEC = 1, the eigenvectors that correspond to the eigenvalues are stored in EV.

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

Two-dimensional array EV(K,MAXNE).

K ..... Input. Size of first-dimension of EV. (K ≥ N).

M ..... Output. Information about multiplicity of eigenvalues calculated.

M(*i*,l) indicates the multiplicity of the *i*-th eigenvalue λ_i. M(*i*,2) indicates the multiplicity of the *i*-th cluster calculated when the adjacent eigenvalues are regarded as approximately multiple eigenvalues (clusters).

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

Two-dimensional array M(MAXNE,2).

ICON ..... Output. Condition code.

See Table DM_VTDEVC-1.

Table DM_VTDEVC-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	During calculation of multiple eigenvalues, the total number of eigenvalues exceeded the MAXNE value.	Processing is discontinued. The eigenvectors cannot be calculated, but the different eigenvalues themselves are already calculated. A suitable value for MAXNE to allow calculation to proceed is returned in NEV(3). (See 3) in a, "Notes," in (3), "Comments on use.")
30000	N < 1, K < 1, NF < 1, NL > N, NL < NF, MAXNE < NL – NF + 1, or N > K.	Processing is discontinued.
30100	$SL(i) \times SU(i-1) \leq 0$. The matrix could not be converted into a symmetrical form.	

(3) Comments on use

a. Notes

1) Problems that can be solved using this function

This routine requires only that $l_i u_{i-1} > 0$, i=2,...,n. Thus it will also solve the generalized eigenvalue problem

$$\boldsymbol{Tx} = \lambda \boldsymbol{Dx} \tag{3.1}$$

where $\boldsymbol{D} > 0$ (every diagonal element is positive) is diagonal by setting $\boldsymbol{T} \leftarrow \boldsymbol{D}^{-1}\boldsymbol{T}$. Also, the eigenvalue problem for T can be reduced to a symmetric generalized problem

$$\boldsymbol{DTv} = \lambda \boldsymbol{Dv} \tag{3.2}$$

where $d_1 = 1$, $d_i = u_{i-1}d_{i-1}/l_i$, $i = 2, ..., n$. If d_i can cause scaling problems then it is preferable to consider the symmetric problem

$$\boldsymbol{D}^{1/2}\,\boldsymbol{TD}^{-1/2}\,\boldsymbol{w} = \lambda\,\boldsymbol{w} \tag{3.3}$$

where $\boldsymbol{w} = \boldsymbol{D}^{1/2}\boldsymbol{v}$.

2) This routine calculates eigenvalues independently from each other by dividing them into nonoverlapping, sequenced sets (parallel processing).

When ε = ETOL, the following condition is satisfied for consecutive eigenvalues λ_j ($j = s - 1, s, ..., s + k$ (k ≥ 0)):

$$\frac{\left|\lambda_i - \lambda_{i-1}\right|}{1 + \max(\left|\lambda_{i-1}\right|, \left|\lambda_i\right|)} \leq \varepsilon \tag{3.4}$$

If formula (3.4) is satisfied for i when $i = s, s + 1, ..., s + k$ but not satisfied when $i = s - 1$ and $i = s + k + 1$, it is assumed that the eigenvalues λ_j ($j = s - 1, s, ..., s + k$) are numerically multiple.

The standard value of ETOL is 3.0D-16 (about the unit round off). In this case, the eigenvalues are refined up to the maximum machine precision.

If formula (3.4) is not satisfied when ε = ETOL, it can be considered that λ_{i-1} and λ_i are distinct eigenvalues.

When ε = ETOL, assume that consecutive eigenvalues λ_m ($m = t - 1, t, ..., t + k$ ($k \geq 0$)) are different eigenvalues. Also, when ε = CTOL, assume that formula (3.4) is satisfied for i when $i = t, t + 1, ..., t + k$ but not satisfied when $i = t - 1$ and $i = t + k + 1$. In this case, it is assumed that the distinct eigenvalues λ_m ($m = t - 1, t, ..., t + k$) are approximately multiple (i.e. form a cluster). In this case, independent starting vectors are generated for inverse iteration, and eigenvectors corresponding to λ_m ($m = t - 1, t, \ldots, t + k$) are reorthogonilized.

3) The maximum number of eigenvalues that can be calculated is specified in MAXNE. When the value of CTOL is increased, the cluster size also increases. Therefore, the total number of eigenvalues calculated might exceed the value of MAXNE. In this case, decrease the value of CTOL or increase the value of MAXNE.

   If the total number of eigenvalues calculated exceeds the value of MAXNE, ICON = 20000 is returned. In this case, the eigenvectors cannot be calculated even if eigenvector calculation is specified. Eigenvalues are calculated, but are not stored repeatedly according to the multiplicity.

   The calculated different eigenvalues are stored in E(1:NEV(1)). The multiplicity of the corresponding eigenvalues is stored in M(1:NEV(1),1).

   When all the eigenvalues are different from each other and there are no approximately multiple eigenvalues, MAXNE can be set to NT (=NL–NF+1). However, when there are multiple eigenvalues and the multiplicity is m, MAXNE must be set to at least NT + 2 × m.

   If the total number of eigenvalues to be calculated exceeds the value of MAXNE, the value required to continue the calculation is returned in NEV(3). The calculation can be continued by allocating the area specified by this returned value and by calling the routine again.

b. Example

   This example calculates ne = nf – nl + 1 eigenvalues and corresponding eigenvectors of a model problem based on a modification to an example problem due to Wilkinson (see [81] in Appendix A, “References”). (This problem is known to have numerically multiple eigenvalues.)

   The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
C
      INTEGER K,N,N0,N1,NE,MAX_CLUS,MAX_NEV,NWR,P1,Q1,IVEC
      REAL*8 EVAL_TOL,CLUS_TOL
C
```

```
      PARAMETER (K=7001)
      PARAMETER (P1=70,Q1=100,N=P1*Q1,N0=6001,N1=7000,
     &           NE=N1-N0+1)
      PARAMETER (MAX_CLUS=2*Q1,MAX_NEV=NE+MAX_CLUS)
      PARAMETER (EVAL_TOL=3.D-16,CLUS_TOL=5.D-12)
      PARAMETER (NWR=2*N+2)
C
C
      REAL*8  A(N),B(N),C(N),EVAL(MAX_NEV),
     &        EVEC(K,MAX_NEV),WR(NWR)
      INTEGER MULT(MAX_NEV,2),NEV(3),ICON,I,J,II,L
      INTEGER N0X,N1X
C
C     W^+_n (Wilkinson): Pathologically close eigenvalues
C
      J = ( P1 + 1 ) / 2
      B(J) = 0.D0
      DO 40 I=1,J-1
         A(I+1) = 1.D0
         C(I) = 1.D0
         A(J+I) = 1.D0
         C(J+I-1) = 1.D0
         B(I) = DFLOAT( J - I )
   40    B(2*J-I) = B(I)
      A(1) = 0.D0
      C(P1) = 0.D0
      DO 45 L=2,Q1
         II = (L-1) * P1
         DO 45 I=1,P1
            A(II+I) = A(I)
            C(II+I) = C(I)
            B(II+I) = B(I)
   45 CONTINUE
C
      A(1)=0.D0
      C(N)=0.D0
C
      N0X=N0
      N1X=N1
      IVEC=1
C
      CALL DM_VTDEVC(B,A,C,N,N0X,N1X,IVEC,EVAL_TOL,CLUS_TOL,NEV,
     &               EVAL,MAX_NEV,EVEC,K,MULT,ICON)
C
      CALL CHECK(A,B,C,N,EVEC,K,EVAL,NEV,WR,WR(N+3))
C
      STOP
      END

      SUBROUTINE CHECK(SL,D,SU,N,EV,LD,E,NEV,W,W2)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION SU(*),D(*),SL(*),EV(LD,*),E(*),NEV(3),
     &          W(N+2),W2(N)
```

```
C
      TMP=0.0
      DO I=1,NEV(3)
C
      DO J=1,N
      W(J+1)=EV(J,I)
      ENDDO
      W(1)=0.0
      W(N+2)=0.0
      DO J=1,N
      W2(J)=SL(J)*W(J)+D(J)*W(J+1)+SU(J)*W(J+2)-E(I)*W(J+1)
      TMP=MAX(TMP,ABS(W2(J)/(ABS(E(I))+1)))
      ENDDO
      ENDDO
C
      PRINT*,'== maximum element error in ||T*x-eig*x||= ',
     &       TMP,' =='

