FUJITSU

FUJITSU Software

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

J2UL-1904-01ENZ0(01)
February 2020

PREFACE

This manual describes the extended capabilities of the Scientific Subroutine Library II (SSL II).

SSL II consists of standard and extended capabilities. Standard capabilities, explained in FUJITSU SSL II User's Guide, are provided for a wide range of scientific calculations performed on general-purpose computers. Extended capabilities are provided for high-speed scientific calculations on FUJITSU VP Series vector processors.

These functions are still included for compatibility.

This manual is organized as follows:

PART I GENERAL DESCRIPTION

Functions are outlined for individual fields and subroutine selection is explained.

PART II USAGE OF SUBROUTINES

The usage of subroutines is discussed. Subroutines are listed in alphabetical order.

For SSL II conventions and standard subroutines, refer to the following manual:

FUJITSU SSL II User's Guide

The asterisk in the table of contents of this manual indicate items added or changed from the previous edition (manual code 99SP4070E-1).

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
February 2020, Version 3.1	J2UL-1904-01ENZ0(01)
October 2014, 3rd Version	J2UL-1904-01ENZ0(00)
August 1991, 2nd Version	—
December 1987, 1st Version	—

Copyright

Update History

Changes	Location	Version
Rework format	Cover, PREFACE	Version 3.1
Rework format	Cover, PREFACE	3rd Version

CONTENTS

ILLUSTRATIONS

FIGURES

SUBROUTINE LIST OF SSL II EXTENDED CAPABILITIES

Linear Equations

Subroutine name	Item
VMGGM	Multiplication of two matrices (real general by real general)
VLSX	A system of linear equations with a positive definite symmetric matrix (modified Cholesky's method)
VSLDL	LDL^T decomposition of a positive definite symmmetric matric (modified Cholesky's method)
VLDLX	A system of linear equations with a positive definite symetric matrix decomposed into L, D, and L^T
VLTX	A system of linear equations with a real tridiagonal matrix (cyclic reduction method)
VLTX1	A system of linear equations with a real constant-tridiagonal matrix (Dirichlet type, cyclic reduction method)
VLTX2	A system of linear equations with a real constant-tridiagonal matrix (Neumann type, cyclic reduction method)
VLTX3	A system of linear equations with a real constant-tridiagonal matric (periodic type, cyclic reduction method)
VLAX	A system of linear equations with a real general matrix (blocking LU-decomposition method)
VALU	LU-decomposition of a real general matrix (blocking LU-decomposition method)
VLUIV	The inverse of a real general matrix decomposed into the factors L and U

Eigenvalues and Eigenvectors

Subroutine name	Item
VSEG2	Selected eigenvalues and corresponding eigenvectors of a real symmetric matrix (Parallel bisection and inverse iteration methods)
VGSG2	Selected eigenvalues and corresponding eigenvectors of a real symmetric generalized matrix system $Ax=\lambda Bx$ (Parallel bisection and inverse iteration methods)

Fourier Transforms

Subroutine name	Item
VCOS1	Discrete cosine transform (radix 2 FFT)
VSIN1	Discrete sine transform (radix 2 FFT)
VRFT1	Discrete real Fourier transform (high performance type, radix 2 FFT)
VRFT2	Discrete real Fourier transform (memory efficient type, radix 2 FFT)
VCFT1	Discrete complex Fourier transform (high performance type radix 2 FFT)
VCFT2	Discrete complex Fourier transfrom (memory efficient type radix 2 FFT)

PART Ⅰ GENERAL DESCRIPTION

CHAPTER 1 OUTLINE

1.1 Extended Capabilities

Scientific computations often require the solution of a variety of mathematical models in areas such as fluid dynamics, structural analysis, molecular science, and nuclear fusion. As these problems become more difficult and complicated, they require faster calculations. The vector processor helps to meet this need by incorporating a different architecture than that of a general-purpose computer, enabling it to perform high-speed calculations for mathematical models, such as special algorithms for numerical analysis.

SSL II extended capabilities perform high-speed calculations on a vector processor. Algorithms have been selected to maximize hardware efficiency. Capabilities in the FUJITSU SSL II User's Guide (99SP4020E-1) are called SSL II standard capabilities in this manual. Standard capabilities perform a wide range of calculations on general-purpose computers.

In this manual, the term SSL II is used to refer to both the standard and extended capabilities.

1.2 Structure of Extended Capabilities

Extended capabilities are divided into two groups (Fig. 1.1). Group 1, which are modifications of SSL II standard subroutines, use vector algorithms, and are provided for high-speed processing on a vector processor. Extended capabilities use different algorithms than those in the standard subroutines. Data is stored differently in array areas, and more work array space is allocated for high-speed processing. Thus, user interfaces differ from those of the corresponding standard capabilities. Also, most standard capabilities provided for a vector processor have been tuned up for vector processor to some extent without changing any user interface. In other words, group 1 can be defined as a set of subroutines that perform high-speed calculations on a vector processor, using different user interfaces than the standard capabilities.

Group 2 provides capabilities for large scale computational problems which are not included in the SSL II standard capabilities. In this group, vector algorithms are also used.

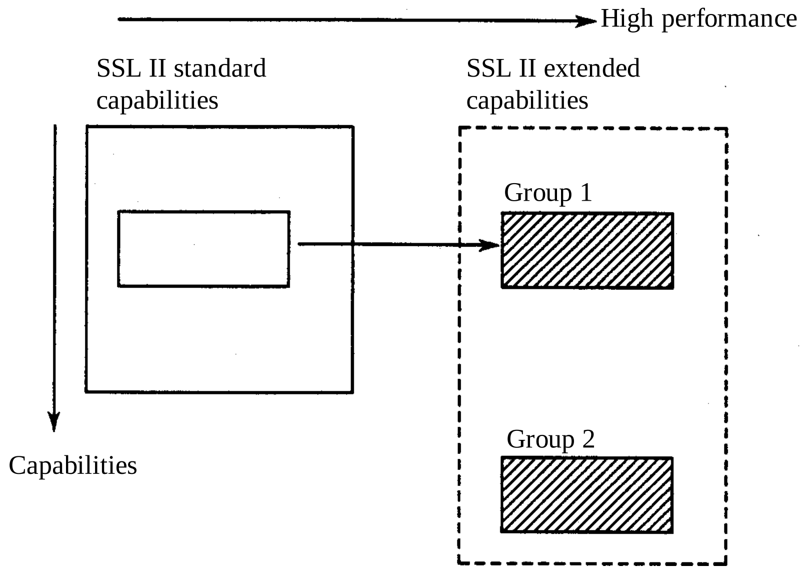


Figure 1.1 Structure of extended capabilites

1.3 Selection between Extended and Standard Capabilities

SSL II is provided for both general-purpose computers and vector processors. Therefore, user programs calling SSL II can be executed on both type of computers without any modification to the call statements.

Group 1 contains subroutines with functions similar to those of standard subroutines. For the purpose of computational efficiency, the user is recommended to select appropriate subroutines between standard and extended capabilities in the following way, when using both general-purpose computers and vector processors.

(1) When a program that calls subroutines of standard capabilities is executed on a vector processor and if the corresponding subroutines are provided in group 1, it is preferable to modify the program to employ the latter ones.

(2) When a program that calls subroutines in group 1 is executed on a general-purpose computer, the program had better be modified to call the corresponding subroutines in standard capabilities. When a general-purpose computer is used only for debugging, no program changes are needed.

The correspondence between group 1 and standard capabilities is explained in the introductory chapter for each field.

Changing the SSL II subroutine call statements in a user program takes time, but it is necessary in order to improve processing efficiency.

These changes should not affect the accuracy calculations. The vector algorithms used in SSL II enable highly accurate calculations.

CHAPTER 2 LINEAR ALGEBRA

2.1Outline

This chapter describes subroutines in linear algebra.

Subroutines of the extended capabilities in this area are listed in Table 2.1 along with the corresponding subroutines from the standard capabilities.

Table 2.1 Subroutines in linear algebra

Functions	Extended capabilities	Standard capabilities
Multiplication of two matrices	VMGGM	MGGM
A system of linear equations with a positive definite symmetric matrix	VLSX (VSLDL) (VLDLX)	LSX (SLDL) (LDLX)
A system of linear equations with a tridiagonal matrix	VLTX VLTX1 VLTX2 VLTX3	LTX LSTX
A system of linear equations with a real general matrix and the inverse of a real general matrix	VLAX (VALU) (VLUIV)	LAX (ALU) (LUIV)

The subroutines in parentheses in Table 2.1 are component subroutines. For example, VSLDL is used to perform LDL^T decomposition of a positive definitive symmetric matrix, and VLDLX is used to obtain a solution based on the decomposed matrices. Both VSLDL and VLDLX are component subroutines of VLSX.

All subroutines use vector algorithms so that they can be executed efficiently on a vector processor. The use of these subroutines and the selection of appropriate subroutines are explained in the following sections.

2.2Notes

Subroutines of the extended capabilities employ different user interfaces from those of the corresponding subroutines of the standard capabilities. Two major differences are as follows:

(1) Storage modes of a positive definite symmetric matrix and a tridiagonal matrix are different from those in the standard capabilities.

(2) Subroutines of the extended capabilities use a larger work area than those of the standard capabilities.

These differences enable memory to be accessed more efficiently when a vector algorithm is constructed. Care should be taken when a subroutine call is changed between the extended and standard capabilities.

2.3Subroutine Selection

As listed in Table 2.1, there are four subroutines for linear equations with tridiagonal matrices, each of which handles a different matrix form.

The tridiagonal matrix treated by any of four subroutines is required to be irreducibly diagonally dominant for the algorithm used to be numerically stable. The term irreducibly diagonally dominant means that the tridiagonal matrix satisfies condition (2.2) when it is of the form (2.1).

$$\begin{bmatrix} d_1 & f_1 & & & & & \\ e_2 & d_2 & f_2 & & & \boldsymbol{0} & \\ & e_3 & d_3 & f_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & \boldsymbol{0} & & & \cdot & \cdot & f_{n-1} \\ & & & & & e_n & d_n \end{bmatrix} \tag{2.1}$$

$|d_i| \geq |e_i| + |f_i|$, $i = 1, 2, \ldots, n$, and a strict inequality is satisfied for at least one i, *where* $e_1 = f_n = 0$. (2.2)

The tridiagonal matrices arising from actual applications usually satisfy the condition (2.2)

A subroutine from the standard capability should be used when the matrix does not satisfy the processing condition.

The first subroutine VLTX is the most commonly used subroutine with matrix form (2.1). However, VLTX1 is a limited version of VLTX, and handles only matrix form (2.3) below.

$$\begin{bmatrix} d & e & & & & & \\ e & d & e & & & \boldsymbol{0} & \\ & e & d & e & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & \boldsymbol{0} & & & \cdot & \cdot & e \\ & & & & & e & d \end{bmatrix} \tag{2.3}$$

As shown in (2.3), a matrix whose element values do not depend on the row or column is called a constant-tridiagonal matrix. More specifically, this matrix is called a constant-tridiagonal matrix of Dirichlet type, because it is related to a specific type of Dirichlet boundary value problem.

The matrix used in subroutines VLTX2 and VLTX3 is a modified version of the matrix in (2.3). The matrix used in VLTX2 contains the element *2e* in the first row and the second column, or in the *n*-th row and (*n*-1) th column, and is called a constant-tridiagonal matrix of Neumann type. Subroutine VLTX3 uses a matrix in which the first row and the *n*-th column element, and the *n*-th row and the first column element take *e*. This matrix is called a constant-tridiagonal matrix of periodic type. These matrices are all derived from boundary value problems of differential equations.

The algorithm used in the above subroutine is the cyclic reduction method, which is suited for vector processors. This method requires larger amount of arithmetic operations than the Gaussian elimination method. However, the cyclic reduction method presents much greater parallelism which is important factor for efficiency in vector processing. Also, for irreducibly diagonally dominant matrices, the cyclic reduction method has the same degree of accuracy as the Gaussian elimination method.

Subroutines VLTX1 and VLTX2 can perform calculations at a higher speed than VLTX , because the matrix forms in these subroutine are less complicated.

CHAPTER 3 EIGENVALUES AND EIGENVECTORS

3.1 Outline

This chapter addresses the subject of matrix eigenvalue problems. Table 3.1 shows subroutines provided as extended capabilities, along with their corresponding standard capability subroutine names.

Table 3.1 Subroutines for eigenvalue problems

Problem type	Matrix type	Extended capability subroutine name	Standard capability subroutine name
$Ax = \lambda x$	A: Real symmetric matrix	VSEG2	SEIG2
$Ax = \lambda Bx$	A: Real symmetric matrix B: Positive definite symmetric matrix	VGSG2	GSEG2

3.2 Notes

Extended capability subroutines use computational methods, in which specified m partial eigenvalues are simultaneously calculated using the parallel bisection method. Therefore, there are differences such as the work area allocation between the extended and standard capabilities. Accordingly, parameter modification is required to change from standard capability subroutine calling to extended capability subroutine calling.

CHAPTER 4 FOURIER TRANSFORMS

4.1Outline

This chapter describes subroutines in discrete Fourier transforms. Subroutines of the extended capabilities in this area are listed in Table 4.1 along with the corresponding subroutines from the standard capabilities.

Table 4.1 Discrete Fourier transform subroutines

Transform	Size of data	Extended capabilities	Characteristics	Standard capabilities
Real transform	Power of 2	VRFT1	High performance	RFT
		VRFT2	Memory efficient	
Complex transform	Power of 2	VRFT1	High performance	CFT
		VCFT2	Memory efficient	
Cosine transform	Power of 2	VCOS1	–	FCOST
Sine transform	Power of 2	VSIN1	–	FSINT

4.2Notes

(1) Selection between extended and standard capabilities

The user should use subroutines of the standard capabilities corresponding to routines of the extended capabilities in Table 3.1 to calculate discrete Fourier transforms on a general-purpose computer.

Although subroutines of the extended capabilities can also be used on a general-purpose computer, subroutines of the standard capabilities are more efficient.

(2) High -performance and memory-efficient subroutines

High-performance subroutines are used to calculate multiple sets of transforms. These subroutines are designed for high-speed calculation by saving in work arrays, the rotation factor (trigonometric function table) and the list vector, both of which can be utilized for the series of transforms. Therefore, high-performance subroutines require more space for work arrays VW and IVW.

When only a single transform is calculated, memory-efficient subroutines should be used.

(3) Effective use of single precision arithmetic routine

The algorithm for single precision arithmetic routine takes account of memory interleave number in order to fully extract the potential power of the vector processor. User can inform the memory interleave number to SSL II through following function.

Function	Initial set of memory interleave number
Calling	CALL SETBNK (INTER)

INTER is input parameter to be specified the interleave number.

User's program can obtain the best performance by calling the above routine in advance of calling Fourier transform routine of single precision arithmetic routine.

If user's program does not call the above routine, SSL II assumes that the interleave number is 64.

PART II USAGE OF SUBROUTINES

VALU

A22-71-0202 VALU, DVALU

LU-decomposition of a real general matrix
(blocking LU-decomposition method)

CALL VALU (A, K, N, EPSZ, IP, IS, VW, ICON)

(1) Function

An $n \times n$ nonsingular real matrix A is LU-decomposed using the using the blocking LU-decomposition method (Gaussian elimination method).

$$PA=LU \tag{1.1}$$

P is the permutation matrix which performs the row exchanges required in partial pivoting, L is a lower triangular matrix, and U is a unit upper triangular matrix.$n \geq 1$.

(2) Parameters

A .... Input. Matrix A
Output. Matrices L and U
Refer to Figure VALU-1
A is a two-dimensional array, A (K,N).

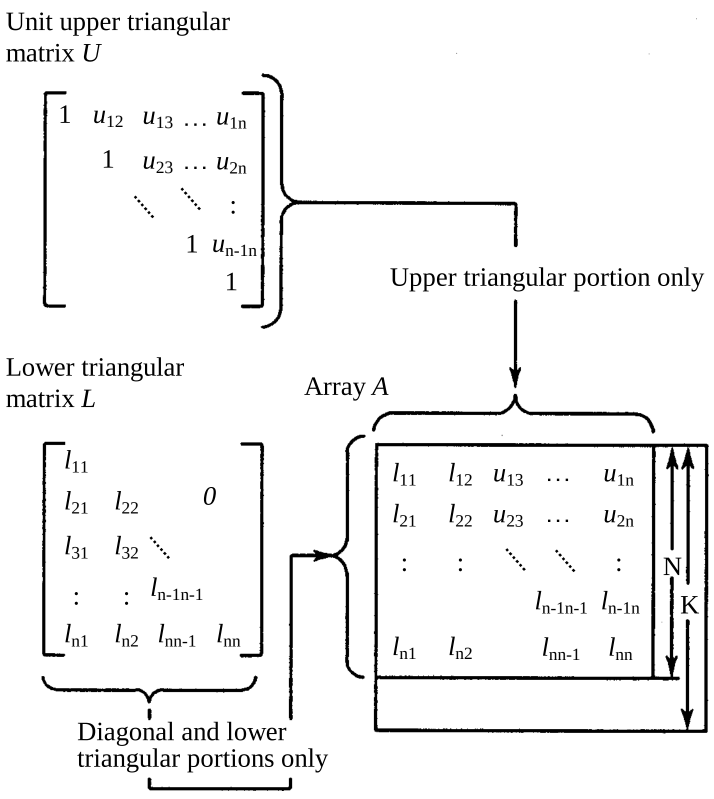

Unit upper triangular matrix *U*
Upper triangular portion only
Lower triangular matrix *L*
Array *A*
N
K
Diagonal and lower triangular portions only

Figure VALU-1 Storage of the elements of *L* and *U* in array *A*

K	Input. Adjustable dimension of array A (≥ N)
N	Input. Order n of matrix *A*
EPSZ....	Input. Tolerance for relative zero test of pivots in decomposition process of *A* (≥ 0.0) When EPSZ is 0.0, a standard value is used. (Refer to Notes.)
IP..........	Output. the transposition vector which indicates the history of row exchanging that occurred in partial pivoting. IP is a one-dimensional array of size n. (Refer to Noter)
IS..........	Output. Information for obtaining the determinant of matrix *A* if the *n* elements of the calculated diagonal of array A are multiplied by IS, the determinant is obtained.
VW.......	Work area. VW is one-dimensional array of size *n*.
ICON ...	Output. Condition code. Refer to Table VALU-1.

Table VALU-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Either all of the elements of some row were zero or the pivot became relatively zero. It is highly probable that the matrix is singular.	Discontinued
30000	K < N, N < 1, or EPSZ < 0.0	Bypassed

(3) Notes

a. Subprograms used

SSL II...... AMACH,MGSSL
FORTRAN intrinsic functions..... ABS

b. Note

(1) If a value is set in the tolerance EPSZ for pivot zero test, this value means the following:

If the selected pivot element is smaller than the product of the largest absolute value of real matrix $A = (a_{ij})$ elements, max $|a_{ij}|$ and EPSZ can be shown as:

$$\left|a_{kk}^{k}\right| \le \max\left|a_{ij}\right|\text{EPSZ}$$

The relative pivot value is assumed to be zero and processing terminates as ICON=20000.

Let u be the unit round-off, and the standard value of EPSZ is 16 u. If the processing is to proceed at a low pivot value, EPSZ will be given the minimum value, but the result is not always guaranteed.

(2) The transposition vector corresponds to the permutation matrix P of LU decomposition in partial pivoting. In this subroutine, the elements of the array A are actually exchanged in partial pivoting. In the J-th stage ($J = 1, \ldots , n$)of decomposition, if the I the row ($I \ge J$)has been selected as the pivotal row the elements of the I-th row and the elements of the Jth row are exchanged. Then, in order to record the history of this exchange ,I is stored in IP (J).

(3) A system of linear equations can be solved by calling subroutine LUX following this subroutine. However ,instead of these subroutines, subroutine VLAX can be normally called to solve such equations in one step.

e. Example

An $n \times n$ matrix is input and LU-decomposition is computed. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),VW(100),IP(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),I=1,N)
      CALL VALU(A,100,N,0.0,IP,IS,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000)GO TO 10
      DET=IS
      DO 20 I=1,N
      DET=DET*A(I,I)
   20 CONTINUE
      WRITE(6,620) (I,IP(I),I=1,N)
      WRITE(6,630) ((I,J,A(I,J),J=1,N),I=1,N)
      WRITE(6,640) DET
      GOTO 10
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT(///10X,'** INPUT MATRIX **'
     */12X,'ORDER=',I5//(10X,4('(',I3,',',I3,')'
     *,E16.8)))
  610 FORMAT('0',10X,'CONDITION CODE =',I5)
  620 FORMAT('0',10X,'TRANSPOSITION VECTOR'
     */(10X,10('(',I3,')',I5)))
  630 FORMAT('0',10X,'OUTPUT MATRICES'
     */(10X,4('(',I3,',',I3,')',E16.8)))
  640 FORMAT('0',10X,
     *'DETERMINANT OF THE MATRIX =',E16.8)
      END
```

(4) Method

The blocking LU-decomposition method is applied by blocking the outer-product Gaussian elimination method.

a. Outer-product Gaussian elimination method

Generally, in exchanging rows using partial pivoting, an $n \times n$ regular real matrix A can be decomposed into the product of a lower triangular matrix L and a unit upper triangular matrix U.

$$PA = LU \qquad (4.1)$$

P is the permutation matrix which performs the row exchanging required in partial pivoting

LU-decomposition is computed by changing $A = (a_{ij})$ as follows:

$A^1 = A \rightarrow, \dots , \rightarrow A^k \rightarrow, \dots , \rightarrow A^n$

$$u_{kj} = a_{kj}^k / a_{kk}^k ,\ j = k,\dots,n \tag{4.2}$$

$$l_{ik} = a_{ik}^k ,i = k,\dots,n \tag{4.3}$$

$$a_{ij}^{k+1} = a_{ij}^k - l_{ij} u_{kj} ,i = k + 1,\dots, n,\ j = k + 1,\dots, n \tag{4.4}$$

The rows are actually exchanged by partial pivoting.

The product of column vectors (4.3) and row vectors (4.2) occur in equation (4.4), and then the rest of the elements will be updated.

b. Blocking method

The outer-product Gaussian elimination method above is determined by the blocked expressions below.

The row and column elements are decomposed with the constant block width bl. The column matrix is taken as L_2^k ,row matrix as U_2^k and the updating part as A^k. They are used for the outer-product Gaussian elimination that is blocked k-th times. (For the location of each matrix, refer to Figure VALU-1.)

The updating corresponding to (4.4) is done in (4.5).

$$A^k = A^k - L_2^k U_2^k \tag{4.5}$$

Before this updating, L_2^k and U_2^k are updated with the expressions below.

First, $\tilde{A}^k$ is decomposed into L_1^k, L_2^k and U_1^k , then U_2^k is updated.

$$\tilde{A}^k = \left(L_1^{kt} ,L_2^{kt} \right)^t U_1^k \tag{4.6}$$

$$U_2^k = \left(L_1^k \right)^{-1} U_2^k \tag{4.7}$$

These expressions are the same as those in the outer-product Gaussian elimination method except that the order is changed.

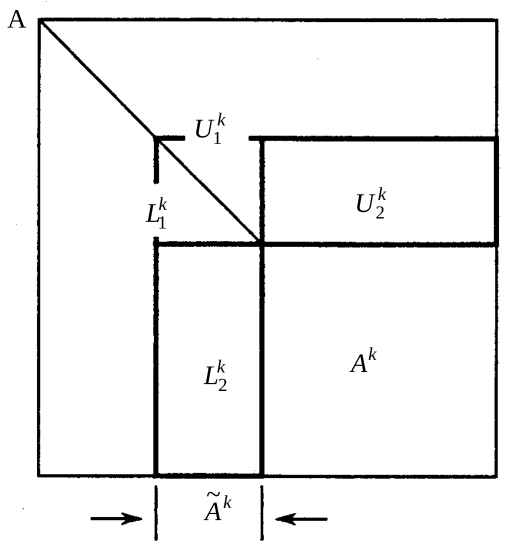


Figure VALU-2 Location of each element in blocked array A

c. Partial pivoting

When matrix A is given as

$$A = \begin{bmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \end{bmatrix}$$

Through the matrix is numerically stable, it can not be LU decomposed. In this state, even if a matrix is numerically stable large errors would occur if LU decomposition were directly computed. So in this subroutine, to avoid such errors partial pivoting with row equilibration is adopted for decomposition.

For more information, see References [9], [10], [11], [12], and [13].

VCFT1

F16-15-0201 VCFT1, DVCFT1

Discrete complex Fourier transform
(high performance,radix 2 FFT)

CALL VCFT1 (A, B, N, ISN, ISW, VW, IVW, ICON)

(1) Function

Given one-dimensional (n-term)complex time-series data $\{x_j\}$, the discrete complex Fourier transform or its inverse transform is calculated by the Fast Fourier Transform (FFT) method suited to a vector processor, where $n = 2^l$ (l is a non-negative integer).

a. Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{-jk}, \quad k = 0,1,...,n-1 \tag{1.1}$$

$$, \omega = \exp(2\pi i/n)$$

b. Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1,2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} a_k \cdot \omega^{jk}, \quad j = 0,1,...,n-1 \tag{1.2}$$

$$, \omega = \exp(2\pi i/n)$$

(2) Parameters

A .... Input. Real part of $\{x_j\}$ or $\{\alpha_k\}$
Output. Real part of $\{n\alpha_k\}$ or $\{x_j\}$
One-dimensional array of size n

B........... Input. Imaginary part of $\{x_j\}$ or $\{\alpha_k\}$
Output. Imaginary part of $\{n\alpha_k\}$ or $\{x_j\}$
One-dimensional array of size n.

N .... Input. Number of terms, n, of the transform

ISN....... Input.Either the transform or the inverse transform is indicated ($\neq 0$).
ISN = +1 for the transform.
ISN = –1 for the inverse transform.
(See Note (3).)

ISW...... Input. Information for controlling the initial state of the transform
ISW = 0 for the first call.
ISW = 1 for the second and subsequent calls.
(See Note (2).)

VW....... Work area. One-dimensional array of size $\max(nl, 1)$.

IVW ..... Work area. One-dimensional array of size $n \cdot \max(l-3, 2)$.

ICON ... Output. Condition code
See Table VCFT1-1.

Table VCFT1-1 Condition codes

Code	Meaning	Processing
0	No error	–
3000	ISN = 0, ISW $\neq$ 0 or 1,or N $\neq 2^l$($l \geq 0$ is an integer)	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: UVTB1,UVF91,UVFA1,UVFB1,UVFX1,UBANK,MGSSL

(2) FORTRAN intrinsic functions: ALOG2,SIN,COS,ATAN,IABS,FLOAT, IAND,MOD

b. Notes

(1) Subroutine use

This subroutine performs high-speed calculation of a complex Fourier transform on a vector processor. On a general-purpose computer, however CFT or CFTM may be more suitable.

This subroutine is used for calculating multiple independent transforms, and because it is a high-performance subroutine, it requires more work array area than VCFT2. If it is difficult to allocate a large work array area, the memory-efficient subroutine VCFT2 should be used, even though it is slower.

(2) Control by ISW

When multiple transforms are calculated, specify ISW = 1 for the second and subsequent subroutine calls. This enables the subroutine to bypass the steps for generating a trigonometric table and a list vector, both of which are needed for the transform, thus improving processing efficiency. The contents of arrays VW and IVW must not be modified when the subroutine is called.

Even if the number of terms, n, of each of the multiple transforms varies, specifying ISW = 1 improves processing efficiency. However, it is desirable to be called so that the maximum number of transforms with the same number of terms are executed consecutively.

When calling this subroutine together with the real Fourier transform subroutine VRFT1, specifying ISW = 1 improves processing efficiency.

(3) ISN specification

Although the ISN parameter is used to specify whether to calculate a transform or an inverse transform, it can also be used as shown below. If the real or imaginary part of $\{x_j\}$ or $\{\alpha_k\}$ is stored at intervals of length I, specify ISN as follows:

For an inverse transform, ISN= + I

For an inverse transform, ISN = − I

The results will also be stored at intervals of length I. Note, however, that when I > 1, specify the size of work array VW to be n (l +2).

When using a vector processor, the interval length I should take the following values in order to access memory more efficiently. (See Example (2)).

For single precision arithmetic (VCFT1), I = 4P + 2, P = 0, 1, 2 , ...

For double precision arithmetic (DVCFT1), I = 2P + 1, P = 1, 2, 3, ...

(4) Work array size conversion table

The table for $16 \le n \le 4096$ is shown as follows:

l	n	VW	IVW
4	16	64 (96)	32
5	32	160 (224)	64
6	64	384 (512)	192
7	128	896 (1152)	512
8	256	2048 (2560)	1280
9	512	4608 (5632)	3072
10	1024	10240 (12288)	7168
11	2048	22528 (26624)	16384
12	4096	49152 (57344)	366864

Figures in ( ) are the sizes when ABS(ISN) > 1.

(5) General definition of Fourier transform

The discrete complex Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \omega^{-ik}, k = 0,1,\ldots,n-1 \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \cdot \omega^{ik}, j = 0,1,\ldots,n-1 \qquad (3.2)$$

where, $\omega = \exp(2\pi i/n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$corresponding to the left hand side of (3.1) or (3.2), respectively. Normalize the results as required.

c. Example

(1) Multiple Fourier transforms

In this example k sets of independent Fourier transforms (with n terms) are calculated.

For $k \le 64$ and $n \le 512$:

```
C     **EXAMPLE**
      DIMENSION A(512,64),B(512,64),
     *          VW(4680),IVW(3072)
      READ(5,500) N,K
      READ(5,510) ((A(I,J),B(I,J),I=1,N),J=1,K)
C
      ISN=1
      ISW=0
      CALL VCFT1(A,B,N,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0) STOP
      ISW=1
      DO 10 J=2,K
      CALL VCFT1(A(1,J),B(1,J),N,ISN,ISW,
     *           VW,IVW,ICON)
   10 CONTINUE
C
      WRITE(6,600) K,N
      DO 20 J=1,K
   20 WRITE(6,610) J,(I,A(I,J),B(I,J),I=1,N)
C
  500 FORMAT(2I5)
  510 FORMAT(2E15.7)
  600 FORMAT(5X,'***',I3,' SET TRANSFORMS'
     *       ' OF',' TERM',I4)
  610 FORMAT(8X,I3,'-TH TRANSFORM'/
     *      (8X,I3,2E16.7))
      STOP
      END
```

(2) Multi-dimensional Fourier transform

In this example a 2-dimensional Fourier transform (with $n1 \times n2$ terms) is calculated.

For $n_1 \leq 512$, $n_2 \leq 64$;

In the example program, the data interval length (the first array declarator of the array) used for the row-wise transform is set at ISN = 514 (= 4p + 2, where p=128).

```
C     **EXAMPLE**
      DIMENSION A(514,64),B(514,64),
     *          VW(4608),IVW(3072)
      READ(5,500) N1,N2
      READ(5,510) ((A(I,J),B(I,J),I=1,N1)
     *                            ,J=1,N2)
C     ---N2 SET TRANSFORMS OF TERM N1---
      ISN=1
      ISW=0
      CALL VCFT1(A,B,N1,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0) STOP
      ISW=1
      DO 10 J=2,N2
      CALL VCFT1(A(1,J),B(1,J),N1,ISN,ISW,
     *           VW,IVW,ICON)
   10 CONTINUE
C     ---N1 SET TRANSFORMS OF TERM N2---
      ISN=514
      CALL VCFT1(A,B,N2,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0) STOP
      DO 20 I=2,N1
      CALL VCFT1(A(I,1),B(I,1),N2,ISN,ISW,
     *           VW,IVW,ICON)
   20 CONTINUE
C
      WRITE(6,600) N1,N2
      DO 30 J=1,N2
   30 WRITE(6,610) J,(I,A(I,J),B(I,J),I=1,N1)
C
  500 FORMAT(2I5)
  510 FORMAT(2E15.7)
  600 FORMAT(5X,'*** 2 DIMENSIONAL TRANSFORM'
     *       ' OF TERM',I4,' BY ',I4)
  610 FORMAT(8X,I3,'-TH COLUMN'//
     *      (8X,I3,2E16.7))
      STOP
      END
```

(4) Method

The discrete complex Fourier transform is calculated using the Fast Fourier Transform method (isogeometric and self-sorting FFTs)suited to a vector processor.

Because of the characteristics of vector processors, this subroutine uses an isogeometric FFT in the single precision arithmetic routine and a self-sorting FFT in the double precision arithmetic routine.

In general, there are two types of FFT algorithms, according to the area used during the computation. One is an in-place type, which uses the input data area only, and the other is a no-in-place type, which uses both the input data area and a work area. The FFT for a general-purpose computer is usually an in-place type, but in this subroutine it is a not-in-place type. Because the not-in-place type FFT can fully utilize parallel processing, it is more suited to a vector processor.

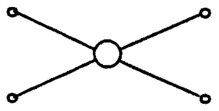
The butterfly operation is the core of the FFT algorithm. The butterfly operation is defined by (4.1) with two arbitrary inputs, a and b, and two outputs, *c* and *d*.

$$c = a + b, \qquad (4.1)$$

$$d = (a - b) \times \omega^{\xi}$$

where a, b, c, d and ω^{ξ} are complex numbers, and ω^{ξ} is a Fourier transform intrinsic coefficient (called rotation factor).

We now introduce the following notation:



(4.2)

In (4.2),a dot (.) represents a data item. The two dots on the left hand side are input (upper dot: *a*, and lower dot: *b*),and the right hand side two dots are output (upper dot: *c*, and lower dot: *d*)

The circle(○) represents the butterfly operation, and the number in the circle, if any represents ξ

Using this notation, the butterfly operations in both isogeometric and self-sorting FFTs are shown in Figures VCFT1-1 and VCFT1-2 (for n=16). In general, assuming $n = 2^l$, an FFT can be composed of l stages of butterfly operations. In the diagram, for example, the FFT is composed of four stages, since $n = 16 = 2^4$. Both types of FFT require the same amount of calculation, but the data transfer pattern at each butterfly stage differs. The characteristics of both FFTs and their adaptability to a vector processor are explained next.

Isogeometric FFT

In this method, the input (and output) transfer patterns are identical during all stages. The algorithm in this method enables a high degree of parallel calculation, and can be accurately described by a program. However, data is in reverse binary order at the end of the butterfly operation, so the data must be permutted. Furthermore ,in a double precision operation, memory conflicts occur because of the characteristics of vector processors.

Self-sorting FFT

In this method, the input transfer patterns are identical during all stages, bat the output transfer patterns vary regularly in each stage. This algorithm enables parallel calculation, just as the isogeometric FFT. A program can made this algorithm by using a list vector. However, in a single precision operation, memory conflicts occur because of the characteristics of vector processors.

This subroutine takes into account the characteristics of both the above methods and their adaptability to a vector processor, to provide higher speed calculations.

Calculation procedure in this subroutine

[Single precision arithmetic routine]

(1) Generation of a trigonometric function table (rotation factor)

All the function values required at every stage are calculated and stored in work array VW.

(2) Generation of list vectors

List vectors, required at the permutation process after the butterfly operation, is calculated and stored in work array IVW.

(3) Butterfly operation

(4) Permutation of data

Steps (1) and (2) above are executed only when this routine is called the first time, i.e., when ISW = 0.

[Double precision arithmetic routine]

(1) Generation of a trigonometric function table (rotation factor)

All the function values required at every stage are calculated and stored in work array VW.

(2) Generation of a list vector

All the list vectors, required at every stage, are calculated and stored in work array IVW.

(3) Butterfly operation

Steps (1) and (2) above are executed only when this routine is called the first time, i.e., when ISW = 0.

For the various FFTs on a vector processor, see reference [5], for the isogeometric FFT,see reference [4], and for the self-sorting FFT, references [2] and [6].

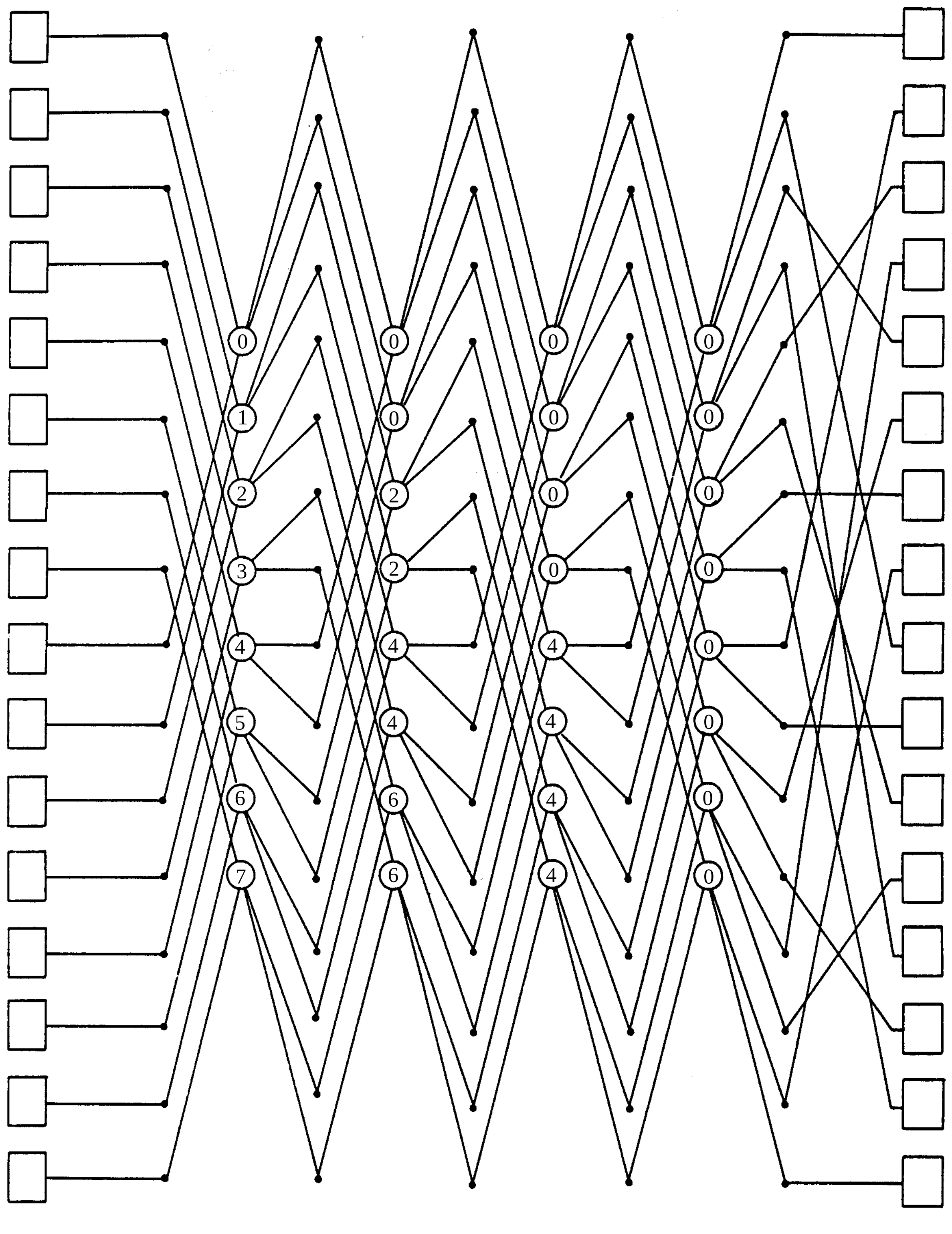


Figure VCFT1-1 Isogeometric FFT flowchart (N=16)

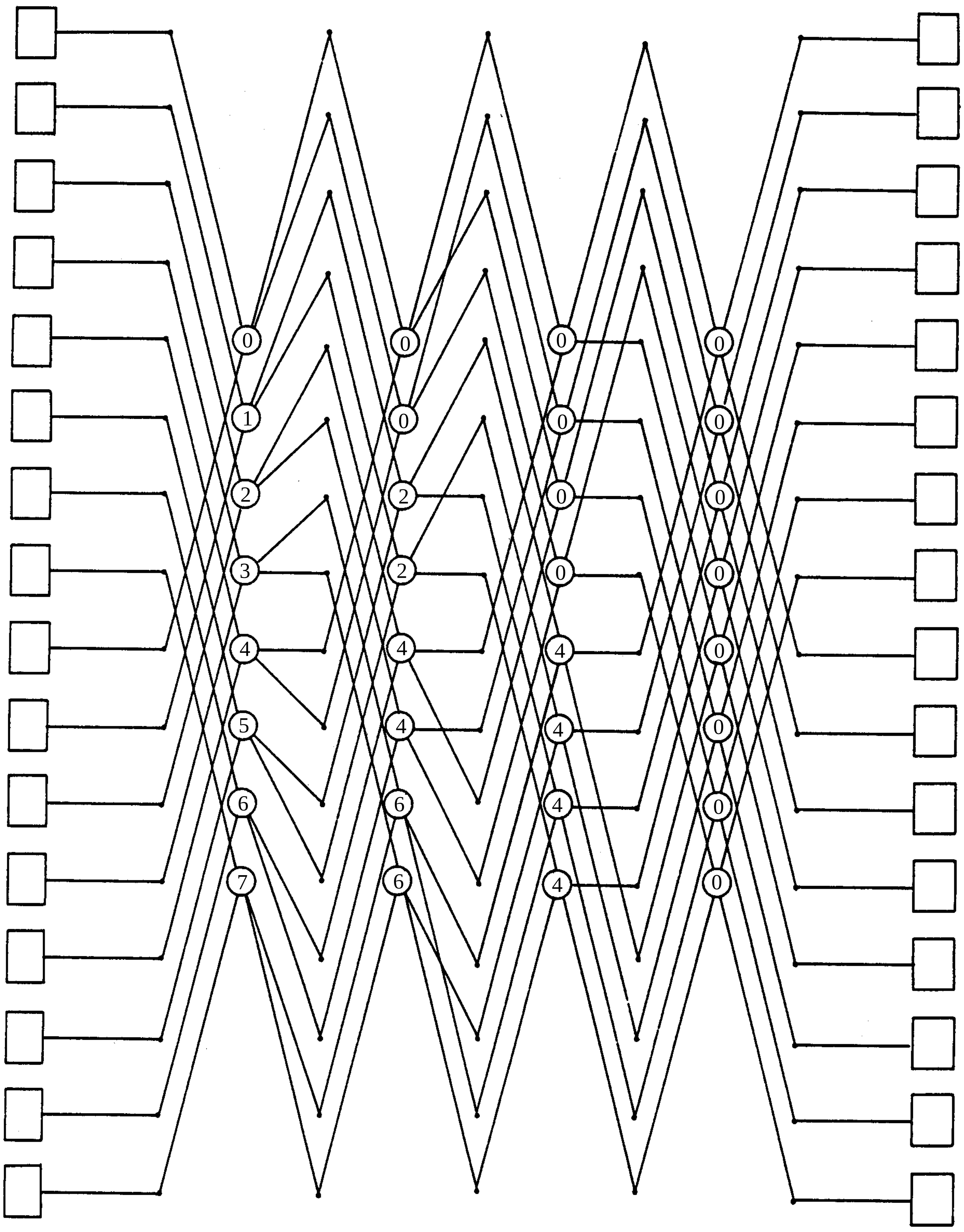


Figure VCFT1-2 Self-sorting FFT flowchart (N = 16)

VCFT2

F16-15-0301 VCFT2, DVCFT2
Discrete complex Fourier transform (memory efficient,radix 2 FFT)
CALL VCFT2(A, B, N, ISN, ISW, VW, IVW, ICON)

(1) Function

Given one-dimensional (n-term) complex time-series data$\{x_j\}$, the discrete complex Fourier transform or its inverse transform is calculated by the Fast Fourier Transform (FFT) method, suited to a vector processor, where $n=2^l$(l is a non-negative integer).

a. Fourier transform

When$\{x_j\}$ is input, the transform defined by (1.1) below is calculated to obtain $\{n\alpha_k\}$.

$$n\alpha_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{-jk}, \quad k = 0,1,...,n-1$$

$$,\omega = \exp(2\pi i/n)$$

b. Fourier inverse transform

When $\{\alpha_k\}$ is input, the transform defined by (1.2) below is calculated to obtain $\{x_j\}$.

$$x_j = \sum_{k=0}^{n-1} \alpha_k \cdot \omega^{jk}, \quad j = 0,1,...,n-1$$

$$,\omega = \exp(2\pi i/n)$$

(2) Parameters

A .... Input. Real part of $\{x_j\}$ or $\{\alpha_k\}$
Output. Real part of $\{n\alpha_k\}$ or $\{x_j\}$
One-dimensional array of size n

B........... Input. Imaginary part of $\{x_j\}$ or $\{\alpha_k\}$
Output. Imaginary part of $\{n\alpha_k\}$ or $\{x_j\}$
One-dimensional array of size n.

N .... Input. Number of terms, *n*, of the transform

ISN....... Input. Either the transform or the inverse transform is indicated (≠ 0).
ISN = +1 for the transform.
ISN = –1 for the inverse transform.
(See Note (3).)

ISW...... Input. Information for controlling the initial state of the transform
ISW = 0 for the first call.
ISW = 1 for the second and subsequent calls.
(See Note (2).)

VW....... Work area. One-dimensional array of size 5*n*

IVW ..... Work area. One-dimensional array of size 3*n*

ICON ... Output. Condition code
See Table VCFT2-1.

(3) Notes

a. Subprogram used

(1) SSL II: UVTB2, UVF92, UVFA2, UVFB2, UVFX2, UBANK, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, SIN, COS, IABS, FLOAT, IAND, MOD

Table VCFT2-1 Condition codes

Code	Meaning	Processing
0	No error	–
3000	ISN = 0, ISW ≠ 0 or 1,or N ≠ 2^l ($l \geq 0$ is an integer)	Bypassed

b. Notes

(1) Subroutine use

This subroutine performs high-speed calculation of a complex Fourier transforms on a vector processor. On a general purpose computer, however, subroutine CFT or CFTM may be more suitable.

This subroutine is suitable for calculating only a single transforms. The work array area is limited to the required minimum; this subroutine is memory efficient. When performing multiple transforms with sufficient work array area available, this high-performance subroutine VCFT1 is more suitable.

(2) Control by ISW

When performing multiple transforms, specify ISW=1 for the second and subsequent subroutine calls. This enables generation of a trigonometric function table to be bypassed, thus improving more processing efficiency.

The contents of array VW and IVW must not be altered when the subroutine is called.

Even if the number of terms, n, in the multiple transforms varies, specifying ISW=1 improves processing efficiency. However, it is desirable to be called so that the maximum number of transforms with the same number of terms are executed consecutively.

When calling this subroutine together with the real Fourier transform subroutine VRFT2, specifying ISW = 1 improves processing efficiency.

(3) ISN specification

Although the ISN parameter is used to indicate whether a transform or an inverse transform is to be calculated, it can also be used as shown below. If the real or imaginary part of $\{x_j\}$ or $\{\alpha_k\}$ is stored at intervals of length I, specify ISN as follows:

For a transform, ISN = + I

For an inverse transform, ISN = – I

The results will also be stored at intervals of length I. Note, however, that when I > 1,specify the size of work array VW to be $7n$.

With a vector processor, the interval length I should take the following values in order to access memory more efficiently. (See Example (2) below.)

For single precision arithmetic (VCFT2), I = 4P + 2, P = 0,1,2, ...

For double precision arithmetic (DVCFT2), I = 2P + 1, P = 1,2,3, ...

(4) Work array size conversion table

The table for $16 \leq n \leq 4096$ is shown as follows:

l	*n*	VW	IVW
4	16	80 (112)	48
5	32	160 (224)	96
6	64	320 (448)	192
7	128	640 (896)	384
8	256	1280 (1792)	768
9	512	2560 (3584)	1536
10	1024	5120 (7168)	3072
11	2048	10240 (14336)	6144
12	4096	20480 (28672)	12288

Figures in ( ) are the sizes when ABS(ISN) > 1.

(5) General definition of Fourier transform

The discrete complex Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$\alpha_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} x_j \cdot \omega^{-jk}, k = 0,1,...,n-1 \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n-1} \alpha_k \cdot \omega^{jk}, j = 0,1,...,n-1 \qquad (3.2)$$

where, $\omega = \exp(2\pi i / n)$

This subroutine calculates $\{n\alpha_k\}$ or $\{x_j\}$ corresponding to the left hand side of (3.1) or (3.2), respectively. Normalize the results as requires.

c. Example

In this example a one-dimensional Fourier transform (with n terms) and its inverse transform are calculated, for $n \leq 1024$.

```
C      **EXAMPLE**
       DIMENSION A(1024),B(1024),VW(5120),IVW(3072)
       READ(5,500) N
       READ(5,510) (A(I),B(I),I=1,N)
C      ---FORWARD TRANSFORM---
       ISN=1
       ISW=0
       CALL VCFT2(A,B,N,ISN,ISW,VW,IVW,ICON)
       IF(ICON.NE.0) STOP
C      ---NORMALIZATION---
       ANOR=1.0/FLOAT(N)
       DO 10 I=1,N
       A(I)=ANOR*A(I)
   10  B(I)=ANOR*B(I)
       WRITE(6,600) N,(I,A(I),B(I),I=1,N)
C      ---BACKWARD TRANSFORM---
       ISN=-1
       ISW=1
       CALL VCFT2(A,B,N,ISN,ISW,VW,IVW,ICON)
       IF(ICON.NE.0)STOP
C
       WRITE(6,610) N,(I,A(I),B(I),I=1,N)
C
  500  FORMAT(I5)
  510  FORMAT(2E15.7)
  600  FORMAT(5X,
     *  '*** FORWARD TRANSFORM OF TERM',
     *  I5//(8X,I3,2E16.7))
  610  FORMAT(5X,
     *  '*** BACKWARD TRANSFORM OF TERM',
     *  I5//(8X,I3,2E16.7))
       STOP
       END
```

(4) Method

The discrete complex Fourier transform is performed using the Fast Fourier Transform (isogeometric and self-sorting FFTs) method, suited to a vector processor.

Because of the characteristics of vector processors, this subroutine uses an isogeometric FFT in the single precision arithmetic routine, and a self-sorting FFT in the double precision arithmetic routine.

For algorithms. see Method for subroutine VCFT1.

Computation procedure in this subroutine

[Single precision arithmetic routine]

(1) Generation of a trigonometric function table (rotation factor)

The function values required for the first stage are calculated and stored in work array VW.

(2) Butterfly operation

(3) Data permutation

(1) above is executed only when this routine is called for the first time, i.e, when ISW = 0.

[Double precision arithmetic routine]

(1) Generation of a trigonometric function table (rotation factor)

The function values required for the first stage are calculated and stored in work array VW.

(2) Butterfly operation

(1) above is executed only when this routine is called for the first time ,i.e, when ISW = 0.

VCOS1

F16-11-0201 VCOS1, DVCOS1
Discrete cosine transform (radix 2 FFT)
CALL VCOS1 (A, N, TAB, VW, IVW, ICON)

(1) Function

Given one-dimensional n+1 sample data $\{x_j\}$ obtained by dividing a 2π period even-function $x(t)$ into n equal parts as defined by the following:

$$x_j = x(\theta j) \qquad j = 0, 1, \ldots, n$$
$$,\theta = \frac{\pi}{n} \tag{1.1}$$

The discrete cosine transform or its inverse transform is calculated by the Fast Fourier Transform (FFT) method suited to a vector processor, where $n = 2l$ (l:a non-negative integer).

a. Cosine transform

When $\{x_j\}$ is input, the transform defined by (1.2) below is calculated to obtain its Fourier coefficient $\{ 2n \cdot a_k \}$

$$2n \cdot a_k = 4 \cdot \sum_{j=0}^{n}{}'' x_j \cos kj\theta \ , k = 0,1,\ldots,n$$
$$,\theta = \frac{\pi}{n} \tag{1.2}$$

Here, $\sum''$ means taking a summation by halving the first and last term.

b. Cosine inverse transform

When $\{a_k\}$ is input, the transform defined by (1.3) is calculated to obtain the Fourier series value $\{4 \cdot x_j\}$.

$$4 \cdot x_j = 4 \cdot \sum_{k=0}^{n}{}'' a_k \cos kj\theta \ , j = 0,1,\ldots,n$$
$$,\theta = \frac{\pi}{n} \tag{1.3}$$

(2) Parameters

A .... Input. $\{x_j\}$ or $\{a_k\}$
Output. $\{2n \cdot a_k\}$ or $\{4 \cdot x_j\}$
One-dimensional array of size n+2
See Figure VCOS1-1.

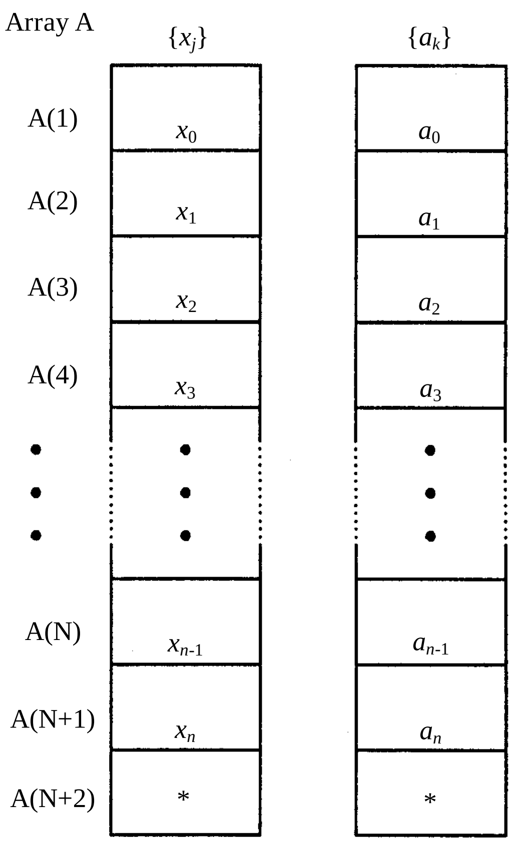
N .... Input. Number of samples minus 1.

TAB ..... Output. Trigonometric function table used in transformation is stored.
One-dimensional array of size 2n+4n

VW....... Work area.
One-dimensional array of size max (n (l + 1) / 2,1)

IVW ..... Work area.
One-dimensional array of size $n \cdot$ max (l – 4, 2) / 2

ICON ... Output. Condition code
See Table VCOS1-1.



Notes:

Same for $\{2na_k\}$ and $\{4x_j\}$
*may be omitted during input.
0.0 is set during output.

Figure VCOS1-1 Data storage method

Table VCOS1-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N ≠ 2^l (l: a non-negative integer)	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: VRFT1, VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK, UVTAB, MGSSL

(2) FORTRAN intrinsic functions:ALOG2, SIN, COS, ATAN, IABS, IAND, MOD, FLOAT

b. Notes

(1) Subroutine use

This subroutine performs high-speed calculation of discrete cosine transforms on a vector processor. On a general-purpose computer, however, subroutine FCOST may be more suitable.

(2) Multiple transforms

Performance of multiple transforms is more efficient, as generation of the trigonometric table and list vector required for transformation is bypassed in the second and subsequent calls of the subroutine. TAB, VW, and IVW arrays must be called without changing their contents.

The contents of TAB, VW, and IVW arrays previously generated are valid even when the number of terms n are different for the multiple transforms. However, it is preferable to call the subroutines in such a way that transforms of equal term numbers are stringed to the maximum extent possible.

(3) Trigonometric table and work array size conversion table

The following table is for16 ≤ n ≤ 4096:

l	n	TAB	VW	IVW
4	16	36	40	16
5	32	68	96	32
6	64	132	224	64
7	128	260	512	192
8	256	516	1152	512
9	512	1028	2560	1280
10	1024	2052	5632	3072
11	2048	4100	11288	7168
12	4096	8196	26624	16384

(4) General definition of discrete cosine transform

The discrete cosine transform and its inverse transform can be defined as in (3.1) and (3.2)

$$a_k = \frac{2}{n}\sum_{j=0}^{n}{}'' x_j \cdot \cos kj\theta, k = 0,1,...,n \qquad (3.1)$$

$$x_j = \sum_{k=0}^{n}{}'' a_k \cdot \cos kj\theta, j = 0,1,...,n \qquad (3.2)$$

This subroutine calculates $\{2n \cdot a_k\}$ or $\{4 \cdot x_j\}$ corresponding to the left-hand side of (3.1) or (3.2), respectively. Therefore, normalize the results as required.

c. Example

In this example, n+1 samples $\{x_j\}$ are input and transformed by this subroutine. Then, the results are normalized and discrete Fourier coefficients $\{a_k\}$ are calculated. Calculation is continued to inverse transformation and $\{x_j\}$ is obtained. The following is an example where $n \leq 512$.

