ADVANCED SCIENTIFIC LIBRARY
ASL
User's Guide
<Shared Memory Parallel Functions>

PROPRIETARY NOTICE

PREFACE

This manual describes general concepts, functions, and specifications for use of the Advanced Scientific Library (ASL).

The manuals corresponding to this product consist of seven volumes, which are divided into the chapters shown below. This manual describes the shared memory parallel functions.

Basic Functions Volume 1

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Storage Mode Conversion	Explanation of algorithms, method of using, and usage example of subroutine related to storage mode conversion of array data.
3	Basic Matrix Algebra	Explanation of algorithms, method of using, and usage example of subroutine related to basic calculations involving matrices.
4	Eigenvalues and Eigenvectors	Explanation of algorithms, method of using, and usage example of subroutine related to **the standard eigenvalue problem** for real matrices, complex matrices, real symmetric matrices, Hermitian matrices, real symmetric band matrices, real symmetric tridiagonal matrices, real symmetric random sparse matrices, Hermitian random sparse matrices and **the generalized eigenvalue problem** for real matrices, real symmetric matrices, Hermitian matrices, real symmetric band matrices.

Basic Functions Volume 2

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Simultaneous Linear Equations (Direct Method)	Explanation of algorithms, method of using, and usage example of subroutine related to simultaneous linear equations corresponding to real matrices, complex matrices, positive symmetric matrices, real symmetric matrices, Hermitian matrices, real band matrices, positive symmetric band matrices, real tridiagonal matrices, real upper triangular matrices, and real lower triangular matrices.

Basic Functions Volume 3

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Fourier Transforms and their applications	Explanation of algorithms, method of using, and usage example of subroutine related to one-, two- and three-dimensional complex Fourier transforms and real Fourier transforms, one-, two- and three-dimensional convolutions, correlations, and power spectrum analysis, wavelet transforms, and inverse Laplace transforms.

Basic Functions Volume 4

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Differential Equations and Their Applications	Explanation of algorithms, method of using, and usage example of subroutine related to **ordinary differential equations initial value problems** for high-order simultaneous ordinary differential equations, implicit simultaneous ordinary differential equations, matrix type ordinary differential equations, stiff problem high-order simultaneous ordinary differential equations, simultaneous ordinary differential equations, first-order simultaneous ordinary differential equations, and high-order ordinary differential equations, and **ordinary differential equations boundary value problems** for high-order simultaneous ordinary differential equations, first-order simultaneous ordinary differential equations, high-order ordinary differential equations, high-order linear ordinary differential equations, and second-order linear ordinary differential equations, and **integral equations** for Fredholm's integral equations of second kind and Volterra's integral equations of first kind, and **partial differential equations** for two- and three-dimensional inhomogeneous Helmholtz equation.
3	Numerical Differentials	Explanation of algorithms, method of using, and usage example of subroutine related to numerical differentials of one-variable functions and multi-variable functions.
4	Numerical Integration	Explanation of algorithms, method of using, and usage example of subroutine related to numerical integration over a finite interval, semi-infinite interval, fully infinite interval, two-dimensional finite interval, and multi-dimensional finite interval.
5	Interpolations and Approximations	Explanation of algorithms, method of using, and usage example of subroutine related to interpolations, surface interpolations, least squares approximations, least squares surface approximations, and Chebyshev's approximations.
6	Spline Functions	Explanation of algorithms, method of using, and usage example of subroutine related to interpolation, smoothing, numerical derivatives, and numerical integrals using cubic splines, bicubic splines and B–splines.

Basic Functions Volume 5

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Special Functions	Explanation of algorithms, method of using, and usage example of subroutine related to Bessel functions, modified Bessel functions, spherical Bessel functions, functions related to Bessel functions, Gamma functions, functions related to Gamma functions, elliptic functions, indefinite integrals of elementary functions, associated Legendre functions, orthogonal polynomials, and other special functions.
3	Sorting and Ranking	Explanation and usage examples of subroutine related to sorting and ranking.
4	Roots of Equations	Explanation of algorithms, method of using, and usage example of subroutine related to roots of algebraic equations, nonlinear equations, and simultaneous nonlinear equations.
5	Extremal Problems and Optimization	Explanation of algorithms, method of using, and usage example of subroutine related to minimization of functions with no constraints, minimization of the sum of the squares of functions with no constraints, minimization of one-variable functions with constraints, minimization of multi-variable functions with constraints, and shortest path problem.

Basic Functions Volume 6

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Random Number Tests	Explanation and usage examples of subroutine related to uniform random number tests, and distribution random number tests.
3	Probability Distributions	Explanation and usage examples of subroutine related to continuous distributions and discrete distributions.
4	Basic Statistics	Explanation and usage examples of subroutine related to basic statistics, variance-covariance and correlation.
5	Tests and Estimates	Explanation and usage examples of subroutine related to interval estimates and tests.
6	Analysis of Variance and Design of Experiments	Explanation and usage examples of subroutine related to one-way layout, two-way layout, multiple-way layout, randomized block design, Greco-Latin square method, cumulative Method.
7	Nonparametric Tests	Explanation and usage examples of subroutine related to tests using χ^2 distribution and tests using other distributions.
8	Multivariate Analysis	Explanation and usage examples of subroutine related to principal component analysis, factor analysis, canonical correlation analysis, discriminant analysis, cluster analysis.
9	Time Series Analysis	Explanation and usage examples of subroutine related to autocorrelation, cross correlation, autocovariance, cross covariance, smoothing and demand forecasting.
10	Regression analysis	Explanation and usage examples of subroutine related to linear Regression and nonlinear Regression.

Shared Memory Parallel Functions

Chapter	Title	Contents
1	Introduction	Explanation of the organization of this manual, how to view each item, and usage limitations.
2	Basic Matrix Algebra	Explanation of algorithms, method of using, and usage example of subroutine related to obtain the product of real matrices and complex matrices.
3	Simultaneous Linear Equations (Direct Method)	Explanation of algorithms, method of using, and usage example of subroutine related to simultaneous linear equations corresponding to real matrices, complex matrices, real symmetric matrices, and Hermitian matrices.
4	Simultaneous Linear Equations (Iteration Method)	Explanation of algorithms, method of using, and usage example of subroutine related to simultaneous linear equations corresponding to real positive definite symmetric sparse matrices, real symmetric sparse matrices and real asymmetric sparse matrices.
5	Eigenvalues and Eigenvectors	Explanation of algorithms, method of using, and usage example of subroutine related to the eigenvalue problem for real symmetric matrices and Hermitian matrices.
6	Fourier Transforms and their applications	Explanation of algorithms, method of using, and usage example of subroutine related to one-, two- and three-dimensional complex Fourier transforms and real Fourier transforms, two- and three-dimensional convolutions, correlations, and power spectrum analysis.
7	Sorting	Explanation and usage examples of subroutine related to sorting and ranking.

Document Version 3.0.0-230301 for ASL, March 2023

Remarks

(1) This manual corresponds to ASL 1.1. All functions described in this manual are program products.

(2) Proper nouns such as product names are registered trademarks or trademarks of individual manufacturers.

(3) This library was developed by incorporating the latest numerical computational techniques. Therefore, to keep up with the latest techniques, if a newly added or improved function includes the function of an existing function may be removed.

Contents

Chapter 1

INTRODUCTION

1.1 OVERVIEW

1.1.1 Introduction to The Advanced Scientific Library ASL

Table 1−1 shows correspondences among product categories, functions of ASL and supported hardware platforms. In the same version of ASL, interfaces of subroutines of the same name are common among hardware platforms.

Table 1−1 Classification of functions included in ASL

Classification of Functions	Volume
Basic functions	Vol. 1-6
Shared memory parallel functions	Vol. 7

1.1.2 Distinctive Characteristics of ASL

ASL has the following distinctive characteristics.

(1) Subroutines are optimized using compiler optimization to take advantage of corresponding system hardware features.

(2) Special-purpose subroutines for handling matrices are provided so that the optimum processing can be performed according to the type of matrix (symmetric matrix, Hermitian matrix, or the like). Generally, processing performance can be increased and the amount of required memory can be conserved by using the special-purpose subroutines.

(3) Subroutines are modularized according to processing procedures to improve reliability of each component subroutine as well as the reliability and efficiency of the entire system.

(4) Error information is easy to access after a subroutine has been used since error indicator numbers have been systematically determined.

1.2 KINDS OF LIBRARIES

Table 1−2 Kinds of libraries providing ASL

Size of variable(byte)		Declaration of arguments	Kind	Kind of library
integer	real			
4	8	INTEGER(4) REAL(8)	32bit integer Double-precision subroutine	32bit integer library (link option: -lasl_openmp)
4	4	INTEGER(4) REAL(4)	32bit integer Single-precision subroutine	
8	8	INTEGER(8) REAL(8)	64bit integer Double-precision subroutine	64bit integer library (link option: -lasl_openmp_i64)
8	4	INTEGER(8) REAL(4)	64bit integer Single-precision subroutine	

(∗1) Functions that appear in this documentation do not always support all of the four kinds of subroutines listed above. For those functions that do not support some of those subroutine kinds, relevant notes will appear in the corresponding subsections.

(∗2) The string "(4)" that specifies 32bit (4 byte) can be omitted.

1.3 ORGANIZATION

This section describes the organization of Chapters 2 and later.

1.3.1 Introduction

The first section of each chapter is a general introduction describing such information as the effective ways of using the subroutines, techniques employed, algorithms on which the subroutines are based, and notes.

1.3.2 Organization of Subroutine Description

The second section of each chapter sequentially describes the following topics for each subroutine.

(1) Function

(2) Usage

(3) Arguments

(4) Restrictions

(5) Error indicator

(6) Notes

(7) Example

Each item is described according to the following principles.

1.3.3 Contents of Each Item

(1) **Function**
Function briefly describes the purpose of the ASL subroutine.

(2) **Usage**
Usage describes the subroutine name and the order of its arguments. In general, arguments are arranged as follows.

CALL subroutine-name (input-arguments, input/output-arguments, output-arguments, ISW, work, IERR)

ISW is an input argument for specifying the processing procedure. IERR is an error indicator. In some cases, input/output arguments precede input arguments. The following general principles also apply.

- Array are placed as far to the left as possible according to their importance.
- The dimension of an array immediately follows the array name. If multiple arrays have the same dimension, the dimension is assigned as an argument of only the first array name. It is not assigned as an argument of subsequent array names.

(3) **Arguments**
Arguments are explained in the order described above in paragraph (2). The explanation format is as follows.

Arguments	Type	Size	Input/Output	Contents
(a)	(b)	(c)	(d)	(e)

(a) Arguments

Arguments are explained in the order they are designated in the Usage paragraph.

(b) Type

Type indicates the data type of the argument. Any of the following codes may appear as the type.

I : Integer type

D : Double precision real

R : Real

Z : Double precision complex

C : Complex

There are 64-bit integer and 32-bit integer for integer type arguments. In a 32-bit (64-bit) integer type subroutine, all the integer type arguments are 32-bit (64-bit) integer. In other words, kinds of libraries determine the sizes of integer type arguments (Refer to 1.4). In the user program, a 32-bit/64-bit integer type argument must be declared by `INTEGER`/ `INTEGER(8)`, respectively.

(c) Size

Size indicates the required size of the specified argument. If the size is greater than 1, the required area must be reserved in the program calling this subroutine.

1 : Indicates that argument is a variable.

N : Indicates that the argument is a vector (one-dimensional array) having N elements. The argument N indicating the size of this vector is defined immediately after the specified vector. However, if the size of a vector or array defined earlier, it is omitted following subsequently defined vectors or arrays. The size may be specified by only a numeric value or in the form of a product or sum such as $3 \times N$ or $N + M$.

M**,** N : Indicates that the argument is a two-dimensional array having M rows and N columns. If M and N indicating the size of this array have not been defined before this array is specified, they are defined as arguments immediately following this array.

(d) Input/Output

Input/Output indicates whether the explanation of argument contents applies to input time or output time.

i. When only “Input” appears

When the control returns to the program using this subroutine, information when the argument is input is preserved. The user must assign input-time information unless specifically instructed otherwise.

ii. When only “Output” appears

Results calculated within the subroutine are output to the argument. No data is entered at input time.

iii. When both “Input” and “Output” appear

Argument contents change between the time control passes to the subroutine and the time control returns from the subroutine. The user must assign input-time information unless specifically instructed otherwise.

iv. When “Work” appears

Work indicates that the argument is an area used when performing calculations within the subroutine. A work area having the specified size must be reserved in the program calling this subroutine. The contents of the work area may have to be maintained so they can be passed along to the next calculation.

(e) Contents

Contents describes information held by the argument at input time or output time.

- A sample Argument description follows.

Example

The statement of the subroutine (DBGMLC, RBGMLC) that obtains the LU decomposition and the condition number of a real matrix is as follows.

Double precision:
CALL DBGMLC (A, LNA, N, IPVT, COND, W1, IERR)
Single precision:
CALL RBGMLC (A, LNA, N, IPVT, COND, W1, IERR)

The explanation of the arguments is as follows.

Table 1−3 Sample Arguments

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	Note {D / R}	LNA, N	Input	Real matrix A(two-dimensional array)
				Output	The matrix A decomposed into the matrix LU where U is a unit upper triangular matrix and L is a lower triangular matrix.
2	LNA	I	1	Input	Adjustable dimension size of array A
3	N	I	1	Input	Order n of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row exchanged with row i in the i-th step.
5	COND	{D / R}	1	Output	Reciprocal of the condition number
6	W1	{D / R}	N	Work	Work area
7	IERR	I	1	Output	Error indicator

To use this subroutine, arrays A, IPVT and W1 must first be allocated in the calling program so they can be used as arguments. A is a {double-precision / single-precision}Note real array of size (LNA , N) , IPVT is an integer array of size N and W1 is a {double-precision / single-precision}real array of size N.
When the 64-bit integer version is used, all integer-type arguments (LNA, N, IPVT and IERR) must be declared by using `INTEGER(8)`, not `INTEGER`.

Note The entries enclosed in brace { } mean that the array should be declared double precision type (code D) when using subroutine DBGMLC and real type (code R) when using subroutine RBGMLC. Braces are used in this manner throughout the remainder of the text unless specifically stated otherwise.

Data must be stored in A, LNA and N before this subroutine is called. The LU decomposition and condition number of the assigned matrix are calculated with in the subroutine, and the results are stored in array A and variable COND. In addition, pivoting information is stored in IPVT for use by subsequent subroutines.

IERR is an argument used to notify the user of invalid input data or an error that may occur during processing. If processing terminates normally, IERR is set to zero.

Since W1 is a work area used only within the subroutine, its contents at input and output time have no special meaning.

(4) **Restrictions**

Restrictions indicate limiting ranges for subroutine arguments.

(5) **Error indicator**

Each subroutine has been given an error indicator as an output argument. This error indicator, which has uniformly been given the variable name IERR, is placed at the end of the arguments. If an error is detected within the subroutine, a corresponding value is output to IERR. Error indicator values are divided into five levels.

Table 1−4 Classification of Error Indicator Output Values

Level	IERR value	Meaning	Processing result
Normal	0	Processing is terminated normally.	Results are guaranteed.
Warning	1000∼2999	Processing is terminated under certain conditions.	Results are conditionally guaranteed.
Fatal	3000∼3499	Processing is aborted since an argument violated its restrictions.	Results are not guaranteed.
	3500∼3999	Obtained results did not satisfy a certain condition.	Obtained results are returned (the results are not guaranteed).
	4000 or more	A fatal error was detected during processing. Usually, processing is aborted.	Results are not guaranteed.

(6) **Notes**

Notes describes ambiguous items and points requiring special attention when using the subroutine.

(7) **Example**

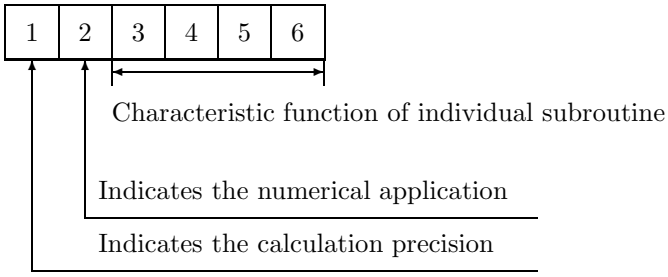
Here gives an example of how to use the subroutine. Note that in some cases, multiple subroutines are combined in a single example. The output results are given in the 32-bit integer version, and may differ within the range of rounding error if the compiler or intrinsic functions are different.

The source codes of examples in this document are included in User's Guide. Input data, if required, is also included in it. To build up an executable files by compiling these example source codes, they should be linked with this product library.

1.4 SUBROUTINE NAMES

The subroutines name of ASL shared memory parallel functions consists of ⟨six alphanumeric characters⟩.

Figure 1−1 Subroutine Name Components



"1" in Figure 1−1 : The following four letters are used to indicate the calculation precision.