      RETURN
      END
```

(4) Method

When each processor calculates eigenvalues by interval refinement the section, the Sturm sequence is calculated at roughly *npts/nev* points. (*npts* ≥ 4*MAXNE.) *nev* indicates the number of eigenvalues to be calculated.

The value of *npts* is determined as explained in [71] in Appendix A, “References.”

A composite data structure is used. An array structure is combined with a *last-in first-out* (LIFO) structure to maintain eigenvalue ordering and multisectioning. This is explained in [61] in Appendix A, “References.” This computation is carried out until the limit of section ETOL refinement is reached. When the standard value is set (3.0d-16), the precision of the eigenvalues approaches machine precision relative to the scale of the matrix.

For an explanation of the Sturm count, see [80] in Appendix A, “References.”

It has the property that the sign count is a monotonic function of the eigenvalue parameter in IEEE floating-point arithmetic. (See [20] in Appendix A, “References.”)

Eigenvectors are calculated by inverse iteration.

The initial vector is determined using the sign structure of the Sturm sequence, except when numerically multiple (or approximately multiple) eigenvalues have been detected.

When the eigenvalues are numerically or approximately multiple, random initial vectors are generated and orthogonalized with respect to other eigenvectors of the cluster. Usually, one step of inverse iteration suffices. The eigenvectors corresponding to the clustered eigenvalues are also reorthogonalized after inverse iteration.

DM_VTFQD

System of linear equations with unsymmetric or indefinite sparse matrices (TFQMR method, diagonal format storage method)
CALL DM_VTFQD(A, K, NDIAG, N, NOFST, B, ITMAX, EPS, IGUSS, X, ITER, ICON)

(1) Function

This subroutine solves, using the transpose-free quasi minimal residual [TFQMR] method, a system of linear equations with unsymmetric or indefinite sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the diagonal format storage method. 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, "Outline," in the SSL II Extended Capability User's Guide II.

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A.

The coefficient matrix is stored in A(1:N,1:NDIAG).

Two-dimensional array A(K,NDIAG)

For an explanation of the diagonal format storage method, see b, "Diagonal format storage method of general sparse matrices," in Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the SSL II Extended Capability User's Guide II.

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

NDIAG ..... Input. Number of columns in array A and size of array NOFST. Must be greater than or equal to the number of nonzero diagonals in matrix $\boldsymbol{A}$. Size of second-dimension of array A.

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

NOFST ..... Input. Offsets of diagonals of $\boldsymbol{A}$ stored A. Main diagonal has offset 0, subdiagonals have negative offsets, and superdiagonals have positive offsets.

One-dimensional array NOFST(NDIAG)

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N).

ITMAX ..... Input. Upper limit of iterative count for TFQMR method. The value of ITMAX should usually be set to about 2000.

EPS ..... Input. Criterion value for judgment of convergence.

When the value of EPS is 0.0 or smaller, EPS is set to 10^{-6}.

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

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VTFQD.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X.

One-dimensional array X(N).

ITER ..... Output. Actual iterative count for TFQMR method.

ICON ..... Output. Condition code.

See Table DM_VTFQD-1.

Table DM_VTFQD-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A breakdown state occurred.	Processing is discontinued.
20001	The iteration count reached the maximum limit.	Processing is discontinued. The already calculated approximate value is output to array X, but its precision is not assured.
30000	N < 1, N > K, NDIAG < 1, ITMAX ≤ 0.	Processing is discontinued.
32001	|NOFST(I)| > N – 1	

(3) Comments on use

a. Notes

1) When the residual Euclidean norm is equal to or smaller than the product of the first residual Euclidean norm and the value of EPS, it is assumed that the solution converged. The error between the correct solution and the calculated approximate solution is roughly equal to the product of the matrix ***A*** condition number and the value of EPS.

2) Conditions for using the diagonal format

The external diagonal vector element of coefficient matrix A must be set to 0. The order in which diagonal vectors (refer to Section 3.2.1.1, "Storage method for general sparse matrices" in the SSL II Extended Capabilities User's Guide II) are stored into array A is not restricted.

The merit of this method is that a matrix vectors can be calculated without using an indirect index. The demerit of this method is that a matrix without a diagonal structure cannot be stored efficiently.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

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

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Diagonal format is generated by the subroutine init_mat_diag.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (EPS = 1D-8)
      PARAMETER (NORD=60,NX = NORD,NY =NORD ,NZ = NORD,
     $      N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (NDIAG = 7)
      PARAMETER(NVW=3*K)

      DIMENSION NOFST(NDIAG)
      DIMENSION A(K,NDIAG)
      DIMENSION X(N),B(N),SOLEX(N),Y(N)
      DIMENSION VW(NVW)

      PRINT *,'    BICGSTAB(L) METHOD'
      PRINT *,'    DIAGONAL FORMAT'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      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)
      NBANDL=0
      NBANDR=0
      DO I=1,NDIAG
      IF(NOFST(I).LT.0)THEN
      NBANDL=MAX(NBANDL,-NOFST(I))
      ELSE
      NBANDR=MAX(NBANDR,NOFST(I))
      ENDIF
```

```
      ENDDO

      VW(1+NBANDL:N+NBANDL) = SOLEX(1:N)
      CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,B,ICON2)

      X(1:N)=0.0D0
      ERR1 = ERRNRM(SOLEX,X,N)
      VW(1+NBANDL:N+NBANDL) = X(1:N)
      CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,Y,ICON2)
      ERR2 = ERRNRM(Y,B,N)

      IGUSS = 0
      ITMAX = 2000

      CALL DM_VTFQD(A,K,NDIAG,N,NOFST,B,ITMAX
     &          ,EPS,IGUSS,X,ITER,ICON)

      ERR3 = ERRNRM(SOLEX,X,N)
      VW(1+NBANDL:N+NBANDL) = X(1:N)
      CALL DM_VMVSD(A,K,NDIAG,N,NOFST,NBANDL,VW,Y,ICON2)
      ERR4 = ERRNRM(Y,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
      PRINT *
      PRINT *,'    DM_VTFQD ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *,'    NBANDL = ',NBANDL,', NBANDR = ',NBANDR
      PRINT *,'    ITER MAX = ',ITMAX
      PRINT *,'    ITER = ',ITER
      PRINT *
      PRINT *,'    EPS = ',EPS
      PRINT *
      PRINT *,'    INITIAL ERROR = ',ERR1
      PRINT *,'    INITIAL RESIDUAL ERROR = ',ERR2
      PRINT *,'    CRITERIA RESIDUAL ERROR = ',ERR2*EPS
      PRINT *
      PRINT *,'    ERROR = ',ERR3
      PRINT *,'    RESIDUAL ERROR = ',ERR4
      PRINT *
      PRINT *

      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF

      STOP
      END

C =======================================
```

```
C     INITIALIZE COEFFICIENT MATRIX
C =====================================
      SUBROUTINE INIT_MAT_DIAG(VA1,VA2,VA3,VC,D_L,OFFSET
     &               ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION D_L(NDIVP,NDIAG)
      INTEGER   OFFSET(NDIAG)
C
      IF (NDIAG .LT. 1) THEN
        WRITE (*,*) 'SUBROUTINE INIT_MAT_DIAG:'
        WRITE (*,*) ' NDIAG SHOULD BE GREATER THAN OR EQUAL TO 1'
        RETURN
      ENDIF

!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+     SHARED(VA1,VA2,VA3,VC,D_L,OFFSET
!$OMP+      ,NX,NY,NZ,XL,YL,ZL,NDIAG,LEN,NDIVP)

C NDIAG CANNOT BE GREATER THAN 7
      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)

!$OMP DO
      DO I = 1,NDIVP
      DO J = 1,NDIAG
      D_L(I,J) = 0.0
      ENDDO
      ENDDO
!$OMP ENDDO

      NXY = NX*NY

C OFFSET SETTING
!$OMP SINGLE
      L = 1
      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
!$OMP END SINGLE

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        JS = J

C DECOMPOSE JS-1 = (K0-1)*NX*NY+(J0-1)*NX+I0-1
        K0 = (JS-1)/NXY+1
        IF (K0 .GT. NZ) THEN
    PRINT*,'ERROR; K0.GH.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C =====================================
* ABSOLUTE ERROR
* | X1 - X2 |
C =====================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

For an explanation of the TFQMR method, see [26] in Appendix A, "References."

DM_VTFQE

System of linear equations with unsymmetric or indefinite sparse matrices (TFQMR method, ELLPACK format storage method)
CALL DM_VTFQE(A, K, IWIDT, N, ICOL, B, ITMAX, EPS, IGUSS, X, ITER, ICON)

(1) Function

This subroutine solves, using the transpose-free quasi minimal residual [TFQMR] method, a system of linear equations with unsymmetric or indefinite sparse matrices as coefficient matrices.

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

The $n \times n$ coefficient matrix is stored using the ELLPACK format storage method. 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, "Outline," in the SSL II Extended Capability User's Guide II.

(2) Parameters

A ..... Input. The nonzero elements of a coefficient matrix are stored in A(1:N,1:IWIDT).

Two-dimensional array A(K,IWIDT)

For an explanation of the ELLPACK format storage method, see Section 3.2.1.1, "Storing the general sparse matrices," in Part I, "Outline," in the SSL II Extended Capability User's Guide II.

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

IWIDT ..... Input. Maximum number of row-vector-direction nonzero elements of coefficient matrix $\boldsymbol{A}$. Size of second-dimension of A and ICOL.

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

ICOL ..... Input. Column index used in ELLPACK format. Used to indicate to which column vector the corresponding element of A belongs.

Two-dimensional array ICOL(K,IWIDT)

B ..... Input. The right-side constant vectors of a system of linear equations are stored in B(1:N).

One-dimensional array B(N)

ITMAX ..... Input. Upper limit of iterative count for TFQMR method. The value of ITMAX should usually be set to about 2000.

EPS ..... Input. Criterion value for judgment of convergence.

When the value of EPS is 0.0 or smaller, EPS is set to 10^{-6}.

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

IGUSS ..... Input. Control information specifying whether iterative computation is to be performed using the approximate values of the solution vectors specified in array X.

When the value of IGUSS is 0, the approximate values of the solution vectors are not specified and set to zero by DM_VTFQE.

When the value of IGUSS is not 0, the iterative computation is performed using the approximate values of the solution vectors specified in array X.

X ..... Input. The approximate values of solution vectors can be specified in X(1:N).

Output. Solution vectors are stored in X(1:N).

One-dimensional array X(N)

ITER ..... Output. Iterative count for TFQMR method.

ICON ..... Output. Condition code.

See Table DM_VTFQE-1.

Table DM_VTFQE-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A breakdown state occurred.	Processing is discontinued.
20001	The iteration count reached the maximum limit.	Processing is discontinued. The already calculated approximate value is output to array X, but its precision is not assured.
30000	K < 1, IWIDT < 1, N < 1, ITMAX ≤ 0, N > K.	Processing is discontinued.
30001	The band width is zero.	

(3) Comments on use

a. Notes

1) When the residual Euclidean norm is equal to or smaller than the product of the first residual Euclidean norm and the EPS, it is assumed that the solution converged. The error between the correct solution and the calculated approximate solution is roughly equal to the product of the matrix A condition number and the EPS.

b. Example

The linear system of equations $\boldsymbol{Ax}=\boldsymbol{f}$ is solved, where $\boldsymbol{A}$ results from the finite difference method applied to the elliptic equation

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

with zero boundary conditions on a cube and the coefficient $a=(a_1,a_2,a_3)$ where a_1, a_2 and a_3 are some constants. The matrix $\boldsymbol{A}$ in Ellpack format is generated by the subroutine init_mat_ell.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT  REAL*8 (A-H,O-Z)
      PARAMETER (EPS = 1D-8)
      PARAMETER (NORD=60,NX =NORD ,NY = NORD,NZ = NORD,
     &           N = NX*NY*NZ)
      PARAMETER (K = N+1)
      PARAMETER (IWIDT = 7)
      DIMENSION ICOL(K,IWIDT)
      DIMENSION A(K,IWIDT)
      DIMENSION X(N),B(N),SOLEX(N),Y(N)

      PRINT *,'    BICGSTAB(L) METHOD'
      PRINT *,'    ELLPACK FORMAT'
      PRINT *

      SOLEX(1:N)=1.0D0
      PRINT *,'    EXPECTED SOLUTIONS'
      PRINT *,'    X(1) = ',SOLEX(1),' X(N) = ',SOLEX(N)
      PRINT *

      VA1 = 3D0
      VA2 = 1D0/3D0
      VA3 = 5D0
      VC = 1.0
      XL = 1.0
      YL = 1.0
      ZL = 1.0
      CALL INIT_MAT_ELL(VA1,VA2,VA3,VC,A,ICOL
     &          ,NX,NY,NZ,XL,YL,ZL,IWIDT,N,K)

      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,SOLEX,B,ICON2)

      X(1:N)=0.0D0
      ERR1 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,X,Y,ICON2)
      ERR2 = ERRNRM(Y,B,N)

      IGUSS = 0
      ITMAX = 2000

      CALL DM_VTFQE(A,K,IWIDT,N,ICOL,B,ITMAX
     &          ,EPS,IGUSS,X,ITER,ICON)

      ERR3 = ERRNRM(SOLEX,X,N)
      CALL DM_VMVSE(A,K,IWIDT,N,ICOL,X,Y,ICON2)
      ERR4 = ERRNRM(Y,B,N)

      PRINT *,'    COMPUTED VALUES'
      PRINT *,'    X(1) = ',X(1),' X(N) = ',X(N)
```

```
      PRINT *
      PRINT *,'    DM_VTFQE ICON = ',ICON
      PRINT *
      PRINT *,'    N = ',N,' :: NX = ',NX,' NY = ',NY,' NZ = ',NZ
      PRINT *,'    ITER MAX = ',ITMAX
      PRINT *,'    ITER = ',ITER
      PRINT *
      PRINT *,'    EPS = ',EPS
      PRINT *
      PRINT *,'    INITIAL ERROR = ',ERR1
      PRINT *,'    INITIAL RESIDUAL ERROR = ',ERR2
      PRINT *,'    CRITERIA RESIDUAL ERROR =',ERR2*EPS
      PRINT *
      PRINT *,'    ERROR = ',ERR3
      PRINT *,'    RESIDUAL ERROR = ',ERR4
      PRINT *
      PRINT *

      IF(ERR4.LE.ERR2*EPS*1.1.AND.ICON.EQ.0)THEN
         WRITE(*,*)'********** OK **********'
      ELSE
         WRITE(*,*)'********** NG **********'
      ENDIF

      STOP
      END

C =======================================
C INITILIZE COEFFICIENT MATRIX
C =======================================
      SUBROUTINE INIT_MAT_ELL(VA1,VA2,VA3,VC,A_L,ICOL_L,NX,NY,NZ
     &          ,XL,YL,ZL,IWIDTH,LEN,NDIVP)
      IMPLICIT  REAL*8(A-H,O-Z)
      DIMENSION A_L(NDIVP,IWIDTH)
      DIMENSION ICOL_L(NDIVP,IWIDTH)
C
      IF (IWIDTH .LT. 1) THEN
         WRITE (*,*) 'SUBROUTINE INIT_MAT_ELL:'
         WRITE (*,*) ' IWIDTH SHOULD BE GREATER THAN OR EQUAL TO 1'
         RETURN
      ENDIF
!$OMP PARALLEL DEFAULT(PRIVATE)
!$OMP+  SHARED(VA1,VA2,VA3,VC,A_L,ICOL_L,NX,NY,NZ
!$OMP+        ,XL,YL,ZL,IWIDTH,LEN,NDIVP)

C IWIDTH CANNOT BE GREATER THAN 7
      IWIDTH_LOC = IWIDTH
      IF (IWIDTH .GT. 7) IWIDTH_LOC = 7

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

```
!$OMP DO
      DO J = 1,IWIDTH
      DO I = 1,NDIVP
      A_L(I,J) = 0.0
      ICOL_L(I,J) = I
      ENDDO
      ENDDO
!$OMP ENDDO

C MAIN LOOP
!$OMP DO
      DO 100 J = 1,LEN
        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) THEN
    PRINT*,' ERROR; K0.GT.NZ '
    GOTO 100
    ENDIF
        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
 100  CONTINUE
!$OMP ENDDO