```
C     **EXAMPLE**
      DIMENSION X(514),TAB(1028),VW(2560)
     *         ,IVW(1280)
    1 READ(5,500) N
      IF(N.EQ.0) STOP
      NP1=N+1
      READ(5,501) (X(I),I=1,NP1)
C     COSINE TRANSFORM
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,NP1)
      CALL VCOS1(X,N,TAB,VW,IVW,ICON)
      IF(ICON.NE.0) GO TO 30
C     NORMALIZE
      CN=1.0/(2.0 *FLOAT(N))
      DO 10 K=1,NP1
      X(K)=X(K)*CN
   10 CONTINUE
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,NP1)
C     COSINE INVERSE TRANSRORM
      CALL VCOS1(X,N,TAB,VW,IVW,ICON)
      IF(ICON.NE.0) GO TO 30
C     NORMALIZE
      CN=0.25
      DO 20 K=1,NP1
      X(K)=X(K)*CN
   20 CONTINUE
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,NP1)
      GO TO 1
   30 WRITE(6,603) ICON
      GO TO 1
  500 FORMAT(I5)
  501 FORMAT(6F12.0)
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA')
  603 FORMAT('0',5X,'CONDITION CODE',I8)
      END
```

(4) Method

Consider performing the discrete cosine transform of term number n+1 (= 2^l + 1, l = 0,1, ...) using the Fast Fourier Transform (FFT) method, suited for a vector processor.

The dixcrete cosine transform may be expressed by (4.1) when samples $\{x_j\}$, j=0,1, ... ,n are given.

$$a_j = \frac{1}{n} x_0 + \frac{2}{n} \sum_{k=1}^{n-1} x_k \cdot \cos(kj\theta) + \frac{1}{n}(-1)^j x_n$$

$$, j = 0,1, \dots ,n$$

$$, \theta = \frac{\pi}{n} \qquad (4.1)$$

Now the samples are an even-function, and the relation expressed by (4.2) can be seen by extending to one period.

$$x_{2n-j} = x_j \, , j = 0, 1, \dots , n \qquad (4.2)$$

Therefore, a_0 to a_n can be calculated by extending x_0 to x_n to x_0 to x_{2n-1} and performing the 2n term discrete real Fourier transform. It is well known that use of (4.2) enables efficient performance of the transform.

Perform the following preprocessing on the $\{x_j\}$ samples:

$$d_j = \frac{1}{2} \cdot \left(x_j + x_{n-j}\right) - \sin(j\theta) \cdot \left(x_j - x_{n-j}\right) \qquad (4.3)$$

$$, j = 0, 1, \dots , n-1$$

Substitution of the discrete cosine inverse transform (4.4) in (4.3) will result in (4.5).

$$x_j = \frac{1}{2} a_0 + \sum_{k=1}^{n-1} a_k \cdot \cos kj\theta + \frac{1}{2}(-1)^j a_n ,$$

$$, j = 0, 1, \dots , n-1 \qquad (4.4)$$

$$d_j = \frac{1}{2}a_0 + \sum_{k=1}^{\frac{n}{2}-1}\left[a_{2k}\cdot\cos(2\cdot kj\theta) + (a_{2k+1} - a_{2k-1})\cdot\cos(2\cdot kj\theta)\right] +$$
$$\frac{1}{2}a_n\cdot(-1)^j, \quad j = 0,1,\ldots,n-1 \qquad (4.5)$$

Expression (4.5) is equivalent to the n term discrete real Fourier transform with samples of $\{d_j\}$ and Fourier coefficients of $\{a_{2k}\}$ and $\{a_{2k+1}–a_{2k-1}\}$. Thus, $\{a_k\}$ can be obtained by using the identical equations:

$$\tilde{a}_k = a_{2k}$$

$$\tilde{b}_k = a_{2k-1} - a_{2k-1}$$

after calculating the Fourier coefficients $\{a_k\}$ and $\{b_k\}$ corresponding to the samples $\{d_j\}$. In other words, $\{a_k\}$ is calculated by (4.6)

$$a_1 = \frac{1}{n}\cdot x_0 \frac{2}{n}\sum_{j=1}^{n-1}x_j\cdot\cos(j\theta) - \frac{1}{n}\cdot x_n,\ a_{2k} = \tilde{a}_k,\ a_{2k+1} = a_{2k-1} + \tilde{b}_k,$$
$$k = 1,\ldots,\frac{n}{2}-1 \qquad (4.6)$$

Now, the last expression in (4.6) is a recurrence formula and is unsuitable for a vector processor. Therefore, this subroutine uses a vector-processor-suited algorithm by eliminating recurrence calculations by tracing the preceding expressions backward, taking advantage of the fact that the discrete cosine transform and its inverse transform are identical except for their normalization constants.

Refer to reference [8] for the details on this algorithm.

VGSG2

B62-21-0201 VGSG2, DVGSG2

Eigenvalue and eigenvector for real symmetric matrix (parallel bisection method and inverse iteration method)

CALL VGSG2(A, B, N, M, EPSZ, EPST,E,EV,K, VW, IVW, ICON)

(1) Function

M eigenvalues for general eigenvalue problem expressed by (1.1)for n order real symmetric matrix A and n order positive definite symmetric matrix B are calculated in descending (or ascending) order using the parallel bisection method.

$$Ax = \lambda Bx \tag{1.1}$$

Also, corresponding m eigenvectors $x_1, x_2, \dots, x_m$ are calculated by the inverse iteration method. Eigenvectors must satisfy the relation expressed in (1.2).

$$X^{\mathrm{T}}BX = I \tag{1.2}$$

Here, $X=[x_1, x_2, \dots, x_m]$, with $1 \le m \le n$.

(2) Parameters

A .... Input. Real symmetric matrix A.
Symmetric matrix compression mode.
One-dimensional array of size $n(n+1)/2$.
Contents are not saved after operation.

B........... Input. Positive definite symmetric matrix B.
Symmetric matrix compression mode.
One-dimensional array of size $n(n+1)/2$.
Contents are not saved after operation.

N .... Input. n order of real symmetric matrix A and of positive definite symmetric matrix B.

M.......... Input. m number of eigenvalues to be calculated.
Calculate in descending order when M = $+m$.
Calculate in ascending order when M = $-m$.

EPSZ .... Input. Relative zero test value of the pivot in the LL^{T} decomposition of B. Default value is used when zero or a negative value is specified.
(See note (2).)

EPST.... Input. Upperbound of absolute errors used in convergence test of eigenvalues. Default value is used when a negative value is specified.
(See note (3).)

E........... Output. Eigenvalues.
One-dimensional array of size *m*.
Output are stored in descending order when M is positive and ascending order when M is negative.

EV........ Output. Eigenvectors.
EV (K, *m*) two-dimensional array.
Eigenvector corresponding to eigenvalue E (J) is stored at EV (I, J), I = 1, ... ,N.

K .... Input. Conformation size ($\geq n$) for array EV.

VW ...... Work area. One-dimensional array of size 15*n*.

IVW..... Work area. One-dimensional array of size 7*n*.

ICON ... Output. Condition code.
See Table VGSG2-1.

(3) Notes

a. Subprograms used

(1) SSL II: GSCHL, TRID1, TRBK, GSBK, UVTG2, UCHLS, UVBCT, AMACH, MGSSL

(2) FORTRAN intrinsic functions: IABS, SQRT, SIGN, ABS, AMAX1

b. Notes

(1) This subroutine is functionally equivalent to the subroutine GSEG2, but it performs at high-speed on a vector processor since the parallel bisection method is used. Note that the methods for work area allocation are different in these subroutines.

(2) Default value for the parameter EPSZ is $16 \cdot u$, when the unit round-off is u.

If EPSZ for this subroutine is set at 10^{-s}, condition code (ICON = 29000) is set assuming the pivot is zero and processing is terminated when the pivot value is truncated for more than the s decimal digits during LL^{T} decomposition of the positive definite symmetric matrix B.

Even when the pivot becomes small, calculation can be continued by specifying a small value for EPSZ, but the calculation accuracy cannot be guaranteed.

On the other hand, when the pivot value becomes negative during decomposition, the matrix B is assumed to be negative and calculation is terminated, setting the condition code (ICON = 28000).

(3) The standard value of the parameter EPST in as in (3.1) when *u* is chosen as the round-off unit.

$$\mathrm{EPST} = u \cdot \max\left(|\lambda_{\max}|, |\lambda_{\min}|\right) \qquad (3.1)$$

Here, max and min are the upperbound and lowerbound of the existence range (given by the Gerschgorin′s theorem) of the eigenvalues for $Ax = \lambda Bx$.

When extremely large and small absolute value eigenvalues coexist and a convergence test is performed using (3.1),it is difficult to obtain the smaller eigenvalues of adequate precision. In such cases, setting EPST at a small value(absolute error) enables calculation of smaller eigenvalues with high precision. However, processing speed slows down as the number of iterations increases.

Table VGSG2-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	N = 1	Make E (1) = A(1)/B (1),and EV (1,1)= 1.0/SQRT (B (1)).
15000	Some eigenvectors were not calculated.	Make uncalculated eigenvectors zero vectors.
20000	No eigenvectors were calculated.	Make all eigenvectors zero vectors.
28000	Pivot became negative during LL^T decomposition of *B*. *B* is negative	Discontinued
29000	Pivot became relatively zero during LL^Tdecomposition of *B*. *B* may be singular	Discontinued
30000	M = 0,N < | M | ,or K < N.	Bypassed

c. Example

In this example, *m* eigenvalues and corresponding eigenvectors are calculated in descending (or ascending) order for the general eigenvalue problem $Ax = \lambda Bx$ for *n* order real symmetric matrix *A* and *n* order positive definite symmetric matrix *B*. This example is for is for cases where $n \leq 100$ and $m \leq 20$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(5050),E(20),
     *          EV(102,20),VW(1500),IVW(700)
   10 READ(5,500,END=900) N,M,EPSZ,EPST
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      READ(5,510) (B(I),I=1,NT)
      WRITE(6,600) N,M,EPSZ,EPST
      WRITE(6,610)
      IJ=0
      DO 20 I=1,N
      IJ=IJ+I
   20 WRITE(6,620) I,(A(J),J=IJ-I+1,IJ)
      WRITE(6,630)
      IJ=0
      DO 30 I=1,N
      IJ=IJ+I
   30 WRITE(6,620) I,(B(J),J=IJ-I+1,IJ)
      CALL VGSG2(A,B,N,M,EPSZ,EPST,
     *           E,EV,102,VW,IVW,ICON)
      WRITE(6,640) ICON
      IF(ICON.GE.20000) GO TO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,102,N,MM)
      GO TO 10
  900 STOP
  500 FORMAT(2I5,2E10.2)
  510 FORMAT(5E15.7)
  600 FORMAT('1'//' ***     N=',I5
     *           /' ***     M=',I5
     *           /' ***  EPSZ=',E15.7
     *           /' ***  EPST=',E15.7)
  610 FORMAT('0'//' *** INPUT MATRIX A'/)
  620 FORMAT('0',2X,I3,5E15.7/(6X,5E15.7))
  630 FORMAT('0'//' *** INPUT MATRIX B'/)
  640 FORMAT('0'//' ***  ICON=',I5)
      END
```

This subroutine SEPRT in this example is used for printing eigenvaluer and eigenvectors of real symmetric matrices. For details, see the example of VSEG2 subroutine use.

(4) Methods

Calculate the eigenvalues and eigenvectors using the following procedures for the general eigenvalue problem expressed by (4.1) for *n* order real symmetric matrix *A* and *n* order positive definite symmetric matrix *B*.

$$Ax = \lambda Bx \qquad (4.1)$$

a. Transformation of general eigenvalue problem to standard format

B in (4.1 can be decomposed into a form expressed by (4.2) since it is a positive definite symmetric matrix.

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

Here, *L* is an order lower triangular matrix. Substituting the values LL^{T} of (4.2) for *B* of (4.1) and rearranging it results in expression (4.3).

$$L^{-1} AL^{-T} (L^{T}x) = \lambda(L^{T}x) \tag{4.3}$$

Here, let

$$S = L^{-1} AL^{-T} \tag{4.4}$$

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

Then, S becomes a real symmetric matrix and (4.3) becomes the standard format, expressed as follows:

$$Sy = \lambda y \tag{4.6}$$

b. Real symmetric matrix eigenvalues and eigenvectors

Transform real symmetric matrix *S* by orthogonal similarity transformation into real symmetric tridiagonal matrix, then calculate the eigen value of *T* and corresponding eigenvector y' using the bisection method and inverse iteration method, respectively. y' is inverse transformed further as eigenvector *y* of *S*.

c. Eigenvectors for general eigenvalue problems

The eigenvector *x* in (4.1) is calculated by (4.7),using vector *y* calculated in *b*.

$$x = L^{T}y \tag{4.7}$$

Subroutine GSCHL calculates *a*., slave subroutines of VSEG2 calculate *b*., and GSBK calculates *c*.

VLAX

A22-71-0101 VLAX, VDLAX

A system of linear equations with a real general matrix (blocking LU-decomposition method)

CALL VLAX (A, K, N, B, EPSZ, ISW, IS, VW, IP, ICON)

(1) Function

This subroutine solves a real coefficient linear equations (1.1) using the blocking LU-decomposition (Gaussian elimination method).

$$Ax=b \qquad (1.1)$$

Where A is an $n \times n$ regular real matrix, b is an n- dimensional real constant vector, and x is the n-dimensional solution vector. $n \geq 1$.

(2) Parameters

A	Input. Coefficient matrix A. The contents of A are altered on output. A is a two-dimensional array, A (K, N).
K	Input. Adjustable dimension of array A ($\geq$N).
N	Input. Order n of the coefficient matrix A.
B...........	Input. Constant vector b Output. Solution vector x B is a one-dimensional array of size n
EPSZ....	Input. Tolerance for relative zero test of pivots in decomposition process of A ($\geq$ 0.0). If EPSZ is 0.0, a standard value is used.
ISW......	Input. Control information. When l ($\geq$1) systems of linear equations with the identical coefficient matrix are to be solved, ISW can be specified as follows: ISW=1, the first system is solved. ISW=2, the 2nd to l-th systems are solved. However, only parameter B is specified for each constant vector b of the systems of equations, with the rest unchanged. (See Notes.)
IS..........	Output. Information for obtaining the determinant of matrix A. If the n elements of the calculated diagonal of array A are multiplied by IS, the determinant is obtained.
VW.......	Work area. VW is a one-dimensional array of size n
IP..........	Work area. IP is a one-dimensional array of size n
ICON....	Output. Condition code. Refer to Table VLAX-1.

Table VLAX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Either all of the elements of some row were zero or the pivot became relatively zero. It is highly probable that the coefficient matrix is singular.	Discontinued
30000	K<N, N<1, EPSZ< 0.0 or ISW ≠ 1, 2	Bypassed

(3) Notes

a. Subprogram used

SSL II .... VALU, LUX, AMACH, MGSSL
FORTRAN intrinsic functions........ ABS

b. Notes

(1) The solution *x* obtained by the subroutine may be refined in accuracy by calling subroutine LAXR successively.

(2) If a value is set in the tolerance EPSZ for pivot relative zero test, this value means the following:

If the selected pivot element is smaller than the product of the largest absolute value of real matrix $A=(a_{ij})$ elements, max $|a_{ij}|$ and EPSZ can be shown as follows;

$$|\, a_{kk}^{k} \,| \le \max|a_{ij}|\text{EPSZ}$$

The relative pivot value is assumed to be zero and processing terminates as ICON=20000. The standard value of EPSZ is 16 *u*, *u* being the unit round off. If the processing is to proceed at a lower pivot value, EPSZ will be given the minimum value but the result is not always guaranteed.

(3) When solving successive systems of linear equations with the identical coefficient matrix, computation can be performed by setting ISW=2 after the first system of equations are processed. By setting ISW=2, LU-decomposition of coefficient matrix *A* is bypassed so the computation time is reduced. In this case, the value of IS is the same as when ISW=1.

c. Example

In this example, *l* systems of linear equations in *n* unknown with the identical coefficient matrix are solved. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),B(100),VW(100),IP(100)
      READ(5,500) N
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),I=1,N)
      READ(5,500) L
      M=1
      ISW=1
      EPSZ=1.0E-6
   10 READ(5,510) (B(I),I=1,N)
      WRITE(6,610) (I,B(I),I=1,N)
      CALL VLAX(A,100,N,B,EPSZ,ISW,IS,VW,IP,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,630) (I,B(I),I=1,N)
      IF(L.EQ.M) GOTO 20
      M=M+1
      ISW=2
      GO TO 10
   20 DET=IS
      DO 30 I=1,N
      DET=DET*A(I,I)
   30 CONTINUE
      WRITE(6,640) DET
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1',10X,'** COEFFICIENT MATRIX'
     */12X,'ORDER=',I5/(10X,4('(',I3,',',I3,
     *')',E15.8)))
  610 FORMAT(///10X,'CONSTANT VECTOR'
     */(10X,5('(',I3,')',E16.8)))
  620 FORMAT('0',10X,'CONDITION CODE=',I5)
  630 FORMAT('0',10X,'SOLUTION VECTOR'
     */(10X,5('(',I3,')',E16.8)))
  640 FORMAT(///10X,
     *'DETERMINANT OF COEFFICIENT MATRIX=',
     *E16.8)
      END
```

(4) Method

A system of linear equations

$$Ax = b \tag{4.1}$$

is solved using the following procedure:

a. LU-decomposition of coefficient matrix A ,(blocking LU-decomposition)

The coefficient matrix A is decomposed into the product of a lower triangular matrix L and a unit upper triangular matrix U. To reduce rounding off errors, the partial pivoting is performed in the decomposition process.

$$PA = LU \tag{4.2}$$

P is the permutation matrix which performs the row exchanges required in partial pivoting. Subroutine VALU is used for this operation.

b. Solving $LU = Pb$ (forward and backward substitutions)

Solving equation (4.1) is equivalent to solving the linear equations (4.3).

$$LUx = Pb \tag{4.3}$$

Equation (4.3) is decomposed into two equations

$$Ly = Pb \tag{4.4}$$

$$Ux = y \tag{4.5}$$

Then the solution is obtained using forward substitution and backward substitution. Subroutine LUX is used for these operations.

VLDLX

A22-61-0302 VLDLX, DVLDLX

A systme of linear equations with a positive definite symmetric matrix decomposed into the factors L, D ans L^T

CALL VLDLX (B, FA, N, ICON)

(1) Function

This subroutine solves a system of linear equations with an LDL^T decomposed positive definite symmetric coefficient matrix

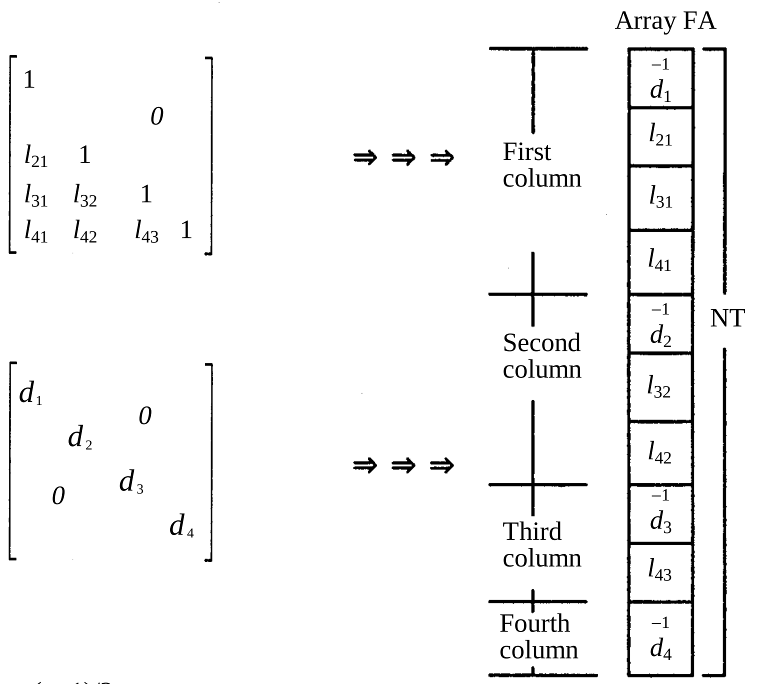
$$LDL^T x = b, \qquad (1.1)$$

where L and D are an $n \times n$ unit lower triangular matrix and a diagonal matrix, respectively, b is an n-dimensional real constant vector, x is an n-dimensional solution vector, and $n \geq 1$.

This subroutine received an LDL^T decomposed matrix from subroutine VSLDL and calculates the solution.

(2) Parameters

B...........	Input. Constant vector b.
	Output. Solution vector x,
	One-dimensional array of size n.
FA	Input. Matrices L and D^{-1}
	One-dimensional array of size $n(n+1)/2$.
	As shown in Figure VLDLX-1, L is input column by column, from the
	first column to the n-th one.
N	Input. Order n of matrices L and D
ICON ...	Output. Condition code
	See Table VLDLX-1.


Array FA
$\begin{bmatrix} 1 & & & \\ l_{21} & 1 & 0 & \\ l_{31} & l_{32} & 1 & \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix}$
$\begin{bmatrix} d_1 & & 0 & \\ & d_2 & & \\ 0 & & d_3 & \\ & & & d_4 \end{bmatrix}$
$\Rightarrow\Rightarrow\Rightarrow$
First column
Second column
Third column
Fourth column
d_1^{-1}
l_{21}
l_{31}
l_{41}
d_2^{-1}
l_{32}
l_{42}
d_3^{-1}
l_{43}
d_4^{-1}
NT

NT=$n\ (n+1)/2$
Correspondence relationship
$l_{ij}\rightarrow$ FA(IJ) IJ=$(2n-j+2)(j-1)/2+(i-j+1)$

Figure VLDLX-1 Storage method of matrices *L* and *D* $^{-1}$

Table VLDLX-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	Coefficient matrix was not positive definite.	Continued
30000	N < 1	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: MGSSL

(2) FORTRAN intrinsic functions: none

b. Notes

(1) A system of linear equations can be solved by calling this subroutine after the VSLDL subroutine. However, subroutine VLSX can usually be called to solve such equations in one step.

c. Example

In this example an LDL^T decomposition is performed for a positive definite symmetric matrix using subroutine VSLDL, then this subroutine is used to solve a system of linear equations. $n\leq$ 100 is assumed.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100),VW(200),IVW(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,640)
      IS=1
      IE=N
      DO 20 J=1,N
      WRITE(6,600) J,(A(I),I=IS,IE)
      IS=IE+1
   20 IE=IE+(N-J)
      CALL VSLDL(A,N,1.0E-6,VW,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      READ(5,510) (B(I),I=1,N)
      CALL VLDLX(B,A,N,ICON)
      WRITE(6,610) ICON
      DET=1.0
      II=1
      NCOL=N
      DO 30 I=1,N
      DET=DET*A(II)
      II=II+NCOL
   30 NCOL=NCOL-1
      DET=1.0/DET
      WRITE(6,620) (B(I),I=1,N)
      WRITE(6,630) DET
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT(' ',I5/(10X,4E16.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(/10X,'SOLUTION VECTOR'
     * //(10X,5E16.8))
  630 FORMAT(/10X,
     *'DETERMINANT OF COEFFICIENT MATRIX='
     *,E16.8)
  640 FORMAT(/10X,'INPUT MATRIX')
      END
```

(4) Method

Suppose that an LDL^T decomposition of a positive definite symmetric matrix A is given as follows:

$$A = LDL^T \tag{4.1}$$

The system of equations,

$$LDL^T x = b \tag{4.2}$$

is solved in the following sequence:

(1) Solve $Ly = b$ (by following substitution)
First, b becomes the initial value of y.

$$y \leftarrow b$$

Next, (4.4) is iterated for $j = 1, 2, \ldots, n-1$.

$$y_i \leftarrow y_i - y_j l_{ij}, i = j+1, j+2, \ldots, n. \tag{4.4}$$

(2) Solve $L^T x = D^{-1}y$ (by backward substitution)
First, $D^{-1}y$ becomes the initial value of x.

$$x \leftarrow D^{-1}y \tag{4.5}$$

Next, (4.6) is iterated for $i = n-1, n-2, \ldots, 1$.

$$x_i \leftarrow x_i - \sum_{j=i+1}^{n} l_{ji} x_j \tag{4.6}$$

For actual calculations, y and x are both obtained on array B, so the substitutions shown above are equivalent to the update procedures for array B.