Q	Parallel function double precision real-type calculation
P	Parallel function single precision real-type calculation
H	Parallel function double precision complex-type calculation
G	Parallel function single precision complex-type calculation

In the Basic Functions Volumes 1 to 6, the following eight letters are used to indicate the calculation precision.

D, W	Double precision real-type calculation
R, V	Single precision real-type calculation
Z, J	Double precision complex-type calculation
C, I	Single precision complex-type calculation

However, the complex type calculations listed above do not necessarily require complex arguments.

"2" in Figure 1−1 : Currently, the following letters letterererererere are used to indicate the application field in the ASL related products.

Letter	Application Field	Volume
A	Storage mode conversion	1
	Basic matrix algebra	1, 7
B	Simultaneous linear equations (direct method)	2, 7
C	Eigenvalues and eigenvectors	1, 7
F	Fourier transforms and their applications	3, 7
	Time series analysis	6
G	Spline function	4
H	Numeric integration	4
I	Special function	5
J	Random number tests	6
K	Ordinary differential equation (initial value problems)	4

Letter	Application Field	Volume
L	Roots of equations	5
M	Extremum problems and optimization	5
N	Approximation and regression analysis	4, 6
O	Ordinary differential equations (boundary value problems), integral equations and partial differential equations	4
P	Interpolation	4
Q	Numerical differentials	4
S	Sorting and ranking	5, 7
X	Basic matrix algebra	1
	Simultaneous linear equations (iterative method)	7
1	Probability distributions	6
2	Basic statics	6
3	Tests and estimates	6
4	Analysis of variance and design of experiments	6
5	Nonparametric tests	6
6	Multivariate analysis	6

"3–6" in Figure 1−1 : These characters indicate the characteristic function of the individual subroutine.

1.5 ASL SHARED MEMORY PARALLEL FUNCTIONS

1.5.1 Overview of Shared Memory Parallel Functions

All the ASL shared memory parallel functions are developed using OpenMP for a shared memory, multi-thread system model. Shared memory parallel functions run across plural processors. As a result, we may expect that an elapsed time will reduce which is required for analysis of large-scale problems.

There are two possible ways to run a program which calls ASL in shared memory parallel as follows:

(1) Use ASL shared memory parallel functions from the user program. In this case, ASL divides up and allocates internal processing among plural processors so that the allocated processing is executed in shared memory parallel.

(2) Parallelize the user program itself by using multi-thread functions, and call either user subroutines and ASL subroutines or plural ASL subroutines in parallel. In this case, the processing of subroutines that are called in parallel will be allocated among plural processors and be executed in parallel.

For the method (2), use ASL non-parallel subroutines (basic functions).

1.5.2 Performance Improvement Due to Parallel Functions

General conditions for improving program performance by using ASL shared memory parallel functions are described below.

(1) Execution environment

Scalability is nothing but an actual reduction in elapsed time, which is achieved through simultaneous use of plural processors. So you can not expect the scalability of shared memory parallel program when all the processors in your parallel system are busy most of time with many jobs.

(2) Problem scale

Shared memory parallel processing subroutines require more overhead for dividing and synchronizing processing than the overhead required when shared memory parallel processing is not performed. Therefore, for small-sized problems, this overhead may surpass the time reduction achieved by parallel processing. In general, the larger the problem size, the greater the effectiveness of shared memory parallel processing since the influence of the surplus overhead becomes relatively small compared to the total processing time. Therefore, to obtain the benefits of shared memory parallel processing, the problem scale should be large.

1.5.3 General Notes Concerning the Use of shared memory Parallel Functions

(1) Execution environment

Shared memory parallel functions can be used only with an operating system associated with a multi-core system. For a detailed explanation of the shared memory parallel processing (multi-thread), refer to the corresponding compiler manuals.

(2) Thread parameter (NT)

Shared memory parallel function subroutines have the number of tasks parameter NT as an argument. This parameter specifies the number of subdivisions of the processing when parallelization is performed within ASL.

(3) Setting the number of processor cores which are used
You can use the `OMP_NUM_THREADS` environment variable to control the number of processor cores which are used. The following command lines show the C shell syntax and Bourne shell syntax to use when setting the variable to 2 processor cores.

- C shell:

  setenv OMP_NUM_THREADS 2 RETURN

- Bourne shell:

  OMP_NUM_THREADS=2 RETURN
  export OMP_NUM_THREADS RETURN

1.6 NOTES

(1) Use the subroutines of double precision version whenever possible. They not only provide higher precision solutions but also are more stable than single precision versions, in particular, for eigenvalue and eigenvector problems.

(2) To suppress compiler operation exceptions, ASL subroutines are set to so that they conform to the compiler parameter indications of a user's main program. Therefore, the main program must suppress any operation exceptions.

(3) The numerical calculation programs generally deal with operations on finite numbers of digits, so the precision of the results cannot exceed the number of operation digits being handled. For example, since the number of operation digits (in the mantissa part) for double-precision operations is on the order of 15 decimal digits, when using these floating point modes to calculate a value that mathematically becomes 1, an error on the order of 10^{-15} may be introduced at any time. Of course, if multiple length arithmetic is emulated such as when performing operations on an arbitrary number of digits, this kind of error can be controlled. However, in this case, when constants such as π or function approximation constants, which are fixed in double-precision operations, for example, are also to be subject to calculations that depend on the length of the multiple length arithmetic operations, the calculation efficiency will be worse than for normal operations.

(4) A solution cannot be obtained for a problem for which no solution exists mathematically. For example, a solution of simultaneous linear equations having a singular (or nearly singular) matrix for its coefficient matrix theoretically cannot be obtained with good precision mathematically. Numerical calculations cannot strictly distinguish between mathematically singular and nearly singular matrices. Of course, it is always possible to consider a matrix to be singular if the calculation value for the condition number is greater than or equal to an established criterion value.

(5) Generally, if data is assigned that causes a floating point exception during calculations (such as a floating point overflow), a normal calculation result cannot be expected. However, a floating point underflow that occurs when adding residuals in an iterative calculation is an exception to this.

(6) For problems that are handled using numerical calculations (specifically, problems that use iterative techniques as the calculation method), there are cases in which a solution cannot be obtained with good precision and cases in which no solution can be obtained at all, by a special-purpose subroutine.

(7) Depending on the problem being dealt with, there may be cases when there are multiple solutions, and the execution result differs in appearance according to the compiler used or the computer or OS under which the program is executed. For example, when an eigenvalue problem is solved, the eigenvectors that are obtained may differ in appearance in this way.

(8) The mark "DEPRECATED" denotes that the subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative practice instead.

Chapter 2

BASIC MATRIX ALGEBRA

2.1 INTRODUCTION

This chapter explains a subroutine that obtains the product of two matrices.
Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

2.1.1 Notes

Since the parallel processing overhead significantly affects the computation cost if the order of the matrix is small, performance may be worse than when a non-parallel subroutine is used.

2.1.2 Algorithms Used

2.1.2.1 Matrix Multiplication

$A = (a_{ik}) \quad (1 \le i \le M, 1 \le k \le N)$
$B = (b_{kj}) \quad (1 \le k \le N, 1 \le j \le L)$
$C = (c_{ij}) \quad (1 \le i \le M, 1 \le j \le L)$

```
for     j = 1, 2, ..., L (Parallel execution)
    for     i = 1, 2, ..., M
            c_ij ← 0.0
```

```
for     j = 1, 2, ..., L (Parallel execution)
    for     k = 1, 2, ..., N
        for     i = 1, 2, ..., M
                c_ij ← c_ij + a_ik b_kj
```

Parallelization is performed for the j loop.
If there are two CPUs, the calculation $C = AB$ is divided into the following two calculations:

$$C_1 = AB_1$$
$$C_2 = AB_2$$

and executed in parallel as shown in Figure 2−1.

$$\underset{\text{Matrix } C}{\left[\, C_1 \mid C_2 \,\right]} = \underset{\text{Matrix } A}{[A]} \cdot \underset{\text{Matrix } B}{\left[\, B_1 \mid B_2 \,\right]}$$

Figure 2−1 Parallelization of Matrix Multiplication for Two CPUs

2.2 BASIC MATRIX ALGEBRA

2.2.1 QAM1MU, PAM1MU
Multiplying Real Matrices (Two-Dimensional Array Type)

(1) **Function**

Obtain the product of two real matrices A and B (two-dimensional array type).

(2) **Usage**

Double precision:

```
CALL QAM1MU  (A, LMA, NM, NN, B, LNB, NL, C, LMC, NT, IERR)
```

Single precision:

```
CALL PAM1MU  (A, LMA, NM, NN, B, LNB, NL, C, LMC, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LMA, NN	Input	Real matrix A (two-dimensional array type).
2	LMA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A (Number of rows in matrix C).
4	NN	I	1	Input	Number of columns in matrix A (Number of rows in matrix B).
5	B	{D / R}	LNB, NL	Input	Real matrix B (two-dimensional array type).
6	LNB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B (Number of columns in matrix C).
8	C	{D / R}	LMC, NL	Output	Product $A \cdot B$ of matrices A, B (two-dimensional array type).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	NT	I	1	Input	Number of tasks to be generated.
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN = 1	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1 & 2 & 0 & -1 \\ -3 & -5 & 1 & 2 \\ 1 & 3 & 2 & -2 \\ 0 & 2 & 1 & -1 \end{bmatrix}$$

$$B = \begin{bmatrix} -3 & -1 & 1 & -1 \\ -3 & -1 & 0 & 1 \\ -4 & -1 & 1 & 0 \\ -10 & -3 & 1 & 1 \end{bmatrix}$$

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, LMA = 11, LNB = 11, LMC = 11, NM = 4, NN = 4, NL = 4 and NT = 2.

(c) Main program

```
      PROGRAM OAM1MU
! *** EXAMPLE OF QAM1MU ***
      IMPLICIT NONE
      INTEGER LMA,LNB,LMC,NT,NM,NN,NL
      PARAMETER( LMA=11, LNB=11, LMC=11, NT=2 )
      PARAMETER( NM=4, NN=4, NL=4 )
      INTEGER IERR,I,J
      REAL(8) A(LMA,NN),B(LNB,NL),C(LMC,NL)
      COMMON A,B,C
!
      DO 100 I=1,NM
         READ(5,*) (A(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LNB,LMC,NM,NN,NL,NT
      DO 120 I=1,NM
         WRITE(6,6010) (A(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6020)
      DO 130 I=1,NN
         WRITE(6,6010) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL QAM1MU(A,LMA,NM,NN,B,LNB,NL,C,LMC,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040)
      DO 140 I=1,NM
         WRITE(6,6010) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  QAM1MU  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
```

```
     1X,'     LMA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
     1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
     1X,'     NT =',I2,/,/,&
     1X,'     INPUT MATRIX A',/)
6010 FORMAT(1X,6X,11(F7.1))
6020 FORMAT(/,&
     1X,'     INPUT MATRIX B',/)
6030 FORMAT(/,&
     1X,' **  OUTPUT  **',/,/,&
     1X,'     IERR = ',I4,/)
6040 FORMAT(1X,'     OUTPUT MATRIX C',/)
     END
```

(d) Output results

```
***  QAM1MU  ***

 **  INPUT  **

     LMA=11   LNB=11   LMC=11

     NM = 4   NN = 4   NL = 4

     NT = 2

     INPUT MATRIX A

          1.0    2.0    0.0   -1.0
         -3.0   -5.0    1.0    2.0
          1.0    3.0    2.0   -2.0
          0.0    2.0    1.0   -1.0

     INPUT MATRIX B

         -3.0   -1.0    1.0   -1.0
         -3.0   -1.0    0.0    1.0
         -4.0   -1.0    1.0    0.0
        -10.0   -3.0    1.0    1.0

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX C

          1.0    0.0    0.0    0.0
          0.0    1.0    0.0    0.0
          0.0    0.0    1.0    0.0
          0.0    0.0    0.0    1.0
```

2.2.2 QAM1MM, PAM1MM
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm AB$)

(1) **Function**

Obtain the product of real matrix A and real matrix $B(C = C \pm AB)$.

(2) **Usage**

Double precision:

```
CALL QAM1MM  (A, LMA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL PAM1MM  (A, LMA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LMA, NN	Input	Real matrix A (two-dimensional array type).
2	LMA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{D / R}	LNB, NL	Input	Real matrix B (two-dimensional array type).
6	LNB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{D / R}	LMC, NL	Input	Initial real matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C \pm]AB$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB$ ISW = 0: Obtain $C = AB$ ISW = −1: Obtain $C = C - AB$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

Obtain $C = AB$.

(b) Input data

Matrices A and B, LMA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 6, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM OAM1MM
! *** EXAMPLE OF QAM1MM ***
      IMPLICIT NONE
      INTEGER LMA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=6 )
      INTEGER IERR,I,J
      REAL(8) A(LMA,NN),B(LNB,NL),C(LMC,NL)
      COMMON A,B,C
!
      DO 100 I=1,NM
         READ(5,*) (A(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LNB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (A(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6020) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL QAM1MM(A,LMA,NM,NN,B,LNB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
```

```
        WRITE(6,6020) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  QAM1MM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,6X,11(F7.1))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  QAM1MM  ***

 **  INPUT  **

     LMA=11   LNB=11   LMC=11

     NM = 4   NN = 5   NL = 6

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
          1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0

     MATRIX B
          1.0    1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0    4.0
          5.0    5.0    5.0    5.0    5.0    5.0

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
         15.0   15.0   15.0   15.0   15.0   15.0
         30.0   30.0   30.0   30.0   30.0   30.0
         45.0   45.0   45.0   45.0   45.0   45.0
         60.0   60.0   60.0   60.0   60.0   60.0
```

2.2.3 QAM1MT, PAM1MT
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm AB^T$)

(1) **Function**

Obtain the product of real matrix A and real matrix $B(C = [C\pm]AB^T)$

(2) **Usage**

Double precision:

CALL QAM1MT (A, LMA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)

Single precision:

CALL PAM1MT (A, LMA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LMA, NN	Input	Real matrix A (two-dimensional array type).
2	LMA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{D / R}	LLB,NN	Input	Real transposed matrix B (two-dimensional array type).
6	LLB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{D / R}	LMC,NL	Input	Initial real matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB^T$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB^T$ ISW = 0: Obtain $C = AB^T$ ISW = −1: Obtain $C = C - AB^T$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NL} \leq \mathrm{LLB}$

(c) $\mathrm{NN} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

Obtain $C = AB^T$.

(b) Input data

Matrices A and B, LMA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 6, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM OAM1MT
! *** EXAMPLE OF QAM1MT ***
      IMPLICIT NONE
      INTEGER LMA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=5 )
      INTEGER IERR,I,J
      REAL(8) A(LMA,NN),B(LLB,NN),C(LMC,NL)
      COMMON A,B,C
!
      DO 100 I=1,NM
         READ(5,*) (A(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (A(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6020) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL QAM1MT(A,LMA,NM,NN,B,LLB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
```

```
         WRITE(6,6020) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  QAM1MT  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,6X,11(F7.1))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  QAM1MT  ***

 **  INPUT  **

     LMA=11   LLB=11   LMC=11

     NM = 4   NN = 5   NL = 5

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
          1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0

     MATRIX B
          1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0
          5.0    5.0    5.0    5.0    5.0

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
          5.0   10.0   15.0   20.0   25.0
         10.0   20.0   30.0   40.0   50.0
         15.0   30.0   45.0   60.0   75.0
         20.0   40.0   60.0   80.0  100.0
```

2.2.4 QAM1TM, PAM1TM
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B$)

(1) **Function**

Obtain the product of real matrix A and real matrix $B(C = [C\pm]A^T B)$

(2) **Usage**

Double precision:

CALL QAM1TM (A, LNA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)

Single precision:

CALL PAM1TM (A, LNA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA, NM	Input	Real transposed matrix A (two-dimensional array type).
2	LNA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{D / R}	LNB, NL	Input	Real matrix B (two-dimensional array type).
6	LNB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{D / R}	LMC, NL	Input	Initial real matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]A^T B$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + A^T B$ ISW = 0: Obtain $C = A^T B$ ISW = −1: Obtain $C = C - A^T B$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \text{NM} \leq \text{LMC}$

(b) $0 < \text{NN} \leq \text{LNA}, \text{LNB}$

(c) $\text{NL} > 0$

(d) $\text{ISW} \in \{0, 1, -1\}$

(e) $\text{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

Obtain $C = A^T B$.