!$OMP END PARALLEL

      RETURN
      END

C ======================================
C ABSOLUTE ERROR
C | X1 - X2 |
C ======================================
      REAL*8 FUNCTION ERRNRM(X1,X2,LEN)
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X1(*),X2(*)
C
      S = 0D0
      DO 100 I = 1,LEN
        SS = X1(I) - X2(I)
        S = S + SS * SS
 100  CONTINUE
C
      ERRNRM = SQRT( S )
      RETURN
      END
```

(4) Method

For an explanation of the TFQMR method, see [26] in Appendix A, "References."

DM_VTRID

Tridiagonalization of real symmetric matrices.
CALL DM_VTRID (A, K, N, D, SL, ICON)

(1) Function

This subroutine reduces the real symmetric matrix $\boldsymbol{A}$ to tridiagonal form using the Housholder reductions.

$$\boldsymbol{T} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Q}$$

where $\boldsymbol{A}$ is an $n \times n$ real symmetric matrix, $\boldsymbol{Q}$ is an $n \times n$ othogonal matrix and $\boldsymbol{T}$ is a real tridiagonal matrix.

(2) Parameters

A ..... Input. The lower triangular part $\{a_{ij} \mid i \geq j\}$ of real symmetric matrix $\boldsymbol{A}$ is stored in the lower triangular part $\{\mathrm{A}(i,j) \mid i \geq j\}$ of A(1:N,1:N).

Output. The information on Householder transforms used for tridiagonalization in stored in the lower triangular part $\{\mathrm{A}(i,j) \mid i \geq j\}$ of A(1:N,1:N)

After calculation, the values in the upper triangular part of A is not assured.

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

Two-dimensional double-precision real array A(K,N).

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

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

D ..... Input. The diagonal elements of the reduced tridiagonal matrix are stored in real double-precision one-dimensional array D(N).

SL ..... Input. The subdiagonal elements of reduced tridiagonal matrix are stored in SL(2:N) of real double-precision one-dimensional array SL(N). SL(1) = 0.

ICON ..... Output. Condition code.

See Table DM_VTRID-1.

Table DM_VTRID-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N < 2, K < N.	Processing is discontinued.

(3) Comments on use

a. Notes

1) Tridiagonalization is performed by the reapeated transforms varying $k = 1, \ldots, n-2$.

$$\boldsymbol{A}^{k} = \boldsymbol{Q}_{k}^{\mathrm{T}}\boldsymbol{A}^{k-1}\boldsymbol{Q}_{k}, \quad \boldsymbol{A}^{0} = \boldsymbol{A}$$

Put $\boldsymbol{b}^{\mathrm{T}} = (0, \ldots, 0, A^{k-1}(k{+}1{:}n, k)^{\mathrm{T}})$.

$\boldsymbol{b}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}, \ldots, b_n)$

$\boldsymbol{b}^{\mathrm{T}} \cdot \boldsymbol{b} = S^2$ and put $\boldsymbol{w}^{\mathrm{T}} = (0, \ldots, 0, b_{k+1}+S, b_{k+2}, \ldots, b_n)$.

The sign of S is chosen same as that of b_{k+1}.

Then the transform matrix is represented as follow.

$$\boldsymbol{Q}_k = \mathrm{I} - \alpha \boldsymbol{w} \cdot \boldsymbol{w}^{\mathrm{T}}, \alpha = \frac{1}{S^2 + |b_{k+i}S|}$$

$\boldsymbol{w}(k{+}1{:}n)$ and α are stored in $\boldsymbol{A}(k{+}1{:}n, k)$ and $\boldsymbol{A}(k, k)$ respectively.

b. Example

This example calculates the tridiagonalization of a real symmetric matrix whose eigenvalues are known.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
c     **example**
      implicit real*8(a-h,o-z)
      parameter(n=2000,k=n)
      parameter(ne=n,max_nev=ne)
      dimension a(k,n),b(k,n),c(k,n),d(k,n),ac(k,n)
      dimension dd(n),sld(n),sud(n)
      dimension nev(5),mult(max_nev,2)
      dimension eval(max_nev),evec(k,max_nev)
cc
      pai=4.0d0*datan(1.0d0)
      coef=dsqrt(2.0d0/(n+1))
      do j=1,n
      do i=1,n
      d(i,j)=coef*dsin(pai/(n+1)*i*j)
      enddo
      enddo
cc
      do j=1,n
      do i=1,n
      if(i.eq.j)then
      c(i,j)=i
      else
      c(i,j)=0.0d0
      endif
      enddo
      enddo
cc
cc    d x c -> b
cc
      call dm_vmggm(d,k,c,k,b,k,n,n,n,icon)
cc
cc    b x d -> a
```

```
cc
      call dm_vmggm(b,k,d,k,a,k,n,n,n,icon)
cc
      do i=1,n
      do j=i,n
      ac(j,i)=a(j,i)
      enddo
      enddo
c
      call dm_vtrid( ac,k,n,dd,sld,icon )
      if(icon.ne.0)then
      print*,' icon of dm_vtrid =',icon
      stop
      endif
c
      do i=2,n
      sud(i-1)=sld(i)
      enddo
      sud(n)=0.0d0
c
      nf=1
      nl=n
      ivec=0
      eval_tol=1.0d-15
      clus_tol=1.0d-10
      call dm_vtdevc( dd,sld,sud,n,nf,nl,ivec,
     &                eval_tol,clus_tol,nev,
     &                eval,max_nev,evec,k,mult,icon )
      do i=1,ne,n/20
      print*,'eigen value in eval(',i,') = ',eval(i)
      enddo
c
      stop
      end
```

(4) Method

This routine reduces a tridiagonal matrix from a real symmetric matrix. The reduction to a tridiagonal form is a parallel version of the Householder reduction to tridiagonal form. (See [30] in Appendix A, “References.”)

DM_V1DCFT

One-dimensional discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V1DCFT(X,KX,Y,KY,N1,N2,ISN,ICON)

(1) Function

The subroutine DM_V1DCFT performs a one-dimensional complex Fourier transform or its inverse transform using a mixed radix FFT.

The length of data transformed $n(=n_1 \times n_2)$ is a product of the powers of 2, 3, 5 and 7.

a. The one-dimensional Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, k = 0, 1, \ldots, n-1 \quad , \omega_n = \exp(2\pi i/n) \tag{1.1}$$

b. The one-dimensional Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, j = 0, 1, \ldots, n-1 \quad , \omega_n = \exp(2\pi i/n) \tag{1.2}$$

It is recommended to use DVCFM1 in "SSL II Extended Capabilities User's Guide II" when the length of data is not large enough.

(2) Parameters

X ..... Input. The complex data. The data is stored in X(1:N1,1:N2).

See Figure DM_V1DCFT-1.

This is a double precision complex two-dimensional array X(KX,N2).

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

KX ..... Input. The size of the first dimension of array X (≥ N1).

Y ..... Output. The complex transformed data. The data is stored in Y(1:N2,1:N1). See Figure DM_V1DCFT-1.

This is a double precision complex two-dimensional array Y(KY,N1).

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

KY ..... Input. The size of the first dimension of array Y (≥ N2).

N1 ..... Input. Assuming that the length of the data transformed (n=N1 × N2) is two-dimensional data, the size of first dimension N1 must be a product of the powers of 2, 3, 5 and 7.

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

N2 ..... Input. Assuming that the length of the data transformed (n=N1 × N2) is two-dimensional data, the size of the second dimension, N2 must be a product of the powers of 2, 3, 5 and 7.

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

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform

ISN = -1 for the inverse transform

ICON ..... Output. Condition code.

See Table DM_V1DCFT-1.

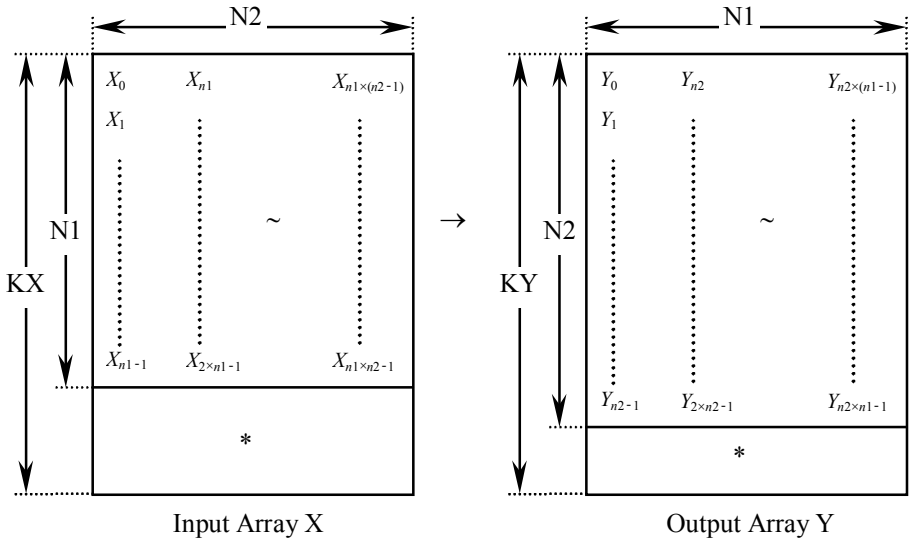


Figure DM_V1DCFT-1 The input/Output data storage method

Table DM_V1DCFT-1 Condition codes

Code	Meaning	Processing
0	No error	–
30001	The dimensions of arrays less than or equal to 0	Processing is discontinued
30002	The leading dimensions are less than the actual dimensions.	
30008	The order of transform is not radix 2/3/5/7.	
30016	The invalid value for the parameter ISN	

(3) Comments on use

a. Notes

1) If the one-dimensional data of $n = n_1 \times n_2$ is numbered $k = 0, ..., n-1$,

$k = k_1 + k_2 \times n_1$, $k_1 = 0, ..., n_1-1$

, $k_2 = 0, ..., n_2-1$

$i = i_1 + i_2 \times n_2$, $i_1 = 0, ..., n_2-1$

, $i_2 = 0, ..., n_1-1$

The input and output data are regarded as two-dimensional arrays with subscripts of (k_1, k_2) and (i_1, i_2), respectively. (See Figure DM_V1DCFT-1.)

2) General definition of a Fourier transform

The one-dimensional discrete complex Fourier transform and its inverse transform is defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0, 1, ..., n-1 \tag{3.1}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, \quad j = 0, 1, ..., n-1 \tag{3.2}$$

where, $\omega_n = \exp(2\pi i/n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A one-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N1=4000,N2=3000)
      PARAMETER (KX=N1+1,KY=N2+1)
      COMPLEX*16 X(KX,N2),Y(KY,N1)
      INTEGER ISN
*
*     Set up the input data arrays
*
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
      DO I=1,N2
      DO J=1,N1
      X(J,I)=DCMPLX(FLOAT(J)+FLOAT(N1)*(I-1),0.0)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO
*
*     Do the forward transform
*
      ISN=1
      CALL DM_V1DCFT(X,KX,Y,KY,N1,N2,ISN,ICON)
      IF(ICON.NE.0) THEN
        WRITE(*,*) 'error occurred : ',ICON
      ENDIF
*
*     Do the reverse transform
```

```
*
      ISN=-1
      CALL DM_V1DCFT(Y,KY,X,KX,N2,N1,ISN,ICON)
      IF(ICON.NE.0) THEN
        WRITE(*,*) 'error occurred : ',ICON
      ENDIF
*
*     Find the error after the forward and
*     inverse transform.
*
      ERROR=0

!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
!$OMP+         REDUCTION(MAX:ERROR)
      DO I=1,N2
      DO J=1,N1
        ERROR=MAX(ABS(DBLE(X(J,I))/N2/N1)-
     &  (FLOAT(J)+FLOAT(N1)*(I-1)),ERROR)
        ERROR=MAX(ABS(DIMAG(X(J,I))/N2/N1),
     &                              ERROR)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO

      WRITE(*,*) 'Error ', ERROR
      STOP
      END
```

(4) Method

DM_V1DCFT is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V1DCFT2

One-dimensional discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V1DCFT2(X,N,Y,ISN,ICON)

(1) Function

The subroutine DM_V1DCFT performs a one-dimensional complex Fourier transform or its inverse transform using a mixed radix FFT.

The length of data transformed *n* is a product of the powers of 2, 3, 5 and 7.

a. The one-dimensional Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,...,\ n-1 \qquad (1.1)$$
$$,\ \omega_n = \exp(2\pi i/n)$$

b. The one-dimensional Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,...,\ n-1 \qquad (1.2)$$
$$,\ \omega_n = \exp(2\pi i/n)$$

It is recommended to use DVCFM1 in "SSL II Extended Capabilities User's Guide II" when the length of data is not large enough.

(2) Parameters

X..... Input. Complex data is stored in X(1:N).

This is a double precision complex one-dimensional array X(N).

N ..... Input. The length of the data transformed. N must be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

Y ..... Output. Transformed complex data is stored in Y(1:N).

This is a double precision complex one-dimensional array Y(N).

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform

ISN = -1 for the inverse transform

Integer (INTEGER*4)

ICON ..... Output. Condition code.

See Table DM_V1DCFT2-1.

Table DM_V1DCFT2-1 Condition codes

Code	Meaning	Processing
0	No error	–
30008	The order of transform is not radix 2/3/5/7.	Processing is discontinued
30016	The invalid notation parameter ISN	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The one-dimensional discrete complex Fourier transform and its inverse transform is defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \qquad (3.2)$$

where, $\omega_n = \exp(2\pi i/n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A one-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N1=1024,N2=N1,N=N1*N2)
      COMPLEX*16 X(N),Y(N),XX(N)
C
      DO I=1,N
      X(I)=DBLE(I)
      XX(I)=X(I)
      ENDDO
C
      CALL DM_V1DCFT2(X,N,Y,1,ICON)
      PRINT*,'ICON =',ICON
C
      CALL DM_V1DCFT2(Y,N,X,-1,ICON)
      PRINT*,'ICON =',ICON
C
      TMP=0.0D0
```

```
      DO I=1,N
      TMP=MAX(ABS(X(I)/DBLE(N)-XX(I)),TMP)
      ENDDO
      PRINT*,' ERROR =',TMP
C
      STOP
      END
```

(4) Method

DM_V1DCFT2 is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V1DMCFT

One-dimensional multiple discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V1DMCFT(X,KX,N,M,ISN,ICON)

(1) Function

The subroutine DM_V1DMCFT performs multiple one-dimensional complex Fourier transforms or its inverse transforms using a mixed radix FFT.