All the above calculations are vectorized on a vector processor.

VLSX

A22-61-0101 VLSX, DVLSX

A system of linear equations with a positive definite symmetric matrix (modified Cholesky's method)

CALL VLSX(A, N, B, EPSZ, ISW, VW, IVW, ICON)

(1) Function

This subroutine solves a system of linear equations with a real coefficient matrix by using the modified Cholesky's method.

$$Ax = b \qquad (1.1)$$

A is an $n \times n$ positive definite symmetric matrix, b is an n-dimensional real constant vector, and x is an n-dimensional solution vector, and $n \geq 1$.

The function of this subroutine is the same as that of subroutine LSX, but this subroutine stores the coefficient matrix differently, which makes it more suitable for a vector processor.

(2) Parameters

A .... Input. Coefficient matrix A.
The contents are altered during calculation.
One-dimensional array of size $n(n+1)/2$.
The lower triangular portion of the symmetric matrix is stored column by column, from the first column to the n-th column, as shown in Figure VLSX-1.

N .... Input. Order n of the coefficient matrix A

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

EPSZ .... Input. Tolerance for relative zero test (≥ 0.0)
When 0.0, a standard value is assigned.
(See Note (2).)

ISW...... Input. Control information
When solving several sets of equations that have an identical coefficient matrix, specify ISW=1 for the first set of equations, and ISW=2 for the second and subsequent sets. Only parameter B is assigned a new constant vector b.

All the other parameters should be unchanged. (See Note (3).)

VW.......	Work area. One -dimensional array of size 2*n*
IVW	Work area. One-dimensional array of size *n*
ICON ...	Output. Condition code See Table VLSX-1.

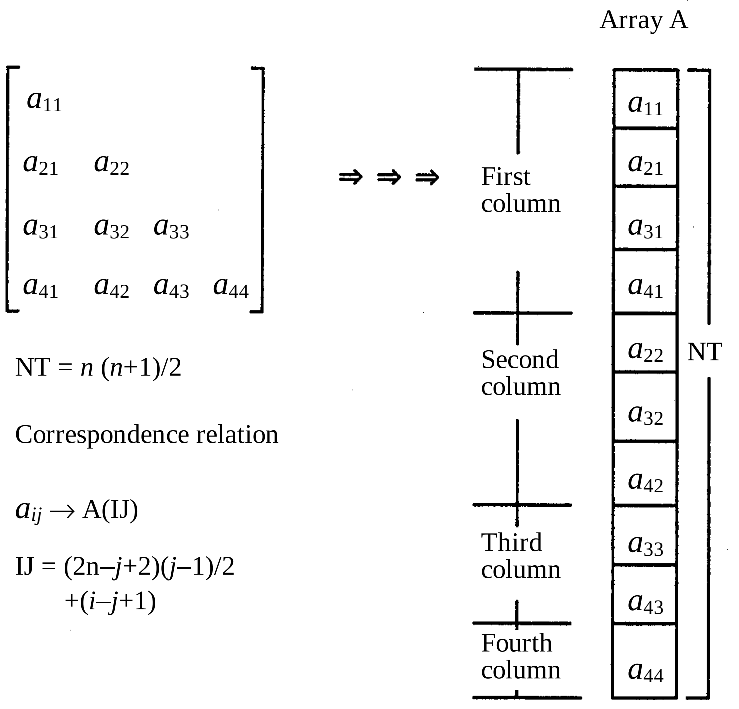


Figure VLSX-1 Storage method of symmetric matrix

Table VLSX-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	Pivot became negative. Coefficient matrix is not positive definite.	Continued
20000	Pivot became smaller than relative zero value. Coefficient matrix might be singular.	Bypassed
30000	N < 1, EPSZ < 0.0, or ISW ≠1 or 2	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: VSLDL, VLDLX, AMACH, MGSSL

(2) FORTRAN intrinsic functions: ABS

b. Notes

(1) This subroutine is provided for high-speed processing on a vector processor by modifying the matrix storage method used in subroutine LSX. Note the differences in the storage methods and calling sequences used by the two subroutines.

(2) If the value 10^{-s} is given as the tolerance for the relative zero test, EPSZ, then the value has the following meaning: if the pivot value loses more than s significant digits during LDL^T decomposition in the modified Cholesky method, the value is assumed to be zero and decomposition is discontinued with ICON=20000. The standard value of EPSZ is normally $16 \cdot u$, where u is the unit round off.

Decomposition can be continued by assigning the smallest value (e.g., 10^{-70}) to EPSZ even when the pivot value becomes smaller than the standard value, although the calculation result may not be as accurate as desired.

(3) When solving several sets of linear equations that have an identical coefficient matrix, specify ISW=2 for subroutine from the second time on. This should reduce the processing time because LDL^T decomposition for the coefficient matrix is bypassed.

(4) If the pivot value becomes negative during decomposition, it means that the coefficient matrix is no longer positive definite. ICON = 10000 is set, but processing continues. Note, however, that the resulting calculation error may be significant, because no pivoting is performed.

(5) To calculate the determinant of the coefficient matrix, multiply all the n diagonal elements of the array A (i.e., diagonal elements of D^{-1}) after calculation is completed, and take the reciprocal of the result.

c. Example

In this example, l sets of n-th order linear equations that have an identical coefficient matrix are solved, where $n \le 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),B(100),VW(200),IVW(100)
      READ(5,500) N
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N
      READ(5,500) L
      ISW=1
      M=1
      EPSZ=1.0E-6
   10 READ(5,510) (B(I),I=1,N)
      CALL VLSX(A,N,B,EPSZ,ISW,VW,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,620) (B(I),I=1,N)
      IF(L.EQ.M) GO TO 20
      M=M+1
      ISW=2
      GO TO 10
   20 DET=1.0
      II=1
      NCOL=N
      DO 30 I=1,N
      DET=DET*A(II)
      II=II+NCOL
   30 NCOL=NCOL-1
      DET=1.0/DET
      WRITE(6,630) DET
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT('1'/10X,'ORDER=',I5)
  610 FORMAT('0',10X,'ICON=',I5)
  620 FORMAT(11X,'SOLUTION VECTOR'
     */(15X,5E16.8))
  630 FORMAT('0',10X,
     *'DETERMINANT OF COEFFICIENT MATRIX='
     *,E16.8)
      END
```

(4) Method

A system of linear equations with a positive definite symmetric coefficient matrix A,

$$Ax = b \tag{4.1}$$

is solved in the following sequence:

a. LDL^T decomposition of coefficient matrix A (modified Cholesky's method)

Using the modified Cholesky method, the coefficient matrix A is decomposed into LDL^T,

$$A = LDL^T \tag{4.2}$$

where L is a unit lower triangular matrix and D is a diagonal matrix. This calculation is performed by subroutine VSLDL.

b. Solution (forward and backward substitutions)

A system of linear equations,

$$LDL^{\mathrm{T}}x = b \qquad (4.3)$$

is solved. This calculation is performed by subroutine VLDLX.

This subroutine is a vector version of subroutine LSX, and is provided for high-speed processing on a vector processor. For further details, see the explanation of subroutine VSLDL and the Method section of VLDLX.

VLTX

A62-11-0101, VLTX, DVLTX

A systme of linear equations with a real tridiagonal matrix (cyclic reduction method)

CALL VLTX(SBD, D, SPD, N, B, ISW, IND, IVW, ICON)

(1) Function

This subroutine solves a tridiagonal matrix equation

$$Ax = b \tag{1.1}$$

using the cyclic reduction method, where A is an $n \times n$ irreducibly diagonally dominant real tridiagonal matrix, b is an n-dimensional real constant vector, and x is the n-dimensional solution vector, and $n \geq 1$.

Matrix A is said irreducibly diagonally dominant if, for the matrix below,

$$A = \begin{bmatrix} d_1 & f_1 & & & & \\ e_2 & d_2 & f_2 & & 0 & \\ & e_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & f_{n-1} \\ & & & & e_n & d_n \end{bmatrix} \tag{1.2}$$

the condition:

$$|\, d_i \,| \geq |\, e_i \,| + |\, f_i \,|, \;\; i=1, 2, \ldots, n \tag{1.3}$$

(where $e_1 = f_n = 0$)

is satisfied, and for at least one i, a strict inequality holds.

(2) Parameters

SBD...... Input. Sub-diagonal portion of coefficient matrix A.
Store as SBD(i)=e_i i=2, 3, ... , n.
See Figure VLTX-1.
The contents are altered during the calculation.
One-dimensional array of size 2n
(See Note (4).)

D .... Input. Diagonal portion of the coefficient matrix *A*.
Store as D(i) = d_i, i = 1, 2, ... , n.
See Figure VLTX-1.
The contents are altered during the calculation.
One-dimensional array of size 2n
(See Note (4).)

SPD...... Input. Super-diagonal portion of coefficient matrix *A*
Store as SPD(i) = f_i, i = 1, 2, ... , n–1.
See Figure VLTX-1.
The contents are altered during the calculation.
One-dimensional array of size 2n.
(See Note (4).)

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

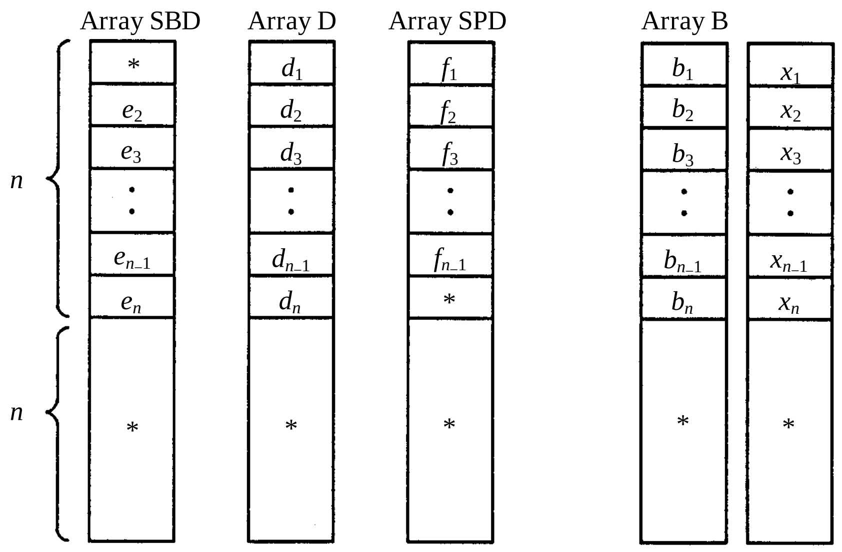
B .... Input. Constant vector *b*.
Store as B(i) = b_i, i = 1, 2, ... , n.
Output. Solution vector *x*.
Store as B(i) = x_i, i = 1, 2, ... , n.
See Figure VLTX-1.
One-dimensional array of size 2n

ISW...... Input. Control information.
When solving several sets of equations that have an identical coefficient matrix, specify ISW = 1 for the first set of the equations, and ISW = 2 for the second and subsequent sets. Only parameter B is assigned a new constant vector *b*. All other parameters should be unchanged.
(See Note (2).)

IND...... Input. Control information.
IND = 0 specifies to check whether the coefficient matrix is irreducibly diagonally dominant. IND = 1 specifies not to check whether the matrix is irreducibly diagonally dominant. Normally, 0 is specified.
(See Note (3).)

IVW..... Work area. One-dimensional array of size [$\log_2 n$] + 10, where [ ] is Gaussian notation.

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



Note:

The portion indicated by an asterisk (*) is used as a work area in this subroutine.

Figure VLTX-1 Storage method of matrix *A*, and vectors *b* and *x*

Table VLTX-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Coefficient matrix is not irreducibly diagonally dominant or the matrix is singular.	Bypassed
30000	N < 1, ISW ≠ 1 or 2 or IND ≠ 0, 1	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: AMACH, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, AMAX1, AMIN1, ABS, FLOAT, MIN0.

b. Notes

(1) This subroutine uses the cyclic reduction method, an algorithm suited to a vector processor. Processing on a vector processor has the following features:

– It is much faster than the Gaussian elimination method used in subroutine LTX.

– Processing time increases almost linearly with N.

– The more diagonally dominant the matrix is, the faster it is processed.

This subroutine is about as accurate as subroutine LTX when processing irreducibly diagonally dominant matrices.

(2) When solving several sets of tridiagonal matrix equations that have an identical coefficient matrix, specify ISW = 2 from the second subroutine call on. This bypasses coefficient matrix elimination, thus speeding up calculation.

(3) If the coefficient matrix is known in advance to be irreducibly diagonally dominant, specify IND = 1 to bypass testing of its irreducible diagonal dominance, thus speeding up calculation. If IND = 1 is specified for a coefficient matrix that is not irreducibly diagonally dominant, the solution may not be as accurate as desired.

(4) If this subroutine is executed with ISW = 1 specified, arrays D(i), SBD(i), and SPD(i), i = 1, 2, ... , n take on the values $1/d_i$, e_i/d_i, and f_i/d_i respectively.

c. Example

In this example, l sets of n-dimensional tridiagonal matrix equations that have an identical coefficient matrix are solved. $n \leq 1000$ is assumed.

```
C     **EXAMPLE**
      DIMENSION SBD(2000),D(2000),SPD(2000),
     *          B(2000),IVW(20)
      READ(5,500) N,L
      IF(N.LE.0) GO TO 30
      NM1=N-1
      READ(5,510) (SBD(I),I=2,N)
      READ(5,510) (D(I),I=1,N)
      READ(5,510) (SPD(I),I=1,NM1)
      WRITE(6,600) N,D(1),SPD(1)
      WRITE(6,610) (I,SBD(I),D(I),SPD(I),I=2,NM1)
      WRITE(6,610) N,SBD(N),D(N)
      ISW=1
      IND=0
      DO 10 II=1,L
      READ(5,510) (B(I),I=1,N)
      WRITE(6,620) (B(I),I=1,N)
      CALL VLTX(SBD,D,SPD,N,B,ISW,IND,IVW,ICON)
      WRITE(6,630) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,640) (B(I),I=1,N)
```

```
    ISW=2
 10 CONTINUE
 30 WRITE(6,650)
    STOP
500 FORMAT(2I5)
510 FORMAT(5E14.7)
600 FORMAT('1',20X,
   *       'LINEAR EQUATIONS (TRIDIAGONAL)',
   *       /' ',20X,'ORDER= ',I5,/,
   *       /' ',25X,'COEFFICIENT MATRIX',/,
   *       /' ','(',4X,'1)',21X,2(2X,E14.7))
610 FORMAT((' ','(',I5,')',5X,3(2X,E14.7)))
620 FORMAT(/' ',78('*'),//,' ',25X,
   *'CONSTANT VECTOR',//,(' ',5(1X,E15.7)))
630 FORMAT(/' ','CONDITION CODE OF VLTX= ',I5)
640 FORMAT(/' ',25X,'SOLUTION VECTOR',//,
   *(' ',5(1X,E14.7)))
650 FORMAT(//' ',30X,'** NORMAL END **')
    END
```

(4) Method

Consider the use of cyclic reduction method to solve a tridiagonal matrix equation (4.1) which is normalized so that the diagonal elements of its coefficient matrix are all 1.

$$Ax = b \qquad (4.1)$$

where:

$$A = \begin{bmatrix} 1 & f_1 & & & & \\ e_2 & 1 & f_2 & & 0 & \\ & e_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & f_{n-1} \\ & & & & e_n & 1 \end{bmatrix}$$

The general form of the cyclic reduction method for (4.1) is explained first, followed by an explanation of the possible improvement in the case where the matrix is diagonally dominant to sufficient extent.

a. General form of cyclic reduction method

This method is used basically to produce a system of tridiagonal matrix equations with respect to even-numbered unknowns, by applying a proper elimination process to the tridiagonal matrix equations being solved.

Suppose n is an odd number for convenience, and select three rows next to each other in (4.1) as follows:

$$\begin{aligned}
e_{i-1}x_{i-2} + x_{i-1} + f_{i-1} &= b_{i-1} \\
e_i x_{i-1} + x_i + f_i x_{i+1} &= b_i \\
e_{i+1}x_i + x_{i+1} + f_{i+1}x_{i+2} &= b_{i+1}
\end{aligned} \qquad (4.2)$$

From the three equations above, x_{i-1} and x_{i+1} can be eliminated in the following way. First, multiply the first equation by ($-e_i$) and the third equation by ($-f_i$),

$$e_i^{(1)} x_{i-2} + x_i + f_i^{(1)} x_{i+2} = b_i^{(1)} \qquad (4.3)$$

where
$$\begin{aligned}
e_i^{(1)} &= e_{i-1}\, e_i\, t_i \\
f_i^{(1)} &= f_i f_{i+1}\, t_i \\
b_i^{(1)} &= (e_i\, b_{i-1} + f_i\, b_{i+1} - b_i)\, t_i \\
t_i &= \frac{1}{e_i f_{i-1} + e_{i+1} f_i - 1}
\end{aligned}$$

Considering only the even-numbered i s in (4.3) i.e., i=2, 4, ... , n–1 (,where $x_0 = x_{n+1} = 0$), the following tridiagonal matrix equation of order [n/2] is obtained.

$$\begin{bmatrix}
1 & f_2^{(1)} & & & & & & \\
e_4^{(1)} & 1 & f_4^{(1)} & & & 0 & & \\
 & e_6^{(1)} & \cdot & \cdot & & & & \\
 & & \cdot & \cdot & \cdot & & & \\
 & & & \cdot & \cdot & \cdot & & \\
 & & & & \cdot & \cdot & \cdot & \\
 & 0 & & & & \cdot & \cdot & f_{n-3}^{(1)} \\
 & & & & & & e_{n-1}^{(1)} & 1
\end{bmatrix}
\begin{bmatrix} x_2 \\ x_4 \\ x_6 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-3} \\ x_{n-1} \end{bmatrix}
=
\begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ b_6^{(1)} \\ \cdot \\ \cdot \\ \cdot \\ b_{n-3}^{(1)} \\ b_{n-1}^{(1)} \end{bmatrix} \qquad (4.4)$$

This operation for reducing the order of equations by half is called reduction. Once x_2, x_4, ... , x_{n-1} are obtained from (4.4), the odd-numbered unknowns can be obtained by substituting them into (4.1), resulting in

$$x_{i-1} = b_{i-1} - e_{i-1}x_{i-2} - f_{i-1}x_i,\ i = 2, 4, \ldots ,.n+1 \qquad (4.5)$$

This is called back ward substitution.

The calculation of $e_i^{(1)}$ through t_i in (4.3), and the calculation of (4.5) can be performed in parallel, and there is no recurrence relation, unlike the Gaussian elimination method. Therefore, the above calculations can be efficiently performed by a vector processor. Thus cyclic reduction is faster than Gaussian elimination on a vector processor.

Next, suppose n is an even number. Then n–1 is an odd number, so the upper limit of i applied in (4.3) is n–2. In return, by using

$$\begin{aligned}
e_{n-1}x_{n-2} + x_{n-1} + f_{n-1}x_n &= b_{n-1} \\
e_n x_{n-1} + x_n &= b_n
\end{aligned} \qquad (4.6)$$

the following equation from which x has been eliminated is added.

$$e_n^{(1)} x_{n-2} + x_n = b_n^{(1)} \qquad (4.7)$$

where $e_n^{(1)} = e_{n-1}\, e_n\, t_n$

$b_n^{(1)} = (e_n\, b_{n-1} - b_n)\, t_n$

$t_n = \dfrac{1}{e_n f_{n-1} - 1}$

Even when n is an even number, the original tridiagonal matrix equations can be reduced to tridiagonal matrix equations of order [n/2], as in (4.4)

The above reduction operation can be applied again to the tridiagonal matrix equations of order [n/2] to reduce the order by half again. By repeating this operation as many times as required, an equation of order 1 will be obtained, and in can be solved for the one corresponding unknown. Then, backward substitution can be repeated to obtain a solution to (4.1). The number of repeated operations required to reduce the equation to order 1 is [$\log_2 n$].

b. Incomplete termination of reduction

By continuing the above reduction operation, the matrix will approach diagonal dominance under certain conditions (i.e., off-diagonal elements will become as small compared to the diagonal elements). Then, some of the components of the modified right-hand-side vector will converge to some of the components of the solution vector. Therefore, if reduction operation is stopped at the proper time and backward substitution is performed, processing efficiency will be improved. The termination of reduction operation before reaching equations of order 1 is called incomplete termination of reduction.

One of the conditions sufficient to enable incomplete termination is that the relation,

$$|\, e_i \,|, |\, f_i \,| < 1/2 \qquad (4.8)$$

is satisfied in the normalized equations given in (4.1). This subroutine, when the above relation is satisfied, determines the number of reductions before incomplete termination takes place, as follows

Under condition (4.8), the lower limit of the rate at which the off-diagonal elements are approaching 0 can be examined. For that purpose, we introduce the value

$$e = \max_i \left(|e_i|, |f_i| \right) < 1/2 \qquad (4.9)$$

and consider a matrix of (4.1) whose e_i and f_i elements are all replaced by e. The ratio of the diagonal elements to e, | 1/e |, is greater that 2, so we represent it as

$$|\, 1/e \,| = 2 + \varepsilon^{(0)} \qquad (\varepsilon^{(0)} > 0) \qquad (4.10)$$

By the first reduction operation, off-diagonal elements become

$$e' = \frac{e^2}{2e^2 - 1} \tag{4.11}$$

Its ratio to diagonal elements is then

$$| 1/e' | = 2 + \varepsilon^{(1),} \text{ where } \varepsilon^{(1)} = 4\varepsilon^{(0)} + (\varepsilon^{(0)})^2. \tag{4.12}$$

The *k*-th ratio is

$$| 1/e^{(k)} | =2 + \varepsilon^{(k)}. \tag{4.13}$$

Therefore

$$\varepsilon^{(k+1)} = 4\varepsilon^{(k)} + (\varepsilon^{(k)})2, k = 1, 2, ... \tag{4.14}$$

Thus, when $\varepsilon^{(0)} <1$, the matrix approaches diagonal dominance linearly but once $\varepsilon^{(k)} >1$, quadratically.

This subroutine estimates in advance the smallest integer k for which

$$\varepsilon^{(k)} \geq 1/u \text{ } (u\text{: unit round off}), \tag{4.15}$$

and then repeats reduction operations k times before performing the substitution. If $k> [\log_2 n]$, however, incomplete termination of reduction will not occur.

The greater the value n is and the smaller the value $\max_i (| e_i |, | f_i |)$is, the greater efficiency can be gained by incomplete termination.

For further details, see References [1], [3] and [7].

VLTX1

A62-21-0101 VLTX1, DVLTX1

A system of linear equations with a real constant-tridiagonal matrix (Dirichlet type and cyclic reduction method)

CALL VLTX1 (D, SD, N, B, ISW, VW, IVW, ICON)

(1) Function

This subroutine solves a real tridiagonal matrix equation

$$Ax = b \tag{1.1}$$

using cyclic reduction, where A is an $n \times n$ irreducibly diagonally dominant real tridiagonal matrix of the form:

$$A = \begin{bmatrix} d & e & & & & \\ e & d & e & & 0 & \\ & e & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & e \\ & & & & e & d \end{bmatrix} \tag{1.2}$$

$d \neq 0, \, | \, d \, | \geq 2 \, | \, e \, |$

Where b is a n-dimensional real constant vector, and x is the n-dimensional solution vector, for $n \geq 1$.

This subroutine restricts the coefficient matrix to the form (1.2) in order to achieve high performance, while subroutine VLTX processes a general tridiagonal matrix.

(2) Parameters

D .... Input. Diagonal element d

SD .... Input. Off-diagonal element e

N .... Input. Order of the coefficient matrix A

B........... Input. Constant vector b
Store as B(i) = b_i , i = 1, 2, ..., n.
Output. Solution vector x
Store as B(i) = x_i , i = 1, 2, ..., n.

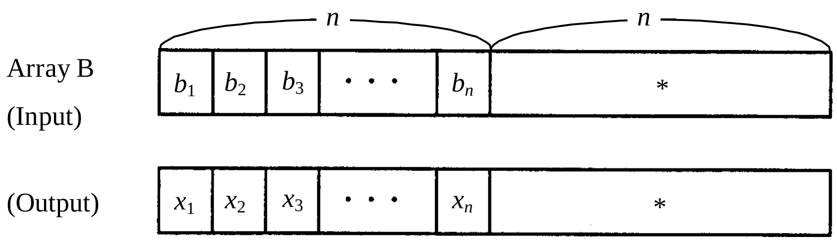
See Figure VLTX1-1.
One-dimensional array of size $2n$

ISW...... Input. Control information
When solving several sets of equations that have an identical coefficient matrix, specify ISW=1 for the first set of equations, and ISW=2 for the second and subsequent sets. Only parameter B is assigned a new constant vector b. All other parameters should be unchanged.
(See Note (3).)

VW....... Work area. One-dimensional array of size $2([\log_2 n]) + 1)$, where [ ] is Gaussian notation.

IVW ..... Work area. One-dimensional array of size $2([\log_2 n] + 1) + 10$

ICON ... Output. Condition code
See Table VLTX1-1.


n
n
Array B
(Input)
b_1 b_2 b_3 • • • b_n *
(Output)
x_1 x_2 x_3 • • • x_n *

Note:

The portion indicated by an asterisk (*) is used as a work area in this subroutine.

Figure VLTX1-1 Storage method of vectors *b* and *x*

Table VLTX1-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Coefficient matrix is not irreducibly diagonally dominant.	Bypassed
30000	N<1, or ISW ≠1, 2	Bypassed

(3) Notes

a Subprograms used

(1) SSL II: AMACH, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, FLOAT, ABS, MIN0

b. Notes

(1) This subroutine uses the cyclic reduction method, an algorithm suited to a vector processor. Processing on a vector processor has the following features:

– It is much faster than the Gaussian elimination method used in subroutine LTX.

– Processing time increases almost linearly with N.

– The more diagonally dominant the matrix is, the faster it is processed.

This subroutine is about as accurate as subroutine LTX or LSTX when processing irreducibly diagonally dominant matrices.

(2) The coefficient matrix (1.2) arises from the discretization of simple Dirichlet boundary value problems.

(3) When solving several sets of tridiagonal matrix equations that have an identical coefficient matrix specify ISW=2 from the second subroutine call on. This bypasses coefficient matrix elimination, thus speeding up calculation.

c. Example

In this example, *l* sets on *n*-dimensional linear equations that have an identical coefficient matrix are solved, for $n \leq 1000$.