(b) Input data

Matrices A and B, LNA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 6, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM OAM1TM
! *** EXAMPLE OF QAM1TM ***
      IMPLICIT NONE
      INTEGER LNA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=5, NL=4 )
      INTEGER IERR,I,J
      REAL(8) A(LNA,NM),B(LNB,NL),C(LMC,NL)
      COMMON A,B,C
!
      DO 100 I=1,NM
         READ(5,*) (A(J,I),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LNB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (A(J,I),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6020) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL QAM1TM(A,LNA,NM,NN,B,LNB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
```

```
        WRITE(6,6020) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  QAM1TM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,6X,11(F7.1))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  QAM1TM  ***

 **  INPUT  **

     LNA=11   LNB=11   LMC=11

     NM = 5   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
          1.0    2.0    3.0    4.0    5.0
          1.0    2.0    3.0    4.0    5.0
          1.0    2.0    3.0    4.0    5.0
          1.0    2.0    3.0    4.0    5.0
          1.0    2.0    3.0    4.0    5.0

     MATRIX B
          1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0
          5.0    5.0    5.0    5.0

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
         55.0   55.0   55.0   55.0
         55.0   55.0   55.0   55.0
         55.0   55.0   55.0   55.0
         55.0   55.0   55.0   55.0
         55.0   55.0   55.0   55.0
```

2.2.5 QAM1TT, PAM1TT
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B^T$)

(1) **Function**

Obtain the product of real matrix A and real matrix $B(C = [C\pm]A^T B^T)$

(2) **Usage**

Double precision:

CALL QAM1TT (A, LNA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)

Single precision:

CALL PAM1TT (A, LNA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA, NM	Input	Real transposed matrix A (two-dimensional array type).
2	LNA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{D / R}	LLB, NN	Input	Real transposed matrix B (two-dimensional array type).
6	LLB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{D / R}	LMC, NL	Input	Initial real matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]A^T B^T$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + A^T B^T$ ISW = 0: Obtain $C = A^T B^T$ ISW = −1: Obtain $C = C - A^T B^T$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 <$ NM $\leq$ LMC

(b) $0 <$ NN $\leq$ LNA

(c) $0 <$ NL $\leq$ LNB

(d) ISW $\in \{0, 1, -1\}$

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

Obtain $C = A^T B^T$.

(b) Input data

Matrices A and B, LNA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 6, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM OAM1TT
! *** EXAMPLE OF QAM1TT ***
      IMPLICIT NONE
      INTEGER LNA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=5, NL=4 )
      INTEGER IERR,I,J
      REAL(8) A(LNA,NM),B(LLB,NN),C(LMC,NL)
      COMMON A,B,C
!
      DO 100 I=1,NN
         READ(5,*) (A(I,J),J=1,NM)
  100 CONTINUE
      DO 110 I=1,NL
         READ(5,*) (B(I,J),J=1,NN)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NN
         WRITE(6,6020) (A(I,J),J=1,NM)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NL
         WRITE(6,6020) (B(I,J),J=1,NN)
  130 CONTINUE
!
      CALL QAM1TT(A,LNA,NM,NN,B,LLB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
```

```
         WRITE(6,6020) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  QAM1TT  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,6X,11(F7.1))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  QAM1TT  ***

 **  INPUT  **

     LNA=11   LLB=11   LMC=11

     NM = 5   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
          1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0
          5.0    5.0    5.0    5.0    5.0

     MATRIX B
          1.0    1.0    1.0    1.0    1.0
          2.0    2.0    2.0    2.0    2.0
          3.0    3.0    3.0    3.0    3.0
          4.0    4.0    4.0    4.0    4.0

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
         15.0   30.0   45.0   60.0
         15.0   30.0   45.0   60.0
         15.0   30.0   45.0   60.0
         15.0   30.0   45.0   60.0
         15.0   30.0   45.0   60.0
```

2.2.6 HAM1MM, GAM1MM
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm AB$)

(1) **Function**

Obtain the product of two complex matrices (Two-dimensional Array Type) ($C = [C\pm]AB$).

(2) **Usage**

Double precision:

```
CALL HAM1MM  (AR, AI, LMA, NM, NN, BR, BI, LNB, NL, CR, CI, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAM1MM  (AR, AI, LMA, NM, NN, BR, BI, LNB, NL, CR, CI, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LMA, NN	Input	Real part of complex matrix A (two-dimensional array type).
2	AI	{D / R}	LMA, NN	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	LMA	I	1	Input	Adjustable dimension of array AR and AI.
4	NM	I	1	Input	Number of rows in matrix A.
5	NN	I	1	Input	Number of columns in matrix A.
6	BR	{D / R}	LNB, NL	Input	Real part of complex matrix B (two-dimensional array type).
7	BI	{D / R}	LNB, NL	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	LNB	I	1	Input	Adjustable dimension of array BR and BI.
9	NL	I	1	Input	Number of columns in matrix B.
10	CR	{D / R}	LMC, NL	Input	Real part of Initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB$).
11	CI	{D / R}	LMC, NL	Input	Imaginary part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB$)
12	LMC	I	1	Input	Adjustable dimension of array CR and CI.
13	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB$ ISW = 0: Obtain $C = AB$ ISW = -1: Obtain $C = C - AB$
14	NT	I	1	Input	Number of tasks to be generated.
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i \\ 2+i & 2+2i & 2+3i & 2+4i \\ 3+i & 3+2i & 3+3i & 3+4i \\ 4+i & 4+2i & 4+3i & 4+4i \\ 5+i & 5+2i & 5+3i & 5+4i \end{bmatrix}$$

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, LMA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 4, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM UAM1MM
! *** EXAMPLE OF HAM1MM ***
      IMPLICIT NONE
      INTEGER LMA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=4 )
      INTEGER IERR,I,J
      REAL(8) AR(LMA,NN),BR(LNB,NL),CR(LMC,NL)
      REAL(8) AI(LMA,NN),BI(LNB,NL),CI(LMC,NL)
!
      DO 100 I=1,NM
         READ(5,*) (AR(I,J),AI(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (BR(I,J),BI(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LNB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (AR(I,J),AI(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6030) (BR(I,J),BI(I,J),J=1,NL)
  130 CONTINUE
!
      CALL HAM1MM(AR,AI,LMA,NM,NN,BR,BI,LNB,NL,CR,CI,LMC,ISW,NT,IERR)
!
      WRITE(6,6040) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6050) 'MATRIX C'
      DO 140 I=1,NM
```

```
        WRITE(6,6030) (CR(I,J),CI(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAM1MM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6030 FORMAT(1X,6X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,6X,5('(',F5.1,',',F5.1,')'))
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6050 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAM1MM  ***

 **  INPUT  **

     LMA=11   LNB=11   LMC=11

     NM = 4   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)
      (  5.0,  1.0)(  5.0,  2.0)(  5.0,  3.0)(  5.0,  4.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
      (  0.0, 60.0)(-15.0, 65.0)(-30.0, 70.0)(-45.0, 75.0)
      ( 15.0, 65.0)(  0.0, 75.0)(-15.0, 85.0)(-30.0, 95.0)
      ( 30.0, 70.0)( 15.0, 85.0)(  0.0,100.0)(-15.0,115.0)
      ( 45.0, 75.0)( 30.0, 95.0)( 15.0,115.0)(  0.0,135.0)
```

2.2.7 HAM1MH, GAM1MH
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm AB^*$)

(1) **Function**

Obtain the product of two complex matrices (Two-dimensional Array Type) ($C = [C\pm]AB^*$)

(2) **Usage**

Double precision:

CALL HAM1MH (AR, AI, LMA, NM, NN, BR, BI, LLB, NL, CR, CI, LMC, ISW, NT, IERR)

Single precision:

CALL GAM1MH (AR, AI, LMA, NM, NN, BR, BI, LLB, NL, CR, CI, LMC, ISW, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LMA, NN	Input	Real part of complex matrix A (two-dimensional array type).
2	AI	{D / R}	LMA, NN	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	LMA	I	1	Input	Adjustable dimension of array AR and AI.
4	NM	I	1	Input	Number of rows in matrix A.
5	NN	I	1	Input	Number of columns in matrix A.
6	BR	{D / R}	LLB, NN	Input	Real part of complex matrix B (two-dimensional array type).
7	BI	{D / R}	LLB, NN	Input	Real part of complex matrix B (two-dimensional array type).
8	LLB	I	1	Input	Adjustable dimension of array BR and BI.
9	NL	I	1	Input	Number of rows in matrix B.
10	CR	{D / R}	LMC, NL	Input	Real part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB^*$).
11	CI	{D / R}	LMC, NL	Input	Imaginary part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB^*$).
12	LMC	I	1	Input	Adjustable dimension of array CR and CI.
13	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB^*$ ISW = 0: Obtain $C = AB^*$ ISW = -1: Obtain $C = C - AB^*$
14	NT	I	1	Input	Number of tasks to be generated.
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NL} \leq \mathrm{LLB}$

(c) $\mathrm{NN} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

Obtain $C = AB^*$.

(b) Input data

Matrix A, matrix B, LMA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 4, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM UAM1MH
! *** EXAMPLE OF HAM1MH ***
      IMPLICIT NONE
      INTEGER LMA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=4 )
      INTEGER IERR,I,J
      REAL(8) AR(LMA,NN),BR(LLB,NN),CR(LMC,NL)
      REAL(8) AI(LMA,NN),BI(LLB,NN),CI(LMC,NL)
!
      DO 100 I=1,NM
         READ(5,*) (AR(I,J),AI(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NL
         READ(5,*) (BR(I,J),BI(I,J),J=1,NN)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (AR(I,J),AI(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NL
         WRITE(6,6020) (BR(I,J),BI(I,J),J=1,NN)
  130 CONTINUE
!
      CALL HAM1MH(AR,AI,LMA,NM,NN,BR,BI,LLB,NL,CR,CI,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6050) (CR(I,J),CI(I,J),J=1,NL)
  140 CONTINUE
```

```
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAM1MH  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6050 FORMAT(1X,6X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,6X,5('(',F5.1,',',F5.1,')'))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAM1MH  ***

 **  INPUT  **

     LMA=11   LLB=11   LMC=11

     NM = 4   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
      ( 60.0,  0.0)( 65.0, 15.0)( 70.0, 30.0)( 75.0, 45.0)
      ( 65.0,-15.0)( 75.0,  0.0)( 85.0, 15.0)( 95.0, 30.0)
      ( 70.0,-30.0)( 85.0,-15.0)(100.0,  0.0)(115.0, 15.0)
      ( 75.0,-45.0)( 95.0,-30.0)(115.0,-15.0)(135.0,  0.0)
```

2.2.8 HAM1HM, GAM1HM
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm A^*B$)

(1) **Function**

Obtain the product of complex matrix A and complex matrix $B(C = [C\pm]A^*B)$

(2) **Usage**

Double precision:

```
CALL HAM1HM  (AR, AI, LNA, NM, NN, BR, BI, LNB, NL, CR, CI, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAM1HM  (AR, AI, LNA, NM, NN, BR, BI, LNB, NL, CR, CI, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA, NM	Input	Real part of complex matrix A (two-dimensional array type).
2	AI	{D / R}	LNA, NM	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	LNA	I	1	Input	Adjustable dimension of array AR and AI.
4	NM	I	1	Input	Number of rows in matrices AR and AI.
5	NN	I	1	Input	Number of columns in matrices AR and AI.
6	BR	{D / R}	LNB, NL	Input	Real part of complex matrix B (two-dimensional array type).
7	BI	{D / R}	LNB, NL	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	LNB	I	1	Input	Adjustable dimension of array BR and BI.
9	NL	I	1	Input	Number of columns in matrix B.
10	CR	{D / R}	LMC, NL	Input	Real part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Real part of product ($C = [C\pm]A^*B$).
11	CI	{D / R}	LMC, NL	Input	Imaginary part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Imaginary part of product ($C = [C\pm]A^*B$).
12	LMC	I	1	Input	Adjustable dimension of array CR and CI.
13	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + A^*B$ ISW = 0: Obtain $C = A^*B$ ISW = -1: Obtain $C = C - A^*B$
14	NT	I	1	Input	Number of tasks to be generated.
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNA}, \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i \\ 2+i & 2+2i & 2+3i & 2+4i \\ 3+i & 3+2i & 3+3i & 3+4i \\ 4+i & 4+2i & 4+3i & 4+4i \end{bmatrix}$$

Obtain $C = A^*B$.

(b) Input data

Matrix A, matrix B, LNA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 6 and ISW = 0.

(c) Main program

```
      PROGRAM UAM1HM
! *** EXAMPLE OF HAM1HM ***
      IMPLICIT NONE
      INTEGER LNA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=4, NL=4 )
      INTEGER IERR,I,J
      REAL(8) AR(LNA,NM),BR(LNB,NL),CR(LMC,NL)
      REAL(8) AI(LNA,NM),BI(LNB,NL),CI(LMC,NL)
!
      DO 100 I=1,NN
         READ(5,*) (AR(I,J),AI(I,J),J=1,NM)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (BR(I,J),BI(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LNB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NN
         WRITE(6,6020) (AR(I,J),AI(I,J),J=1,NM)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6030) (BR(I,J),BI(I,J),J=1,NL)
  130 CONTINUE
!
      CALL HAM1HM(AR,AI,LNA,NM,NN,BR,BI,LNB,NL,CR,CI,LMC,ISW,NT,IERR)
!
      WRITE(6,6040) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6050) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6030) (CR(I,J),CI(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
```

```
 6000 FORMAT(/,&
             1X,'***  HAM1HM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6030 FORMAT(1X,4X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,4X,5('(',F5.1,',',F5.1,')'))
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6050 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAM1HM  ***

 **  INPUT  **

     LNA=11   LNB=11   LMC=11

     NM = 5   NN = 4   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
    ( 34.0,  0.0)( 38.0, 10.0)( 42.0, 20.0)( 46.0, 30.0)
    ( 38.0,-10.0)( 46.0,  0.0)( 54.0, 10.0)( 62.0, 20.0)
    ( 42.0,-20.0)( 54.0,-10.0)( 66.0,  0.0)( 78.0, 10.0)
    ( 46.0,-30.0)( 62.0,-20.0)( 78.0,-10.0)( 94.0,  0.0)
    ( 50.0,-40.0)( 70.0,-30.0)( 90.0,-20.0)(110.0,-10.0)
```

2.2.9 HAM1HH, GAM1HH
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm A^* B^*$)

(1) **Function**

Obtain the product of complex matrix A and complex matrix $B(C = [C\pm]A^* B^*)$

(2) **Usage**

Double precision:

```
CALL HAM1HH  (AR, AI, LNA, NM, NN, BR, BI, LLB, NL, CR, CI, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAM1HH  (AR, AI, LNA, NM, NN, BR, BI, LLB, NL, CR, CI, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA, NM	Input	Real part of complex matrix A (two-dimensional array).
2	AI	{D / R}	LNA, NM	Input	Imaginary part of complex matrix A (two-dimensional array).
3	LNA	I	1	Input	Adjustable dimension of array AR and AI.
4	NM	I	1	Input	Number of rows in matrix A.
5	NN	I	1	Input	Number of columns in matrix A.
6	BR	{D / R}	LLB, NN	Input	Real part of complex matrix A (two-dimensional array type).
7	BI	{D / R}	LLB, NN	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	LLB	I	1	Input	Adjustable dimension of array BR and BI.
9	NL	I	1	Input	Number of columns in matrix B.
10	CR	{D / R}	LMC, NL	Input	Real part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]A^*B^*$).
11	CI	{D / R}	LMC, NL	Input	Imaginary part of initial complex matrix C (If ISW = ± 1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]A^*B^*$).
12	LMC	I	1	Input	Adjustable dimension of array CR and CI.
13	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + A^*B^*$ ISW = 0: Obtain $C = A^*B^*$ ISW = -1: Obtain $C = C - A^*B^*$
14	NT	I	1	Input	Number of tasks to be generated.
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNA}$

(c) $0 < \mathrm{NL} \leq \mathrm{LNB}$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \\ 5+i & 5+2i & 5+3i & 5+4i & 5+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

Obtain$C = A^*B^*$.

(b) Input data

Matrix A, matrix B, LNA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 6 and ISW = 0.

(c) Main program

```
      PROGRAM UAM1HH
! *** EXAMPLE OF HAM1HH ***
      IMPLICIT NONE
      INTEGER LNA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=5, NL=4 )
      INTEGER IERR,I,J
      REAL(8) AR(LNA,NM),BR(LLB,NN),CR(LMC,NL)
      REAL(8) AI(LNA,NM),BI(LLB,NN),CI(LMC,NL)
!
      DO 100 I=1,NN
         READ(5,*) (AR(I,J),AI(I,J),J=1,NM)
  100 CONTINUE
      DO 110 I=1,NL
         READ(5,*) (BR(I,J),BI(I,J),J=1,NN)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NN
         WRITE(6,6020) (AR(I,J),AI(I,J),J=1,NM)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NL
         WRITE(6,6020) (BR(I,J),BI(I,J),J=1,NN)
  130 CONTINUE
!
      CALL HAM1HH(AR,AI,LNA,NM,NN,BR,BI,LLB,NL,CR,CI,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'REAL PART OF MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6050) (CR(I,J),CI(I,J),J=1,NL)
  140 CONTINUE
```

```
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAM1HH  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,4X,5('(',F5.1,',',F5.1,')'))
 6050 FORMAT(1X,4X,4('(',F6.1,',',F6.1,')'))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAM1HH  ***