The length of data transformed *n* is a product of the powers of 2, 3, 5 and 7.

a. The one-dimensional Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jk}, k = 0, 1, ..., n-1 \quad , \omega_n = \exp(2\pi i/n) \tag{1.1}$$

b. The one-dimensional Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, j = 0, 1, ..., n-1 \quad , \omega_n = \exp(2\pi i/n) \tag{1.2}$$

(2) Parameters

X ..... Input. The complex data. Store the data in X(1:N,1:M).

Output. The complex transformed data. The data is stored in X(1:N,1:M).

This is a double precision complex two-dimensional array X(KX,M).

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

KX ..... Input. The size of the first dimension of array X (≥ N).

N ..... Input. The length of the data transformed must be a product of the powers of 2, 3, 5 and 7.

M ..... Input. The multiplicity of the data transformed.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform

ISN = -1 for the inverse transform

ICON ..... Output. Condition code.

See Table DM_V1DMCFT-1.

Table DM_V1DMCFT-1 Condition codes

Code	Meaning	Processing
0	No error	–
30001	The dimensions of arrays less than or equal to 0	Processing is discontinued
30002	The leading dimensions are less than the actual dimensions.	
30008	The order of transform is not radix 2/3/5./7	
30016	The invalid value for the parameter ISN	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The one-dimensional discrete complex Fourier transform and its inverse transform is defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk}, \quad k = 0,\ 1,\ldots,\ n-1 \tag{3.1}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk}, \quad j = 0,\ 1,\ldots, n-1 \tag{3.2}$$

where, $\omega_n = \exp(2\pi i/n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

Multiple one-dimensional FFTs are computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N1=2048,M=256)
      PARAMETER (KX=N1+1)
      COMPLEX*16 X(KX,M)
      INTEGER ISN

*
*     Set up the input data arrays
*
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
      DO I=1,M
      DO J=1,N1
      X(J,I)=dcmplx(FLOAT(J)+FLOAT(N1)*(I-1),0.0)
      ENDDO
```

```
      ENDDO
!$OMP END PARALLEL DO

*
*     Do the forward transform
*
      ISN=1
      CALL dm_v1dmcft(X,KX,N1,M,ISN,ICON)
      IF(ICON.NE.0) THEN
        WRITE(*,*) 'error occurred : ',ICON
      ENDIF
*
*     Do the reverse transform
*
       ISN=-1
       CALL dm_v1dmcft(X,KX,N1,M,ISN,ICON)
       IF(ICON.NE.0) THEN
         WRITE(*,*) 'error occurred : ',ICON
       ENDIF
*
*     Find the error after the forward and
*     inverse transform.
*
      ERROR=0

!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
!$OMP+         REDUCTION(MAX:ERROR)
      DO I=1,M
      DO J=1,N1
      ERROR=MAX(ABS(dble(X(J,I))/N1-
     &  (FLOAT(J)+FLOAT(N1)*(I-1))),ERROR)
      ERROR=MAX(ABS(dimag(X(J,I))/N1),
     &                             ERROR)
      ENDDO
      ENDDO
!$OMP END PARALLEL DO

      WRITE(*,*) 'Error ', ERROR
      STOP
      END
```

(4) Method

DM_V1DMCFT is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V2DCFT

Two-dimensional discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V2DCFT(X,KX,N1,N2,ISN,ICON)

(1) Function

The subroutine DM_V2DCFT performs a two-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of two-dimensional data (n_1, n_2) is a product of the powers of 2, 3, 5 and 7.

a. The two-dimensional Fourier transform

When $\{x_{j1j2}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1 n_2 \alpha_{k1k2}\}$.

$$n_1 n_2 \alpha_{k1k2} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2}$$
$$, k_1 = 0, 1, ..., n_1 - 1$$
$$, k_2 = 0, 1, ..., n_2 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp(2\pi i/n_1)$$
$$, \omega_{n2} = \exp(2\pi i/n_2)$$

b. The two-dimensional Fourier inverse transform

When $\{\alpha_{k1k2}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2}\}$.

$$x_{j1j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2}$$
$$, j_1 = 0, 1, ..., n_1 - 1$$
$$, j_2 = 0, 1, ..., n_2 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp(2\pi i/n_1)$$
$$, \omega_{n2} = \exp(2\pi i/n_2)$$

(2) Parameters

X ..... Input. The complex data.

The data is stored in X(1:N1,1:N2).

Output. The transformed complex data.

The results are stored in X(1:N1,1:N2).

This is a double precision complex two-dimensional array X(KX,N2).

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

KX ..... Input. The size of the first dimension of input data array X.

N1 ..... Input. The size n_1 of data in the first dimension of the two-dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

N2 ..... Input. The size n_2 of data in the second dimension of the two-dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

ICON ..... Output. Condition code.

See Table DM_V2DCFT-1.

Table DM_V2DCFT-1 Condition codes

Code	Meaning	Processing
0	No error	–
30001	The dimensions of arrays less than or equal to 0	Processing is discontinued
30002	The leading dimensions are less than the actual dimensions.	
30008	The order of transform is not radix 2/3/5/7.	
30016	The invalid value for the parameter ISN	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The two-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2} = \frac{1}{n_1 n_2} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \quad , k_1 = 0, 1, ..., n_1 - 1 \quad , k_2 = 0, 1, ..., n_2 - 1 \tag{3.1}$$

$$x_{j1j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j2k2} \omega_{n2}^{j2k2} \quad , j_1 = 0, 1, ..., n_1 - 1 \quad , j_2 = 0, 1, ..., n_2 - 1 \tag{3.2}$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$

This subroutine calculates $\{n_1 n_2 \alpha_{k1k2}\}$ or $\{x_{j1j2}\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A two-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8 (A-H,O-Z)
      PARAMETER (N1=4000,N2=3000)
      PARAMETER (KX=4400)
      COMPLEX*16 X(KX,N2)
      INTEGER ISN

*
*      Set up the input data arrays
*
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
       DO I=1,N2
       DO J=1,N1
       X(J,I)=DCMPLX(FLOAT(J)+FLOAT(N1)*(I-1),0.0)
       ENDDO
       ENDDO
!$OMP END PARALLEL DO

*
*     Do the forward transform
*
       ISN=1
       CALL DM_V2DCFT(X,KX,N1,N2,ISN,ICON)
       IF(ICON.NE.0) THEN
         WRITE(*,*) 'error occurred : ',ICON
       ENDIF
*
*     Do the reverse transform
*
       ISN=-1
       CALL DM_V2DCFT(X,KX,N1,N2,ISN,ICON)
       IF(ICON.NE.0) THEN
         WRITE(*,*) 'error occurred : ',ICON
       ENDIF
*
*     Find the error after the forward and
*     inverse transform.
*
      ERROR=0

!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
!$OMP+         REDUCTION(MAX:ERROR)
      DO I=1,N2
      DO J=1,N1
        ERROR=MAX(ABS(DBLE(X(J,I))/(N2*N1)-
     &  (FLOAT(J)+FLOAT(N1)*(I-1))),ERROR)
        ERROR=MAX(ABS(DIMAG(X(J,I))/(N2*N1)),
     &                             ERROR)
        ENDDO
```

```
       ENDDO
!$OMP END PARALLEL DO

      WRITE(*,*) 'Error ', ERROR
      STOP
      END
```

(4) Method

DM_V2DCFT is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V3DCFT

Three-dimensional discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V3DCFT(X,KX,N1,N2,N3,ISN,ICON)

(1) Function

The subroutine DM_V3DCFT performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, k_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, j_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, j_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$

(2) Parameters

X ..... Input. The complex data.

Data is stored in X(1:N1,1:N2,1:N3).

Output. The transformed complex data.

The results are stored in X(1:N1,1:N2,1:N3).

This is a double precision complex three-dimensional array X(KX,N2,N3).

KX ..... Input. The size of the first dimension of input data arrays X (≥ N1).

N1 ..... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

N2 ..... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

N3 ..... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

ICON ..... Output. Condition code.

See Table DM_V3DCFT-1.

Table DM_V3DCFT-1 Condition codes

Code	Meaning	Processing
0	No error	–
30001	The dimensions of arrays less than or equal to 0	Processing is discontinued
30002	The leading dimensions are less than the actual dimensions.	
30008	The order of transform is not radix 2/3/5/7.	
30016	The invalid value for the parameter ISN	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad (3.1)$$
$$, k_1 = 0, 1, ..., n_1 - 1$$
$$, k_2 = 0, 1, ..., n_2 - 1$$
$$, k_3 = 0, 1, ..., n_3 - 1$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad (3.2)$$
$$, j_1 = 0, 1, ..., n_1 - 1$$
$$, j_2 = 0, 1, ..., n_2 - 1$$
$$, j_3 = 0, 1, ..., n_3 - 1$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A three-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C      **EXAMPLE**
       IMPLICIT REAL*8 (A-H,O-Z)
       PARAMETER (N1=400,N2=100,N3=200)
       PARAMETER (KX=440)
       COMPLEX*16 X(KX,N2,N3)
       INTEGER ISN
*
*      Set up the input data arrays
*
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
       DO K=1,N3
       DO I=1,N2
       DO J=1,N1
       X(J,I,K)=DCMPLX(FLOAT(J)+FLOAT(N1)*(I-1),0.0)
       ENDDO
       ENDDO
       ENDDO
!$OMP END PARALLEL DO

*
*      Do the forward transform
*
       ISN=1
       CALL DM_V3DCFT(X,KX,N1,N2,N3,ISN,ICON)
       IF(ICON.NE.0) THEN
         WRITE(*,*) 'error occurred : ',ICON
       ENDIF
*
*     Do the reverse transform
*
       ISN=-1
       CALL DM_V3DCFT(X,KX,N1,N2,N3,ISN,ICON)
       IF(ICON.NE.0) THEN
         WRITE(*,*) 'error occurred : ',ICON
       ENDIF
*
*     Find the error after the forward and
*     inverse transform.
*
       ERROR=0
```

```
!$OMP PARALLEL DO DEFAULT(PRIVATE) SHARED(X)
!$OMP+         REDUCTION(MAX:ERROR)

      DO K=1,N3
      DO I=1,N2
      DO J=1,N1
      ERROR=MAX(ABS(DBLE(X(J,I,K))/(N3*N2*N1)-
     &      (FLOAT(J)+FLOAT(N1)*(I-1))),ERROR)
      ERROR=MAX(ABS(DIMAG(X(J,I,K))/(N3*N2*N1)),
     &                           ERROR)
      ENDDO
      ENDDO
      ENDDO
!$OMP END PARALLEL DO