```
C     **EXAMPLE**
      DIMENSION B(2000),VW(20),IVW(30)
      READ(5,500) N
      READ(5,510) D,SD
      WRITE(6,600) N,D,SD
      READ(5,500) L
      ISW=1
      DO 10 II=1,L
      READ(5,510) (B(I),I=1,N)
      WRITE(6,610) (B(I),I=1,N)
      CALL VLTX1(D,SD,N,B,ISW,VW,IVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,630) (B(I),I=1,N)
      ISW=2
   10 CONTINUE
```

```
    WRITE(6,640)
    STOP
500 FORMAT(I5)
510 FORMAT(5E14.7)
600 FORMAT('1',
   *  20X,'LINEAR EQUATIONS (TRIDIAGONAL)'
   *  /' ',20X,'ORDER= ',I5/
   *  /' ',25X,'COEFFICIENT MATRIX'/
   *  /' ',30X,'D =',E14.7/
   *  /' ',30X,'SD=',E14.7)
610 FORMAT(/' ',78('*')//' ',
   *  25X,'CONSTANT VECTOR'//
   *  (' ',5(1X,E14.7)))
620 FORMAT(/' ','CONDITION CODE OF VLTX1= ',
   *       I5)
630 FORMAT(/' ',25X,'SOLUTION VECTOR'//
   *  (' ',5(1X,E14.7)))
640 FORMAT(//' ',30X,'** NORMAL END **')
    END
```

(4) Method

Cyclic reduction can be used to solve tridiagonal matrix equation (4.1), which is normalized so that the off diagonal elements of its coefficient matrix are all 1.

$$Ax = b \tag{4.1}$$

where

$$A = \begin{bmatrix} d & 1 & & & & \\ 1 & d & 1 & & \mathit{0} & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & \mathit{0} & & \cdot & \cdot & 1 \\ & & & & 1 & d \end{bmatrix} \tag{4.2}$$

$|d| \geq 2$

The cyclic reduction method for a general tridiagonal matrix is explained in Method for subroutine VLTX, but for the restricted form as (4.2), the amount of calculation can be greatly reduced. The reduction of the coefficient matrix at each step requires only a few scalar calculations, and most of the calculation involves reduction of the right hand side vector.

Here, the cyclic reduction method for coefficient matrix (4.2) is explained. When the matrix is diagonally dominant to sufficient extent, reduction operation will be incompletely terminated. For further details of it, see the explanation of subroutine VLTX.

Suppose *n* is an odd number, and select three rows next to each other in (4.1) as follows:

$$\begin{aligned} x_{i-2} + dx_{i-1} + x_i \quad &= b_{i-1} \\ x_{i-1} + dx_i + x_{i+1} \quad &= b_i \\ x_i + dx_{i+1} + x_{i+2} &= b_{i+1} \end{aligned} \tag{4.3}$$

x_{i-1} and x_{i+1} can be eliminated from the three above equations in the following way. First, multiply the second equation by ($-d$), and add to its result the first and the third equations to obtain (4.4).

$$x_{i-2} + d^{(1)}x_i + x_{i+2} = b_i^{(1)} \tag{4.4}$$

where $d^{(1)} = 2-d^2$

$b_i^{(1)} = b_{i-1}b_{i+1} - db_i$

Considering only the even-numbered i's in (4.4), i.e., i = 2, 4, ..., n–1 (, where $x_0 = x_{n+1} = 0$), the following tridiagonal matrix equation of order [n/2]is obtained.

$$\begin{bmatrix} d^{(1)} & 1 & & & & \\ 1 & d^{(1)} & 1 & 0 & & \\ & 1 & d^{(1)} & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & d^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ x_6 \\ \cdot \\ x_{n-3} \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ b_6^{(1)} \\ \cdot \\ b_{n-3}^{(1)} \\ b_{n-1}^{(1)} \end{bmatrix} \tag{4.5}$$

Once x_2, x_4, ..., x_{n-1} are obtained from (4.5), the odd-numbered unknowns can be obtained by substituting them into (4.1), resulting in

$$x_{i-1} = (b_{i-1} - x_{i-2} - x_i)/d \tag{4.6}$$

i = 2, 4, ... , n+1

The calculations for $b_i^{(1)}$ and (4.6) can be performed very efficiently on a vector processor.

Next, suppose n is an even number. Then n–1 is an odd number, so the upper limit of i applied in (4.4) is n–2. In return, using

$$\begin{aligned} x_{n-2} + dx_{n-1} + x_n &= b_{n-1} \\ x_{n-1} + dx_n &= b_n, \end{aligned} \tag{4.7}$$

The following equation from which x_{n-1} has been eliminated is added:

$$x_{n-2} + c^{(1)}x_n = b_n^{(1)}, \tag{4.8}$$

where $c^{(1)} = 1 - d^2$

$b_n^{(1)} = b_{n-1} - db_n$.

Then, (4.5) becomes

$$\begin{bmatrix} d^{(1)} & 1 & & & & \\ 1 & d^{(1)} & 1 & 0 & & \\ & 1 & d^{(1)} & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ x_6 \\ \cdot \\ x_{n-2} \\ x_n \end{bmatrix} = \begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ b_6^{(1)} \\ \cdot \\ b_{n-2}^{(1)} \\ b_n^{(1)} \end{bmatrix} \tag{4.9}$$

Thus, the problem can still be reduced to a tridiagonal matrix equation of order [*n*/2]. The first reduction operation has been explained. By repeating this operation as many as required, an equation of order 1 can be obtained. The coefficient matrix at each reduction step contains all 1 in its off-diagonal elements, and its diagonal elements all have the same value except for the last element. The last diagonal element is handled differently because the order of the coefficient matrix alternates between odd and even at reduction step.

In conclusion, general step of the reduction operation can be described as follows: We represent the equation which is going to be reduced by (4.10), and suppose that it is of order *n*.

$$\begin{bmatrix} d & 1 & & & & \\ 1 & d & 1 & 0 & & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdot \\ b_{n-1} \\ b_n \end{bmatrix} \tag{4.10}$$

The reduction operation produces, from (4.10), an equation with even-numbered unknowns. To do that the processing explained above is performed according to whether *n* is odd or even. The resulting reduced equation can be written as shown in (4.11).

$$\begin{bmatrix} d^{(1)} & 1 & & & & \\ 1 & d^{(1)} & 1 & 0 & & \\ & 1 & d^{(1)} & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ x_6 \\ \cdot \\ x_{l-2} \\ x_l \end{bmatrix} = \begin{bmatrix} b_2^{(1)} \\ b_4^{(1)} \\ b_6^{(1)} \\ \cdot \\ b_{l-2}^{(1)} \\ b_l^{(1)} \end{bmatrix} \tag{4.11}$$

where $d^{(1)} = 2-d^2$

$b_{2r}^{(1)} = b_{2r-1}+b_{2r+1} - db_{2r}$,

$r = 1, 2, \ldots, [n/2] -1$

when n is odd

$$l = n-1,\ c^{(1)} = 1 - d^2 + d/c$$

$$b_l^{(1)} = b_{n-2} - db_{n-1} + (d/c)\, b_n$$

when n is even

$$l = n,\ c^{(1)} = 1 - dc, \tag{4.12}$$

$$b_l^{(1)} = b_{n-1} - db_n$$

Repeating the reduction of (4.10) into (4.11) as may times as required, an equation of order 1 can be obtained. By solving the equation, and using backward substitution, the final solution can be obtained.

As explained above, this subroutine requires few calculations to reduce a coefficient matrix. Most of the calculations involve reduction of the right hand side vector and backward substitution, both of which can be vectorized on a vector processor.

VLTX2

A62-31-0101 VLTX2, DVLTX2

A system of linear equations with a real constant tridiagonal matrix (Neumann type and cyclic reduction method)

CALL VLTX2 (D, SD, N, B, ISW, IND, VW, IVW, ICON)

(1) Function

This subroutine solves a real tridiagonal matrix equation

$$Ax = b \tag{1.1}$$

using cyclic reduction, where A is an $n \times n$ irreducibly diagonally dominant real tridiagonal matrix of either form below:

$$\begin{bmatrix} d & 2e & & & & \\ e & d & e & 0 & & \\ & e & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & e \\ & & & & e & d \end{bmatrix} d \neq 0, |d| \geq 2|e| \tag{1.2}$$

$$\begin{bmatrix} d & e & & & & \\ e & d & e & 0 & & \\ & e & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & e \\ & & & & 2e & d \end{bmatrix} d \neq 0, |d| \geq 2|e| \tag{1.3}$$

$$\begin{bmatrix} d & 2e & & & & \\ e & d & e & 0 & & \\ & e & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & e \\ & & & & 2e & d \end{bmatrix} d \neq 0, |d| \geq 2|e| \tag{1.4}$$

In equation (1.1), b is an n-dimensional real constant vector, and x is the n-dimensional solution vector, and $n \geq 1$.

This subroutine restricts the coefficient matrix to the above forms to achieve high performance, while subroutine VLTX processes general tridiagonal matrices.

(2) Parameters

D .... Input. Diagonal element d

SD .... Input. Off-diagonal element e

N .... Input. Order n of the coefficient matrix A

B .... Input. Constant vector b
Store as B (i) = b_i, i = 1, 2, ... , n.
Output. Solution vector x
Store as B (i) = x_i, i = 1, 2, ... , n.
See Figure VLTX2-1.
One dimensional array of size $2n + [\log_2 n]$

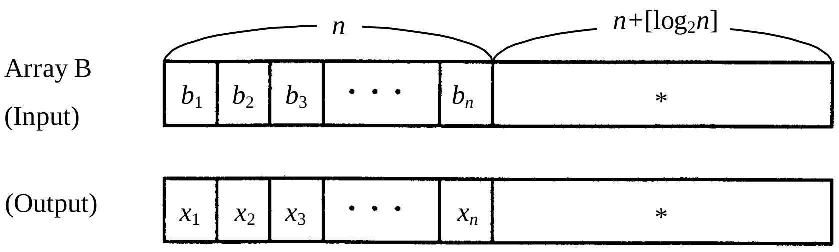
ISW...... Input. Control information
When solving several sets of equations that have an identical coefficient matrix, specify ISW=1 for the first set of equations, and ISW=2 for the second and subsequent sets. Only parameter B is assigned a new constant vector b. All other parameters should be unchanged. (See Note (3).)

IND ...... Input. Control information to specify the form of the coefficient matrix.
IND=1 for (1.2)
IND=2 for (1.3)
IND=3 for (1.4)

VW....... Work area. One-dimensional array of size $2([\log_2 n] + 1)$ where [ ] is Gaussian notation

IVW ..... Work area. One-dimensional array of size $2([\log_2 n] + 1) + 10$

ICON ... Output. Condition code
See Table VLTX2-1


n
$n+[\log_2 n]$
Array B
(Input)
(Output)
b_1 b_2 b_3 $\cdots$ b_n $*$
x_1 x_2 x_3 $\cdots$ x_n $*$

Note:

The portion indicated by an asterisk (*) is used as a work area in this subroutine.

Figure VLTX2-1 Storage method of vectors *b* and *x*

Table VLTX2-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Coefficient matrix is not irreducibly diagonally dominant.	Bypassed
30000	N < 1, IND ≠ 1, 2, or 3, ISW ≠ 1 or 2	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: AMACH, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, FLOAT, ABS, MIN0

b. Notes

(1) This subroutine uses the cyclic reduction-method, an algorithm suited to a vector processor. Processing on a vector processor has the following features:

– It is much faster than Gaussian elimination method used in subroutine LTX.

– Processing time increases almost linearly with N.

– The more diagonally dominant the matrix is, the faster it is processed.

This subroutine is about as accurate as subroutine LTX when processing irreducibly diagonally dominant matrices.

(2) The coefficient matrices in (1.2) to (1.4) arises from the discretization of simple Neumann boundary value problems.

(3) When solving several sets of tridiagonal matrix equations that have an identical coefficient matrix, specify ISW=2 from the second routine call on. This bypasses coefficient matrix elimination, thus speeding up calculation.

c. Example

In this examples, *l* sets of *n*-dimensional linear equations that have an identical coefficient matrix are solved. Here the coefficient matrix is assumed to be of the form (1.2) and $n \leq 1000$.

```
C     **EXAMPLE**
      DIMENSION B(2010),VW(20),IVW(30)
      READ(5,500) N
      READ(5,510) D,SD
      WRITE(6,600) N,D,SD
      READ(5,500) L
      ISW=1
      IND=1
      DO 10 II=1,L
      READ(5,510) (B(I),I=1,N)
      WRITE(6,610) (B(I),I=1,N)
      CALL VLTX2(D,SD,N,B,ISW,IND,VW,IVW,
     *ICON)
      WRITE(6,620) ICON
      IF(ICON.NE.0) STOP
      WRITE(6,630) (B(I),I=1,N)
      ISW=2
   10 CONTINUE
      WRITE(6,640)
      STOP
  500 FORMAT(I5)
  510 FORMAT(5E14.7)
  600 FORMAT('1',
     *  20X,'LINEAR EQUATIONS (TRIDIAGONAL)'
     *  /' ',20X,'ORDER= ',I5/
     *  /' ',25X,'COEFFICIENT MATRIX'/
     *  /' ',30X,'D= ',E14.7/
     *  /' ',30X,'SD=',E14.7)
  610 FORMAT(/' ',78('*')//' ',
     *  25X,'CONSTANT VECTOR'//
     *  (' ',5(1X,E14.7)))
  620 FORMAT(/' ','CONDITION CODE OF VLTX2= ',
     *  I5)
  630 FORMAT(/' ',25X,'SOLUTION VECTOR'//
     * (' ',5(1X,E14.7)))
  640 FORMAT(//' ',30X,'** NORMAL END **')
      END
```

(4) Method

Cyclic reduction can be used to solve tridiagonal matrix equation (4.1), which is normalized so that the off diagonal elements of its coefficient matrix are all 1.

$$Ax = b, \tag{4.1}$$

where A takes one of the following forms:

$$\begin{bmatrix} d & 2 & & & \\ 1 & d & 1 & & \mathit{0} \\ & 1 & d & \cdot & \\ & & \cdot & \cdot & \cdot \\ & \mathit{0} & & \cdot & \cdot & 1 \\ & & & & 1 & d \end{bmatrix}, |d| \geq 2 \tag{4.2}$$

$$\begin{bmatrix} d & 1 & & & \\ 1 & d & 1 & & \mathit{0} \\ & 1 & d & \cdot & \\ & & \cdot & \cdot & \cdot \\ & \mathit{0} & & \cdot & \cdot & 1 \\ & & & & 2 & d \end{bmatrix}, |d| \geq 2 \tag{4.3}$$

$$\begin{bmatrix} d & 2 & & & \\ 1 & d & 1 & & \mathit{0} \\ & 1 & d & \cdot & \\ & & \cdot & \cdot & \cdot \\ & \mathit{0} & & \cdot & \cdot & 1 \\ & & & & 2 & d \end{bmatrix}, |d| \geq 2 \tag{4.4}$$

Dividing the n-th row of the matrix (4.3) by 2, all the off-diagonal elements become 1, and the last diagonal element becomes $d/2$. This type of matrix can be solved as explained in Method for subroutine VLTX1, so only solution of forms (4.2) and (4.4) need to be explained.

Dividing the n-th row of (4.4) by 2, the matrix becomes of the same form as (4.2) except for the last diagonal element. Therefore, we now consider (4.2) and (4.4) to be of the same form as matrix (4.5), and explain cyclic reduction for this matrix.

$$\begin{bmatrix} d & 2 & & & \\ 1 & d & 1 & & \mathit{0} \\ & 1 & d & \cdot & \\ & & \cdot & \cdot & \cdot \\ & \mathit{0} & & \cdot & \cdot & 1 \\ & & & & 1 & c \end{bmatrix} \tag{4.5}$$

Here $c=d$ for matrix (4.2) and $c=d/2$ for matrix (4.4). We assign b to be a constant vector of the matrix equation with coefficient matrix (4.5).

The cyclic reduction method here generates a matrix equation (of order $[(n-1)/2]+1$)with respect to the odd-numbered unknowns, x_1 , x_3 , x_5 , ... This differs from subroutine VLTX1. First, by eliminating x_2 from the following two equations:

$$\begin{aligned} dx_1 + 2x_2 &= b_1 \\ x_1 + dx_2 + x_3 &= b_2 \end{aligned} \tag{4.6}$$

we obtain

$$(2-d^2)x_1 + 2x_3 = 2b_2 - db_1. \tag{4.7}$$

Next, eliminating unknowns x_{2j} and x_{2j+2} from the three equations constructed using the $2j$-th row, $(2j + 1)$ st row and $(2j + 2)$ nd row of (4.5), we obtain

$$x_{2j-1} + (2-d^2)x_{2j+1} + x_{2j+3} = b_{2j} + b_{2j+2} - db_{2j+1}. \tag{4.8}$$

This calculation is repeated for each value of j =1, 2, ... , m (where m is the largest integer satisfying $2j+1 \le ... n-2$). One more equation is added to these two equations depending on whether n is even or odd. If n is even, eliminating x_{n-2} and x_n from the three equations,

$$\begin{aligned} x_{n-3} + dx_{n-2} + x_{n-1} \quad &= b_{n-2} \\ x_{n-2} + dx_{n-1} + x_n &= b_{n-1} \\ x_{n-1} + cx_n &= b_n, \end{aligned} \tag{4.9}$$

we obtain

$$x_{n-3} + (1 - d^2 + d/c)x_{n-1} = b_{n-2} + (d/c)b_n - db_{n-1} \tag{4.10}$$

When n is odd, eliminating x_{n-1} from the second and third equations of (4.9) we obtain

$$x_{n-2} + (1-dc)x_n = b_{n-1} - db_n. \tag{4.11}$$

Thus the equations of order $[(n-1)/2]$ consisting of (4.7), (4,8), and either (4.10) or (4.11), are obtained with respect to the odd-numbered unknowns only. These equations can be rewritten as (4.12).

$$\begin{bmatrix} d^{(1)} & 2 & & & & \\ 1 & d^{(1)} & 1 & 0 & & \\ & 1 & d^{(1)} & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ \cdot \\ x_{l-2} \\ x_l \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_3^{(1)} \\ b_5^{(1)} \\ \cdot \\ b_{l-2}^{(1)} \\ b_l^{(1)} \end{bmatrix} \tag{4.12}$$

where $d^{(1)} = 2 - d^2$

$b_1^{(1)} = 2b_2 - db_1$

$b_{2j+1}^{(1)} = \mathrm{b}_{2j} + b_{2j+2} - db_{2j+1}$

$j = 1, 2, ... ,m$

when n is even,

$l = n - 1$

$c^{(1)} = 1 - d^2 + d/c$

$$b_l^{(1)} = b_{n-2} + (d/c)b_n - db_{n-1}$$

when n is odd,

$l = n$

$c^{(1)} = 1 - dc$

$$bl^{(1)} = b_{n-1} - db_n$$

Looking at equation (4.12), we see that the coefficient matrix obtained by performing this single reduction is of the same form as (4.5), which is one of the characteristics of this method. Once the solution to (4.12) is obtained, the even-numbered unknowns can also be obtained by substituting them into the original matrix equation.

Applying the same reduction operation to (4.12), an equation of half the order can be obtained. By repeating the operation as many times as required, a matrix equation with coefficient matrix (4.13) can be obtained.

$$\begin{bmatrix} d^{(k)} & 2 \\ 1 & c^{(k)} \end{bmatrix} \qquad (4.13)$$

By solving this matrix, followed by substitution, the original equation can be solved.

If | d | is greater than 2, the reduction terminates incompletely for efficiency in the same way as explained for subroutine VLTX.

VLTX3

A62-41-0101 VLTX3, DVLTX3

A system of linear equations with a real constant tridiagonal matrix (periodic type and cyclic reduction method)

CALL VLTX3 (D, SD, N, B, ISW, VW, IVW, ICON)

(1) Function

This subroutine solves a real tridiagonal matrix equation

$$Ax = b \qquad (1.1)$$

using cyclic reduction, where A is an $n \times n$ irreducibly diagonally dominant real and almost tridiagonal matrix of the form:

$$\begin{bmatrix} d & e & & & & e \\ e & d & e & & 0 & \\ & e & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & e \\ e & & & & e & d \end{bmatrix} \begin{matrix} d \neq 0 \\ , |d| > 2|e| \end{matrix} \qquad (1.2)$$

Here b is an n-dimensional real constant vector and x is the n-dimensional solution vector, and $n \geq 1$.

(2) Parameters

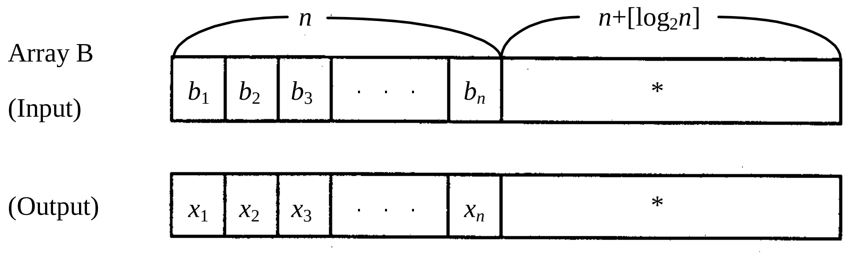
D	Input. Diagonal element d
SD	Input. Off-diagonal element e
N	Input. Order n of the coefficient matrix A
B...........	Input. Constant vector b Store as B(i) = b_i, i = 1, 2, ... , n Output. Solution vector x Store as B(i) = x_i, i =1, 2, ... , n See Figure VLTX3-1. One dimensional array of size $2n + [\log_2 n]$
ISW......	Input. Control information When solving several sets of equations that have an identical coefficient matrix, specify ISW=1 for the first set of equations, and

ISW=2 for the second and subsequent sets. Only parameter B is assigned a new constant vector *b*. All other parameters should be unchanged. (See Note (3)).

VW....... Work area. One-dimensional array of size 3 ([$\log_2 n$]+1), where [ ] is Gaussian notation.

IVW ..... Work area. One-dimensional array of size 4 ([$\log_2 n$]+1)+10.

ICON ... Output. Condition code
See Table VLTX3-1.



Note:

The portion indicated by an asterisk (*) is used as a work area in this subroutine.

Figure VLTX3-1 Storage method of vectors *b* and *x*

Table VLTX3-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	Coefficient matrix is not irreducibly diagonally dominant.	Bypassed
30000	N<1, or ISW ≠ 1, 2	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: AMACH, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, FLOAT, ABS, MIN0

b) Notes

(1) This subroutine uses cyclic reduction, an algorithm suited to a vector processor. Processing on a vector processor has the following features:

- It is much faster than the Gaussian elimination method
- Processing time increases almost linearly with N.
- The more diagonally dominant the matrix is, the faster it is processed.

This subroutine is about as accurate as the Gaussian elimination method.

(2) The coefficient matrix (1.2) arises from the discretization of simple periodic boundary value problems.

(3) When solving several sets of tridiagonal matrix equations that have an identical coefficient matrix, specify ISW=2 for the second and subsequent subroutine call. This bypasses coefficient matrix elimination, thus speeding up calculation.

c. Example

In this example, *l* sets of *n*-dimensional linear equations that have an identical coefficient matrix are solved, for $n \leq 1000$.