 **  INPUT  **

     LNA=11   LLB=11   LMC=11

     NM = 5   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)
    (  5.0,  1.0)(  5.0,  2.0)(  5.0,  3.0)(  5.0,  4.0)(  5.0,  5.0)

     MATRIX B
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     REAL PART OF MATRIX C
    (   0.0, -60.0)(  15.0, -65.0)(  30.0, -70.0)(  45.0, -75.0)
    ( -15.0, -65.0)(   0.0, -75.0)(  15.0, -85.0)(  30.0, -95.0)
    ( -30.0, -70.0)( -15.0, -85.0)(   0.0,-100.0)(  15.0,-115.0)
    ( -45.0, -75.0)( -30.0, -95.0)( -15.0,-115.0)(   0.0,-135.0)
    ( -60.0, -80.0)( -45.0,-105.0)( -30.0,-130.0)( -15.0,-155.0)
```

2.2.10 HAN1MM, GAN1MM
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB$)

(1) **Function**

Obtain the product of two complex matrices (Two-dimensional Array Type) ($C = [C\pm]AB$).

(2) **Usage**

Double precision:

```
CALL HAN1MM  (A, LMA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAN1MM  (A, LMA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ D / R }	LMA, NN	Input	Complex matrix A (two-dimensional array type).
2	LMA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{ D / R }	LNB, NL	Input	Complex matrix B (two-dimensional array type).
6	LNB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{ D / R }	LMC, NL	Input	Initial complex matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB$ ISW = 0: Obtain $C = AB$ ISW = −1: Obtain $C = C - AB$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i \\ 2+i & 2+2i & 2+3i & 2+4i \\ 3+i & 3+2i & 3+3i & 3+4i \\ 4+i & 4+2i & 4+3i & 4+4i \\ 5+i & 5+2i & 5+3i & 5+4i \end{bmatrix}$$

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, LMA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 4, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM UAN1MM
! *** EXAMPLE OF HAN1MM ***
      IMPLICIT NONE
      INTEGER LMA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=4 )
      INTEGER IERR,I,J
      COMPLEX(8) A(LMA,NN),B(LNB,NL),C(LMC,NL)
!
      DO 100 I=1,NM
         READ(5,*) (A(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LNB,LMC,NM,NN,NL,ISW,NT
```

```
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
         WRITE(6,6020) (A(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6030) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL HAN1MM(A,LMA,NM,NN,B,LNB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6040) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6050) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6030) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAN1MM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6030 FORMAT(1X,6X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,6X,5('(',F5.1,',',F5.1,')'))
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6050 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAN1MM  ***

 **  INPUT  **

     LMA=11   LNB=11   LMC=11

     NM = 4   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)
      (  5.0,  1.0)(  5.0,  2.0)(  5.0,  3.0)(  5.0,  4.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
      (  0.0, 60.0)(-15.0, 65.0)(-30.0, 70.0)(-45.0, 75.0)
      ( 15.0, 65.0)(  0.0, 75.0)(-15.0, 85.0)(-30.0, 95.0)
      ( 30.0, 70.0)( 15.0, 85.0)(  0.0,100.0)(-15.0,115.0)
      ( 45.0, 75.0)( 30.0, 95.0)( 15.0,115.0)(  0.0,135.0)
```

2.2.11 HAN1MH, GAN1MH
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB^*$)

(1) **Function**

Obtain the product of two complex matrices (Two-dimensional Array Type) ($C = [C\pm]AB^*$)

(2) **Usage**

Double precision:

```
CALL HAN1MH  (A, LMA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAN1MH  (A, LMA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LMA, NN	Input	Complex matrix A (two-dimensional array type).
2	LMA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{D / R}	LLB, NN	Input	Complex matrix B (two-dimensional array type).
6	LLB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of rows in matrix B.
8	C	{D / R}	LMC, NL	Input	Initial complex matrix C (If ISW = $\pm$1) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB^*$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + AB^*$ ISW = 0: Obtain $C = AB^*$ ISW = -1: Obtain $C = C - AB^*$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMA}, \mathrm{LMC}$

(b) $0 < \mathrm{NL} \leq \mathrm{LLB}$

(c) $\mathrm{NN} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 1+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

Obtain $C = AB^*$.

(b) Input data

Matrix A, matrix B, LMA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 4, ISW = 0 and NT = 2.

(c) Main program

```
      PROGRAM UAN1MH
! *** EXAMPLE OF HAN1MH ***
      IMPLICIT NONE
      INTEGER LMA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LMA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=4, NN=5, NL=4 )
      INTEGER IERR,I,J
      COMPLEX(8) A(LMA,NN),B(LLB,NN),C(LMC,NL)
!
      DO 100 I=1,NM
         READ(5,*) (A(I,J),J=1,NN)
  100 CONTINUE
      DO 110 I=1,NL
         READ(5,*) (B(I,J),J=1,NN)
  110 CONTINUE
!
      WRITE(6,6000) LMA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NM
```

```
         WRITE(6,6020) (A(I,J),J=1,NN)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NL
         WRITE(6,6020) (B(I,J),J=1,NN)
  130 CONTINUE
!
      CALL HAN1MH(A,LMA,NM,NN,B,LLB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6050) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAN1MH  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LMA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6050 FORMAT(1X,6X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,6X,5('(',F5.1,',',F5.1,')'))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAN1MH  ***

 **  INPUT  **

     LMA=11   LLB=11   LMC=11

     NM = 4   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
      (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
      (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
      (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
      (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
      ( 60.0,  0.0)( 65.0, 15.0)( 70.0, 30.0)( 75.0, 45.0)
      ( 65.0,-15.0)( 75.0,  0.0)( 85.0, 15.0)( 95.0, 30.0)
      ( 70.0,-30.0)( 85.0,-15.0)(100.0,  0.0)(115.0, 15.0)
      ( 75.0,-45.0)( 95.0,-30.0)(115.0,-15.0)(135.0,  0.0)
```

2.2.12 HAN1HM, GAN1HM
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm A^*B$)

(1) **Function**

Obtain the product of complex matrix A and complex matrix $B(C = [C\pm]A^*B)$

(2) **Usage**

Double precision:

```
CALL HAN1HM  (A, LNA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAN1HM  (A, LNA, NM, NN, B, LNB, NL, C, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ Z / C }	LNA, NM	Input	Complex Matrix A (two-dimensional array type).
2	LNA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns of matrix A.
5	B	{ Z / C }	LNB, NL	Input	Complex matrix B (two-dimensional array type).
6	LNB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns of matrix B.
8	C	{ Z / C }	LMC, NL	Input	Initial complex matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]A^*B$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW = 1: Obtain $C = C + A^*B$ ISW = 0: Obtain $C = A^*B$ ISW = −1: Obtain $C = C - A^*B$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNA}, \mathrm{LNB}$

(c) $\mathrm{NL} > 0$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i \\ 2+i & 2+2i & 2+3i & 2+4i \\ 3+i & 3+2i & 3+3i & 3+4i \\ 4+i & 4+2i & 4+3i & 4+4i \end{bmatrix}$$

Obtain $C = A^*B$.

(b) Input data

Matrix A, Matrix B, LNA = 11, LNB = 11, LNC = 11, NM = 4, NN = 5, NL = 6 and ISW = 0.

(c) Main program

```
      PROGRAM UAN1HM
! *** EXAMPLE OF HAM1HM ***
      IMPLICIT NONE
      INTEGER LNA,LNB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LNB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=4, NL=4 )
      INTEGER IERR,I,J
      COMPLEX(8) A(LNA,NM),B(LNB,NL),C(LMC,NL)
!
      DO 100 I=1,NN
         READ(5,*) (A(I,J),J=1,NM)
  100 CONTINUE
      DO 110 I=1,NN
         READ(5,*) (B(I,J),J=1,NL)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LNB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NN
         WRITE(6,6020) (A(I,J),J=1,NM)
  120 CONTINUE
```

```
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NN
         WRITE(6,6030) (B(I,J),J=1,NL)
  130 CONTINUE
!
      CALL HAN1HM(A,LNA,NM,NN,B,LNB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6040) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6050) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6030) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAN1HM  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LNB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6030 FORMAT(1X,4X,4('(',F5.1,',',F5.1,')'))
 6020 FORMAT(1X,4X,5('(',F5.1,',',F5.1,')'))
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6050 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAN1HM  ***

 **  INPUT  **

     LNA=11   LNB=11   LMC=11

     NM = 5   NN = 4   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

     MATRIX B
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
    ( 34.0,  0.0)( 38.0, 10.0)( 42.0, 20.0)( 46.0, 30.0)
    ( 38.0,-10.0)( 46.0,  0.0)( 54.0, 10.0)( 62.0, 20.0)
    ( 42.0,-20.0)( 54.0,-10.0)( 66.0,  0.0)( 78.0, 10.0)
    ( 46.0,-30.0)( 62.0,-20.0)( 78.0,-10.0)( 94.0,  0.0)
    ( 50.0,-40.0)( 70.0,-30.0)( 90.0,-20.0)(110.0,-10.0)
```

2.2.13 HAN1HH, GAN1HH
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm A^*B^*$)

(1) **Function**

Obtain the product of complex matrix A and complex matrix $B(C = [C\pm]A^*B^*)$

(2) **Usage**

Double precision:

```
CALL HAN1HH (A, LNA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)
```

Single precision:

```
CALL GAN1HH (A, LNA, NM, NN, B, LLB, NL, C, LMC, ISW, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ Z / C }	LNA, NM	Input	Complex matrix A (two-dimensional array type).
2	LNA	I	1	Input	Adjustable dimension of array A.
3	NM	I	1	Input	Number of rows in matrix A.
4	NN	I	1	Input	Number of columns in matrix A.
5	B	{ Z / C }	LLB, NN	Input	Complex matrix B (two-dimensional array type).
6	LLB	I	1	Input	Adjustable dimension of array B.
7	NL	I	1	Input	Number of columns in matrix B.
8	C	{ Z / C }	LMC, NL	Input	Initial complex matrix C (If ISW $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]A^*B^*$).
9	LMC	I	1	Input	Adjustable dimension of array C.
10	ISW	I	1	Input	Processing switch. ISW $= 1$: Obtain $C = C + A^*B^*$ ISW $= 0$: Obtain $C = A^*B^*$ ISW $= -1$: Obtain $C = C - A^*B^*$
11	NT	I	1	Input	Number of tasks to be generated.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{NM} \leq \mathrm{LMC}$

(b) $0 < \mathrm{NN} \leq \mathrm{LNA}$

(c) $0 < \mathrm{NL} \leq \mathrm{LNB}$

(d) $\mathrm{ISW} \in \{0, 1, -1\}$

(e) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	NN was equal to 1.	Processing continues.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	

(6) **Notes**

None

(7) **Example**

(a) Problem

$$A = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \\ 5+i & 5+2i & 5+3i & 5+4i & 5+5i \end{bmatrix}$$

$$B = \begin{bmatrix} 1+i & 1+2i & 1+3i & 1+4i & 1+5i \\ 2+i & 2+2i & 2+3i & 2+4i & 2+5i \\ 3+i & 3+2i & 3+3i & 3+4i & 3+5i \\ 4+i & 4+2i & 4+3i & 4+4i & 4+5i \end{bmatrix}$$

Obtain $C = A^*B^*$.

(b) Input data

Matrix A, matrix B, LNA = 11, LLB = 11, LNC = 11, NM = 4, NN = 5, NL = 6 and ISW = 0.

(c) Main program

```
      PROGRAM UAN1HH
! *** EXAMPLE OF HAN1HH ***
      IMPLICIT NONE
      INTEGER LNA,LLB,LMC,ISW,NT,NM,NN,NL
      PARAMETER( LNA=11, LLB=11, LMC=11, ISW=0, NT=2 )
      PARAMETER( NM=5, NN=5, NL=4 )
      INTEGER IERR,I,J
      COMPLEX(8) A(LNA,NM),B(LLB,NN),C(LMC,NL)
!
      DO 100 I=1,NN
         READ(5,*) (A(I,J),J=1,NM)
  100 CONTINUE
      DO 110 I=1,NL
         READ(5,*) (B(I,J),J=1,NN)
  110 CONTINUE
!
      WRITE(6,6000) LNA,LLB,LMC,NM,NN,NL,ISW,NT
      WRITE(6,6010) 'MATRIX A'
      DO 120 I=1,NN
```

```
         WRITE(6,6020) (A(I,J),J=1,NM)
  120 CONTINUE
      WRITE(6,6010) 'MATRIX B'
      DO 130 I=1,NL
         WRITE(6,6020) (B(I,J),J=1,NN)
  130 CONTINUE
!
      CALL HAN1HH(A,LNA,NM,NN,B,LLB,NL,C,LMC,ISW,NT,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6040) 'MATRIX C'
      DO 140 I=1,NM
         WRITE(6,6050) (C(I,J),J=1,NL)
  140 CONTINUE
      STOP
!
 6000 FORMAT(/,&
             1X,'***  HAN1HH  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA=',I2,'   LLB=',I2,'   LMC=',I2,/,/,&
             1X,'     NM =',I2,'   NN =',I2,'   NL =',I2,/,/,&
             1X,'     ISW=',I2,'   NT =',I2,/,/,&
             1X,'     INPUT MATRIX')
 6010 FORMAT(/,&
             1X,5X,A)
 6020 FORMAT(1X,4X,5('(',F5.1,',',F5.1,')'))
 6050 FORMAT(1X,4X,4('(',F6.1,',',F6.1,')'))
 6030 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I4,/)
 6040 FORMAT(1X,'     OUTPUT MATRIX',/,/,&
             1X,5X,A)
      END
```

(d) Output results

```
***  HAN1HH  ***

 **  INPUT  **

     LNA=11   LLB=11   LMC=11

     NM = 5   NN = 5   NL = 4

     ISW= 0   NT = 2

     INPUT MATRIX

     MATRIX A
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)
    (  5.0,  1.0)(  5.0,  2.0)(  5.0,  3.0)(  5.0,  4.0)(  5.0,  5.0)

     MATRIX B
    (  1.0,  1.0)(  1.0,  2.0)(  1.0,  3.0)(  1.0,  4.0)(  1.0,  5.0)
    (  2.0,  1.0)(  2.0,  2.0)(  2.0,  3.0)(  2.0,  4.0)(  2.0,  5.0)
    (  3.0,  1.0)(  3.0,  2.0)(  3.0,  3.0)(  3.0,  4.0)(  3.0,  5.0)
    (  4.0,  1.0)(  4.0,  2.0)(  4.0,  3.0)(  4.0,  4.0)(  4.0,  5.0)

 **  OUTPUT  **

     IERR =    0

     OUTPUT MATRIX

     MATRIX C
    (   0.0, -60.0)(  15.0, -65.0)(  30.0, -70.0)(  45.0, -75.0)
    ( -15.0, -65.0)(   0.0, -75.0)(  15.0, -85.0)(  30.0, -95.0)
    ( -30.0, -70.0)( -15.0, -85.0)(   0.0,-100.0)(  15.0,-115.0)
    ( -45.0, -75.0)( -30.0, -95.0)( -15.0,-115.0)(   0.0,-135.0)
    ( -60.0, -80.0)( -45.0,-105.0)( -30.0,-130.0)( -15.0,-155.0)
```

Chapter 3

SIMULTANEOUS LINEAR EQUATIONS (DIRECT METHOD)

3.1 INTRODUCTION

This chapter describes subroutines that solve simultaneous linear equations and obtain the determinant value and inverse matrix of a matrix.

Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

In this library, subroutines having the following functions are provided individually for each set of matrix characteristics and storage mode.

(1) Perform triangular decomposition of coefficient matrix, then solve simultaneous linear equations.

(2) Perform triangular decomposition of coefficient matrix.

(3) Perform triangular decomposition of coefficient matrix and obtain condition number.

(4) Solve simultaneous linear equations after triangular decomposition of coefficient matrix

(5) Obtain determinant value and inverse matrix.

You can freely combine the various types of subroutines (1) through (5) to suit your processing needs. This enables you to perform efficient processing by eliminating unnecessary calculation steps.
For explanations of subroutines (4) and (5) (See Chapter 2 of <Basic Functions Vol. 2> .)

3.1.1 Methods of using subroutines

Methods of using subroutines are described below using a real matrix (two–dimensional array type) as an example.

(1) Simultaneous linear equations

(a) Using QBGMSL

CALL QBGMSL $(A, \cdots, \boldsymbol{b}, \cdots)$

Performs a triangular decomposition of coefficient matrix A and solves $A\boldsymbol{x} = \boldsymbol{b}$.

(b) Using QBGMLU and $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$

CALL QBGMLU $(A, \cdots)$

CALL $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$ $(A, \cdots, \boldsymbol{b}, \cdots)$

QBGMLU performs a triangular decomposition of coefficient matrix A, and $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$.