      WRITE(*,*) 'Error ', ERROR
      STOP
      END
```

(4) Method

DM_V3DCFT is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V3DCFT2

Three-dimensional discrete complex Fourier transforms (mixed radices of 2, 3, 5 and 7)
CALL DM_V3DCFT2(X,K1,K2,N1,N2,N3,ISN,ICON)

(1) Function

The subroutine DM_V3DCFT2 performs a three-dimensional complex Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of three-dimensional arrays (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1}\omega_{n2}^{-j2k2}\omega_{n3}^{-j3k3}$$
$$, k_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, k_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1}\omega_{n2}^{j2k2}\omega_{n3}^{j3k3}$$
$$, j_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, j_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, j_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$

(2) Parameters

X ..... Input. The complex data.

Data is stored in X(1:N1,1:N2,1:N3).

Output. The transformed complex data.

The results are stored in X(1:N1,1:N2,1:N3).

This is a double precision complex three-dimensional array X(K1,N2,N3).

K1 ..... Input. The size of the first dimension of input data arrays X (≥ N1).

K2 ..... Input. The size of the second dimension of input data arrays X (≥ N2).

N1 ..... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

N2 ..... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

N3 ..... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

ICON ..... Output. Condition code.

See Table DM_V3DCFT2-1.

Table DM_V3DCFT2-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	n_1, n_2 or n_3 less than or equal to 0, or K1<N1, or K2<N2, or invalid value for the parameter ISN.	Processing is discontinued
30008	The order of transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad , k_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , k_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , k_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.1}$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad , j_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , j_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , j_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.2}$$

where, $\omega_{n1} = \exp(2\pi i/n_1)$, $\omega_{n2} = \exp(2\pi i/n_2)$,

$\omega_{n3} = \exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A three-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
c     **example**
      implicit real*8 (a-h,o-z)
      parameter (n1=128,n2=128,n3=128)
      parameter (k1=n1+1,k2=n2)
      complex*16 x(k1,k2,n3)
      integer isn
*
*     set up the input data arrays
*
!$omp parallel do default(private) shared(x)
      do k=1,n3
      do i=1,n2
      do j=1,n1
      x(j,i,k)=dcmplx(float(j)+float(n1)*(i-1),0.0)
      enddo
      enddo
      enddo
!$omp end parallel do

*
*     do the forward transform
*
      isn=1
      call dm_v3dcft2(x,k1,k2,n1,n2,n3,isn,icon)
      if(icon.ne.0) then
        write(*,*) 'error occurred : ',icon
      endif
*
*    do the reverse transform
*
      isn=-1
      call dm_v3dcft2(x,k1,k2,n1,n2,n3,isn,icon)
      if(icon.ne.0) then
        write(*,*) 'error occurred : ',icon
      endif
*
*    find the error after the forward and
*    inverse transform.
*
      error=0
```

```
!$omp parallel do default(private) shared(x)
!$omp+         reduction(max:error)

      do k=1,n3
      do i=1,n2
      do j=1,n1
      error=max(abs(dble(x(j,i,k))/(n3*n2*n1)-
     &      (float(j)+float(n1)*(i-1))),error)
      error=max(abs(dimag(x(j,i,k))/(n3*n2*n1)),
     &                           error)
      enddo
      enddo
      enddo
!$omp end parallel do

      write(*,*) 'error=', error
      stop
      end
```

(4) Method

DM_V3DCFT2 is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V1DRCF

One-dimensional discrete real Fourier transform (mixed radix of 2, 3, 5 and 7)
CALL DM_V1DRCF(X,KX,Y,KY,N1,N2,ISIN,ISN,ICON)

(1) Function

The subroutine DM_V1DRCF performs a one-dimensional real Fourier transform or its inverse transform using a mixed radix FFT.

The data count n ($=n_1 \times n_2$) is a product of the powers of 2, 3, 5 and 7.

a. One-dimensional Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jkr},\ k = 0,\ 1,\ ...,\ n-1 \quad , \ \omega_n = \exp\ (2\pi i/n) \tag{1.1}$$

$r = 1$ or $r = -1$

b. One-dimensional Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jkr},\ j = 0,\ 1,\ ...,\ n-1 \quad , \ \omega_n = \exp\ (2\pi i/n) \tag{1.2}$$

$r = 1$ or $r = -1$

This routine can perform about 30% faster than DM_V1DRCF2 or more, provided that the n is factorized into n_1 and n_2 appropriately.

(2) Parameters

X ..... Input/output. Real data. is stored in X(1:N1,1:N2).

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = –1), data is output. For ISN = 1, the input data is not saved.

This is a double precision real two-dimensional array X(KX,N2).

See Figure DM_V1DRCF-1.

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

KX ..... Input. The size of the first dimension of array X (≥ N1).

Integer (INTEGER*4)

Y ..... Output/input. Transformed complex data.

Data is stored in Y(1:N2/2 + 1,1:N1).

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = –1), data is input.

The input data is not guaranteed when ISN = –1.

The complex data obtained from real data by Fourier transformation has the conjugate complex relation. About half data is stored.

This is a double precision complex two-dimensional array Y(KY,N1).

(See note 3), (3) "Comments on use" and Figure DM_V1DRCF-1.)

KY ..... Input. The size of the first dimension of arrays Y (KY ≥ N2/2 + 1).

Integer (INTEGER*4).

N1 ..... Input. The size of the first dimension assuming that the real data to be transformed ($n = n1 \times n2$) is two-dimensional data.

N1 must be a product of the powers of 2, 3, 5 and 7.

N1*N2 must be the length of the data sequence to be transformed.

Integer (INTEGER*4).

(See note 1),4) in (3), "Comments on use.")

N2 ..... Input. The size of the second dimension assuming that the real data to be transformed ($n = n1 \times n2$) is two-dimensional data.

N2 must be a product of the powers of 2, 3, 5 and 7.

N1*N2 must be the length of the data sequence to be transformed.

Integer (INTEGER*4).

(See note 1),4) in (3), "Comments on use.")

ISIN ..... Input. The direction of transformation.

ISIN=1 for $r = 1$.

ISIN=–1 for $r = -1$.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = –1 for the inverse transform.

Integer (INTEGER*4).

ICON ..... Output. Condition code.

See Table DM_V1DRCF-1.

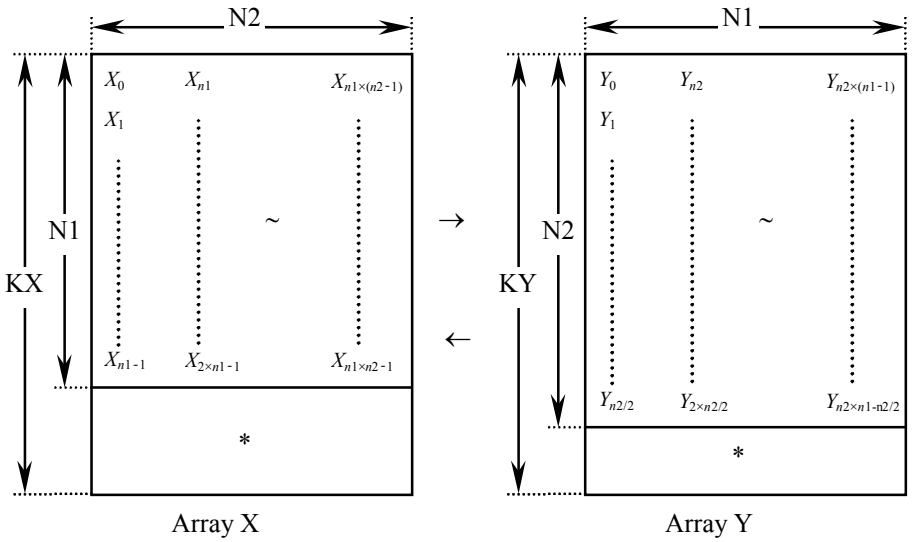


Figure DM_V1DRCF-1 Input/Output data storage method

Table DM_V1DRCF-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	KX < N1, KY < N2/2+1, N1 < 1, N2 < 1, ISIN ≠ 1,–1, or ISN ≠ 1, –1.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) If one-dimensional data of $n = n_1 \times n_2$ is numbered $k = 0$, ..., n–1,

$k = k_1 + k_2 \times n_1$, $k_1 = 0, ..., n_1-1$

, $k_2 = 0, ..., n_2-1$

$k = i_1 + i_2 \times n_2$, $i_1 = 0, ..., n_2-1$

, $i_2 = 0, ..., n_1-1$

Real data and complex data are regarded as two-dimensional data with subscripts of (k_1, k_2) and (i_1, i_2), respectively. However, $i_1 = 0$, ..., $n_2/2$ are stored in Y.

(See Figure DM_V1DRCF-1.)

2) General definition of a Fourier transform

The one-dimensional discrete complex Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \qquad (3.2)$$

where, $\omega_n = exp(2\pi i/n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

3) The result of the one-dimensional real Fourier transform has the following complex conjugate relation (indicated by $\overline{\ }$).

$\alpha_k = \overline{\alpha_{n-k}}$ k=1, ..., n–1

$n = n_1 \times n_2$

$i_1 = 0, 1, ..., n_2-1$

$i_2 = 0, 1, ..., n_1-1$

If $k = i_1 + i_2 \times n_2$ is assumed,

$n - k = n_2 - i_1 + (n_1-1-i_2) \times n_2$

The rest of data can be obtained from data numbered i_1 = 1, ..., n_2/2 (the first part excluding zeros).

4) The performance of this routine will be the best when the n can be factorized into adequately large n_1 and n_2 which are about the same size.

b. Example

A one-dimensional real FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N1=1024,N2=1024,KX=N1+1,KY=N2/2+1+1)
      REAL*8     X(KX,N2), XX(N1,N2)
      COMPLEX*16 Y(KY,N1)
CC
      DO I=1,N2
      DO J=1,N1
      X(J,I)=J+N1*(I-1)
      XX(J,I)=X(J,I)
      ENDDO
      ENDDO
      ISW=1
      CALL DM_V1DRCF(X,KX,Y,KY,N1,N2,1,ISW,ICON)
      PRINT*,' ICON =',ICON
CC
      ISW=-1
      CALL DM_V1DRCF(X,KX,Y,KY,N1,N2,1,ISW,ICON)
      PRINT*,' ICON =',ICON
CC
      TMP=0.0D0
```

```
      DO I=1,N2
      DO J=1,N1
      TMP=MAX(DABS(DBLE(X(J,I))/DBLE(N1)/DBLE(N2)
     $        -DBLE(XX(J,I))),TMP)
      ENDDO
      ENDDO
CC
      PRINT*,' ERROR  =',TMP
      STOP
      END
```

(4) Method

DM_V1DRCF is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V1DRCF2

One-dimensional discrete real Fourier transform (mixed radix of 2, 3, 5 and 7)
CALL DM_V1DRCF2(X,N,Y,ISIN,ISN,ICON)

(1) Function

This subroutine performs a one-dimensional real Fourier transform or its inverse transform using a mixed radix FFT.

The data count n is a product of the powers of 2, 3, 5 and 7.

a. One-dimensional Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \omega_n^{-jkr},\ k = 0,\ 1,\ ...,\ n-1 \qquad (1.1)$$

$$,\ \omega_n = \exp\ (2\pi i/n)$$

$r = 1$ or $r = -1$

b. One-dimensional Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jkr},\ j = 0,\ 1,\ ...,\ n-1 \qquad (1.2)$$

$$,\ \omega_n = \exp\ (2\pi i/n)$$

$r = 1$ or $r = -1$

(2) Parameters

X ..... Input/output. Real data is stored in X(1:N).

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = −1), data is output.

This is a double precision real One-dimensional array X(N).

N ..... Input. The size of the data to be transformed.

N must be an even number and a product of the powers of 2, 3, 5 and 7.

Y ..... Output/input. About a half of the complex is stored in Y(1:N/2 + 1).

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = −1), data is input.

(See note 1), (3) “Comments on use.”)

This is a double precision complex one-dimensional array Y(N/2+1).

ISIN ..... Input. The direction of transformation.

ISIN=1 for $r = 1$.

ISIN=–1 for $r = -1$.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = –1 for the inverse transform.

Integer (INTEGER*4).

ICON ..... Output. Condition code.

See Table DM_V1DRCF2-1.

Table DM_V1DRCF2-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N is not a multiple of 2, or N is not a product of the powers of 2, 3, 5 and 7, or ISIN ≠ 1,-1, or ISN ≠ 1, -1.	Processing is discontinued.

(3) Comments on use

a. Notes

1) The result of the one-dimensional real Fourier transform has the following complex conjugate relation (indicated by $\overline{\ }$).

$\alpha_k = \overline{\alpha_{n-k}}$ k=1, ..., n–1 (excluding 0).

2) General definition of a Fourier transform

The one-dimensional discrete complex Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega_n^{-jk},\ k = 0,\ 1,\ ...,\ n-1 \quad , \ \omega_n = \exp\ (2\pi i/n) \tag{3.1}$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \omega_n^{jk},\ j = 0,\ 1,\ ...,\ n-1 \quad , \ \omega_n = \exp\ (2\pi i/n) \tag{3.2}$$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A one-dimensional real FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N1=1024,N2=N1,N=N1*N2)
      REAL*8     X(N)
      COMPLEX*16 Y(N/2+1),XX(N)
C
      DO I=1,N
      X(I)=DBLE(I)
      XX(I)=X(I)
      ENDDO
C
      CALL DM_V1DRCF2(X,N,Y,1,1,ICON)
      PRINT*,'ICON =',ICON
C
      CALL DM_V1DRCF2(X,N,Y,1,-1,ICON)
      PRINT*,'ICON =',ICON
C
      TMP=0.0D0
      DO I=1,N
      TMP=MAX(ABS(X(I)/DBLE(N)-XX(I)),TMP)
      ENDDO
      PRINT*,' ERROR =',TMP
C
      STOP
      END
```

DM_V2DRCF

Two-dimensional discrete real Fourier transform (mixed radices of 2, 3, 5 and 7)
CALL DM_V2DRCF(X,K,N1,N2,ISIN,ISN,ICON)

(1) Function

The subroutine DM_V2DRCF performs a two-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of the two-dimensional data (n_1, n_2) can be a product of the powers of 2, 3, 5 and 7.

a. The two-dimensional Fourier transform

When $\{x_{j1j2}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1 n_2 \alpha_{k1k2}\}$.

$$n_1 n_2 \alpha_{k1k2} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1r} \omega_{n2}^{-j2k2r}$$
$$, k_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1,\ ...,\ n_2 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$r = 1,\ \text{or}\ r = -1$$

b. The two-dimensional Fourier inverse transform

When $\{\alpha_{k1k2}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2}\}$.

$$x_{j1j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1r} \omega_{n2}^{j2k2r}$$
$$, j_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, j_2 = 0,\ 1,\ ...,\ n_2 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$r = 1,\ \text{or}\ r = -1$$

(2) Parameters

X ..... Input/Output. Two-dimensional real data is stored in X(1:N1,1:N2).