```
C      **EXAMPLE**
       DIMENSION B(2010),VW(30),IVW(50)
       READ(5,500) N
       READ(5,510) D,SD
       WRITE(6,600) N,D,SD
       READ(5,500) L
       ISW=1
       DO 10 II=1,L
       READ(5,510) (B(I),I=1,N)
       WRITE(6,610) (B(I),I=1,N)
       CALL VLTX3(D,SD,N,B,ISW,VW,IVW,ICON)
       WRITE(6,620) ICON
       IF(ICON.NE.0) STOP
       WRITE(6,630) (B(I),I=1,N)
       ISW=2
   10  CONTINUE
       WRITE(6,640)
       STOP
  500  FORMAT(I5)
  510  FORMAT(5E14.7)
  600  FORMAT('1',
      *  20X,'LINEAR EQUATIONS (TRIDIAGONAL)'
      *  /' ',20X,'ORDER= ',I5/
      *  /' ',25X,'COEFFICIENT MATRIX'/
      *  /' ',30X,'D= ',E14.7/
      *  /' ',30X,'SD=',E14.7)
```

```
610 FORMAT(/' ',78('*')//' ',
   *  25X,'CONSTANT VECTOR'//
   *  (' ',5(1X,E14.7)))
620 FORMAT(/' ','CONDITION CODE OF VLTX3= ',
   *  I5)
630 FORMAT(/' ',25X,'SOLUTION VECTOR'//
   *  (' ',5(1X,E14.7)))
640 FORMAT(//' ',30X,'** NORMAL END **')
    END
```

(4) Method

Cyclic reduction can be used to solve tridiagonal matrix equation (4.1), which is normalized so that the off diagonal elements of its coefficient matrix are all 1.

$$Ax = b \tag{4.1}$$

where

$$A = \begin{bmatrix} d & 1 & & & & 1 \\ 1 & d & 1 & & 0 & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ 1 & & & & 1 & d \end{bmatrix}, |d| > 2 \tag{4.2}$$

Because the above equation has nonzero elements at (*n*, 1) and (1, *n*), in its matrix, cyclic reduction cannot be applied directly. However, by transforming variables, the equation can be separated into two independent tridiagonal matrix equations each of which can then be solved using the cyclic reduction method described in Method for subroutine VLTX1 or VLTX2. The separation method is explained here for both even and odd *n*, because processing differs for the two cases.

(1) When *n* is even

Assuming n=2l we introduce two new variables *y* and *z* as follows:

$$y_j = x_{l-j} - x_{l+j}, \quad j = 1,2,\dots,l-1 \tag{4.3}$$

$$z_{j+1} = x_{l-j} + x_{l+j}, \quad j = 0,1,\dots,l$$

where $x_0 = x_n$ With these variables, the equations pertaining to *y* and *z* are given by (4.4) and (4.5), respectively.

$$\begin{bmatrix} d & 1 & & & & \\ 1 & d & 1 & & 0 & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & d \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ y_{l-2} \\ y_{l-1} \end{bmatrix} = \begin{bmatrix} b_{l-1} - b_{l+1} \\ b_{l-2} - b_{l+2} \\ b_{l-3} - b_{l+3} \\ \cdot \\ b_2 - b_{n-2} \\ b_1 - b_{n-1} \end{bmatrix} \tag{4.4}$$

$$\begin{bmatrix} d & 2 & & & & \\ 1 & d & 1 & & 0 & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \cdot \\ z_l \\ z_{l+1} \end{bmatrix} = \begin{bmatrix} 2b_l \\ b_{l-1} + b_{l+1} \\ b_{l-2} + b_{l+2} \\ \cdot \\ b_1 + b_{n-1} \\ b_n \end{bmatrix}, c = d/2 \qquad (4.5)$$

Equations (4.4) and (4.5) can be solved using the methods of subroutinesVLTX1 and VLTX2, respectively. Given *y* and *z*, *x* can be obtained as follows:

$$\begin{aligned} &x_l = z_1/2, \quad x_n = z_{l+1}/2 \\ &x_{l-j} = (y_i + z_{j+1})/2, \quad x_{l+j} = (z_{j+1} - y_i)/2 \\ &\qquad\qquad j = 1,2,...,l-1 \end{aligned} \qquad (4.6)$$

(2) When *n* is odd

Assuming *n* = 2*l* − 1 we introduce two new variables *y* and *z* as follows:

$$\begin{aligned} &y_j = x_{l-j} - x_{l+j},\ j = 1,2,...,l-1 \\ &z_{j+1} = x_{l-j} + x_{l+j},\ j = 0,1,...,l-1 \end{aligned} \qquad (4.7)$$

With these variables, the equations pertaining to *y* and *z* are given by (4.8) and (4.9), respectively.

$$\begin{bmatrix} d & 1 & & & & \\ 1 & d & 1 & & 0 & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c_1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ y_{l-2} \\ y_{l-1} \end{bmatrix} = \begin{bmatrix} b_{l-1} - b_{l+1} \\ b_{l-2} - b_{l+2} \\ b_{l-3} - b_{l+3} \\ \cdot \\ b_2 - b_{n-1} \\ b_1 - b_n \end{bmatrix}, c_1 = d - 1 \qquad (4.8)$$

$$\begin{bmatrix} d & 2 & & & & \\ 1 & d & 1 & & 0 & \\ & 1 & d & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & 0 & & \cdot & \cdot & 1 \\ & & & & 1 & c_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \cdot \\ z_{l-1} \\ z_l \end{bmatrix} = \begin{bmatrix} 2b_l \\ b_{l-1} + b_{l+1} \\ b_{l-2} + b_{l+2} \\ \cdot \\ b_2 + b_{n-1} \\ b_1 + b_n \end{bmatrix}, c_2 = d + 1 \qquad (4.9)$$

Similarly, the two equations above can be solved using the methods of subroutines VLTX1 and VLTX2, respectively.

Given *y* and *z*, *x* can be obtained as follows:

$$\begin{aligned} &x_l = z_1/2 \\ &x_{l-j} = (y_i + z_{j+1})/2,\ x_{l+j} = (z_{j+1} - y_j)/2 \\ &\qquad\qquad j = 1,2,...,l-1 \end{aligned} \qquad (4.10)$$

VLUIV

A22-71-0602 VLUIV, VDLUIV

The inverse of a real general matrix decomposed into the factors *L* and *D*

CALL VLUIV (FA, K, N, IP, AI, ICON)

(1) Function

This subroutine computes the inverse A^{-1} of an $n \times n$ real general matrix A given in decomposed form $PA = LU$

$A^{-1} = U^{-1} L^{-1} P$

L and U are respectively the $n \times n$ lower triangular and unit upper triangular matrices, and P is the permutation matrix which performs the row exchanges in partial pivoting for LU decomposition. $n \geq 1$.

(2) Parameters

FA	Input. Matrix L and matrix U. FA is a two-dimensional array, FA (K, N). Refer to Fig. VLUIV-1.
K	Input. Adjustable dimensional of array FA and AI (≥N).
N	Input. Order n of the matrices L and U.
IP..........	Input. Transposition vector which indicates the history of row exchanges in partial pivoting. One-dimensional array of size n.
AI	Output. Inverse A^{-1}. AI is a two-dimensional array, AI (K, N).
ICON....	Output. Condition code. See Table VLUIV-1.

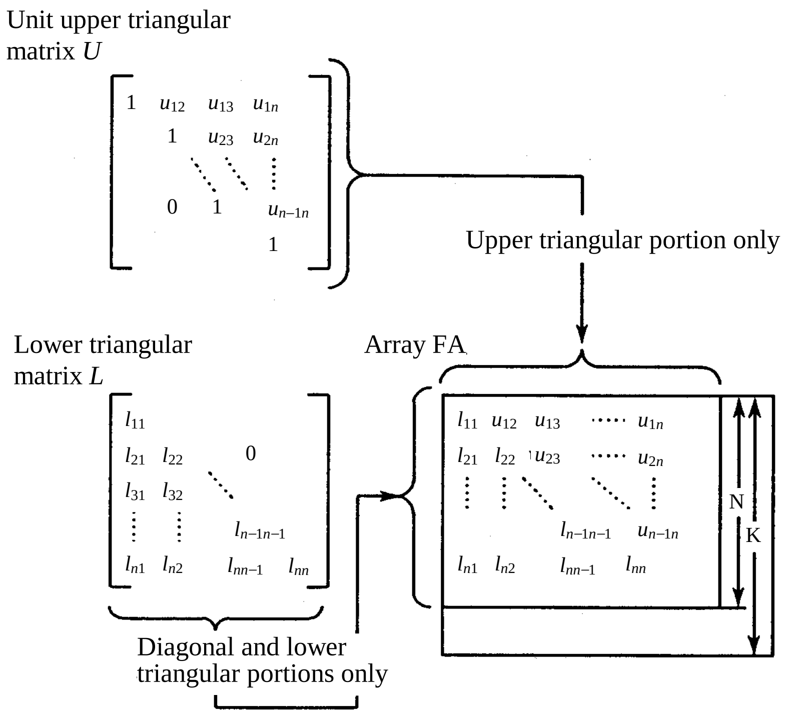

Unit upper triangular matrix U
Upper triangular portion only
Lower triangular matrix L
Array FA
Diagonal and lower triangular portions only
N
K

Figure VLUIV-1 Storage of the elements of *L* and *U* in array FA

Table VLUIV-1 Condition codes

Code	Meaning	Processing
0	No error	–
20000	A real matrix was singular.	Discontinued
30000	K>N of N<1 or there was an error in IP.	Bypassed

(3) Notes

a. Subprograms used

SSL II ......MGSSL

FORTRAN intrinsic functions........None

b. Notes

Prior to calling this subroutine, LU-decomposed matrix must be obtained by subroutine VALU and must be input as the parameters FA and IP to be used for this subroutine. The subroutine VLAX should be used for solving linear equations. Obtaining the solution by first computing the inverse matrix requires more steps of calculation, so subroutine VLUIV should be used only when the inverse matrix is inevitable. The transposition vector corresponds to the permutation matrix P of

$PA = LU$

When performing LU decomposition with partial pivoting. Refer to Notes of the subroutine VALU.

c. Example

The inverse of an $n \times n$ real general matrix is obtained. $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(100,100),VW(100),IP(100),AI(100,100)
      READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,510) ((A(I,J),I=1,N),J=1,N)
      WRITE(6,600) N,((I,J,A(I,J),J=1,N),I=1,N)
      CALL VALU(A,100,N,0.0,IP,IS,VW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) STOP
      CALL VLUIV(A,100,N,IP,AI,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) STOP
      WRITE(6,630) ((I,J,AI(I,J),I=1,N),J=1,N)
      STOP
  500 FORMAT(I5)
  510 FORMAT(4E15.7)
  600 FORMAT(//11X,'**INPUT MATRIX**'/12X,
     *'ORDER=',I5/(2X,4('(',I3,',',I3,')',E16.8)))
  610 FORMAT('0',10X,'CONDITION CODE(VALU)=',I5)
  620 FORMAT('0',10X,'CONDITION CODE(VLUIV)=',I5)
  630 FORMAT('0',10X,'**INVERSE MATRIX**',
     */(2X,4('(',I3,',',I3,')',E16.8)))
      END
```

(4) Method

This subroutine computes the inverse of an $n \times n$ real general matrix, giving the LU-decomposed matrices L, U and the permutation matrix P which indicates row exchanges in partial pivoting.

$$PA = LU \tag{4.1}$$

then, the inverse of A can be represented using (4.1) as follows:
The inverse of L and U are computed Eq. $UB = L^{-1}$ is solved to determine $B = U^{-1} L^{-1}$, and then the inverse of A is obtained as (4.2).

$$A^{-1} = \left(P^{-1}LU\right)^{-1} = U^{-1}L^{-1}P \tag{4.2}$$

L and U are as shown in Eq. (4.3) for the following explanation.

$$L = \left(l_{ij}\right), \quad U = \left(u_{ij}\right) \tag{4.3}$$

a. Calculating L^{-1}

Since the inverse L^{-1} of a lower triangular matrix L is also a lower triangular matrix, if we represent L^{-1} by

$$L^{-1} = \left(\tilde{l}_{ij}\right) \tag{4.4}$$

then Eq. (4.5) is obtained based on the relation

$LL^{-1} = I.$

$$\sum_{k=1}^{n} l_{ik}\tilde{l}_{kj} = \delta_{ij},$$

$$\delta_{ij} \begin{Bmatrix} 1, i = j \\ 0, i \neq j \end{Bmatrix} \tag{4.5}$$

(4.5) is rewritten as

$$\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} + l_{ii}\tilde{l}_{ij} = \delta_{ij}$$

and the elements $\tilde{l}_{ij}$ of the j-th column ($j = 1,...,n$) of the matrix L^{-1} are obtained as follows:

$$\tilde{l}_{ij} = \left(-\sum_{k=j}^{i-1} l_{ik}\tilde{l}_{kj} \right) / l_{ii}, \quad i = j+1,...,n$$

$$\tilde{l}_{jj} = 1/l_{jj} \tag{4.6}$$

where, $l_{ii} \neq 0 \left(i = j,...,n\right)$

b. Solving $UB = L^{-1}$

$Eq.UB = L^{-1}$ is solved by (4.7).

$$Ub_j = \tilde{l}_j \tag{4.7}$$

However,

$b_j = \left(b_{1j},..., b_{nj}\right)$: the column vector in B

$\tilde{l}_j = \left(\tilde{l}_{1j},..., \tilde{l}_{nj}\right)$: the column vector in L^{-1}

From (4.8), B is determined successively with $i = n,...,1$

$$b_{ij} = \tilde{l}_{ij} - \sum_{k=i+1}^{n} u_{ik} \tilde{b}_{kj} \qquad (4.8)$$

VMGGM

A61-11-0301 VMGGM, DVMGGM

Multiplication of two matrices
(real general by real general)

CALL VMGGM (A, KA, B, KB, C, KC, M, N, L, ICON)

(1) Function

This subroutine performs multiplication of an $m \times n$ real general matrix A by an $n \times l$ real general matrix B.

$C = AB$

Where C is an $m \times l$ real matrix. $m, n, l \geq 1$.

(2) Parameters

A	Input. Matrix *A*, two-dimensional array, A (KA, L).
KA........	Input. The adjustable dimension of array A, (≥M).
B...........	Input. Matrix *B*, two-dimensional array, B (KB, L).
KB........	Input. The adjustable dimension of array B, (≥N).
C...........	Output. Matrix *C*, two-dimensional array, C(KC, L). (See “Notes.”)
KC........	Input. The adjustable dimension of array C, (≥M).
M..........	Input. The number of rows *m* in matrix *A* and *C*.
N	Input. The number of columns *n*in matrix *A* and the number of rows *n* in matrix *B*.
L...........	Input. The number of columns *l* in matrices *B* and *C*.
ICON....	Output. Condition codes. SEE Table VMGGM-1.

Table VMGGM-1 Condition code

Code	Meaning	Processing
0	No error	–
30000	M<1, N<1, L<1, KA<M, KB<N, or KC<M	Bypassed

(3) Notes

a. Subprograms used

(1) SSL: MGSSL

(2) FORTRAN intrinsic function: FLOAT, MOD

b. Notes

The VMGGM subroutine differs from the standard function subroutine MGGM in one important respect.

The VMGGM subroutine performs high-speed calculation on a vector processor.

The performance of MGGM is changed by the adjustable dimensions of arrays A, B, and C, but the performance of the subroutine is not changed in essence.

Saving the storage area

To store matrix *C* in array A, the user must use MGGM.

c. Example

The following shows an example of obtaining the multiplication of matrices *A* and *B*. Here, $m \leq 200$, $n \leq 400$, and $l \leq 300$.

```
C     **EXAMPLE**
      DIMENSION A(202,400),B(402,300),C(202,300)
      CHARACTER*4 IA,IB,IC
      DATA IA/'A   '/,IB/'B   '/,IC/'C   '/
      DATA KA/202/,KB/402/,KC/202/
   10 READ(5,100) M,N,L
      IF(M.EQ.0) STOP
      WRITE(6,150)
      READ(5,200) ((A(I,J),I=1,M),J=1,N)
      READ(5,200) ((B(I,J),I=1,N),J=1,L)
      CALL VMGGM(A,KA,B,KB,C,KC,M,N,L,
     *ICON)
      IF(ICON.NE.0)GOTO 10
      CALL PGM(IA,1,A,KA,M,N)
      CALL PGM(IB,1,B,KB,N,L)
      CALL PGM(IC,1,C,KC,M,L)
      GOTO 10
  100 FORMAT(3I5)
  200 FORMAT(4E15.7)
  150 FORMAT('1'///10X,
     *'** MATRIX MULTIPLICATION **')
      END

C     ** MATRIX PRINT(REAL NON-SYMMETRIC) **
      SUBROUTINE PGM(ICOM,L,A,K,M,N)
      DIMENSION A(K,N)
      CHARACTER*4 ICOM(L)
      WRITE(6,600) (ICOM(I),I=1,L)
```

```
    DO 10 I=1,M
    WRITE(6,610) I,(J,A(I,J),J=1,N)
 10 CONTINUE
    RETURN
600 FORMAT(/10X,35A2)
610 FORMAT(/5X,I3,3(4X,I3,E17.7),
   *(/8X,3(4X,I3,E17.7)))
    END
```

Subroutine PGM in the example is for printing a real matrix.

F15-31-0201 VRFT1, DVRFT1
Discrete real Fourier transform (high performance, radix 2 FFT)
CALL VRFT1 (A, N, ISN, ISW, VW, IVW, ICON)

(1) Function

Given one-dimensional (n-term) real time-services data $\{x_j\}$, the discrete real Fourier transform or its inverse transform is calculated by the Fast Fourier Transform (FFT) method, suited to a vector processor, where n=2^l (l is a non-negative integer).

a. Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) is calculated to obtain Fourier coefficients $\{na_k\}$ and $\{nb_k\}$.

$$\begin{aligned} na_k &= 2 \cdot \sum_{j=0}^{n-1} x_j \cdot \cos kj\theta \quad ,k = 0,1,...,n/2, \\ nb_k &= 2 \cdot \sum_{j=0}^{n-1} x_j \cdot \sin kj\theta \quad ,k = 1,2,...,n/2-1 \\ &,\theta = 2\pi / n \end{aligned} \tag{1.1}$$

b. Fourier inverse transform

When $\{a_k\}$ and $\{b_k\}$ are input, the transform defined by (1.2) is calculated to obtain sum of Fourier series $\{2x_j\}$.

$$\begin{aligned} 2x_j &= a_0 + a_{n/2} \cdot \cos \pi j \\ &+ 2 \cdot \sum_{k=1}^{n/2-1} \left(a_k \cdot \cos kj\theta + b_k \cdot \sin kj\theta \right), \\ & j = 0,1,...,n-1, \theta = 2\pi / n \end{aligned} \tag{1.2}$$

(2) Parameters

A .... Input. $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$
Output. $\{na_k\}$, $\{nb_k\}$, or $\{2x_j\}$
One-dimensional array of size n+2
See Figure VRFT1-1.

N .... Input. Number of terms, n, of the transform

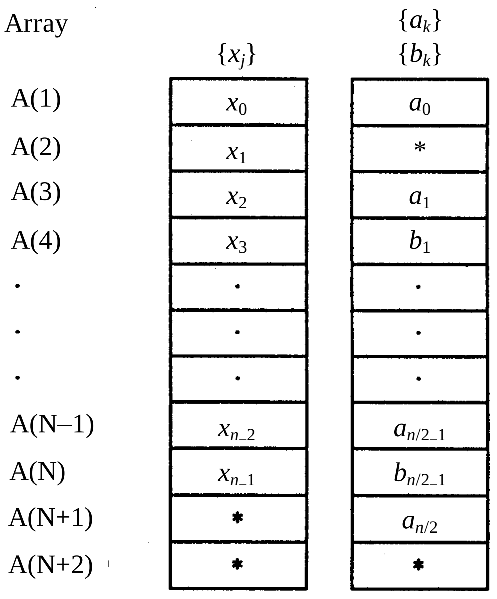
ISN....... Input. Either the transform or the inverse transform is indicated (≠0).
ISN=+1 for the transform.
ISN=−1 for the inverse transform.
(See Note(4).)

ISW...... Input. Information for controlling the initial state of the transform
ISW=0 for the first call.
ISW=1 for the second and subsequent calls.
(See Note(2).)

VW....... Work area
One-dimensional array of size max (n(l+1)/2, 1).

IVW ..... Work area. One-dimensional array of size n max (l – 4, 2)/2.

ICON ... Output. Condition code
See Table VRFT1-1.



Note:

The portion indicated by *has an arbitrary value at input, and is set to 0.0 at output.

Figure VRFT1-1 Data storage method

Table VRFT1-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	ISN=0, ISW≠0, 1 or N≠2^l (l: 0 or positive integer)	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK, MGSSL

(2) FORTRAN intrinsic functios: ALOG2, SIN, COS, ATAN, IABS, IAND, MOD, FLOAT

b. Notes

(1) Subroutine use

This subroutine performs high-speed calculation of a real Fourier transform on a vector processor. On a general-purpose computer, however, the subroutine RFT may be more suitable.

The function of this subroutine is the same as that of subroutine VRFT2, which is also suited to a vector processor. This subroutine can perform multiple independent transforms, but it requires more work array area than VRFT2; it is a high-performance subroutine. If it is difficult to allocate a large work array area, memory-efficient subroutine VCFT2 may be more suitable, even though it is slower.

(2) Control by ISW

When calculating multiple sets of transforms, specify ISW=1 for the second and subsequent subroutine calls. This bypasses trigonometric table and list vector generation, both of which are needed for the transform, thus increasing processing efficiency. The contents of the arrays VW and IVW must not be altered, however, when calling the subroutine.

Even the number of transforms, n, of each of the multiple transforms varies, specifying ISW=1 improves processing efficiency. However, it is desirable to be called so that the maximum number of transforms with the same number of terms are executed consecutively.

When calling this subroutine in together with the complex Fourier transform subroutine VCFT1, specifying ISW=1 improves processing efficiency.

(3) Work array size conversion table

The table for $16 \leq n \leq 4096$ is shown as follows:

l	n	VW	IVW
4	16	40	16
5	32	96	32
6	64	224	64
7	128	512	192
8	256	1152	512
9	512	2560	1280
10	1024	5632	3072
11	2048	12288	7168
12	4096	26624	16384

(4) ISN specification

Although the ISN parameter is used to specify whether a transform or an inverse transform is to be calculated, it can also be used as shown below. If $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$ is stored at intervals of length I, specify the ISN as follows:

ISN=+I for the transform.

ISN=–I for the inverse transform.

The results are also stored at intervals of length I.

With a vector processor, interval length I should take one the following values in order to access memory more efficiently. (see Example (2).)

I=4p+2, p=0, 1, 2, ... , for single precision arithmetic. (VRFT1)

I=2p+1, p=1, 2, 3, ... , for double precision arithmetic. (DVRFT1)

(5) General definition of Fourier transform

In general, the discrete real Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$a_k = \frac{2}{n}\sum_{j=0}^{n-1} x_j \cdot \cos kj\theta, k = 0,1,...,n/2,$$

$$b_k = \frac{2}{n}\sum_{j=0}^{n-1} x_j \cdot \sin kj\theta, k = 1,2,...,n/2-1 \qquad (3.1)$$

$$,\theta = 2\pi / n$$

$$\begin{aligned} x_j &= \frac{1}{2}a_0 + \frac{1}{2}a_{n/2}\cdot\cos\pi j \\ &+ \sum_{k=0}^{n/2-1}\left(a_k\cdot\cos kj\theta + b_k\cdot\sin kj\theta\right), \\ j &= 0,1,\ldots,n-1,\quad \theta = 2\pi/n \end{aligned} \tag{3.2}$$

This subroutine obtains $\{na_k\}$, $\{nb_k\}$ or $\{2x_j\}$ corresponding to the left hand side of (3.1) or (3.2), respectively. The result must be normalized as required.

c. Example

(1) Multiple Fourier transforms

In this example, k sets of independent Fourier transforms (with n terms) are calculated, for $k \le 64$ and $n \le 512$.

```
C     **EXAMPLE**
      DIMENSION A(514,64),VW(2560),IVW(1280)
      READ(5,500) N,K
      READ(5,510) ((A(I,J),I=1,N),J=1,K)
C
      ISN=1
      ISW=0
      CALL VRFT1(A,N,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0) STOP
      ISW=1
      DO 10 J=2,K
      CALL VRFT1(A(1,J),N,ISN,ISW,VW,IVW
     *         ,ICON)
   10 CONTINUE
C
      WRITE(6,600) K,N
      DO 20 J=1,K
   20 WRITE(6,610) J,(I,A(I,J),I=1,N+2)
C
  500 FORMAT(2I5)
  510 FORMAT(E15.7)
  600 FORMAT(5X,'***',I3,' SET TRANSFORMS'
     *       ' OF',' TERM',I4//)
  610 FORMAT(8X,I3,'-TH TRANSFORM'/
     *      (8X,I3,E16.7))
      STOP
      END
```

(2) Multi-dimensional Fourier transform

In this example, a 2-dimensional Fourier transform (with $n1\times n2$ terms) is calculated, for $n1\le 512$ and $n2\le 64$.

In the example program, the row-wise transform is calculated by subroutine VCFT1, using a complex Fourier transform.

Here, the data interval length (the first array declarator of the array), ISN=514, is suited to a vector processor (514=4p+2, p=128). For a double precision alogrithm, ISN=517 is better.

```
C     **EXAMPLE**
      DIMENSION A(514,64),VW(2560),IVW(1280)
      READ(5,500) N1,N2
      READ(5,510) ((A(I,J),I=1,N1),J=1,N2)
C     ----N2 SET REAL TRANSFORMS OF TERM
C                                      N1----
      ISN=1
      ISW=0
      CALL VRFT1(A,N1,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0)STOP
      ISW=1
      DO 10 J=2,N2
      CALL VRFT1(A(I,J),N1,ISN,ISW,VW,IVW,
     *           ICON)
   10 CONTINUE
C     ----HALF SET COMPLEX TRANS. OF TERM
C                                       N2----
      ISN=514
      CALL VCFT1(A,A(2,1),N2,
     *           ISN,ISW,VW,IVW,ICON)
      IF (ICON.NE.0) STOP
      DO 20 I=3,N1+2,2
      CALL VCFT1(A(I,1),A(I+1,1),N2,
     *           ISN,ISW,VW,IVW,ICON)
   20 CONTINUE
C
      WRITE(6,600) N1,N2
      DO 30 J=1,N2
   30 WRITE(6,610) J,(I,A(I,J),
     *                  A(I+1,J),I=1,N1+2,2)
C
  500 FORMAT(2I5)
  510 FORMAT(E15.7)
  600 FORMAT(5X,'***2-DIMENSIONAL TRANSFORM',
     *           ' OF TERM',I4,' BY ',I4)
  610 FORMAT(8X,I3,'-TH COLUMN'//
     *      (8X,I3,2E16.7))
      STOP
      END
```

(4) Method

A discrete real Fourier transform with n terms (where n=2^l) is calculated using the fast Fourier transform (isogeometric type and self-sorting type FFTs) method, suited to a vector processor.

A real Fourier transform can be calculated by assuming the real data $\{x_j\}$ to be complex with its imaginary part equal to 0.0, and by applying a discrete complex Fourier transform to the data.

However in such case, the complex Fourier transform can be done efficiently by taking account of the characteristics of complex transform.

We now define a complex transform by (4.1).

$$\alpha_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{jk}, k = 0,1,\ldots,n-1$$
$$,\omega = \exp\left(2\pi i / n\right) \tag{4.1}$$

If $\{x_j\}$ is real data, relation (4.2) can be satisfied.

$$\alpha_{n-k} = \alpha_k^*, k = 1,2,\ldots,n-1 \tag{4.2}$$

* represents the complex conjugate.