(c) Obtaining the condition number in addition to solving simultaneous linear equations

CALL QBGMLC $(A, \cdots,$ COND, $\cdots)$

CALL $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$ $(A, \cdots, \boldsymbol{b}, \cdots)$

QBGMLC calculates the condition number and performs a triangular decomposition of coefficient matrix A, and $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$.

(2) Determinant and inverse matrix

CALL QBGMLU $(A, \cdots)$

CALL $\begin{Bmatrix}\text{DBGMDI}\\ \text{RBGMDI}\end{Bmatrix}$ $(A, \cdots,$ DET, $\cdots)$

QBGMLU performs a triangular decomposition of matrix A, and $\begin{Bmatrix}\text{DBGMDI}\\ \text{RBGMDI}\end{Bmatrix}$ (See Section 2.2.7 of <Basic Functions Vol. 2>) obtains the determinant and inverse matrix.

(3) Improving the solution

(a) Using QBGMSL

$A_2 \leftarrow A$

$\boldsymbol{b_2} \leftarrow \boldsymbol{b}$

CALL QBGMSL $(A_2, \cdots, \boldsymbol{b}_2, \cdots)$

CALL $\begin{Bmatrix}\text{DBGMLX}\\ \text{RBGMLX}\end{Bmatrix}$ $(A, \cdots, A_2, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_2, \cdots)$

The subroutine shown above improves the solution obtained by using QBGMSL.

(b) Using QBGMLU and $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$

$A_2 \leftarrow A$

$\boldsymbol{b_2} \leftarrow \boldsymbol{b}$

CALL QBGMLU $(A_2, \cdots)$

CALL $\begin{Bmatrix}\text{DBGMLS}\\ \text{RBGMLS}\end{Bmatrix}$ $(A_2, \cdots, \boldsymbol{b}_2, \cdots)$

CALL $\left\{\begin{matrix}\text{DBGMLX}\\\text{RBGMLX}\end{matrix}\right\}$ $(A, \cdots, A_2, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_2, \cdots)$

QBGMLU performs a triangular decomposition of matrix A, $\left\{\begin{matrix}\text{DBGMLS}\\\text{RBGMLS}\end{matrix}\right\}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$, and $\left\{\begin{matrix}\text{DBGMLX}\\\text{RBGMLX}\end{matrix}\right\}$ (See Section 2.2.8 of <Basic Functions Vol. 2>) improves the solution.

3.1.2 Notes

(1) Since the parallel processing overhead significantly affects the computation cost if the order of the matrix is small, performance may be worse than when a non–parallel subroutine is used.

(2) To solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$, you could use the mathematical formula $\boldsymbol{x} = A^{-1}\boldsymbol{b}$. However, it would be ill–advised to solve these equations by obtaining the inverse matrix A^{-1} and multiplying it by the constant vector. For example, for a real matrix (two–dimensional array type), if you compare this method to one in which you obtain the solution by performing a triangular decomposition of the coefficient matrix, you would find that for n variables the inverse matrix method requires approximately n^3 multiplications, while the triangular decomposition method requires approximately $n^3/3$ multiplications. Clearly, the triangular decomposition method is preferable. Therefore, you should obtain the inverse matrix A^{-1} only if you actually need the inverse matrix itself.

(3) If you need to perform calculations many times for the same matrix such as when solving multiple sets of simultaneous linear equations where only the constant vector differs, it is more efficient to first perform the triangular decomposition once and then use that result repetitively thereafter.
Example :
To solve the equations:

$$\begin{aligned} A\boldsymbol{x}_1 &= \boldsymbol{b}_1 \\ A\boldsymbol{x}_2 &= \boldsymbol{b}_2 \end{aligned}$$

execute either:

CALL QBGMSL $(A, \cdots, \boldsymbol{b}_1, \cdots)$
CALL $\left\{\begin{matrix}\text{DBGMLS}\\ \text{RBGMLS}\end{matrix}\right\}$ $(A, \cdots, \boldsymbol{b}_2, \cdots)$

or

CALL QBGMLU $(A, \cdots)$
CALL $\left\{\begin{matrix}\text{DBGMLS}\\ \text{RBGMLS}\end{matrix}\right\}$ $(A, \cdots, \boldsymbol{b}_1, \cdots)$
CALL $\left\{\begin{matrix}\text{DBGMLS}\\ \text{RBGMLS}\end{matrix}\right\}$ $(A, \cdots, \boldsymbol{b}_2, \cdots)$

QBGMSL or QBGMLU performs the triangular decomposition of coefficient matrix A, and this result is only referred thereafter without its contents being changed.

(4) Two subroutines are provided for performing triangular decomposition. One obtains the condition number and the other does not. The subroutine that obtains the condition number requires many more calculations just to obtain the condition number. For a matrix of order n, it requires approximately n^2 more multiplications than the subroutine that does not obtain the condition number. Therefore, unless you specifically need the condition number, you can save execution time by performing triangular decomposition without obtaining the condition number.

3.1.3 Algorithms Used

3.1.3.1 Solution of Simultaneous Linear Equations

Assume that the following simultaneous linear equations are to be solved :

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ & & & & \vdots & & & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{cases}$$

If we define the coefficient matrix $A = (a_{ij})$ for $(i, j = 1, 2, \cdots, n)$, the right–hand side vector $\boldsymbol{b} = (b_i)$ for $(i = 1, 2, \cdots, n)$, and the solution vector $\boldsymbol{x} = (x_i)$ for $(i = 1, 2, \cdots, n)$, then the simultaneous linear equations shown above can be represented in matrix format as:

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

Now, if we decompose (LU decomposition) the coefficient matrix $A = (a_{ij})$ for $(i, j = 1, 2, \cdots, n)$ into $PA = LU$ as the product of the lower triangular matrix $L = (l_{ij})$ for $(i, j = 1, 2, \cdots, n)$ and upper triangular matrix $U = (u_{ij})$ for $(i, j = 1, 2, \cdots, n)$, the solution vector $\boldsymbol{x} = (x_i)$ for $(i = 1, 2, \cdots, n)$ is obtained by sequentially solving the following matrix equations:

$$\begin{cases} L\boldsymbol{y} = P\boldsymbol{b} \\ U\boldsymbol{x} = \boldsymbol{y} \end{cases}$$

where, the matrix P is an $n \times n$ row exchange matrix corresponding to n row exchanges, and the vector $\boldsymbol{y} = (y_i)$ for $(i = 1, 2, \cdots, n)$ is a work vector that is assigned intermediate calculation results. These matrix equations can be solved easily by using forward substitution and back substitution since the coefficient matrices are triangular matrices.

3.1.3.2 LU Decomposition (Gauss Method)

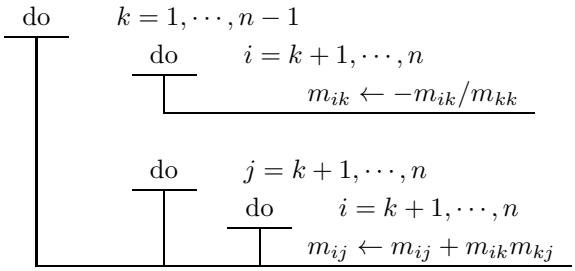
Consider the decomposition of the regular matrix $A = (a_{ij})$ for $(i, j = 1, 2, \cdots, n)$ into $PA = LU$ as the product of the lower triangular matrix $L = (l_{ij})$ for $(i, j = 1, 2, \cdots, n)$ and upper triangular matrix $U = (u_{ij})$ for $(i, j = 1, 2, \cdots, n)$. This kind of decomposition can be done at any time for a regular matrix. For simplicity, let $A' = PA = (a'_{ij})$ for $(i, j = 1, 2, \cdots, n)$. Then, from the definition of matrix multiplication, we have:

$$a'_{ij} = \begin{cases} \displaystyle\sum_{k=1}^{j-1} l_{ik}u_{ki} + u_{ij} & (i \leq j) \\ \displaystyle\sum_{k=1}^{j-1} l_{ik}u_{ki} + l_{ij}u_{jj} & (i > j) \end{cases}$$

To uniquely determine the decomposition, let $l_{ii} = 1$ for $(i = 1, 2, \cdots, n)$. By transforming this expression, various algorithms for calculating the LU decomposition are obtained. Consider an algorithm in which the matrix A' is assigned as the initial values of the $n \times n$ matrix $M = (m_{ij})$ for $(i, j = 1, 2, \cdots, n)$, and after the calculations are performed, the each element of the lower triangular matrix $L = (l_{ij})$ for $(i, j = 1, 2, \cdots, n)$ and upper triangular matrix $U = (u_{ij})$ for $(i, j = 1, 2, \cdots, n)$ is assigned for the element of that matrix respectively as follows:

$$M = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{ln} \\ -l_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\ -l_{31} & -l_{32} & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ -l_{n1} & -l_{n2} & \cdots & -l_{nn-1} & u_{nn} \end{bmatrix}$$

This algorithm (which is called the kji–SAXPY format Gauss method) is represented as follows.

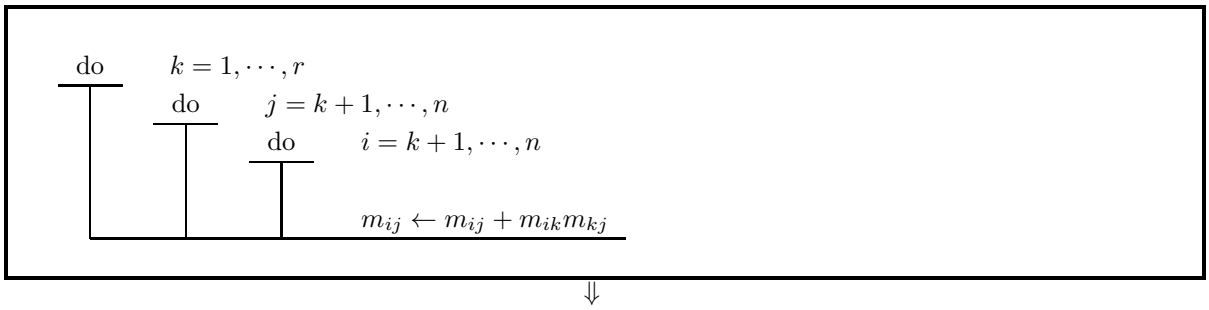


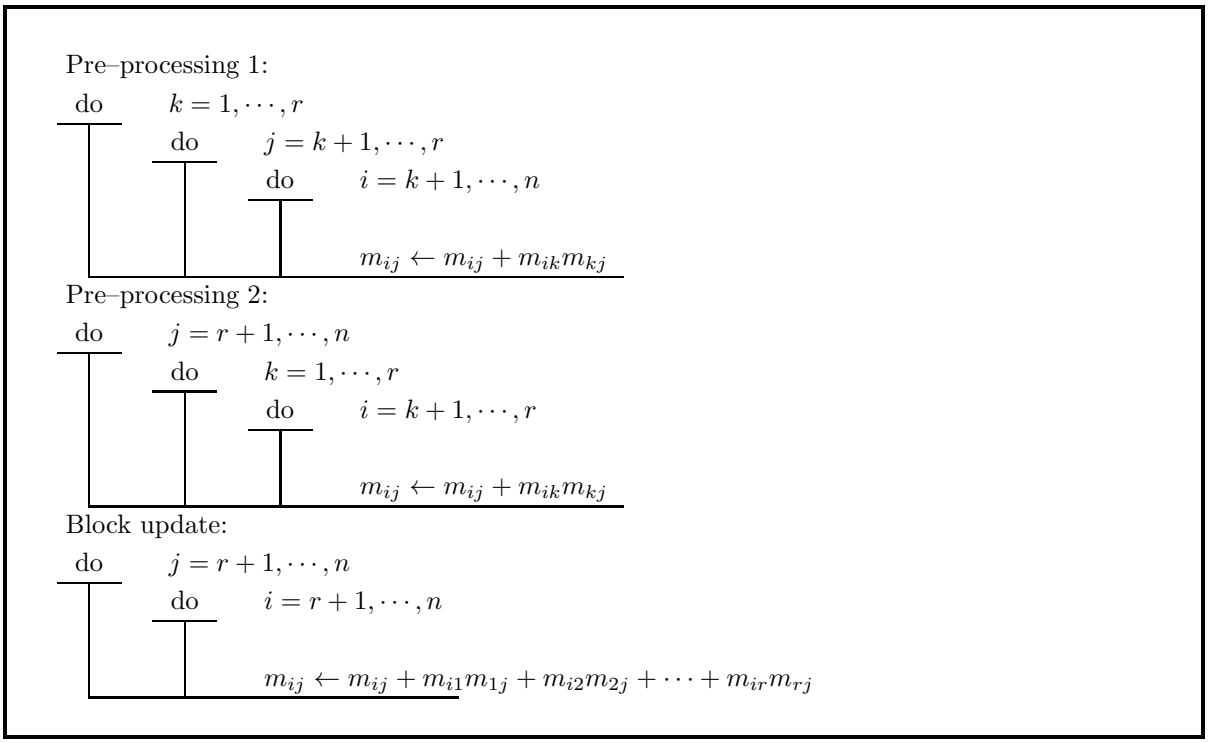
To prevent division by an element $m_{kk} \simeq 0$ when implementing the algorithm shown above, select the element m_{pk} for which the following equation holds:

$| m_{pk} | = \max_{k \le i \le n} | m_{ik} |$

and perform partial pivoting to exchange rows p and k before the division. The matrix that expresses the accumulation of these row exchange operations is the row exchange matrix P.

This library reduce the number of substitutions in order to increase calculation speed by taking r loops for k at a time and repeatedly applying the following algorithm transformation (blocking).





Also, by modifying r, speed is increased by executing this kind of transformation recursively even for pre–processing 1.

Parallelization of this algorithm is described below.

The algorithm mentioned above can be parallelized by taking the pre–processing 2 and block update portions of the algorithm and dividing up the loop for j of these portions according to the number of parallel processing tasks, and executing these loop subdivisions in parallel. This library increase efficiency by using task division that also takes into account pre–processing 1 for the next block update. (See reference bibliography (1).)

3.1.4 Reference Bibliography

(1) Robert, Y. and Sguazzero, P. , "The LU decomposition algorithm and its efficient FORTRAN implementation on IBM 3090 Vector Multiprocessor", IBM Tech. Rep. , ICE-0006, (1987).

3.2 REAL MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

3.2.1 QBGMSM

Simultaneous Linear Equations with Multiple Right-Hand Sides (Real Matrix)

(1) **Function**

QBGMSM uses Gauss' method to solve the simultaneous linear equations $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ having real matrix A (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the subroutine obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:

CALL QBGMSM (AB, LNA, N, M, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AB	D	LNA, (N + M)	Input	Matrix (real matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Matrix (real matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array AB
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	Number of right-hand side vectors, m
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step. (See Note (a))
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $0 < \mathrm{M}$

(c) $\mathrm{NT} \geq 1$

(5) **Error indicator**

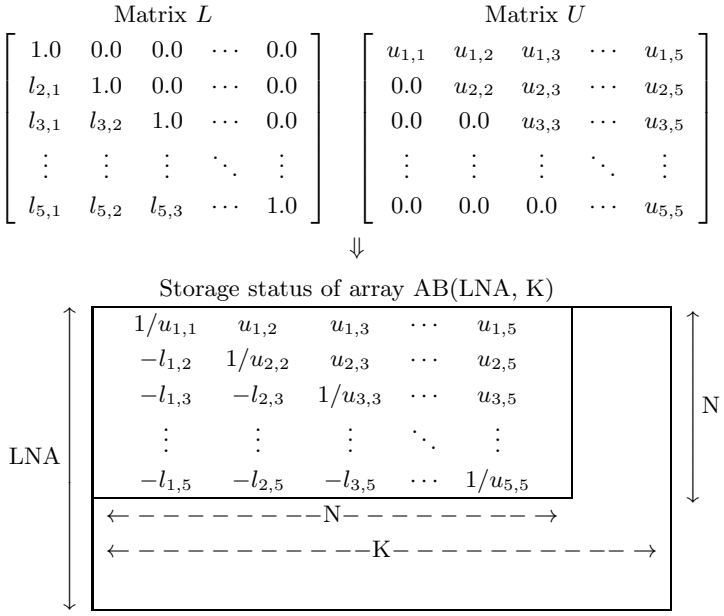
IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1	AB(1, N+i) ← AB(1, N+i)/AB(1, 1) ($i = 1, 2, \cdots,$ M) is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) This subroutine perform partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(b) The unit lower triangular matrix L is stored in the lower triangular portion of array AB with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array AB. In addition, the reciprocals of the diagonal components of U are stored.

Figure 3−1 Storage Status of Matrices L and U



Remarks

a. LNA ≥ N and N+M ≤ K must be hold.

(c) Shared memory parallel function for single-precision is not supported.