For the real to complex transform (INS = 1), data is input; for the complex to real transform (INS = -1), data is output.

Output/input. The real and imaginary parts of the transformed complex data are stored as follows:

The real and imaginary parts are stored in X(1,1:N1/2+1,1:N2) and X(2,1:N1/2+1,N2) respectively assuming that the array X was a three-dimensional array X(2,K/2,N2).

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = -1), data is input.

The complex data transformed Fourier has the complex conjugate relation. And about half data is stored.

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

This is a double precision real two-dimensional array X(K,N2).

K ..... Input. The size of the first dimension of array X ($\geq 2\times(n_1/2+1)$).

K must be an even number.

Integer (INTEGER*4)

N1 ..... Input. The length n_1 of data in the first dimension of the two- dimensional array to be transformed.

n_1 must be a value that can be a product of powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 ..... Input. The length n_2 of data in the second dimension of the two- dimensional array to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

ISIN ..... Input. The direction of transformation.

ISIN=1 for $r = 1$.

ISIN=–1 for $r = -1$.

Integer (INTEGER*4)

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = –1 for the inverse transform.

Integer (INTEGER*4).

ICON ..... Output. Condition code.

See Table DM_V2DRCF-1.

Table DM_V2DRCF-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	K < 2×(N1/2+1), K is not an even number, N1 < 1, N2 < 1, ISIN ≠ 1, –1,or ISN ≠ 1, –1.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The two-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3,1) and (3,2).

$$\alpha_{k1k2} = \frac{1}{n_1 n_2} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1j2} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2}$$
$$, k_1 = 0, 1, ..., n_1 - 1 \qquad (3.1)$$
$$, k_2 = 0, 1, ..., n_2 - 1$$

$$x_{j1j2} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \alpha_{k1k2} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2}$$
$$, j_1 = 0, 1, ..., n_1 - 1 \qquad (3.2)$$
$$, j_2 = 0, 1, ..., n_2 - 1$$

where $\omega_{n1} = exp\ (2\pi\ i/n_1)$, $\omega_{n2} = exp\ (2\pi\ i/n_2)$

This subroutine calculates $\{n_1 n_2 \alpha_{k1k2}\}$ or $\{x_{j1j2}\}$ corresponding to the left term of (3.1) or (3.2), respectively. Normalization of the results is required, if necessary.

2) The results of the two-dimensional real Fourier transform that has the following complex conjugate relation (indicated by $^-$).

$$\alpha_{k1k2} = \overline{\alpha_{n1-k1\ n2-k2}} \qquad (3.3)$$

The remainder of the data is obtained from the data in $k_1 = 0, ..., n_1/2$ and $k_2 = 0, ..., n_2-1$.

b. Example

A two-dimensional real FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N1=2048,N2=2048,K=(N1/2+1)*2)
      REAL*8    X(K,N2), Y(N1,N2)
CC
      DO I=1,N2
      DO J=1,N1
      X(J,I)=J+N2*(I-1)
      Y(J,I)=X(J,I)
      ENDDO
      ENDDO
      ISW=1
      CALL DM_V2DRCF(X,K,N1,N2,1,ISW,ICON)
      PRINT*,' ICON =',ICON
CC
      ISW=-1
      CALL DM_V2DRCF(X,K,N1,N2,1,ISW,ICON)
      PRINT*,' ICON =',ICON
CC
      TMP=0.0D0
```

```
      DO I=1,N2
      DO J=1,N1
      TMP=MAX(DABS(DBLE(X(J,I))/DBLE(N1)/DBLE(N2)
     $             -DBLE(Y(J,I))),TMP)
      ENDDO
      ENDDO
CC
      PRINT*,' ERROR  =',TMP
      STOP
      END
```

(4) Method

DM_V2DRCF is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V3DRCF

Three-dimensional discrete real Fourier transform (mixed radices of 2, 3, 5 and 7)
CALL DM_V3DRCF(X,K,N1,N2,N3,ISIN,ISN,ICON)

(1) Function

The subroutine DM_V3DRCF performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of the three-dimensional array (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$
\begin{aligned}
n_1n_2n_3\alpha_{k1k2k3} = & \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r} \\
& , k_1 = 0,\ 1,\ ...,\ n_1-1 \\
& , k_2 = 0,\ 1,\ ...,\ n_2-1 \\
& , k_3 = 0,\ 1,\ ...,\ n_3-1 \\
& , \omega_{n1} = \exp\ (2\pi i/n_1) \\
& , \omega_{n2} = \exp\ (2\pi i/n_2) \\
& , \omega_{n3} = \exp\ (2\pi i/n_3) \\
& r = 1 \text{ or } r = -1
\end{aligned}
\qquad (1.1)
$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$
\begin{aligned}
x_{j1j2j3} = & \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r} \\
& , j_1 = 0,\ 1,\ ...,\ n_1-1 \\
& , j_2 = 0,\ 1,\ ...,\ n_2-1 \\
& , j_3 = 0,\ 1,\ ...,\ n_3-1 \\
& , \omega_{n1} = \exp\ (2\pi i/n_1) \\
& , \omega_{n2} = \exp\ (2\pi i/n_2) \\
& , \omega_{n3} = \exp\ (2\pi i/n_3) \\
& r = 1 \text{ or } r = -1
\end{aligned}
\qquad (1.2)
$$

(2) Parameters

X ..... Input/Output. Three-dimensional real data is stored in X(1:N1,1:N2,1:N3).

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = –1), data is output.

Output/input. The real and imaginary parts of the transformed complex data are stored as follows:

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = –1), data is input.

The complex data obtained from real data by Fourier transformation has the complex conjugate relation. And about half data is stored.

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

The real and imaginary parts are stored in X(1,1:N1/2+1,1:N2,1:N3) and X(2,1:N1/2+1,1:N2,1:N3) respectively assuming that the array X was a four-dimensional array X(2,K/2,N2,N3).

This is a double precision real three-dimensional array X(K,N2,N3).

K ..... Input. The size of the first dimension of array X ($\geq 2\times(n_1/2+1)$). Even number.

Integer (INTEGER*4)

N1 ..... Input. The length n_1 of real data in the first dimension to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 ..... Input. The length n_2 of real data in the second dimension to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 ..... Input. The length n_3 of real data in the third dimension to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

ISIN ..... Input. The direction of the transformation.

ISIN=1 for $r = 1$.

ISIN=–1 for $r = -1$.

Integer (INTEGER*4)

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = –1 for the inverse transform.

Integer (INTEGER*4).

ICON ..... Output. Condition code.

See Table DM_V3DRCF-1.

Table DM_V3DRCF-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	K < 2 × (N1/2+1), K is not an even number, N1 < 1, N2 < 1, N3 < 1, ISIN ≠ 1, –1, or ISN ≠ 1, –1.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad , k_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , k_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , k_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.1}$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad , j_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , j_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , j_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.2}$$

where, $\omega_{n1} = exp(2\pi i/n_1)$, $\omega_{n2} = exp(2\pi i/n_2)$,

$\omega_{n3} = exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left term of (3.1) or (3.2), respectively. The normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by ¯).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \tag{3.3}$$

The remainder of the data is obtained from data in $k_1 = 0, ..., n_1/2$, $k_1 = 0, ..., n_2-1$, and $k_3 = 0, ..., n_3-1$.

b. Example

A three-dimensional real FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
C     **EXAMPLE**
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (N1=128,N2=128,N3=128,K=(N1/2+1)*2)
      DIMENSION YY(K,N2,N3), YR(K,N2,N3)
C
      DO I3=1,N3
      DO I2=1,N2
      DO I1=1,N1
      YY(I1,I2,I3)=DBLE(I1+N1*(I2-1)+N1*N2*(I3-1))
      YR(I1,I2,I3)=YY(I1,I2,I3)
      ENDDO
      ENDDO
      ENDDO
C
      ISW=1
      CALL DM_V3DRCF(YY,K,N1,N2,N3,1,ISW,ICON)
      PRINT*,'ICON =',ICON
C
      ISW=-1
      CALL DM_V3DRCF(YY,K,N1,N2,N3,1,ISW,ICON)
      PRINT*,'ICON =',ICON
C
      TMP=0.0D0
      DO I3=1,N3
      DO I2=1,N2
      DO I1=1,N1
      TMP=MAX(DABS(YY(I1,I2,I3)/DBLE(N1)/DBLE(N2)/DBLE(N3)
     $             -YR(I1,I2,I3)),TMP)
      ENDDO
      ENDDO
      ENDDO
      PRINT*,' ERROR =',TMP
C
      STOP
      END
```

(4) Method

DM_V3DRCF is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V3DRCF2

Three-dimensional discrete real Fourier transform (mixed radices of 2, 3, 5 and 7)
CALL DM_V3DRCF2(X,K1,K2,N1,N2,N3,ISIN,ISN,ICON)

(1) Function

The subroutine DM_V3DRCF2 performs a three-dimensional real Fourier transform or its inverse Fourier transform using a mixed radix FFT.

The size of each dimension of the three-dimensional array (n_1, n_2, n_3) can be a product of the powers of 2, 3, 5 and 7.

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1n_2n_3\alpha_{k1k2k3}\}$.

$$n_1n_2n_3\alpha_{k1k2k3} = \sum_{j1=0}^{n1-1}\sum_{j2=0}^{n2-1}\sum_{j3=0}^{n3-1} x_{j1j2j3}\omega_{n1}^{-j1k1r}\omega_{n2}^{-j2k2r}\omega_{n3}^{-j3k3r}$$
$$, k_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, k_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, k_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$
$$r = 1 \text{ or } r = -1$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1}\sum_{k2=0}^{n2-1}\sum_{k3=0}^{n3-1} \alpha_{k1k2k3}\omega_{n1}^{j1k1r}\omega_{n2}^{j2k2r}\omega_{n3}^{j3k3r}$$
$$, j_1 = 0,\ 1,\ ...,\ n_1 - 1$$
$$, j_2 = 0,\ 1,\ ...,\ n_2 - 1$$
$$, j_3 = 0,\ 1,\ ...,\ n_3 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp\ (2\pi i/n_1)$$
$$, \omega_{n2} = \exp\ (2\pi i/n_2)$$
$$, \omega_{n3} = \exp\ (2\pi i/n_3)$$
$$r = 1 \text{ or } r = -1$$

(2) Parameters

X ..... Input/Output. Three-dimensional real data is stored in X(1:N1,1:N2,1:N3).

For the real to complex transform (ISN = 1), data is input; for the complex to real transform (ISN = –1), data is output.

Output/input. The real and imaginary parts of the transformed complex data are stored as follows:

For the real to complex transform (ISN = 1), data is output; for the complex to real transform (ISN = −1), data is input.

The complex data obtained from real data by Fourier transformation has the complex conjugate relation. And about half data is stored.