The result of the real Fourier transform, $\{a_k\}$ and $\{b_k\}$ and the result of the complex Fourier transform, $\{\alpha_k\}$, are related as follows:

$$\begin{aligned} a_0 &= 2\cdot\alpha_0, a_{n/2} = 2\cdot\alpha_{n/2} \\ a_k &= (\alpha_k + \alpha_{n-k}), k = 1,2,\ldots,n/2-1 \\ b_k &= i(\alpha_k - \alpha_{n-k}), k = 1,2,\ldots,n/2-1 \end{aligned} \tag{4.3}$$

Therefore, when calculating a real Fourier transform, it can be seen that the complex Fourier transform,

$$\alpha_k = \sum_{j=0}^{n-1} x_j \omega^{jk}, k = 0,1,\ldots,n/2$$
$$,\omega = \exp\left(2\pi i / n\right) \tag{4.4}$$

should be calculated first, followed by application of (4.2) and (4.3).

This subroutine calculates the complex Fourier transform of (4.4) using the fast Fourier transform method, suited to a vector processor.

For further details on calculating real Fourier transforms by using complex Fourier transforms, see Method for subroutine RFT, and for details on the fast Fourier transform method for a vector processor, see Method for subroutine VCFT1.

VRFT2

F15-31-0301 VRFT2, DVRFT2

Discrete real Fourier transform
(Memory efficient, radix 2 FFT)

CALL VRFT2 (A, N, ISN, ISW, VW, IVW, ICON)

(1) Function

Given one-dimensional (n-term) real time-service data $\{x_j\}$, the discrete real Fourier transform or its inverse transform is calculated by the Fast Fourier Transform (FFT) method, suited to a vector processor, where $n = 2^l$(l is a non- negative integer).

a. Fourier transform

When $\{x_j\}$ is input, the transform defined by (1.1) is calculated to obtain Fourier coefficients $\{na_k\}$ and $\{nb_k\}$.

$$na_k = 2 \cdot \sum_{j=0}^{n-1} x_j \cdot \cos kj\theta,\ k = 0,1,...,n/2$$

$$nb_k = 2 \cdot \sum_{j=0}^{n-1} x_j \cdot \sin kj\theta,\ k = 1,2,...,n/2-1 \qquad (1.1)$$

$$,\theta = 2\pi / n$$

b. Fourier inverse transform

When $\{a_k\}$ and $\{b_k\}$ are input, the transform defined by (1.2) is calculated to obtain sum of Fourier series $\{2x_j\}$

$$2x_j = a_0 + a_{n/2} \cos \pi j$$
$$+ 2 \cdot \sum_{k=0}^{n/2-1} \left(a_k \cos kj\theta + b_k \sin kj\theta \right), \qquad (1.2)$$
$$j = 0,1,...,n-1 \text{ and } \theta = 2\pi / n$$

(2) Parameters

A .... Input. $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$
Output. $\{na_k\}$, $\{nb_k\}$ or $\{2x_j\}$
One-dimensional array or size n+2
see Figure VRFT2-1

N .... Input. Number of terms, n, of the transform

ISN....... Input. Either the transform or the inverse transform is indicated ($\neq$0)
ISN=+1 for the transform.
ISN=–1 for the inverse transform.
(See Note(4).)

ISW...... Input. Information for controlling the intial state of the transform
ISW = 0 for the first call.
ISW = 1 for the second and subsequent calls.
(See Note (2).)

VW....... Work area.
One-dimensional array of size 7n/2.

IVW ..... Work area. One-dimensional array of size 3n/2

ICON ... Output. Condition code
See Table VRFT2-1

Array A	$\{x_j\}$	$\{a_k\}$ $\{b_k\}$
A(1)	x_0	a_0
A(2)	x_1	*
A(3)	x_2	a_1
A(4)	x_3	b_1
•	•	•
•	•	•
•	•	•
A(N–1)	x_{n-2}	$a_{n/2-1}$
A(N)	x_{n-1}	$b_{n/2-1}$
A(N+1)	*	$a_{n/2}$
A(N+2)	*	*

Note:

The portion indicated by * has an arbitrary value at input, and is set to 0.0 at output

Figure VRFT2-1 Data storage method

Table VRFT2-1 Condition Codes

Code	Meaning	Processing
0	No error	–
30000	ISN = 0, ISW ≠ 0, 1 or N ≠ 2^l ($l \geq 0$ is integer)	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: VCFT2, UVRFT, UVTB2, UVF92, UVFA2, UVFB2, UVFX2, UBANK, MGSSL

(2) FORTRAN intrinsic function: ALOG2, SIN, COS, ATAN, IABS

b. Notes

(1) Subroutine use

This subroutine performs high-speed calculation of real Fourier transform on a vector processor. On a general-purpose computer, however, subroutine RFT may be more suitable.

The function of this subroutine is the same as that of subroutine VRFT1, which is also suited to a vector processor. This subroutine is suitable for calculating only a single transform. The work array area is limited to the required minimum; it is a memory-efficient subroutine. For multiple transform, if there is sufficient work array area available, the high- performance subroutine VRFT1 is more suitable.

(2) Control by ISW

When performing multiple transform, specify ISW=1 for the second and subsequent subroutine calls. This bypasses trigonometric function table and list vector generation, both of which are needed for the transform, thus Increasing processing efficiency. The contents of the arrays VW and IVW must not be altered, however, when calling the subroutine.

Even when the number of transform, n, of each of the multiple transforms varies, specifying ISW=1 improves efficiency. However, it is desirable to be called so that the maximum number of transforms with the same number of terms are executed consecutively.

When calling this subroutine in together with the complex Fourier transform subroutine VCFT2, specifying ISW=1 improves processing efficiency.

(3) Work array size conversion table

The table for $16 \le n \le 4096$ is shown below.

l	n	VW	IVW
4	16	56	24
5	32	112	48
6	64	224	96
7	128	448	192
8	256	896	384
9	512	1792	768
10	1024	3584	1536
11	2048	7168	3072
12	4096	14336	6144

(4) ISN specification

Although the ISN parameter is used to specify whether a transform or an inverse transform is to be calculated, it can also be used as shown below. If $\{x_j\}$ or $\{a_k\}$, $\{b_k\}$ is stored at intervals of length I, specify ISN as follow:

ISN=+I for the transform.

ISN=−I for the inverse transform.

The results are also stored at intervals of length I.

With a vector computer, the interval length I should take the following values in order to access memory more efficiently. (see Example(2).)

I=4p+2, p=0, 1, 2, ... , for single precision arithmetic. (VRFT2)

I=2p+1, p=1, 2, 3, ... , for double precision arithmetic. (DVRFT2)

(5) General definition of Fourier transform

In general, the discrete Fourier transform and its inverse transform can be defined as in (3.1) and (3.2).

$$a_k = \frac{2}{n}\sum_{j=0}^{n-1} x_j \cdot \cos kj\theta, k = 0,1,...,n/2$$

$$b_k = \frac{2}{n}\sum_{j=0}^{n-1} x_j \cdot \sin kj\theta, k = 1,2,...,n/2-1 \qquad (3.1)$$

$$,\theta = 2\pi/n$$

$$x_j = \frac{1}{2}a_0 + \frac{1}{2}a_{n/2}\cos \pi j$$

$$+ \sum_{k=0}^{n/2-1} (a_k \cdot \cos kj\theta + b_k \cdot \sin kj\theta),$$

$$j = 0,1,...,n-1, \theta = 2\pi / n \qquad (3.2)$$

This subroutine obtains $\{na_k\}$, $\{nb_k\}$ or $\{2x_j\}$ corresponding to the left hand side of (3.1) or (3.2), respectively.

Normalized the results as required.

c. Example

In this example, a one-dimensional Fourier transform (with n terms) and its inverse transform are calculated, for $n \le 1024$.

```
C     **EXAMPLE**
      DIMENSION A(1026),VW(3584),IVW(1536)
      READ(5,500) N
      READ(5,510) (A(I),I=1,N)
C     ----FOURIER ANALYSIS----
      ISN=1
      ISW=0
      CALL VRFT2(A,N,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0)STOP
C     ----NORMALIZATION----
      ANOR=2.0/FLOAT(N)
      DO 10 I=1,N+2
   10 A(I)=ANOR*A(I)
      WRITE(6,600) N,(I,A(I),A(I+1),I=1,N+2,2)
C     ----FOURIER SYNTHESIS----
      ISN=-1
      ISW=1
      CALL VRFT2(A,N,ISN,ISW,VW,IVW,ICON)
      IF(ICON.NE.0) STOP
C     ----NORMALIZATION----
      ANOR=0.5
      DO 20 I=1,N
   20 A(I)=ANOR*A(I)
      WRITE(6,610) N,(I,A(I),I=1,N)
C
  500 FORMAT(I5)
  510 FORMAT(E15.7)
  600 FORMAT(5X,
     *  '***FOURIER ANALYSIS OF TERM',I5//
     *  (8X,I3,2E16.7))
  610 FORMAT(5X,
     *  '***FOURIER SYNTHESIS OF TERM',I5//
     *  (8X,I3,E16.7))
      STOP
      END
```

(4) Method

A discrete real Fourier transform with n terms (where n=2^l) is calculated using the fast Fourier transform (isogeometric type and self-sorting type FFTs) method, suited to a vector processor.

The real Fourier transform can be calculated by assuming the real data $\{x_j\}$ to be complex data with its imaginary part equal to 0.0, and by applying a discrete complex Fourier transform to the data.

However in such case, the complex Fourier transform can be done efficiently by taking account of the characteristics of complex transform.

We now define a complex transform by (4.1).

$$\alpha_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{jk}, k = 0,1,\ldots,n-1 \\ ,\omega = \exp(2\pi i / n) \tag{4.1}$$

If $\{x_j\}$ is real data, relation (4.2) can be satisfied.

$$\alpha_{n-k} = \alpha_k^*, \quad k = 1,2,\ldots,n-1 \tag{4.2}$$

* represents the complex conjugate.

The result of the real Fourier transform, $\{a_k\}$ and $\{b_k\}$, and the result of the complex transform, $\{\alpha_k\}$, are related as follows:

$$\begin{aligned} & a_0 = 2 \cdot \alpha_0 \cdot a_{n/2} = 2 \cdot \alpha_{n/2} \\ & a_k = (\alpha_k + \alpha_{n-k}), \quad k = 1,2,\ldots,n/2-1 \\ & b_k = i(\alpha_k - \alpha_{n-k}), \quad k = 1,2,\ldots,n/2-1 \end{aligned} \tag{4.3}$$

To calculate a real Fourier transform, the complex Fourier transform.

$$\alpha_k = \sum_{j=0}^{n-1} x_j \cdot \omega^{jk}, k = 0,1,\ldots,n/2 \\ ,\omega = \exp(2\pi i / n) \tag{4.4}$$

should be calculated, followed by application of (4.2) and (4.3).

This subroutine calculates the complex Fourier transform of (4.4) using the fast Fourier transform method, suited to a vector processor.

For further details on calculating real Fourier transforms by using complex Fourier transforms, see Method for subroutine RFT, and for details on the fast Fourier transform method for a vector processor, see Method for subroutine VCFT1.

VSEG2

B61-21-0201 VSEG2, DVSEG2

Eigenvalue and engenvector of real symmetric matrix
(parallel bisection method, reverse iteration method)

CALL VSEG2 (A, N, M, EPST, E, EV, K, VW, IVW, ICON)

(1) Function

This subroutine calculates *m* number of eigenvalues of an *n* order real symmetric matrix *A* in descending (or ascending) order, using the parallel bisection method. It also calculates corresponding m number of eigenvectors, using the inverse iteration method. Eigenvectors are normalized such that $||x||_2=1$. The result must be such that $1 \le m \le n$.

(2) Parameters

A	Input. Real symmetric matrix *A*. Symmetric matrix compression mode. One-dimensional array of size $n(n+1)/2$. The content is altered at output.
N	Input. Order *n* of real symmetric matrix *A*.
M	Input. Number *m* of eigenvalues to be calculated. Calculate in descending order when M = $+m$. Calculate in ascending order when M = $-m$.
EPST	Input. Upper bound of the absolute errors used in eigenvalue convergence test. The default value is used when a negative value is specified. (See note (2).)
E	Output. Eigenvalues. One-dimensional array of size *m*. Store in descending order when M is positive and in ascending order when M is negative.
EV	Output. Eigenvectors. Two-dimensional array of EV (K, *m*). Eigenvector corresponding to eigenvalue E(J) is stored at EV(I, J), I=1, ... ,N.
K	Input. Conformation size ($\ge n$)for array EV.
VW	Work area. One-dimensional array of size 15*n*.
IVW	Word area. One-dimensional array of size 7*n*.
ICON	Output. Condition codes See Table VSEG2-1.

(3) Notes

a. Subprograms used

(1) SSL II: TRID1, UVTG2, TRBK, AMACH, MGSSEL, UVBCT

(2) FORTRAN intrinsic functions: IABS, SQRT, SIGN, ABS, AMAX1

b. Notes

(1) This subroutine is functionally equivalent to subroutine SEIG2, but is designed for high-speed execution on a vector processor using the parallel bisection method. Note that the methods of allocating work areas are different in these subroutines.

(2) Default value of the parameter EPST is as expressed by (3.1) when unit round off is u.

$$\lambda\,\mathrm{EPST} = u \cdot \max\left(\left|\lambda_{\max}\right|, \left|\lambda_{\min}\right|\right) \qquad (3.1)$$

Here, $\lambda_{\max}$ and $\lambda_{\min}$ are the upper and lower bounds of the existence range (given by Gerschgorin's theorem) of eigenvalues of $Ax = \lambda x$.

When very large and small absolute value eigenvalues coexist and a convergence test is performed using (3.1), it is generally difficult to calculate smaller eigenvalues with adequate precision. In such cases, smaller eigenvalues may be calculated with higher precision by setting EPST at a small value (absolute error). However, processing speed slows down, as the number of iterations increases.

See the section on the method of obtaining the convergence criterion.

Table VSEG2-1 Condition codes

Code	Meaning	Processing
0	No error	–
10000	N=1	Set EV (1, 1)=1.0 and E(1)=A(1).
15000	Some eigenvectors were not calculated.	Make the uncalculated eigenvectors zero vectors.
20000	No eigenvectors were calculated.	Make all eigenvectors zero vectors.
30000	M=0, N<|M|, or K<N.	Bypassed

c. Example

In this example, *m* number of eigenvalues and their corresponding eigenvectors are calculated for an *n* order real symmetric matrix *A* in descending (or ascending) order.

The following example is for *n*≤100 and *m*≤20.

```
C     **EXAMPLE**
      DIMENSION A(5050),E(20),EV(102,20),
     *          VW(1500),IVW(700)
   10 READ(5,500,END=900) N,M,EPST
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,600) N,M
      IJ=0
      DO 20 I=1,N
      IJ=IJ+I
   20 WRITE(6,610) I,(A(J),J=IJ-I+1,IJ)
      CALL VSEG2(A,N,M,EPST,E,EV,102,
     *           VW,IVW,ICON)
      WRITE(6,620) ICON
      IF(ICON.GE.20000) GO TO 10
      MM=IABS(M)
      CALL SEPRT(E,EV,102,N,MM)
      GO TO 10
  900 STOP
  500 FORMAT(2I5,E10.2)
  510 FORMAT(5E15.7)
  600 FORMAT('1',//'*** ORIGINAL MATRIX N=',I4,
     *       2X,'M=',I4//)
  610 FORMAT('0',2X,I3,5E15.7/(6X,5E15.7))
  620 FORMAT('0'//'*** ICON= ',I5)
      END
```

The subroutine SEPRT in this example is used for printing eigenvalues and eigenvectors of real symmetric matrices. The following illustrates the contents of this subroutine.

```
      SUBROUTINE SEPRT(E,EV,K,N,M)
      DIMENSION E(M),EV(K,M)
      WRITE(6,600)
      KAI=(M-1)/5+1
      LST=0
      DO 10 KK=1,KAI
      INT=LST+1
      LST=LST+5
      IF(LST.GT.M) LST=M
      WRITE(6,610) (J,J=INT,LST)
      WRITE(6,620) (E(J),J=INT,LST)
      DO 10 I=1,N
      WRITE(6,630) I,(EV(I,J),J=INT,LST)
   10 CONTINUE
      RETURN
  600 FORMAT('1',20X,
     *      'EIGENVALUE AND EIGENVECTOR')
```

```
610 FORMAT('0',5I20)
620 FORMAT('0',5X,'ER',3X,5E20.8/)
630 FORMAT(5X,I3,3X,5E20.8)
    END
```

(4) Method

This subroutine calculates *m* number of eigenvalues of an *n* order real symmetric matrix *A* in descending (or ascending) order using the parallel bisection method, and their corresponding eigenvectors using the inverse iteration method.

First, it transforms real symmetric matrix *A*, using the Householder method, into real symmetric tridiagonal matrix *T* shown in Fig. VSEG2-1. This operation is shown by expression (4.1).

$$T = Q_H^T A Q_H \tag{4.1}$$

Here, Q_H is an orthogonal matrix. This operation is performed using the subroutine TRID1.

Next, m number of eigenvalues are calculated by applying the parallel bisection method on transformed matrix *T*. Then, the eigenvector for matrix *T* corresponding to the *m* eigenvalues are calculated using the inverse iteration method. This method calculates eigenvectors by repeatedly solving expression (4.2).

$$(T - \lambda I)y_r = y_{r-1}, \quad r = 1,2,... \tag{4.2}$$

Note that in (4.2), λ is the eigenvalue calculated by the parallel bisection method and y_r is the iteration vector. The parallel bisection method is explained in later paragraphs. See the section on subroutine TEIG2 for the inverse iteration method.

Next, calculate the eigenvectors of *A*. Eigenvector *x* of *A* can be calculated by using Q_H of equation (4.1) in (4.3), by letting *y* be the eigenvector of *T*.

$$x = Q_H y \tag{4.3}$$

This operation is performed using subroutine TRBK.

$$\begin{bmatrix} c_1 & b_2 & & & & & \\ b_2 & c_2 & b_3 & & & & \\ & b_3 & c_3 & b_4 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & b_n \\ & & & & & b_n & c_n \end{bmatrix}$$

Figure VSEG2-1 Real symmetric tridiagonal matrix T

Parallel bisection method

The following paragraphs present the calculation of m number of eigenvalues in descending order to simplify its explanation.

Here, letting λ be a variable and $p_i(\lambda)$ be the value of the leading principle minor of matrix $(T - \lambda I)$ from the upper left results in the following recurrence relation:

$$p_0(\lambda) = 1, p_1(\lambda) = c_1 - \lambda,$$

$$p_i(\lambda) = (c_i - \lambda) \times p_{i-1}(\lambda) - b_i^2 \times p_{i-2}(\lambda), \quad (4.4)$$

$$i=2, 3,..., n$$

The polynomial sequence $p_0(\lambda)$, $p_1(\lambda)$,..., $p_n(\lambda)$ in (4.4) constitutes a Sturm sequence. Therefore, if the number of times the codes of consecutive terms $p_0(\lambda)$ through $p_n(\lambda)$ invert is defined as $\alpha(\lambda)$, then $\alpha(\lambda)$ is equal to the number of eigenvalues smaller than λ. The bisection method is a method of calclating eigenvalues one by one by repeatedly bisecting the eigenvalue existence interval, applying such theorem. In general, an underflow or overflow can easily occur in the calculation of (4.4) so that the ploynomial sequence $q_i(\lambda)$ expressed as (4.5) is used for evaluation to avoid underflows and overflows.

$$q_i(\lambda) = p_i(\lambda) / p_{i-1}(\lambda) \quad (4.5)$$

In this case, the number of times $q_i(\lambda)$ becomes negative is equal to the number of eigenvalues smaller than λ. In the following paragraph, the number of times $q_i(\lambda)$ becomes negative is defined as $\alpha(\lambda)$.

The parallel bisection method applies the bisection method simultaneously on *m* number of eigenvalues λ_j, j = 1, 2, ..., *m*, by setting an existence interval for each eigenvalue. Now, express the existence interval for the *j*-th eigenvalue λ_j as $\left[a_j^{(k)}, b_j^{(k)}\right]$. *k* is the number of iterations. The initial existence interval is $\left[a_j^{(0)}, b_j^{(0)}\right]$ and it is set to satisfy the relation ship of (4.6).

$$\alpha\left(a_j^{(0)}\right) = j - 1,$$

$$\alpha\left(b_j^{(0)}\right) = j \tag{4.6}$$

The parallel bisection method iterates the following steps (1) through (3) for *k*=0, 1, 2,... to sufficiently reduce $\left[a_j^{(k)}, b_j^{(k)}\right]$ and approximates the value of λ_j at its midpoint.

(1) Approximate λ_j at the midpoint of the interval.

$$h_j^{(k)} = \left(a_j^{(k)} + b_j^{(k)}\right)/2, j = 1,2,...,m \tag{4.7}$$

$$\alpha\left(h_j^{(k)}\right) = 0, j = 1,2,...,m \tag{4.8}$$

(2) Evaluate the Sturm sequence q_i, i=1, 2, ...n, and obtain the number of times the code becomes negative.

$$q_i\left(h_j^{(k)}\right),$$

$$q_i\left(h_j^{(k)}\right) < 0 \qquad \alpha\left(h_j^{(k)}\right) = \alpha\left(h_j^{(k)}\right) + 1, \tag{4.9}$$

$$j = 1,2,...,m$$

(3) Revise the existence interval.

$$\alpha\left(h_j^{(k)}\right) = j - 1,$$

$$a_j^{(k+1)} = h_j^{(k)}, b_j^{(k+1)} = b_j^{(k)},$$

$$\alpha\left(h_j^{(k)}\right) = j, \tag{4.10}$$

$$a_j^{(k+1)} = a_j^{(k)}, b_j^{(k+1)} = h_j^{(k)},$$

$$j = 1,2,...,m$$

Eigenvalue convergence criterion and EPST specifying method Convergence test in this subroutine is performed by (4.11)

$$b_j^{(k)} - a_j^{(k)} \le 2u\left(\left|b_j^{(k)}\right| + \left|a_j^{(k)}\right|\right) + \text{EPST} \tag{4.11}$$

Here, *u* is the unit round off and EPST is the value specified as the upper bound of absolute errors for the eigenvalues to be calculated. When the relation expressed by (4.11) is satisfied, $\left(b_j^{(k)} - a_j^{(k)}\right)/2$ is made the *j*-th eigenvalue λ_j. EPST has he function to control process termination at the required precision level. If EPST = 0.0, (4.11) becomes (4.12).

$$b_j^{(k)} - a_j^{(k)} \le 2u\left(\left|b_j^{(k)}\right| + \left|a_j^{(k)}\right|\right) \tag{4.12}$$

At this time, bisection is performed repeatedly until the least significant digits of $b_j^{(k)}$ and $a_j^{(k)}$ are nearly equal. On the other hand, if EPST > 0.0 iteration stops when the specified precision level is reached. Specification of EPST > 0.0 is specifically require when eigenvalues include a zero.

When EPST < 0.0 is specified, this subroutine uses the following as the default value.

$$\mathrm{EPST} = u \cdot \max(\{\lambda_{\max}\},\{\lambda_{\min}\})$$

Here, $\lambda_{\max}$ and $\lambda_{\min}$ are the lower bound and upper bound values of the interval that includes all eigenvalues obtained using the Gerschgorin's theorem.

VSIN1

F16-21-0201 VSIN1, DVSIN1	
Discrete sine transform	(radix 2 FFT)
CALL VSIN 1 (A, N, TAB, VW, IVW, ICON)	

(1) Function

This subroutine calculates discrete sine transform and its inverse transform using the Fast Fourier Transform (FFT) suited to a vector processor, when n number of samples $\{x_j\}$ obtained by dividing half a period of an odd-function $x(t)$ of period 2π into n, equal sections as expressed by (1.1), with $n = 2^l$ where l is a positive integer.

$$x_j = x(\theta j), j = 0,1,..., n-1 \qquad (1.1)$$

$$,\theta = \pi / n$$

a. Sine transform

When $\{x_j\}$ is input, Fourier coefficients $\{2n \cdot b_k\}$ are calculated using the transform defined by (1.2).

$$2n \cdot b_k = 4 \cdot \sum_{j=0}^{n-1} x_j \cdot \sin kj\theta, k = 0,1,..., n-1 \qquad (1.2)$$

$$,\theta = \pi / n$$

Note that $x_0 = 0$.

b. Sine inverse transform

When $\{b_k\}$ is input, Fourier series values $\{4 \cdot x_j\}$ are calculated using the transform defined by (1.3).

$$4 \cdot x_j = 4 \cdot \sum_{k=0}^{n-1} b_k \cdot \sin kj\theta, \quad j = 0,1,..., n-1 \qquad (1.3)$$

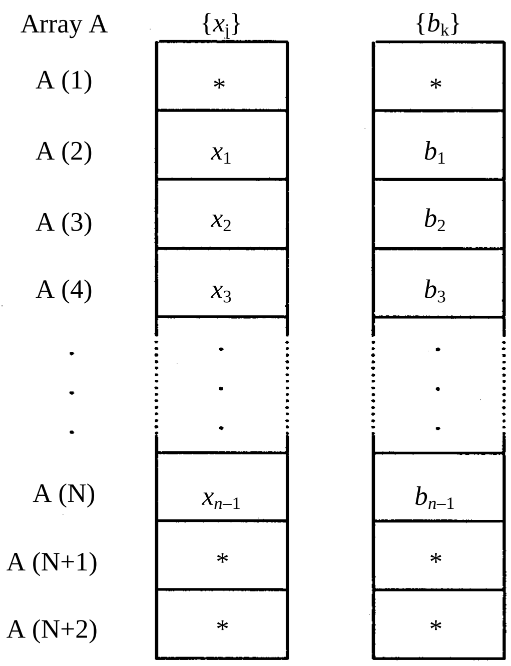
$$,\theta = \pi / n$$

Note that $b_0 = 0$

(2) Parameters

A	Input. $\{x_j\}$ or $\{b_k\}$
	Output. $\{2n \cdot b_k\}$ or $\{4 \cdot x_j\}$
	One-dimensional array of size n+2
	See Fig. VSIN 1-1.
N	Input. Number of samples n
TAB	Output. Trigonometric table used by transform is stored.
	One-dimensional array of size $2n$+4
VW	Word area.
	One-dimensional array of size max(n(l+1) /2,1)
IVW	Work area.
	One-dimensional array of size $n \cdot$ max(l–4,2) / 2
ICON	Output. Condition codes

See Table VSIN1-1.



Notes:

Same for $\{2nb_k\}$ and $\{x_j\}$.
* is and arbitrary value during input.
0.0 is set at the time of output.

Figure VSIN1-1 Data storage mode

Table VSIN1-1 Condition codes

Code	Meaning	Processing
0	No error	–
30000	N ≠ 2^l (l is a positive integer)	Bypassed

(3) Notes

a. Subprograms used

(1) SSLII: VRFT1, VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK, UVTAB, MGSSL

(2) FORTRAN intrinsic functions: ALOG2, SIN, COS, ATAN, IABS, IAND, MOD, FLOAT

b. Notes

(1) subroutine use

this subroutine performs high-speed calculation of discrete sine transform on a vector processor. The subroutine FSINT may be more suited on a general-purpose computer.