(7) **Example**

(a) ProblemSolve the following simultaneous linear equations.

$$\begin{bmatrix} 2 & 4 & -1 & 6 \\ -1 & -5 & 4 & 2 \\ 1 & 2 & 3 & 1 \\ 3 & 5 & -1 & -3 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix} = \begin{bmatrix} 36 & 11 \\ 15 & 0 \\ 22 & 7 \\ -6 & 4 \end{bmatrix}$$

(b) Input data

Array AB in which coefficient matrix A and constant vectors $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are stored, LNA=11, N=4 and M=2.

(c) Main program

```
      PROGRAM OBGMSM
! *** EXAMPLE OF QBGMSM ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11)
      PARAMETER (LMA = 5)
      PARAMETER (NT = 2)
      DIMENSION AB(LNA,LNA+LMA),IPVT(LNA)
      COMMON AB,IPVT
!
      READ (5,*) N
      READ (5,*) M
      WRITE (6,1000) N, M, NT
      DO 10 I = 1, N
         READ (5,*) (AB(I,J),J=1,N)
         WRITE (6,1100) (AB(I,J),J=1,N)
   10 CONTINUE
      WRITE (6,1200)
      DO 20 I = 1, N
         READ (5,*) (AB(I,N+J),J=1,M)
         WRITE (6,1100) (AB(I,N+J),J=1,M)
   20 CONTINUE
      WRITE (6,1300)
      CALL QBGMSM (AB,LNA,N,M,IPVT,NT,IERR)
      WRITE (6,1400) 'QBGMSM',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1600)
      DO 30 I = 1, N
         WRITE (6,1100) (AB(I,N+J),J=1,M)
```

```
   30 CONTINUE
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  QBGMSM  ***',/,&
            2X,'**  INPUT  **',/,&
            6X,'N =',I3,/,&
            6X,'M =',I3,/,&
            6X,'NT =',I3,/,&
            6X,'COEFFICIENT MATRIX')
 1100 FORMAT(7X,10(F11.4))
 1200 FORMAT(6X,'CONSTANT VECTOR')
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1600 FORMAT(6X,'SOLUTION')
      END
```

(d) Output results

```
***  QBGMSM  ***
 **  INPUT  **
     N =  4
     M =  2
     NT =  2
     COEFFICIENT MATRIX
           2.0000     4.0000    -1.0000     6.0000
          -1.0000    -5.0000     4.0000     2.0000
           1.0000     2.0000     3.0000     1.0000
           3.0000     5.0000    -1.0000    -3.0000
     CONSTANT VECTOR
          36.0000    11.0000
          15.0000     0.0000
          22.0000     7.0000
          -6.0000     4.0000
 **  OUTPUT  **
     IERR (QBGMSM) =    0
     SOLUTION
           1.0000     1.0000
           2.0000     1.0000
           4.0000     1.0000
           5.0000     1.0000
```

3.2.2 QBGMSL
Simultaneous Linear Equations (Real Matrix)

(1) **Function**

QBGMSL uses the Gauss method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:

CALL QBGMSL (A, LNA, N, B, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	D	LNA,N	Input	Coefficient matrix A (real matrix, two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (b) and (c))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	D	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step. (See Note (b))
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1	B(1) ← B(1)/A(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the subroutine 3.2.1 QBGMSM to perform the calculations. However, when 3.2.1 QBGMSM cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this subroutine only once and then call subroutine 2.2.5 $\left\{\begin{matrix}\text{DBGMLS}\\ \text{RBGMLS}\end{matrix}\right\}$ (in <Basic Functions Vol. 2>) the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculations by performing the LU decomposition of matrix A only once.

(b) This subroutine performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) The unit lower triangular matrix L is stored in the lower triangular portion of array A with sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U.

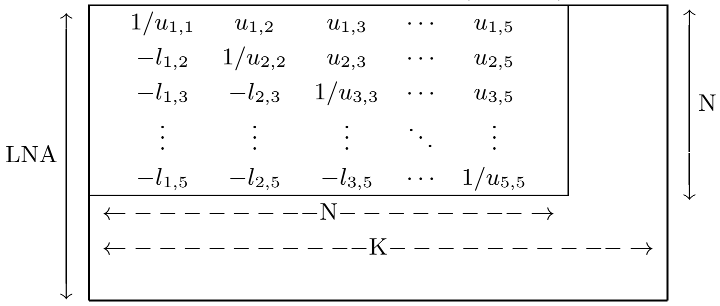
(d) Shared memory parallel function for single-precision is not supported.

$$
\text{Matrix } L \qquad\qquad\qquad\qquad \text{Matrix } U
$$

$$
\begin{bmatrix}
1.0 & 0.0 & 0.0 & \cdots & 0.0 \\
l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\
l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0
\end{bmatrix}
\quad
\begin{bmatrix}
u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\
0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\
0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & u_{5,5}
\end{bmatrix}
$$

$$\Downarrow$$

Storage status within array A(LNA, K)



Remarks

a. LNA $\geq$ N and N $\leq$ K must hold.

Figure 3−2 Storage Status of Matrices L and U

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$
\begin{bmatrix}
2 & 4 & -1 & 6 \\
-1 & -5 & 4 & 2 \\
1 & 2 & 3 & 1 \\
3 & 5 & -1 & -3
\end{bmatrix}
\begin{bmatrix}
x_1 \\ x_2 \\ x_3 \\ x_4
\end{bmatrix}
=
\begin{bmatrix}
36 \\ 15 \\ 22 \\ -6
\end{bmatrix}
$$

(b) Input data

Coefficient matrix A, LNA=11, N=4, constant vector $\boldsymbol{b}$ and NT=2.

(c) Main Program

```
      PROGRAM OBGMSL
! *** EXAMPLE OF QBGMSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11, NT = 2)
      DIMENSION A(LNA,LNA),B(LNA),IPVT(LNA)
      COMMON A,B,IPVT
!
      READ (5,*) N
      WRITE (6,1000) N, NT
      DO 10 I = 1, N
         READ (5,*) (A(I,J),J=1,N)
         WRITE (6,1100) (A(I,J),J=1,N)
   10 CONTINUE
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1200) (B(I),I=1,N)
      WRITE (6,1300)
      CALL QBGMSL (A,LNA,N,B,IPVT,NT,IERR)
      WRITE (6,1400) 'QBGMSL',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1600) (I,B(I),I=1,N)
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  QBGMSL  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'N =',I3,/,&
             6X,'NT =',I3,/,&
             6X,'COEFFICIENT MATRIX')
```

```
 1100 FORMAT(7X,10(G11.4))
 1200 FORMAT(6X,'CONSTANT VECTOR',/,(7X,F10.4))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1600 FORMAT(6X,'SOLUTION',/,(8X,'X(',I2,') =',D18.10))
      END
```

(d) Output results

```
***  QBGMSL  ***
 **  INPUT  **
     N =  4
     NT =  2
     COEFFICIENT MATRIX
        2.000      4.000     -1.000      6.000
       -1.000     -5.000      4.000      2.000
        1.000      2.000      3.000      1.000
        3.000      5.000     -1.000     -3.000
     CONSTANT VECTOR
         36.0000
         15.0000
         22.0000
         -6.0000
 **  OUTPUT  **
     IERR (QBGMSL) =    0
     SOLUTION
       X( 1) =  0.1000000000D+01
       X( 2) =  0.2000000000D+01
       X( 3) =  0.4000000000D+01
       X( 4) =  0.5000000000D+01
```

3.2.3 QBGMLU
LU Decomposition of a Real Matrix

(1) **Function**

QBGMLU uses the Gauss method to perform an LU decomposition of the real matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL QBGMLU (A, LNA, N, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }

No.	Argument	Type	Size	Input/ Output	Contents
1	A	D	LNA,N	Input	Real matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step. (See Note (b))
5	NT	I	1	Input	Number of tasks to be generated
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N is equal to 1	Contents of array A are not changed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular	

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array A with sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U. (See Fig. 3−2.)

(b) This subroutine performs partial pivoting. Pivoting information is stored in array IPVT for use by subsequent subroutines. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Shared memory parallel function for single-precision is not supported.

3.2.4 QBGMLC
LU Decomposition and Condition Number of a Real Matrix

(1) **Function**

QBGMLC uses the Gauss method to perform an LU decomposition and obtain the condition number of the real matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL QBGMLC (A, LNA, N, IPVT, COND, W1, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	D	LNA,N	Input	Real matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step (See Note (b))
5	COND	D	1	Output	Reciprocal of the condition number
6	W1	D	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N is equal to 1	Contents of array A are not changed and COND ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular.	Processing is aborted. The condition number is not obtained.

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array A with sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U. (See Fig. 3−2.)

(b) This subroutine partial performs pivoting. Pivoting information is stored in array IPVT for use by subsequent subroutines. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Although the condition number is defined by $\|A\| \cdot \|A^{-1}\|$, the one obtained by this subroutine is an approximation.

(d) Shared memory parallel function for single-precision is not supported.

3.3 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (REAL ARGUMENT TYPE)

3.3.1 HBGMSM

Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix)

(1) **Function**

HBGMSM uses Gauss' method to solve the simultaneous linear equations $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ having complex matrix A (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the subroutine obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:

CALL HBGMSM (ABR, ABI, LNA, N, M, IPVT, W1, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	ABR	D	LNA, (N + M)	Input	Real part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Real part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	ABI	D	LNA, (N + M)	Input	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
3	LNA	I	1	Input	Adjustable dimension of arrays ABR and ABI
4	N	I	1	Input	Order of matrix A
5	M	I	1	Input	Number of right-hand side vectors, m
6	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step. (See Note (a))
7	W1	D	N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $0 < \mathrm{M}$

(c) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1	ABR(1, N+i) ← (ABR(1, N+i) × ABR(1, 1) + ABI(1, N+i) × ABI(1, 1)) / (ABR(1, 1)^2 + ABI(1, 1)^2) , ABI(1, N+i) ← (ABI(1, N+i) × ABR(1, 1) − ABR(1, N+i) × ABI(1, 1)) / (ABR(1, 1)^2 + ABI(1, 1)^2) (i=1, 2, ⋯, M)
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) This subroutine perform partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(b) The unit lower triangular matrix L is stored in the lower triangular portion of array ABR and ABI with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array ABR and ABI. In addition, the reciprocals of the diagonal components of U are stored.

(c) Shared memory parallel function for single-precision is not supported.

Figure 3−3 Storage Status of Matrices L and U

Matrix L

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0 \end{bmatrix}$$

Matrix U

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\ 0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\ 0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & u_{5,5} \end{bmatrix}$$

$\Downarrow$

Storage status within array ABR(LNA, K)

$\Re\{1/u_{1,1}\}$	$\Re\{u_{1,2}\}$	$\Re\{u_{1,3}\}$	$\cdots$	$\Re\{u_{1,5}\}$
$\Re\{-l_{1,2}\}$	$\Re\{1/u_{2,2}\}$	$\Re\{u_{2,3}\}$	$\cdots$	$\Re\{u_{2,5}\}$
$\Re\{-l_{1,3}\}$	$\Re\{-l_{2,3}\}$	$\Re\{1/u_{3,3}\}$	$\cdots$	$\Re\{u_{3,5}\}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\Re\{-l_{1,5}\}$	$\Re\{-l_{2,5}\}$	$\Re\{-l_{3,5}\}$	$\cdots$	$\Re\{1/u_{5,5}\}$

LNA ↕ N

←−−−−−−−−−−−−−−−−N−−−−−−−−−−−−−−−→

←−−−−−−−−−−−−−−−−−−K−−−−−−−−−−−−−−−−−−→

Storage status within array ABI(LNA, K)

$\Im\{1/u_{1,1}\}$	$\Im\{u_{1,2}\}$	$\Im\{u_{1,3}\}$	$\cdots$	$\Im\{u_{1,5}\}$
$\Im\{-l_{1,2}\}$	$\Im\{1/u_{2,2}\}$	$\Im\{u_{2,3}\}$	$\cdots$	$\Im\{u_{2,5}\}$
$\Im\{-l_{1,3}\}$	$\Im\{-l_{2,3}\}$	$\Im\{1/u_{3,3}\}$	$\cdots$	$\Im\{u_{3,5}\}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\Im\{-l_{1,5}\}$	$\Im\{-l_{2,5}\}$	$\Im\{-l_{3,5}\}$	$\cdots$	$\Im\{1/u_{5,5}\}$

LNA ↕ N

←−−−−−−−−−−−−−−−−N−−−−−−−−−−−−−−−→

←−−−−−−−−−−−−−−−−−−K−−−−−−−−−−−−−−−−−−→

Remarks

a. LNA $\geq$ N and N+M $\leq$ K must be hold.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 4+2i & 3+9i & 4+i & 7+9i \\ 5+7i & 4i & 4+7i & 2+5i \\ 9+3i & 6+2i & 9+5i & 8+5i \\ 1+5i & 7+9i & 3+5i & 2+4i \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Input data

Array ABR and ABI in which coefficient matrix A and constant vectors $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are stored, LNA=11, N=4 and M=4.

(c) Main program

```
      PROGRAM UBGMSM
! *** EXAMPLE OF HBGMSM ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11)
      PARAMETER (LMA = 5)
      PARAMETER (NT = 2)
      DIMENSION ABR(LNA,LNA+LMA),ABI(LNA,LNA+LMA),IPVT(LNA),W(LNA)
!
```

```
      READ (5,*) N
      READ (5,*) M
      WRITE (6,1000) N, M
      DO 10 I = 1, N
         READ (5,*) (ABR(I,J),ABI(I,J),J=1,N)
         WRITE (6,1100) (ABR(I,J),ABI(I,J),J=1,N)
   10 CONTINUE
      WRITE (6,1200)
      DO 20 I = 1, N
         READ (5,*) (ABR(I,N+J),ABI(I,N+J),J=1,M)
         WRITE (6,1100) (ABR(I,N+J),ABI(I,N+J),J=1,M)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBGMSM (ABR,ABI,LNA,N,M,IPVT,W,NT,IERR)
      WRITE (6,1400) 'HBGMSM',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1600)
      DO 30 I = 1, N
         WRITE (6,1100) (ABR(I,N+J),ABI(I,N+J),J=1,M)
   30 CONTINUE
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  HBGMSM  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'N =',I3,/,&
             6X,'M =',I3,/,&
             6X,'COEFFICIENT MATRIX')
 1100 FORMAT(7X,4('(',F8.4,',',F8.4,')'))
 1200 FORMAT(6X,'CONSTANT VECTORS')
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1600 FORMAT(6X,'SOLUTION')
      END
```

(d) Output results

```
***  HBGMSM  ***
 **  INPUT  **
     N =  4
     M =  4
     COEFFICIENT MATRIX
      (  4.0000,  2.0000)(  3.0000,  9.0000)(  4.0000,  1.0000)(  7.0000,  9.0000)
      (  6.0000,  7.0000)(  0.0000,  4.0000)(  4.0000,  7.0000)(  2.0000,  5.0000)
      (  9.0000,  3.0000)(  6.0000,  2.0000)(  9.0000,  5.0000)(  8.0000,  5.0000)
      (  1.0000,  5.0000)(  7.0000,  9.0000)(  3.0000,  5.0000)(  2.0000,  4.0000)
     CONSTANT VECTORS
      (  1.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  1.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  0.0000,  0.0000)(  1.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)(  1.0000,  0.0000)
 **  OUTPUT  **
     IERR (HBGMSM) =    0
     SOLUTION
      (  0.0133, -0.0730)(  0.1814, -0.2467)( -0.1840,  0.1782)( -0.1039, -0.0560)
      ( -0.0178, -0.0189)( -0.0680, -0.0696)( -0.0128,  0.1001)(  0.0415, -0.0657)
      ( -0.0353,  0.1382)( -0.0585,  0.1700)(  0.1333, -0.2410)(  0.1314,  0.0191)
      (  0.0494, -0.0686)( -0.0096,  0.1300)(  0.0885, -0.0709)( -0.0462,  0.0662)
```

3.3.2 HBGMSL
Simultaneous Linear Equation (Complex Matrix)

(1) **Function**

HBGMSL uses the Gauss method to solve the simultaneous equations $A\boldsymbol{x} = \boldsymbol{b}$ having the coefficient matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL HBGMSL (AR, AI, LNA, N, BR, BI, IPVT, W, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	D	LNA,N	Input	Real part of coefficient matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (b) and (c))
2	AI	D	LNA,N	Input	Imaginary part of coefficient matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (b) and (c))
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	BR	D	N	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution vector $\boldsymbol{x}$
6	BI	D	N	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution vector $\boldsymbol{x}$
7	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step (See Note (b))
8	W	D	N	Work	Work area
9	NT	I	1	Input	Number of tasks to be generated
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1	BR(1) $\leftarrow$ {BR(1) $\times$ AR(1,1) + BI(1) $\times$ AI(1,1)}/{AR(1,1)^2 + AI(1,1)^2} BI(1) $\leftarrow$ {BI(1) $\times$ AR(1,1) − BR(1) $\times$ AI(1,1)}/{AR(1,1)^2 + AI(1,1)^2} are performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of the coefficient matrix A. A is singular.	