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

The real and imaginary parts are stored in X(1,1:N1/2+1,1:N2,1:N3) and X(2,1:N1/2+1,1:N2,1:N3) respectively assuming that the array X was a four-dimensional array X(2,K1/2,K2,N3).

This is a double precision real three-dimensional array X(K1,K2,N3).

K1 ..... Input. The size of the first dimension of array X ($\geq 2\times(n_1/2+1)$). Even number.

Integer (INTEGER*4)

K2 ..... Input. The size of the second dimension of array X ($\geq n_2$).

Integer (INTEGER*4)

N1 ..... Input. The length n_1 of real data in the first dimension to be transformed.

n_1 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N2 ..... Input. The length n_2 of real data in the second dimension to be transformed.

n_2 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

N3 ..... Input. The length n_3 of real data in the third dimension to be transformed.

n_3 must be a value that can be a product of the powers of 2, 3, 5 and 7.

Integer (INTEGER*4)

ISIN ..... Input. The direction of the transformation.

ISIN=1 for $r = 1$.

ISIN=−1 for $r = -1$.

Integer (INTEGER*4)

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = −1 for the inverse transform.

Integer (INTEGER*4).

ICON ..... Output. Condition code.

See Table DM_V3DRCF2-1.

Table DM_V3DRCF2-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	K1 < 2 × (N1/2+1), K1 is not an even number, K2 < N2, N1 < 1, N2 < 1, N3 < 1, ISIN ≠ 1, −1, or ISN ≠ 1, −1.	Processing is discontinued.
30008	The order of the transform is not radix 2/3/5/7.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad , k_1 = 0, 1, ..., n_1 - 1 \quad , k_2 = 0, 1, ..., n_2 - 1 \quad , k_3 = 0, 1, ..., n_3 - 1 \tag{3.1}$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad , j_1 = 0, 1, ..., n_1 - 1 \quad , j_2 = 0, 1, ..., n_2 - 1 \quad , j_3 = 0, 1, ..., n_3 - 1 \tag{3.2}$$

where, $\omega_{n1} = exp(2\pi i/n_1)$, $\omega_{n2} = exp(2\pi i/n_2)$,

$\omega_{n3} = exp(2\pi i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left term of (3.1) or (3.2), respectively. The normalization of the results may be required.

2) The results of the three-dimensional real Fourier transform has the following complex conjugate relation (indicated by ¯).

$$\alpha_{k1k2k3} = \overline{\alpha_{n1-k1\ n2-k2\ n3-k3}} \tag{3.3}$$

The remainder of the data is obtained from data in $k_1 = 0, ..., n_1/2$, $k_1 = 0, ..., n_2-1$, and $k_3 = 0, ..., n_3-1$.

b. Example

A three-dimensional real FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
c     **example**
      implicit real*8(a-h,o-z)
      parameter(n1=128,n2=128,n3=128,k1=(n1/2+1)*2,k2=n2+1)
      dimension yy(k1,k2,n3),yr(k1,k2,n3)
c
      do i3=1,n3
      do i2=1,n2
      do i1=1,n1
      yy(i1,i2,i3)=dble(i1+n1*(i2-1)+n1*n2*(i3-1))
      yr(i1,i2,i3)=yy(i1,i2,i3)
      enddo
      enddo
      enddo
c
      isw=1
      call dm_v3drcf2(yy,k1,k2,n1,n2,n3,1,isw,icon)
      print*,'icon =',icon
c
      isw=-1
      call dm_v3drcf2(yy,k1,k2,n1,n2,n3,1,isw,icon)
      print*,'icon =',icon
c
      tmp=0.0d0
      do i3=1,n3
      do i2=1,n2
      do i1=1,n1
      tmp=max(dabs(yy(i1,i2,i3)
     $             /dble(n1)/dble(n2)/dble(n3)
     $            -yr(i1,i2,i3)),tmp)
      enddo
      enddo
      enddo
      print*,' error =',tmp
c
      stop
      end
```

(4) Method

DM_V3DRCF2 is implemented using DVCFM1 which is the routine of one-dimensional complex Fourier transform highly adapted to a scalar computer. Refer to “SSL II Extended Capabilities User's Guide II” in detail.

DM_V3DCPF

Three-dimensional prime factor discrete complex Fourier transforms
CALL DM_V3DCPF(X,K1,K2,N1,N2,N3,ISN,ICON)

(1) Function

The subroutine DM_V3DCPF performs a three-dimensional complex Fourier transform or its inverse Fourier transform.

The size of each dimension of three-dimensional data (n_1, n_2, n_3) 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 ($p \in \{2, 3, 4, 5, 7, 8, 9, 16, 25\}$)

a. The three-dimensional Fourier transform

When $\{x_{j1j2j3}\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$.

$$n_1 n_2 n_3 \alpha_{k1k2k3} = \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3}$$
$$, k_1 = 0, 1, ..., n_1 - 1$$
$$, k_2 = 0, 1, ..., n_2 - 1$$
$$, k_3 = 0, 1, ..., n_3 - 1 \qquad (1.1)$$
$$, \omega_{n1} = \exp(2\pi i / n_1)$$
$$, \omega_{n2} = \exp(2\pi i / n_2)$$
$$, \omega_{n3} = \exp(2\pi i / n_3)$$

b. The three-dimensional Fourier inverse transform

When $\{\alpha_{k1k2k3}\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_{j1j2j3}\}$.

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3}$$
$$, j_1 = 0, 1, ..., n_1 - 1$$
$$, j_2 = 0, 1, ..., n_2 - 1$$
$$, j_3 = 0, 1, ..., n_3 - 1 \qquad (1.2)$$
$$, \omega_{n1} = \exp(2\pi i / n_1)$$
$$, \omega_{n2} = \exp(2\pi i / n_2)$$
$$, \omega_{n3} = \exp(2\pi i / n_3)$$

(2) Parameters

X ..... Input. The complex data.

Data is stored in X(1:N1,1:N2,1:N3).

Output. The transformed complex data.

The results are stored in X(1:N1,1:N2,1:N3).

This is a double precision complex three-dimensional array X(K1,K2,N3).

K1 ..... Input. The size of the first dimension of input data arrays X (≥ N1).

K2 ..... Input. The size of the second dimension of input data arrays X (≥ N2).

N1 ..... Input. The length n_1 of data in the first dimension of the three- dimensional array to be transformed.

N2 ..... Input. The length n_2 of data in the second dimension of the three- dimensional array to be transformed.

N3 ..... Input. The length n_3 of data in the third dimension of the three- dimensional array to be transformed.

ISN ..... Input. Either the transform or the inverse transform is indicated.

ISN = 1 for the transform.

ISN = -1 for the inverse transform.

ICON ..... Output. Condition code.

See Table DM_V3DCPF-1.

Table DM_V3DCPF-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	n_1, n_2 or n_3 can not be factored into the product of the factors in 2, 3, 4, 5, 7, 8, 9, 16, 25.	Processing is discontinued
30000	n_1, n_2 or n_3 less than or equal 0, or K1<N1, or K2<N2, or invalid value for the parameter ISN.	

(3) Comments on use

a. Notes

1) General definition of a Fourier transform

The three-dimensional discrete complex Fourier transform and its inverse transform can generally be defined as in (3.1) and (3.2).

$$\alpha_{k1k2k3} = \frac{1}{n_1 n_2 n_3} \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1j2j3} \omega_{n1}^{-j1k1} \omega_{n2}^{-j2k2} \omega_{n3}^{-j3k3} \quad , k_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , k_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , k_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.1}$$

$$x_{j1j2j3} = \sum_{k1=0}^{n1-1} \sum_{k2=0}^{n2-1} \sum_{k3=0}^{n3-1} \alpha_{k1k2k3} \omega_{n1}^{j1k1} \omega_{n2}^{j2k2} \omega_{n3}^{j3k3} \quad , j_1 = 0,\ 1,\ ...,\ n_1 - 1 \quad , j_2 = 0,\ 1,\ ...,\ n_2 - 1 \quad , j_3 = 0,\ 1,\ ...,\ n_3 - 1 \tag{3.2}$$

where, $\omega_{n1} = \exp(2\pi\, i/n_1)$, $\omega_{n2} = \exp(2\pi\, i/n_2)$,

$\omega_{n3} = \exp(2\pi\, i/n_3)$

This subroutine calculates $\{n_1 n_2 n_3 \alpha_{k1k2k3}\}$ or $\{x_{j1j2j3}\}$ corresponding to the left-hand-side term of (3.1) or (3.2), respectively. Normalization of the results may be required.

b. Example

A three-dimensional FFT is computed.

The number of the threads can be specified with an environment variable (OMP_NUM_THREADS). For example, set OMP_NUM_THREADS to be 4 when this program is to be executed in parallel with 4 threads on the system of 4 processors.

```
c     **example**
      implicit real*8 (a-h,o-z)
      parameter (n1=40,n2=240,n3=90)
      parameter (k1=n1,k2=n2)
      complex*16 x(k1,k2,n3)
      integer isn
*
*     set up the input data arrays
*
!$omp parallel do default(private) shared(x)
      do k=1,n3
      do i=1,n2
      do j=1,n1
      x(j,i,k)=dcmplx(float(j)+float(n1)*(i-1),0.0)
      enddo
      enddo
      enddo
!$omp end parallel do

*
*     do the forward transform
*
      isn=1
      call dm_v3dcpf(x,k1,k2,n1,n2,n3,isn,icon)
      if(icon.ne.0) then
        write(*,*) 'error occurred : ',icon
      endif
*
*    do the reverse transform
*
      isn=-1
      call dm_v3dcpf(x,k1,k2,n1,n2,n3,isn,icon)
      if(icon.ne.0) then
        write(*,*) 'error occurred : ',icon
      endif
*
*    find the error after the forward and
*    inverse transform.
*
      error=0
```

```
!$omp parallel do default(private) shared(x)
!$omp+         reduction(max:error)

      do k=1,n3
      do i=1,n2
      do j=1,n1
      error=max(abs(dble(x(j,i,k))/(n3*n2*n1)-
     &      (float(j)+float(n1)*(i-1))),error)
      error=max(abs(dimag(x(j,i,k))/(n3*n2*n1)),
     &                           error)
      enddo
      enddo
      enddo
!$omp end parallel do