(2) Multiple transforms

When performing multiple transforms, generation of trigonometric table and list vectors is omitted in the second and subsequent subroutine calls, resulting in processing efficiency. The contents of arrays TAB, VW, and IVW must be called without altering them.

Even when the number of terms n for the multiple transforms differs, the previously generated contents of arrays TAB, VW, and IVW are valid. However, it is preferable to call them in such a way that transforms with identical number of terms are stringed together to the maximum extent possible.

(3) Trigonometric table and work array size table

The following shows the sizes for $16 \le n \le 4096$.

l	n	TAB	VW	IVW
4	16	36	40	16
5	32	68	96	32
6	64	132	224	64
7	128	260	512	192
8	256	516	1152	512
9	512	1028	2560	1280
10	1024	2052	5632	3072
11	2048	4100	12288	7168
12	4096	8196	26624	16384

(4) General definition of discrete sine transform

Discrete sine transform and its inverse transform are generally defined by (3.1),and (3.2), respectively.

$$b_k = \frac{2}{n}\sum_{j=1}^{n-1} x_j \cdot \sin kj\theta, \quad k = 1,2,...,n-1 \tag{3.1}$$

$$x_j = \sum_{k=1}^{n-1} b_k \cdot \sin kj\theta, \quad j = 1,2,...,n-1 \tag{3.2}$$

This subroutine calculates $\{2n \cdot b_k\}$ or $\{4 \cdot x_j\}$corresponding the left hand sides of (3.1) and (3.2), respectively. Therefore, normalize the results as required.

c. Example

Input n number of samples $\{x_j\}$ and transform by this subroutine. Then normalize the results and obtain discrete Fourier coefficients$\{b_k\}$. Calculate $\{x_j\}$ by proceeding to inverse transformation. This example is for $n \le 512$.

```
C     **EXAMPLE**
      DIMENSION X(514),TAB(1028),VW(2560),
     *          IVW(1280)
    1 READ(5,500) N
      IF(N.EQ.0) STOP
      READ(5,501) (X(I),I=1,N)
C     SINE TRANSFORM
      WRITE(6,600) N
      WRITE(6,601) (X(I),I=1,N)
      CALL VSIN1(X,N,TAB,VW,IVW,ICON)
      IF(ICON.NE.0) GO TO 30
C     NORMALIZE
      CN=1.0/(2.0*FLOAT(N))
      DO 10 K=1,N
      X(K)=X(K)*CN
   10 CONTINUE
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,N)
C     SINE INVERSE TRANSFORM
      CALL VSIN1(X,N,TAB,VW,IVW,ICON)
      IF(ICON.NE.0) GO TO 30
C     NORMALIZE
      CN=0.25
      DO 20 K=1,N
      X(K)=X(K)*CN
   20 CONTINUE
      WRITE(6,602)
      WRITE(6,601) (X(I),I=1,N)
      GO TO 1
   30 WRITE(6,603) ICON
      GO TO 1
  500 FORMAT(I5)
  501 FORMAT(6F12.0)
  600 FORMAT('0',5X,'INPUT DATA N=',I5)
  601 FORMAT(5F15.7)
  602 FORMAT('0',5X,'OUTPUT DATA')
  603 FORMAT('0',5X,'CONDITION CODE',I8)
      END
```

(4) Method

Consider calculating discrete sine transform of n terms ($=2^l$, l = 1, 2,...) using the Fast Fourier Transform (FFT) suited for a vector processor.

Discrete sine transform is generally expressed by (4.1), when samples$\{x_j\}$, j=0,1, ... ,n−1, are given.

$$b_k = \frac{2}{n}\sum_{k=1}^{n-1} x_j \cdot \sin(kj\theta)$$

$$, j = 0,1,...,n-1 \qquad (4.1)$$

$$,\theta = \pi / n$$

Now the samples are an odd-function and the relation expressed by (4.2) exists when extended to one period.

$$x_{2n-j} = -x_j, \quad j = 0,1,...,n-1$$

$$\text{and}, x_0 = x_n = 0 \qquad (4.2)$$

Therefore, $b_0 \sim b_n$ can be calculated by extending $x_0 \sim x_{n-1}$ to $x_0 \sim x_{2n-1}$ and performing 2n term (discrete real fourier transform.
It is well known that efficient transformation can be achieved by taking advantage of the symmetry of (4.2), in this case.

Now, perform following preprocessing on the samples$\{x_j\}$:

$$d_j = \frac{1}{2} \cdot \left(x_j - x_{n-j}\right) + \sin\left(j\theta\right) \cdot \left(x_j + x_{n-j}\right), \quad J = 0,1,\ldots,n-1 \qquad (4.3)$$

At this point, substituting of discrete sine inverse transform (4.4) into (4.3) results in (4.5).

$$b_k = \frac{2}{n} \sum_{k=1}^{n-1} x_j \cdot \sin\left(kj\theta\right), \quad j = 0,1,\ldots,n-1 \qquad (4.4)$$

$$d_j = b_1 + \sum_{k=1}^{n/2-1} \left[\left(b_{2k+1} - b_{2k-1}\right) \cdot \cos\left(2 \cdot kj\theta\right) + b_{2k} \cdot \sin\left(2 \cdot kj\theta\right)\right] - \left(-1\right)^j \cdot b_{n-1} \qquad (4.5)$$

$$, j = 0,1,\ldots, n-1$$

Expression (4.5) is equivalent to an *n* term discrete real Fourier transform with Fourier coefficients of $\{b_{2k+1} - b_{2k-1}\}$ and $\{b_{2k}\}$. Thus, calculation of Fourier coefficients $\{\tilde{a}_k\}$ and $\{\tilde{b}_k\}$ for the samples $\{d_j\}$ will enable obtaining of $\{b_k\}$ using the identities:

$$\tilde{a}_k = b_{2k+1} - b_{2k-1}$$

$$\tilde{b}_k = b_{2k}$$

In other words, $\{b_k\}$ is calculated using (4.6) which follows.

$$b_1 = 1/2 \cdot \tilde{a}_0, b_{n-1} = -1/2 \cdot \tilde{a}_{n/2},$$

$$b_{2k} = \tilde{b}_k,$$

$$b_{2k+1} = b_{2k-1} + \tilde{a}_k, k = 1,\ldots,n/2-1 \qquad (4.6)$$

The last expression in (4.6) is a recurrence formula and is unsuitable for a vector processor. Therefore, this subroutine is designed as a vector processor suited algorithm by back tracing these calculations, which avoids performing reference calculation, taking advantage of the fact that discrete sine transform and its inverse transform are identical expressions, except for the normalization constants.

Refer to reference [8] for details on this algorithm.

VSLDL

A22-61-0202 VSLDL, DVSLDL

LDL^T decomposition of a positive definite symmetric matrix

CALL VSLDL(A, N, EPSZ, VW, IVW, ICON)

(1) Function

This subroutine decomposes an $n \times n$ positive definite symmetric matrix A into LDL^T using the modified Cholesky's method:

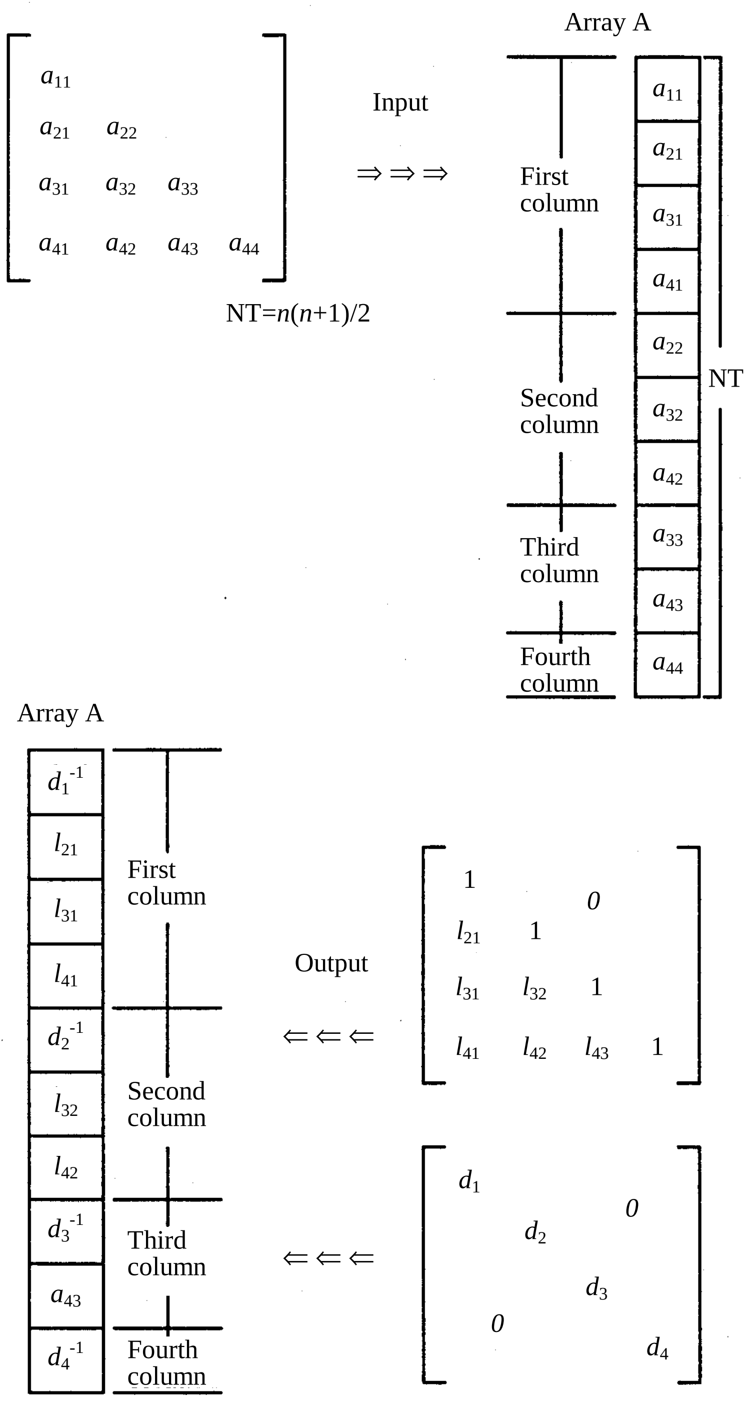
$$A = LDL^T \qquad (1.1)$$

Where L is a unit lower triangular matrix, D is a diagonal matrix, and $n \geq 1$.

The function of this subroutine is similar to that of subroutine SLDL, but the coefficient matrix is stored differently, and this subroutine is more suited to a vector processor.

(2) Parameters

A	Input. Coefficient matrix A Output. Matrices L and D^{-1} The lower triangular portion of the symmetric matrix is stored column by column,from the first to the n-th column, in a one-dimensional array of size $n(n+1)/2$, as shown in Figure VSLDL-1.
N	Input. Order n of matrix A
EPSZ	Input. Tolerance for relative zero test of pivots ($\geq$0.0) When EPSZ=0.0, a standard value is used. (See Note (2).)
VW.......	Work area. One- dimensional array of size 2n
IVW	Work area. One-dimensional array of size n
ICON....	Output. Condition code See Table VSLDL-1.



Array A
a_{11}
a_{21} a_{22}
a_{31} a_{32} a_{33}
a_{41} a_{42} a_{43} a_{44}
Input
⇒⇒⇒
NT=$n(n+1)/2$
First column
Second column
Third column
Fourth column
a_{11} a_{21} a_{31} a_{41} a_{22} a_{32} a_{42} a_{33} a_{43} a_{44}
NT
Array A
d_1^{-1} l_{21} l_{31} l_{41} d_2^{-1} l_{32} l_{42} d_3^{-1} a_{43} d_4^{-1}
First column
Second column
Third column
Fourth column
Output
⇐⇐⇐
1
0
l_{21} 1
l_{31} l_{32} 1
l_{41} l_{42} l_{43} 1
⇐⇐⇐
d_1
0
d_2
d_3
0
d_4

Figure VSLDL-1 Storage method of a symmetric matrix

Table VSLDL-1 Condition codes

code	Meaning	Processing
0	No error	–
10000	Pivot became negative. Coefficient matrix is not positive definite.	Continued
20000	Pivot became smaller than relative zero value. Coefficient matrix might be singular.	Bypassed
30000	N<1 or EPSZ < 0.0	Bypassed

(3) Notes

a. Subprograms used

(1) SSL II: AMACH, MGSSL

(2) FORTRAN intrinsic functions:ABS

b. notes

(1) This subroutine is designed to speed up processing on a vector processor by using a different matrix storage method than the one used in subroutine SLDL. Note how the storage methods and calling sequences of the two subroutines differ.

(2) Suppose that 10^{-s} was given as the tolerance value for relative zero test EPSZ. This value has the following meaning: if the pivot value loses more than *S* sihnificant digits during LDL^T decomposition in the modified Cholesky's method, the value is assumed to be zero and decomposition processing is discontinued with ICON = 20000. The standard value of EPSZ is normally 16*u*, where *u* is the unit round off.

Processing can be continued by assigning the smallest value to EPSZ, even when the pivot value becomes smaller than the standard value. However, the calculation result may not be as accurate as desired.

(3) If the pivot value becomes negative during decomposition, it means that the coefficient matrix is nor longer positive definite. ICON = 10000 is set, and processing continues. Note, however, that the resulting calculation error may be significant, because no pivoting operation is performed.

(4) To obtain the determinant of the coefficient matrix, multiply all the *n* diagonal elements of array A (i.e., diagonal elements of D^{-1}) afer calculations are completed, and take the reciprocal of the result.

b. Example

An $n \times n$ matrix is input, an LDL^Tdecomposition is performed for $n \leq 100$.

```
C     **EXAMPLE**
      DIMENSION A(5050),VW(200),IVW(100)
   10 READ(5,500) N
      IF(N.EQ.0) STOP
      NT=N*(N+1)/2
      READ(5,510) (A(I),I=1,NT)
      WRITE(6,630)
      IS=1
      IE=N
      DO 20 J=1,N
      WRITE(6,600) J,(A(I),I=IS,IE)
      IS=IE+1
   20 IE=IE+(N-J)
      CALL VSLDL(A,N,1.0E-6,VW,IVW,ICON)
      WRITE(6,610) ICON
      IF(ICON.GE.20000) GO TO 10
      WRITE(6,640)
      IS=1
      IE=N
      DET=1.0
      DO 30 J=1,N
      WRITE(6,600) J,(A(I),I=IS,IE)
      DET=DET*A(IS)
      IS=IE+1
   30 IE=IE+(N-J)
      DET=1.0/DET
      WRITE(6,620) DET
      GO TO 10
  500 FORMAT(I5)
  510 FORMAT(5E15.7)
  600 FORMAT(' ',I5/(10X,5E16.8))
  610 FORMAT(/10X,'ICON=',I5)
  620 FORMAT(//10X,
     *'DETERMINANT OF MATRIX=',E16.8)
  630 FORMAT(/10X,'INPUT MATRIX')
  640 FORMAT(/10X,'DECOMPOSED MATRIX')
      END
```

(4) Method

LDL^T decomposition using the modified cholesky’s method is explained in Method for subroutine SLDL. This subroutine, however, is well suited to a vector processor, because decomposition is basically treated as calculation of a matrix-vector product.

In addition, the coefficient matrix storage method is very important. In order to perform efficient vector processing, the lower triangular portion of the coefficient matrix is stored column by column.

In LDL^{T} decomposition of a positive define symmetric matrix,

$$A = LDL^{\mathrm{T}} \tag{4.1}$$

We define $\tilde{L}$ such that $\tilde{L} = LD$. For $L = (l_{ij})$ and $D = \mathrm{diag}(d_i)$, $\tilde{L}$ is of the following form:

$$\tilde{L} = \begin{bmatrix} d_1 & & & & & \\ l_{21}d_1 & d_2 & & & 0 & \\ l_{31}d_1 & l_{32}d_2 & d_3 & & & \\ \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \\ l_{n1}d_1 & l_{n2}d_2 & \cdot & \cdot & \cdot & d_n \end{bmatrix} \tag{4.2}$$

During decomposition processing, this subroutine stores the subsets of elements of the coefficient matrices A, $\tilde{L}$, L and D^{-1} in one-dimensional array A, but at the end of decomposition, it stores only the elements of L and D^{-1}.

Figure VSLDL-2 shows the contents of array A at the r-th stage of the decomposition (where r =2,3,..., n) In the diagram, array A is depicted in the form of the lower triangular portion of a matrix. Elements marked by X are the L elements obtained so far, *'s are D^{-1} elements, ○'s are $\tilde{L}$ elements,and △'s are the elements of coefficient matrix A. (M_r and a_r are defined in the following paragraph (2).)

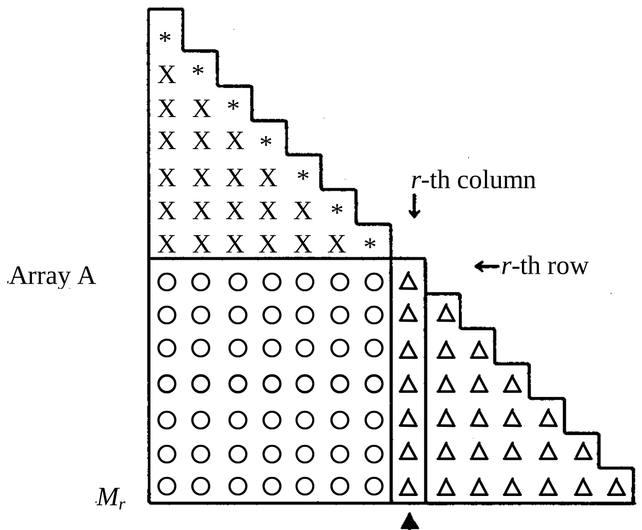


Figure VSLDL-2 Contens of array A

At the stage, the following calculations are performed:

(1) The r-th row of L is determined from the r-th row of array A as follows. Because the r-th row of array A is ($l_{r1}d_1$, $l_{r2}d_2$,..., $l_{r,r\text{-}1}d_{r\text{-}1}$, a_{rr}),

l_{rj} is readily obtained.

$$l_{rj}=(l_{rj}d_j)\,d_j^{-1},\ j = 1,2,3,...,r\text{–}1 \qquad (4.3)$$

These elements are temporarily stored in work array VW.

(2) The r-th column of $\tilde{L}$ determined by updating the r-th column of array A. This calculation, which is the main part of this method, is basically calculation of a matrix-vector product.

We now introduce several symbols. First, let $\tilde{l}_r$ be the r-th column vector of $\tilde{L}$ that is to be determined, i.e.,

$$\tilde{l}_r = (d_r, l_{r+1,r}d_r, l_{r+2,r}d_r, ..., l_{nr}d_r)^{\mathrm{T}} \qquad (4.4)$$

Next, let vector l_r, matrix M_r, and vector a_r be defined as follows:

$$l_r = (l_{r1}, l_{r2}, ..., l_{r,r-1})^{\mathrm{T}} \qquad (4.5)$$

$$M_r = \begin{bmatrix} l_{r1}d_1 & \cdots & l_{r,r-1}d_{r-1} \\ : & & : \\ l_{n1}d_1 & \cdots & l_{n,r-1}d_{r-1} \end{bmatrix} \qquad (4.6)$$

$$a_r = (a_{rr}, a_{r+1,r}, ..., a_{nr})^{\mathrm{T}} \qquad (4.7)$$

l_r , obtained in (1) above, is the r-th row vector of L, Mr is a submatrix of $\tilde{L}$, and a_r is the r-th column of coefficient matrix A. (See Figure VSLDL-2.)

The vectors and matrices defined above are related as follows:

$$a_r = \left[M_r \vdots \tilde{l}_r\right] \begin{bmatrix} l_r \\ \cdots \\ 1 \end{bmatrix}$$

Therefore,

$$\tilde{l}_r = a_r - M_r l_r \qquad (4.8)$$

can be obtained, which means that $\tilde{l}_r$ is basically calculated from a matrix-vector product. This calculation is well suited to a vector processor.

(3) Last, we update the r-th row of array A using the r-th row of L, and store d_r^{-1} as diagonal elements. For the above update, { l_{rj} ; j = 1,2,..., r –1} that have been saved in array VW are copied into the r-th row of array A. To store d_r^{-1}, take the reciprocal of d_r and store it, since it is the first element of $\tilde{l}_r$ obtained in

(2) above. d_r is checked here to see if the coefficient matrix is nonsingular and positive definite.

The above explanation concerns the r-th stage. By repeating (1), (2), and (3) above for r=2,3,...,n, array A will contain the lower triangular portion of matrix L (except for diagonal element 1's) and the inverse of diagonal matrix D.

APPENDIXES

APPENDIX A ALPHABETIC GUIDE FOR EXTENDED SUBROUTINES

A.1 General Subroutines

Subroutine	Classification code	Subprograms used
VALU	A22-71-0202	AMACH
VCFT1	F16-15-0201	UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK
VCFT2	F16-15-0301	UVTB2, UVF92, UVFA2, UVFB2, UVFX2, UBANK
VLDLX	A22-61-0302	
VCOS1	F16-11-0201	VRFT1, VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK, UVTAB
VGSG2	B62-21-0201	GSCHL, TRID1, TRBK, GSBK, UVTG2, UCHLS, AMACH, UVBCT
VLAX	A22-61-0101	VALU,LUX,AMACH
VLSX	A22-61-0101	AMACH,VSLDL,VLDLX
VLTX	A62-11-0101	AMACH
VLTX1	A62-21-0101	AMACH
VLTX2	A62-31-0101	AMACH
VLTX3	A62-41-0101	
VLUIV	A22-71-0602	
VMGGM	A61-11-0301	
VRFT1	F15-31-0201	VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK
VRFT2	F15-31-0301	VCFT2, UVRFT, UVTB2, UVF92, UVFA2, UVFB2, UVFX2, UBANK
VSEG2	B61-21-0201	TRID1, UVTG2, TRBK, AMACH, UVBCT
VSIN1	F16-21-0201	VRFT1, VCFT1, UVRFT, UVTB1, UVF91, UVFA1, UVFB1, UVFX1, UBANK, UVTAB
VSLDL	A22-61-0202	AMACH

A.2 Slave Subroutines

Slave routine	Calling subroutine
UBANK	VCFT1, VRFT1, VCFT2, VRFT2, VCOS1, VSIN1
UVBCT	VGSG2, VSEG2
UVFA1	VCFT1, VRFT1, VCOS1, VSIN1
UVFA2	VCFT2, VRFT2
UVFB1	VCFT1, VRFT1, VCOS1, VSIN1
UVFB2	VCFT2, VRFT2
UVFX1	VCFT1, VRFT1, VCOS1, VSIN1
UVFX2	VCFT2, VRFT2
UVF91	VCFT1, VRFT1, VCOS1, VSIN1
UVF92	VCFT2, VRFT2
UVTAB	VCOS1, VSIN1
UVRFT	VRFT1, VRFT2, VCOS1, VSIN1
UVTB1	VCFT1, VRFT1, VCOS1, VSIN1
UVTB2	VCFT2, VRFT2
UVTG2	VGSG2, VSEG2

APPENDIX B CLASSIFICATION CODES AND SUBROUTINES

Linear Algebra

Classification code	Subroutine
A22-61-0101	VLSX
A22-61-0202	VSLDL
A22-61-0302	VLDLX
A22-71-0202	VALU
A22-71-0101	VLAX
A22-71-0602	VLUIV
A61-11-0301	VMGGM
A62-11-0101	VLTX
A62-21-0101	VLTX1
A62-31-0101	VLTX2
A62-41-0101	VLTX3

Eigenvalues and Eigenvectors

Classification code	Subroutine
B61-21-0201	VSEG2
B62-21-0201	VGSG2

Transform

Classification code	Subroutine
F15-31-0201	VRFT1
F15-31-0301	VRFT2
F16-11-0201	VCOS1
F16-21-0201	VSIN1
F16-15-0201	VCFT1
F16-15-0301	VCFT2

APPENDIX C REFERENCES

[1] Stone, H. S.
Parallel Tridiagonal Equation Solvers,
ACM Trans. on Math. Soft., Vol. 1, No. 4, 1975,
pp. 289-307

[2] Temperton, C.
Fast Fourier Transforms and Poisson
solvers on CRAY-1
INFOTECH, 1979

[3] Hiraiwa,K.
The modified cyclic reduction algorithm for solving a system of
linear equations with a tridiagonal matrix on the parallel computer.
Transactions of Information Processing Society of Japan,
Vol.20, No.2, 1979, pp.190-193 (Japanese only)

[4] Hiraiwa,K.
The new parallel calculation method for FFT on the pipeline computer
which has the bit vector and the indirect vector.
Transactions of Information Processing Society of Japan,
Vol.21, No.2, 1980 (Japanese only)

[5] Mikami,J., Ina,H., Akita,T. and Yamashita,S.
The FFT algorithm on the vector computer.
The 28th IPSJ National Conference, 1984 (Japanese only)

[6] Matsuura,T. and Miura,K.
The data editing function of the super computer and the improvement
of the FFT performance.
The 28th IPSJ National Conference, 1984 (Japanese only)

[7] Akita,T., Mikami,J., Ina,H. and Yamashita,S.
The solver of a system of linear equations with a tridiagonal matrix
on the vector computer.
The 28th IPSJ National Conference, 1984 (Japanese only)

[8] Swarztrauber, P.N.
Vectorizing the FFTs,
Parallel Computations,
Academic Press, 1982, pp. 51-83

[9] Forsythe. G.E. and Moler, C.B.
Computer Solution of Linear Algebraic Systems,
Prentice-Hall, Inc., 1967

[10] Bowdler, H.J., Martin, R.S. and Wilkinson, J.H.
Solution of Real and Complex Systems of Linear Equations, Linear Algebra,
Handbook for Automatic Computation, Vol. 2, pp. 93-110,
Springer-Verlag, Berlin-Heidelberg-New York, 1971

[11] Parlett, B.N. and Wang, Y.
The Influence of The Compiler on The Cost of Mathematical Software-in Particular on The Cost of Triangular Factorization,
ACM Transactions on Mathematical Software, Vol. 1, No. 1, pp. 35-46, March, 1975

[12] 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

[13] Shimasaki,M.
Supercomputer and Programming.
Kyoritsu Shuppan Co., 1989 (Japanese only)

INDEX