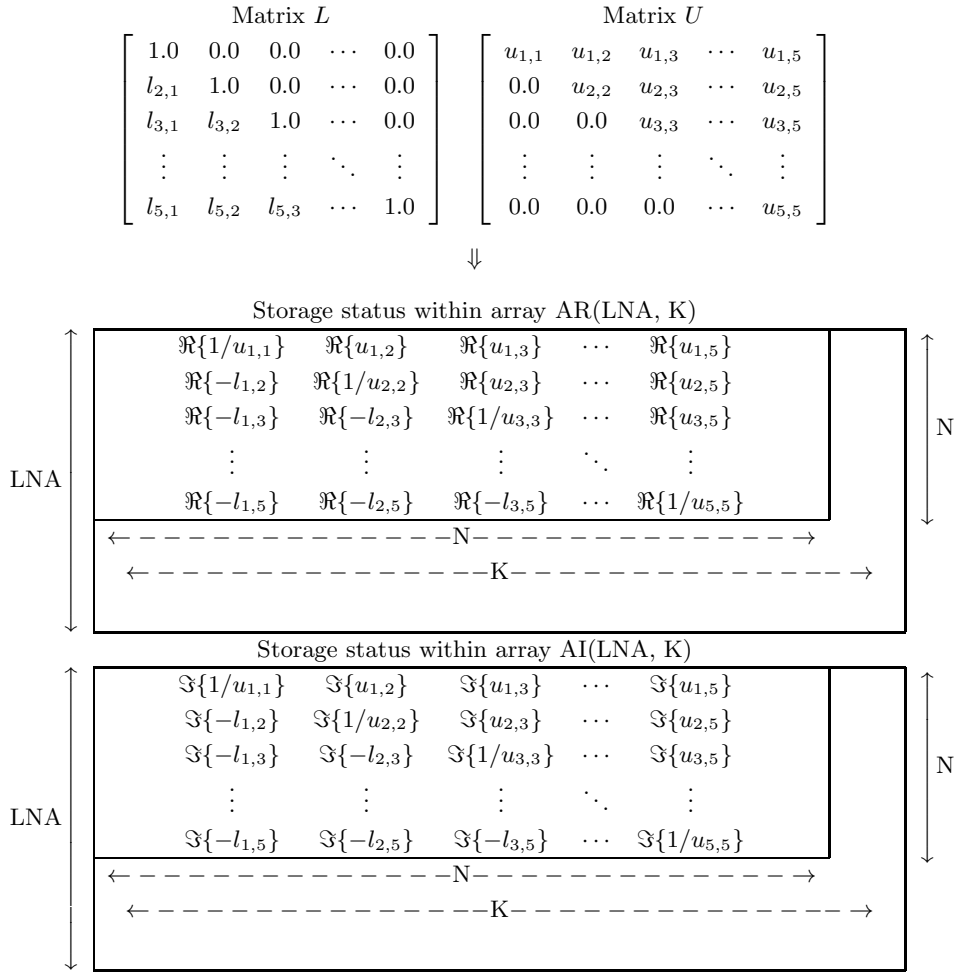
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the subroutine3.3.1 HBGMSM to perform the calculations. However, when 3.3.1 HBGMSM cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this subroutine only once and then call subroutine 2.3.5 $\left\{\begin{matrix}\text{ZBGMLS}\\ \text{CBGMLS}\end{matrix}\right\}$ (in <Basic Functions Vol. 2>) the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculations by performing the LU decomposition of matrix A only once.

(b) This subroutine performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) The unit lower triangular matrix L is stored in the lower triangular portions of arrays AR and AI with sign changed, and the upper triangular matrix U is stored in the upper triangular portions. However, since the diagonal components of L always are 1.0, they are not stored in arrays AR and AI. Also, reciprocals are stored for the diagonal components of U. In Fig. 3−4, $\Re\{z\}$ and $\Im\{z\}$ denote a real part and an imaginary part of a complex number z, respectively.

(d) Shared memory parallel function for single-precision is not supported.



Remarks

a. LNA ≥ N, N ≤ K must hold.

Figure 3−4 Storage Status of Matrices L and U

(7) **Example**

(a) Problem

Obtain the condition number by solving the following simultaneous linear equations.

$$\begin{bmatrix} 5+8i & 7+i & 6+3i & 1+2i \\ 1+i & 9+5i & 4+i & 5 \\ 4i & 3+3i & 4+2i & 6+9i \\ 7+8i & 6 & 7+6i & 10+4i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3+20i \\ -6+7i \\ -6i \\ 13i \end{bmatrix}$$

(b) Input data

Coefficient matrix real part AR and imaginary part AI, LNA=11, N=4 and constant vector $\boldsymbol{b}$.

(c) Main program

```
      PROGRAM UBGMSL
!     *** EXAMPLE OF HBGMLC,ZBGMLS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NT
      PARAMETER (LNA = 11,LNW = 22,NT=2)
      DIMENSION AR(LNA,LNA),AI(LNA,LNA),BR(LNA),BI(LNA),IPVT(LNA)
      DIMENSION W1(LNW)
!
      READ (5,*) N
```

```
      WRITE (6,1000) N
      DO 10 I = 1, N
         READ (5,*) (AR(I,J),AI(I,J),J=1,N)
         WRITE (6,1100) (AR(I,J),AI(I,J),J=1,N)
   10 CONTINUE
      READ (5,*) (BR(I),BI(I),I=1,N)
      WRITE (6,1200)
      DO 20 I = 1, N
         WRITE (6,1300) BR(I),BI(I)
   20 CONTINUE
      WRITE (6,1400)
      CALL HBGMLC (AR,AI,LNA,N,IPVT,COND,W1,NT,IERR)
      WRITE (6,1500) 'ZBGMLC',IERR
      IF (IERR .GE. 3000) STOP
      COND = 1.0D0/COND
      CALL ZBGMLS (AR,AI,LNA,N,BR,BI,IPVT,KERR)
      WRITE (6,1500) 'ZBGMLS',KERR
      WRITE (6,1600) COND
      WRITE (6,1700)
      DO 30 I = 1, N
         WRITE (6,1800) I,BR(I),BI(I)
   30 CONTINUE
      STOP
!
 1000 FORMAT (' ',/,/,' ***  HBGMLC,ZBGMLS  ***',&
             /,2X,'**  INPUT  **',&
             /,6X,'N =',I3,&
             /,6X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (6X,4(' (',F5.1,' ,',F5.1,' )'))
 1200 FORMAT (6X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1300 FORMAT (6X,'  ' (',F5.1,' ,',F5.1,' )')
 1400 FORMAT (2X,'**  OUTPUT  **')
 1500 FORMAT (6X,'IERR (',A6,') =',I5)
 1600 FORMAT (6X,'CONDITION NUMBER =',D18.10)
 1700 FORMAT (6X,'SOLUTION ( REAL, IMAGINARY )')
 1800 FORMAT (6X,'    X(',I2,') = (',D18.10,' ,',D18.10,' )')
      END
```

(d) Output results

```
***  HBGMLC,ZBGMLS  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  5.0 ,  8.0 ) (  7.0 ,  1.0 ) (  6.0 ,  3.0 ) (  1.0 ,  2.0 )
      (  1.0 ,  1.0 ) (  9.0 ,  5.0 ) (  4.0 ,  1.0 ) (  5.0 ,  0.0 )
      (  0.0 ,  4.0 ) (  3.0 ,  3.0 ) (  4.0 ,  2.0 ) (  6.0 ,  9.0 )
      (  7.0 ,  8.0 ) (  6.0 ,  0.0 ) (  7.0 ,  6.0 ) ( 10.0 ,  4.0 )
     CONSTANT VECTOR ( REAL, IMAGINARY )
      (  3.0 , 20.0 )
      ( -6.0 ,  7.0 )
      (  0.0 , -6.0 )
      (  0.0 , 13.0 )
 **  OUTPUT  **
     IERR (ZBGMLC) =    0
     IERR (ZBGMLS) =    0
     CONDITION NUMBER =  0.6279263302D+01
     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.1000000000D+01 ,  0.1000000000D+01 )
         X( 2) = ( -0.2220446049D-15 ,  0.1000000000D+01 )
         X( 3) = (  0.1000000000D+01 , -0.4996003611D-15 )
         X( 4) = ( -0.1000000000D+01 , -0.1000000000D+01 )
```

3.3.3 HBGMLU
LU Decomposition of a Complex Matrix

(1) **Function**

HBGMLU uses the Gauss method to perform an LU decomposition of the complex matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL HBGMLU (AR, AI, LNA, N, IPVT, W, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	D	LNA,N	Input	Real part of complex matrix A (two-dimensional array type)
				Output	Real part of upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	AI	D	LNA,N	Input	Imaginary part of complex matrix A (two-dimensional array type)
				Output	Imaginary part of upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i int the i-th processing step. (See Note (b))
6	W	D	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1	The contents of arrays AR and AI are unchanged.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular.	

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portions of arrays AR and AI with sign changed, and the upper triangular matrix U is stored in the upper triangular portions. However, since the diagonal components of L always are 1.0, they are not stored in arrays AR and AI. Also, reciprocals are stored for the diagonal components of U. (Refer to Fig. 3−4.)

(b) This subroutine performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Shared memory parallel function for single-precision is not supported.

3.3.4 HBGMLC
LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

HBGMLC uses the Gauss method to perform an LU decomposition and obtain the condition number of the complex matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL HBGMLC (AR, AI, LNA, N, IPVT, COND, W, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	D	LNA,N	Input	Real part of complex matrix A (two-dimensional array type)
				Output	Real part of upper triangular matrix L and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	AI	D	LNA,N	Input	Imaginary part of complex matrix A (two-dimensional array type)
				Output	Imaginary part of upper triangular matrix L and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step (See Note (b))
6	COND	D	1	Output	Reciprocal of the condition number
7	W	D	$2 \times$ N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1	The contents of arrays AR and AI are unchanged. COND ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular.	Processing is aborted. The condition number is not obtained.

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portions of arrays AR and AI with sign changed, and the upper triangular matrix U is stored in the upper triangular portions. However, since the diagonal components of L always are 1.0, they are not stored in arrays AR and AI. Also, reciprocals are stored for the diagonal components of U. (Refer to Fig. 3−4.)

(b) This subroutine performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j(i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Although the condition number is defined by $\|A\| \cdot \|A^{-1}\|$, the one obtained by this subroutine is an approximation.

(d) Shared memory parallel function for single-precision is not supported.

3.4 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (COMPLEX ARGUMENT TYPE)

3.4.1 HBGNSM

Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix)

(1) **Function**

HBGNSM uses Gauss' method to solve the simultaneous linear equations $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ having complex matrix A (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the subroutine obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:

CALL HBGNSM (AB, LNA, N, M, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AB	Z	LNA, (N + M)	Input	Matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array AB
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	Number of right-hand side vectors, m
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of row exchanged with row i in the i-th processing step. (See Note (a))
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $0 < \mathrm{M}$

(c) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	$\mathrm{AB}(1, \mathrm{N}+\mathrm{i}) \leftarrow \mathrm{AB}(1, \mathrm{N}+\mathrm{i})/\mathrm{AB}(1,1)$ $(i = 1, 2, \cdots, \mathrm{M})$ is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) This subroutine perform partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j ($\mathrm{i} \leq \mathrm{j}$), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(b) The unit lower triangular matrix L is stored in the lower triangular portion of array AB with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array AB. In addition, the reciprocals of the diagonal components of U are stored. (See Figure 3−1 in Section 3.2.1).

(c) Shared memory parallel function for single-precision is not supported.

(7) **Example**

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} 4+2i & 3+9i & 4+i & 7+9i \\ 5+7i & 4i & 4+7i & 2+5i \\ 9+3i & 6+2i & 9+5i & 8+5i \\ 1+5i & 7+9i & 3+5i & 2+4i \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Input data
Array AB in which coefficient matrix A and constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, LNA=11, N=4, M=4 and NT=2.

(c) Main program

```
      PROGRAM UBGNSM
! *** EXAMPLE OF HBGNSM ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11)
      PARAMETER (LMA = 5)
      PARAMETER (NT = 2)
      COMPLEX(8) AB
      DIMENSION AB(LNA,LNA+LMA),IPVT(LNA)
!
      READ (5,*) N
      READ (5,*) M
      WRITE (6,1000) N, M, NT
      DO 10 I = 1, N
         READ (5,*) (AB(I,J),J=1,N)
         WRITE (6,1100) (AB(I,J),J=1,N)
   10 CONTINUE
      WRITE (6,1200)
      DO 20 I = 1, N
         READ (5,*) (AB(I,N+J),J=1,M)
         WRITE (6,1100) (AB(I,N+J),J=1,M)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBGNSM (AB,LNA,N,M,IPVT,NT,IERR)
      WRITE (6,1400) 'HBGNSM',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1600)
      DO 30 I = 1, N
         WRITE (6,1100) (AB(I,N+J),J=1,M)
   30 CONTINUE
      STOP
!
 1000 FORMAT(1X,/,/,&
             1X,'***  ZBGNSM  ***',/,/,&
             1X,1X,'**  INPUT  **',/,/,&
             1X,5X,'N  =',I3,/,&
             1X,5X,'M  =',I3,/,&
             1X,5X,'NT =',I3,/,&
             /,1X,5X,'COEFFICIENT MATRIX')
 1100 FORMAT(1X,6X,4('(',F8.4,',',F8.4,')'))
 1200 FORMAT(/,1X,5X,'CONSTANT VECTORS')
 1300 FORMAT(/,1X,1X,'**  OUTPUT  **',/)
 1400 FORMAT(1X,5X,'IERR (',A6,') =',I5)
 1600 FORMAT(/,1X,5X,'SOLUTION')
      END
```

(d) Output results

```
***  ZBGNSM  ***

 **  INPUT  **

     N  =  4
     M  =  4
     NT =  2

     COEFFICIENT MATRIX
      (  4.0000,  2.0000)(  3.0000,  9.0000)(  4.0000,  1.0000)(  7.0000,  9.0000)
      (  6.0000,  7.0000)(  0.0000,  4.0000)(  4.0000,  7.0000)(  2.0000,  5.0000)
      (  9.0000,  3.0000)(  6.0000,  2.0000)(  9.0000,  5.0000)(  8.0000,  5.0000)
      (  1.0000,  5.0000)(  7.0000,  9.0000)(  3.0000,  5.0000)(  2.0000,  4.0000)

     CONSTANT VECTORS
      (  1.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  1.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  0.0000,  0.0000)(  1.0000,  0.0000)(  0.0000,  0.0000)
      (  0.0000,  0.0000)(  0.0000,  0.0000)(  0.0000,  0.0000)(  1.0000,  0.0000)

 **  OUTPUT  **

     IERR (HBGNSM) =    0

     SOLUTION
      (  0.0133, -0.0730)(  0.1814, -0.2467)( -0.1840,  0.1782)( -0.1039, -0.0560)
      ( -0.0178, -0.0189)( -0.0680, -0.0696)( -0.0128,  0.1001)(  0.0415, -0.0657)
      ( -0.0353,  0.1382)( -0.0585,  0.1700)(  0.1333, -0.2410)(  0.1314,  0.0191)
      (  0.0494, -0.0686)( -0.0096,  0.1300)(  0.0885, -0.0709)( -0.0462,  0.0662)
```

3.4.2 HBGNSL
Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

HBGNSL uses the Gauss method or the Crout method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the complex matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:

CALL HBGNSL (A, LNA, N, B, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	Z	LNA,N	Input	Coefficient matrix (complex matrix, two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (b) and (c))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	Z	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row exchanged with row i in the i-th processing step. (See Note (b))
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$ and $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	B(1) ← B(1)/A(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the subroutine3.4.1 HBGNSM to perform the calculations. However, when 3.4.1 HBGNSM cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this subroutine only once and then call subroutine 2.4.5 $\left\{\begin{matrix}\text{ZBGNLS}\\\text{CBGNLS}\end{matrix}\right\}$ the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculation by performing the LU decomposition of matrix A only once.

(b) This subroutine performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j (i ≤ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) The unit lower triangular matrix L is stored in the lower triangular portion of array A with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U. (See Figure 3−2 in Section 3.2.2).

(d) Shared memory parallel function for single-precision is not supported.

(7) **Example**

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5+8i & 7+i & 6+3i & 1+2i \\ 1+i & 9+5i & 4+i & 5 \\ 4i & 3+3i & 4+2i & 6+9i \\ 7+8i & 6 & 7+6i & 10+4i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3+20i \\ -6+7i \\ -6i \\ 13i \end{bmatrix}$$

(b) Input data
Coefficient matrix A, LNA = 11, N = 4, constant vector $\boldsymbol{b}$ and NT=2.