      write(*,*) 'error ', error
      stop
      end
```

Appendixes

Appendix A References

[1] P. AMESTOY, M. DAYDE and I. DUFF

Use of computational kernels in the solution of full and sparse linear equations, M. COSNARD, Y. ROBERT, Q. QUINTON and M. RAYNAL, PARALLEL & DISTRIBUTED ALGORITHMS, North-Holland, 1989, pp. 13-19.

[2] P.R.AMESTOY and C.PUGLISH

AN UNSYMMETRIZED MULTIFRONTAL LU FACTORIZATION, SIAM J. MATRIX ANAL. APPL. Vol. 24, No. 2, pp. 553-569, 2002

[3] A.A. Anda and H. Park

Fast Plane Rotations with Dynamic Scaling, to appear in SIAM J, Matrix Analysis and Applications, 1994.

[4] S.L. Anderson

Random number generators on vector supercomputers and other advanced architectures, SIAM Rev. 32 (1990), 221-251.

[5] C. Ashcraft

The distributed solution of linear systems using the torus wrap data mapping, Tech. Report ECA-TR-147, Boeing Computer Services, October 1990.

[6] O.Axelsson and M.Neytcheva

Algebraic multilevel iteration method for Stieltjes matrices. Num. Lin. Alg. Appl., 1:213-236, 1994.

[7] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors.

Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia, 2000.

[8] Å.Björck

Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT, 7:1-21, 1967.

[9] R.P. Brent

Uniform random number generators for supercomputers, Proc. Fifth Australian Supercomputer Conference, Melbourne, Dec. 1992, 95-104.

[10] 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

[11] 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.

[12] R.P.Brent

A Fast Vectorised Implementation of Wallace's Normal Random Number Generator, Technical Report, Computer Sciences Laboratory, Australian National University, to appear.

[13] R.Burkard, M.Dell'Amico and S.Martello

Assignment Problems, SIAM Philadelphia, 2009

[14] J. Choi, J. Dongarra, R. Pozo, and D. Walker

ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers., Technical Report 53, LAPACK Working Note, 1993.

[15] A. Cleary

A comparison of algorithms for Cholesky factorization on a massively parallel MIMD computer, Parallel Processing for Scientific Computing, 1991.

[16] A. Cleary

A Scalable Algorithm for Triangular System Solution Using the Torus Wrap Mapping, ANU-CMA Tech Report, series 1994.

[17] T.H.CORMEN, C.E.LEISERSON, R.L.RIVEST and C.STEIN

INTRODUCTION TO ALGORITHMS, SECOND EDITION, The MIT Press, 2001

[18] J.K. Cullum and R.A. Willoughby

"Lanczos algorithm for large symmetric eigenvalue computations", Birkhauser, 1985.

[19] T.Davis,
Direct Methods for Sparse Linear Systems, SIAM 2006.

[20] J. Demmel and W. Kahan

Accurate singular values of bidiagonal matrices, SISSC 11, 873-912, 1990.

[21] J.J. Dongarra and R.A. Van de Geijn

Reduction to condensed form for the eigenvalue problem on distributed memory architectures, Parallel Computing, 18, pp. 973-982, 1992.

[22] I.S.DUFF, A.M.ERISMAN and J.K.REID

Direct Methods for Sparse Matrices, OXFORD SCIENCE PUBLICATIONS, 1986

[23] I.S.DUFF and J.KOSTER

ON ALGORITHMS FOR PERMUTING LARGE ENTRIES TO THE DIAGONAL OF A SPARSE MATRIX, SIAM J. MATRIX ANAL. APPL. Vol. 22, No. 4, pp. 973-996, 2001

[24] 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.

[25] G. Fox

Square matrix decomposition — Symmetric, local, scattered, Caltech Publication Hm-97, California Institute of Technology, Pasadena, CA, 1985.

[26] R. Freund

"A transpose-free quasi-minimal residual algorithm for nonhermitian linear systems, SIAM J. Sci. Comput. 14, 1993, pp. 470-482.

[27] R. Freund and N. Nachtigal

"QMR: a quasi minimal residual method for non-Hermitian linear systems", Numer. Math. 60, 1991, pp. 315-339.

[28] K.A. Gallivan, R.J. Plemmons, and A.H. Sameh

Parallel Algorithms for Dense Linear Algebra Computations, SIAM Review, 1990.

[29] Martin B. van Gijzen and Peter Sonneveld

"An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties", Delft university of technology, Report 08-21, 2008.

[30] G.H. Golub, C.F. van Loan

Matrix Computations Second Edition, The Johns Hopkins University Press, 1989

[31] Marcus J. Grote and Thomas Huckle

"Parallel preconditioning with sparse approximate inverse", SIAM J. Sci. Comput., Vol.18, No.3, pp838-853, May 1997.

[32] M.H.Gutknecht

Variants of BiCGStab for matrices with complex spectrum,IPS Research report No. 91-14, 1991.

[33] E. Hairer, S.P.Norsett, and G. Wanner

"Solving Ordinary Differential Equations I: Nonstiff Problems." Second Revised Edition, Springer, 2000.

[34] E. Hairer, and G. Wanner

"Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems." Second Revised Edition, Springer, 2002

[35] Japanese translation of [33] , Springer, 2007

[36] Japanese translation of [34], Springer, 2008

[37] Markus Hegland

An implementation of multiple and multi-variate Fourier transforms on vector processors, submitted to SIAM J. Sci. Comput., 1992.

[38] Markus Hegland

Block Algorithms for FFTs on Vector and Parallel Computers. PARCO 93, Grenoble, 1993.

[39] Markus Hegland

On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization, Numer. Math. 59, 453-472, 1991.

[40] B. Hendrickson and D.Womble

The torus-wrap mapping for dense matrix calculations on massively parallel computers, SAND Report SAND 92-0792, Sandia National Laboratories, Albuquerque, NM, 1992.

[41] 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.

[42] F. James

A review of pseudorandom number generators, Computer Physics Communications 60 (1990), 329-344.

[43] G.KARYPIS AND V.KUMAR

A fast and high quality multilevel scheme for partitioning irregurar graphs, SIAM J. Sci. Comput., 20 pp.359-392, 1998

[44] G.KARYPIS AND V.KUMAR

METIS

A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices

Version 4.0

University of Minnesota, Department of Computer Science / Army HPC Research Center Minneapolis, MN 55455

September 20, 1998

[45] D. Kincaid, T. Oppe

ITPACK on supercomputers, Numerical methods, Lecture Notes in Mathematics 1005 (1982).

[46] 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.

[47] 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.

[48] X.S.Li AND J.W.DEMMEL

A scalable sparse direct solver using static pivoting, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, 1999, CD-ROM, SIAM, Philadelphia, PA, 1999

[49] Charles Van Loan

Computational Frameworks for the Fast Fourier Transform, SIAM, 1992.

[50] F.T. Luk

Computing the Singular-Value Decomposition on the ILIAC IV, ACM Trans. Math. Softw., 6, 1980, pp. 259-273.

[51] F.T. Luk and H. Park

On Parallel Jacobi Orderings, SIAM J. Sci. Comput., 10, 1989, pp. 18-26.

[52] 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

[53] 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.

[54] M. Nakanishi, H. Ina, K. Miura

A high performance linear equation solver on the VPP500 parallel supercomputer, Proceedings of Supercomputing ’94, Washington D.C., Nov. 1994.

[55] M. Nakanishi, J. Mikami

Tuning techniques for blocking LU decomposition in VP2000 series, 42nd IPSJ Conference (1991) [in Japanese]

[56] M. Nakanishi, J. Mikami

High performance methods for solving linear equations, IPSJ Tech Report 91-OS-54, Vol.92, No.22, pp.33-40 (1992) [in Japanese]

[57] M.OLSCHOWKA and A.NEUMAIER

A new pivoting strategy for Gaussian elimination, Linear Algebra Appl., 240(1996), pp.131-151

[58] T. Oppe, W. Joubert and D. Kincaid

An overview of NSPCG: a nonsymmetric preconditioned conjugate gradient package, Computer Physics communications 53 p283 (1989).

[59] T.C. Oppe and D.R. Kincaid

“Are there iterative BLAS?”, Int. J. Sci. Comput. Modeling (to appear or has appeared).

[60] M.R. Osborne

Solving least squares problems on parallel vector processors, Area 4 working notes no. 17, 1994.

[61] M.R. Osborne

Computing the eigenvalues of tridiagonal matrices on parallel vector processors, Mathematics Research Report No. MRR 044-94, Australian National University, 1994.

[62] J.R. Rice and R.F. Boisvert

Solving Elliptic Problems Using Ellpack, Springer-Verlag, New York, 1985.

[63] D. Ruiz

A scaling algorithm to equilibrate both rows and columns norms in matrices, Tech. rep. RAL-TR-2001-034, Rutherford Appleton Laboratory, Chilton, U.K., 2001

[64] Y.Saad

A dual threshold incomplete LU factorization. Research Report UMSI 92/38, University of Minnesota, Supercomputer Institute, 1200 Washington Avenue South, Minneapolis, Minnesota 55415, USA, 1992.

[65] Y.Saad

A multi-elimination ILU preconditioner for general sparse 591 matrices. SIAM J.Sci.Comput, 17:830-847, 1996.

[66] Y.Saad

"Iterative methods for sparse linear systems, second edition",
Univ.Minnesota,SIAM, 2003

[67] 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.

[68] O.Schenk , K.Gärtner

Solving unsymmetric sparse systems of linear equations with PARDISO, Future Generation Computer Systems 20(2004)475-487

[69] J.A.SCOTT

Scaling and Pivoting in an Out-of-Core Sparse Direct Solver
ACM Transactions on Mathematical Software, Vol. 37, No. 2, Article 19, April 2010

[70] M. Shimasaki

Supercomputer and Programming, Kyoritsu Publishers (1989) [in Japanese]

[71] H.D. Simon

Bisection is not optimal on vector processors, SISSC 10, 205-209, 1989.

[72] G. Sleijpen, D. Fokkema

BCG for linear equations involving unsymmetric matrices with complex spectrum, Electronic Transactions on Numerical Analysis, 1 p11 1993

[73] Gerard L.G. Sleijpen and Martin B. van Gijzen

"Exploiting BICGSTAB(l) Strategies to Induce Dimension Reduction",
Delft university of technology, Report 09-02, 2009.

[74] Tomohiro Sogabe,Shao-Liang Zhang

"A COCR method for solving complex symmetric linear systems",
Journal of Computational and SIAM Applied Mathematics,199(2007)297-303.

[75] J.C. Strikwerda

Finite Difference Schemes and Partial Differential Equations. Wadsworth and Brooks/Cole, Pacific Grove, 1989.

[76] Paul N. Swarztrauber

Multiprocessor FFTs. Parallel Comput. 5, 197-210, 1987.

[77] 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

[78] C.S.Wallace

"Fast Pseudo-Random Generators for Normal and Exponential Variates", ACM Trans. on Mathematical Software 22 (1996), 119-127.

[79] R.Weiss

Parameter-Free Iterative Linear Solvers. Mathematical Research, vol. 97. Akademie Verlag, Berlin, 1996.

[80] J.H. Wilkinson

The Algebraic Eigenvalue Problem, O.U.P., 1965.

[81] B.B. Zhou and R.P. Brent

A Parallel Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations, to appear in Proc. Sixty IASTED-ISMM International Conference on Parallel and Distributed Computing Systems, 1994.

[82] K. Miura

Full Polynomial Multiple Recursive Generator(MRG) Revisited, MCQMC 2006, Ulm, Germany

[83] Kenta Hongo, Ryo Maezono, and Kenichi Miura

Random Number Generators Tested on Quantum Monte Carlo Simulations, Journal of Computational Chemistry, 31, 2186-2194, 2010

[84] P. L'Ecuyer and R. Simard

TestU01: A C Library for Empirical Testing of Random Number Generators, ACM Transactions on Mathematical Software, Vol. 33, article 22, 2007.

Appendix B
Contributors and Their Work

The almost full or partial parts of the codes and algorithms developed for SSL II/VPP are used or tailored to implement the functions in SSL II Thread-Parallel Capabilities. The following table shows the contributors for the SSL II/VPP subroutines used.

Author	**Subroutine in SSLII/VPP (Subroutine in Thread-Parallel Capabilities)**	**Item**
Richard Peirce Brent Peter Frederick Price	DP_VRANU4 (DM_VRANU4)	Generation of uniform random numbers [0,1)
Richard Peirce Brent Margaret Helen Kahn	DP_VRANN3 (DM_VRANN3)	Generation of normal random numbers
Richard Peirce Brent	DP_VRANN4 (DM_VRANN4)	Generation of normal random numbers (Wallace's method)
Murry Leslie Dow	DP_VBCSD (DM_VBCSD)	System of linear equations with unsymmetric or indefinite sparse matrices (BICGSTAB(*l*) method, diagonal format storage method)
	DP_VBCSE (DM_VBCSE)	System of linear equations with unsymmetric or indefinite sparse matrices (BICGSTAB(*l*) method, ELLPACK format storage method)
	DP_VCGD (DM_VCGD)	A system of linear equations with symmetric positive definite sparse matrices (preconditioned CG method, diagonal format storage method)
	DP_VCGE (DM_VCGE)	A system of linear equations with symmetric positive definite sparse matrices (preconditioned CG method, ELLPACK format storage method)
Lutz Grosz	DP_VAMLID (DM_VAMLID)	System of linear equations with sparse matrices of M-matrix (Algebraic multilevel iteration method [AMLI Method], diagonal format storage method)
	– (DM_VMLBIFE)	System of linear equations with sparse matrices (Multilevel iteration method based on incomplete block factorization, ELLPACK format storage method)
	DP_VPDE2D (DM_VPDE2D)	Generation of System of linear equations with sparse matrices by the finite difference discretization of a two dimensional boundary value problem for second order partial differential equation

Author	Subroutine in SSLII/VPP (Subroutine in Thread-Parallel Capabilities)	Item
Lutz Grosz	DP_VPDE3D (DM_VPDE3D)	Generation of System of linear equations with sparse matrices by the finite difference discretization of a three dimensional boundary value problem for second order partial differential equation
Zbigniew Leyk	DP_VMVSD (DM_VMVSD)	Multiplication of real sparse matrices and real vectors (diagonal format storage method)
	DP_VMVSE (DM_VMVSE)	Multiplication of real sparse matrices and real vectors (ELLPACK format storage method)
	DP_VTFQD (DM_VTFQD)	System of linear equations with unsymmetric or indefinite sparse matrices (TFQMR method, diagonal format storage method)
	DP_VTFQE (DM_VTFQE)	System of linear equations with unsymmetric or indefinite sparse matrices (TFQMR method, ELLPACK format storage method)
Michael Robert Osborne David Lawrence Harrar II	DP_VTDEVC (DM_VTDEVC)	Eigenvalues and eigenvectors of real tridiagonal matrices

Subroutine DM_VRADAU5 is based on the free software RADAU5 developed by Ernst Hairer and distributed under the following condition.

Copyright (c) 2004, Ernst Hairer

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON

ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index