(c) Main program

```
      PROGRAM OBGNSL
!     *** EXAMPLE OF OBGNLC,ZBGNLS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11,LNW = 22)
      PARAMETER (NT = 2)
      COMPLEX(8) A(LNA,LNA),B(LNA),W1(LNW)
      DIMENSION IPVT(LNA)
!
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
         READ (5,*) (A(I,J),J=1,N)
         WRITE (6,1100) (A(I,J),J=1,N)
   10 CONTINUE
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1200)
      DO 20 I = 1, N
         WRITE (6,1300) B(I)
   20 CONTINUE
      WRITE (6,1400)
      CALL HBGNLC (A,LNA,N,IPVT,COND,W1,NT,IERR)
      WRITE (6,1500) 'OBGNLC',IERR
      IF (IERR .GE. 3000) STOP
      COND = 1.0D0/COND
      CALL ZBGNLS (A,LNA,N,B,IPVT,KERR)
      WRITE (6,1500) 'OBGNLS',KERR
      WRITE (6,1600) COND
      WRITE (6,1700)
      DO 30 I = 1, N
         WRITE (6,1800) I,B(I)
   30 CONTINUE
      STOP
!
 1000 FORMAT (1X,/,/,1X ,'***  OBGNLC,OBGNLS  ***',/,&
             /,1X,1X,'**  INPUT  **',/,&
             /,1X,5X,'N =',I3,/,&
             /,1X,5X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (1X,5X,4(' (',F5.1,' ,',F5.1,' )'))
 1200 FORMAT (/,1X,5X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1300 FORMAT (1X,5X,  ' (',F5.1,' ,',F5.1,' )')
 1400 FORMAT (/,1X,1X,'**  OUTPUT  **',/)
 1500 FORMAT (1X,5X,'IERR (',A6,') =',I5)
 1600 FORMAT (/,1X,5X,'CONDITION NUMBER =',D18.10)
 1700 FORMAT (/,1X,5X,'SOLUTION ( REAL, IMAGINARY )')
 1800 FORMAT (1X,5X,'    X(',I2,') = (',D18.10,' ,',D18.10,' )')
      END
```

(d) Output results

```
***  OBGNLC,OBGNLS  ***

 **  INPUT  **

     N =  4

     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  5.0 ,  8.0 ) (  7.0 ,  1.0 ) (  6.0 ,  3.0 ) (  1.0 ,  2.0 )
      (  1.0 ,  1.0 ) (  9.0 ,  5.0 ) (  4.0 ,  1.0 ) (  5.0 ,  0.0 )
      (  0.0 ,  4.0 ) (  3.0 ,  3.0 ) (  4.0 ,  2.0 ) (  6.0 ,  9.0 )
      (  7.0 ,  8.0 ) (  6.0 ,  0.0 ) (  7.0 ,  6.0 ) ( 10.0 ,  4.0 )

     CONSTANT VECTOR ( REAL, IMAGINARY )
      (  3.0 , 20.0 )
      ( -6.0 ,  7.0 )
      (  0.0 , -6.0 )
      (  0.0 , 13.0 )

 **  OUTPUT  **

     IERR (OBGNLC) =    0
     IERR (OBGNLS) =    0

     CONDITION NUMBER =  0.5807863993D+01

     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.1000000000D+01 ,  0.1000000000D+01 )
         X( 2) = ( -0.1665334537D-15 ,  0.1000000000D+01 )
         X( 3) = (  0.1000000000D+01 , -0.2775557562D-15 )
         X( 4) = ( -0.1000000000D+01 , -0.1000000000D+01 )
```

3.4.3 HBGNLU
LU Decomposition of a Complex Matrix

(1) **Function**

HBGNLU uses the Gauss method or the Crout method to perform an LU decomposition of the complex matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL HBGNLU (A, LNA, N, IPVT, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	Z	LNA,N	Input	Complex matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row exchanged with row i in the i-th processing step. (See Note (b))
5	NT	I	1	Input	Number of tasks to be generated
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$ and $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	The contents of array A are unchanged.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular.	

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array A with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U. (See Fig. 3−2 in Section 3.2.2.)

(b) This subroutine performs partial pivoting. Pivoting information is stored in array IPVT for use by subsequent subroutines. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Shared memory parallel function for single-precision is not supported.

3.4.4 HBGNLC
LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

HBGNLC uses the Gauss method or the Crout method to perform an LU decomposition and obtain the condition number of the complex matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL HBGNLC (A, LNA, N, IPVT, COND, W1, NT, IERR)

Single precision:

Nothing

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	Z	LNA,N	Input	Complex matrix (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row exchanged with row i in the i-th processing step. (See Note (b))
5	COND	D	1	Output	Reciprocal of the condition number
6	W1	Z	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$ and $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	The contents of array A are unchanged. COND ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step. A is singular.	

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array A with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array A. Also, reciprocals are stored for the diagonal components of U. (See Fig. 3−2 in Section 3.2.2.)

(b) This subroutine performs partial pivoting. Pivoting information is stored in array IPVT for use by subsequent subroutines. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in IPVT(i). In addition, among the column elements corresponding to row i and row j of matrix A, elements from column 1 to column n actually are exchanged at this time.

(c) Although the condition number is defined by $\|A\| \cdot \|A^{-1}\|$, an approximate value is obtained by this subroutine.

(d) Shared memory parallel function for single-precision is not supported.

3.5 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)

3.5.1 QBSPSL, PBSPSL
Simultaneous Linear Equations (Real Symmetric Matrix)

(1) **Function**

QBSPSL or PBSPSL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real symmetric matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

CALL QBSPSL (A, LNA, N, B, IPVT, WK, NT, IERR)

Single precision:

CALL PBSPSL (A, LNA, N, B, IPVT, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	{D / R}	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	WK	{D / R}	N	Work	Work Area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	$\mathrm{B}(1) \leftarrow \mathrm{B}(1)/\mathrm{A}(1,1)$ is performed.
2100	There existed the diagonal element which was close to zero in the $\mathrm{LDL}^{\mathrm{T}}$ decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LDL^{T} decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.6.4 $\left\{\begin{matrix}\mathrm{DBSPLS}\\ \mathrm{RBSPLS}\end{matrix}\right\}$ you to eliminate unnecessary calculations by performing the $\mathrm{LDL}^{\mathrm{T}}$ decomposition of matrix A only once.

(b) The upper triangular matrix L^T is stored in array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array A. The matrix L is the transpose of matrix L^T, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of matrix L^T as components.
This subroutine uses only the upper triangular portion of array A.

Matrix L^T

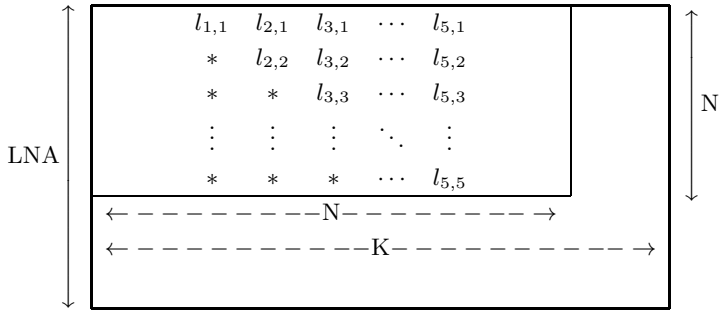
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

Matrix D

$$\begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix}$$

$\Downarrow$

Storage status within array A(LNA, K)



Remarks

a. LNA $\geq$ N and N $\leq$ K must hold.

b. Input time values of elements indicated by asterisks ($*$) are not guaranteed.

Figure 3−5 Storage Status of Matrix L^T and Contents of Matrix D

(c) This subroutine performs partial pivoting when obtaining the $\mathrm{LDL^T}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 4 \\ -4 \end{bmatrix}$$

(b) Input data

Coefficient matrix A, LNA = 11, N = 4 and constant vector $\boldsymbol{b}$.

(c) Main Program

```
      PROGRAM BBSPSL
! *** EXAMPLE OF DBSPSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11)
      DIMENSION A(LNA,LNA),B(LNA),W1(LNA),IPVT(LNA)
!
      NT = 2
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
         READ (5,*) (A(I,J),J=I,N)
         WRITE (6,1100) (A(J,I),J=1,I-1),(A(I,J),J=I,N)
   10 CONTINUE
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1200) (B(I),I=1,N)
      WRITE (6,1300)
```

```
      CALL QBSPSL (A,LNA,N,B,IPVT,W1,NT,IERR)
      WRITE (6,1400) 'QBSPSL',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1500) (I,B(I),I=1,N)
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  QBSPSL  ***',/,&
            2X,'**  INPUT  **',/,&
            6X,'N =',I3,/,&
            6X,'COEFFICIENT MATRIX')
 1100 FORMAT(7X,10(G11.4))
 1200 FORMAT(6X,'COEFFICIENT VECTOR',/,(7X,F10.4))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1500 FORMAT(6X,'SOLUTION',/,(8X,'X(',I2,') =',D18.10))
      END
```

(d) Output results

```
***  QBSPSL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX
        5.000      4.000      1.000      1.000
        4.000      5.000      1.000      1.000
        1.000      1.000      4.000      2.000
        1.000      1.000      2.000      4.000
     COEFFICIENT VECTOR
          1.0000
         -1.0000
          4.0000
         -4.0000
 **  OUTPUT  **
     IERR (QBSPSL) =    0
     SOLUTION
       X( 1) =  0.1000000000D+01
       X( 2) = -0.1000000000D+01
       X( 3) =  0.2000000000D+01
       X( 4) = -0.2000000000D+01
```

3.5.2 QBSPUD, PBSPUD
LDL$^{\mathrm{T}}$ Decomposition of a Real Symmetric Matrix

(1) **Function**

QBSPUD or PBSPUD uses the modified Cholesky method to perform an LDL$^{\mathrm{T}}$ decomposition of the real symmetric matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

CALL QBSPUD (A, LNA, N, IPVT, WK, NT, IERR)

Single precision:

CALL PBSPUD (A, LNA, N, IPVT, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	WK	{D / R}	N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of array A are not changed.
2100	There existed the diagonal element which was close to zero in the LDL$^{\mathrm{T}}$ decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LDL^{T} decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) The upper triangular matrix L^T is stored in array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array A. (See Section 3.5.1, Figure 3−5.)

(b) This subroutine performs partial pivoting when obtaining the LDL$^{\mathrm{T}}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

3.6 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE, LOWER TRIANGULAR TYPE) (NO PIVOTING)

3.6.1 QBSNSL, PBSNSL
Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

(1) **Function**

QBSNSL or PBSNSL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real symmetric matrix A (two-dimensional array type, lower triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

CALL QBSNSL (A, LNA, N, B, NT, IERR)

Single precision:

CALL PBSNSL (A, LNA, N, B, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, lower triangular type)
				Output	lower triangular matrix U^T when A is decomposed into $A = U^T DU$ (See Note (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	{D / R}	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	NT	I	1	Input	Number of tasks to be generated
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

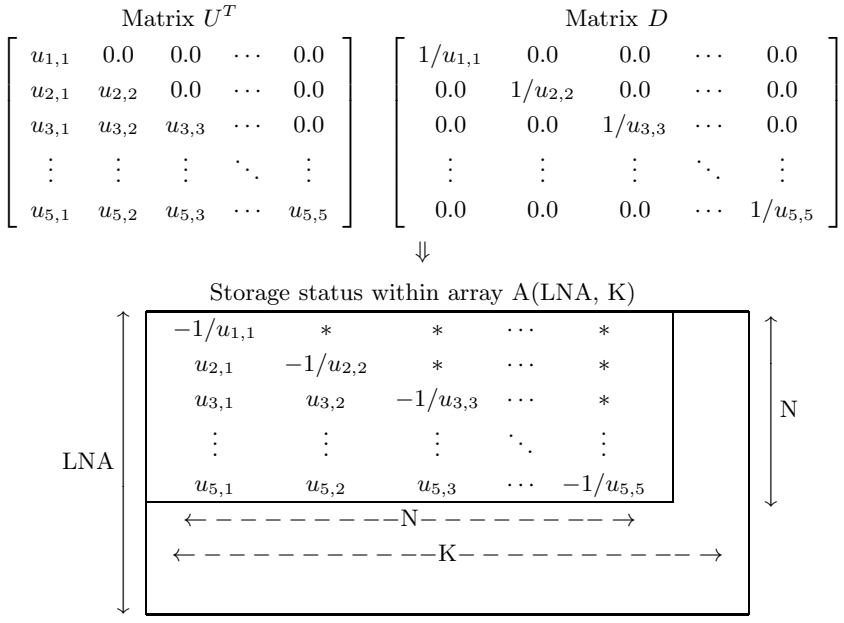
(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	B(1) ← B(1)/A(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the U^TDU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the U^TDU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.8.3 $\left\{\begin{matrix}\text{DBSNLS}\\\text{RBSNLS}\end{matrix}\right\}$ you to eliminate unnecessary calculations by performing the U^TDU decomposition of matrix A only once.

(b) The lower triangular matrix U^T is stored in array A. For the diagonal components of U^T, their reciprocals are stored in array A with the sign changed. Since the diagonal matrix D and the upper triangular matrix U are calculated from U^T, they are not stored in array A. The matrix U is the transpose of matrix U^T, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of matrix U^T as components.
This subroutine uses only the lower triangular portion of array A.



Remarks

a. LNA $\geq$ N and N $\leq$ K must hold.

b. Input time values of elements indicated by asterisks ($*$) are not guaranteed.

Figure 3−6 Storage Status of Matrix U^T and Contents of Matrix D

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 4 \\ -4 \end{bmatrix}$$

(b) Input data

Coefficient matrix A, LNA $= 11$, N $= 4$ and constant vector $\boldsymbol{b}$.

(c) Main Program

```
      PROGRAM OBSNSL
! *** EXAMPLE OF QBSNSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11)
      DIMENSION A(LNA,LNA),B(LNA)
      COMMON A,B
!
      NT=2
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
         READ (5,*) (A(I,J),J=1,I)
   10 CONTINUE
      DO 20 I = 1, N
         WRITE (6,1100) (A(I,J),J=1,I),(A(J,I),J=I+1,N)
   20 CONTINUE
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1200) (B(I),I=1,N)
      WRITE (6,1300)
      CALL QBSNSL (A,LNA,N,B,NT,IERR)
      WRITE (6,1400) 'QBSNSL',IERR
      IF (IERR .GE. 3000) STOP
      WRITE (6,1500) (I,B(I),I=1,N)
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' *** QBSNSL  ***',/,&
```

```
            2X,'**  INPUT  **',/,&
            6X,'N =',I3,/,&
            6X,'COEFFICIENT MATRIX')
 1100 FORMAT(7X,10(G11.4))
 1200 FORMAT(6X,'COEFFICIENT VECTOR',/,(7X,F10.4))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1500 FORMAT(6X,'SOLUTION',/,(8X,'X(',I2,') =',D18.10))
      END
```

(d) Output results

```
***  QBSNSL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX
        5.000      4.000      1.000      1.000
        4.000      5.000      1.000      1.000
        1.000      1.000      4.000      2.000
        1.000      1.000      2.000      4.000
     COEFFICIENT VECTOR
          1.0000
         -1.0000
          4.0000
         -4.0000
 **  OUTPUT  **
     IERR (QBSNSL) =    0
     SOLUTION
       X( 1) =  0.1000000000D+01
       X( 2) = -0.1000000000D+01
       X( 3) =  0.2000000000D+01
       X( 4) = -0.2000000000D+01
```

3.6.2 QBSNUD, PBSNUD
U^TDU Decomposition of a Real Symmetric Matrix (No Pivoting)

(1) **Function**

QBSNUD or PBSNUD uses the modified Cholesky method to perform an U^TDU decomposition of the real symmetric matrix A (two-dimensional array type) (lower triangular type).

(2) **Usage**

Double precision:

CALL QBSNUD (A, LNA, N, NT, IERR)

Single precision:

CALL PBSNUD (A, LNA, N, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type) (lower triangular type)
				Output	Lower triangular matrix U^T when A is decomposed into $A = U^TDU$ (See Note (a))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	NT	I	1	Input	Number of tasks to be generated
5	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of array A are not changed.
2100	There existed the diagonal element which was close to zero in the U^TDU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the U^TDU decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) The lower triangular matrix U^T is stored in array A. For the diagonal components of U^T, their reciprocals are stored in array A with the sign changed. Since the diagonal matrix D and the upper triangular matrix U are calculated from U^T, they are not stored in array A. (See Section 3.6.1, Figure 3−6.)

3.7 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

3.7.1 HBHPSL, GBHPSL
Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

HBHPSL or GBHPSL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a Hermitian matrix (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

```
CALL HBHPSL  (AR, AI, LNA, N, BR, BI, IPVT, W1, NT, IERR)
```

Single precision:

```
CALL GBHPSL  (AR, AI, LNA, N, BR, BI, IPVT, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA,N	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	AI	{D / R}	LNA,N	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	BR	{D / R}	N	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	BI	{D / R}	N	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
8	W1	{D / R}	N	Work	Work area
9	NT	I	1	Input	Number of tasks to be generated
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

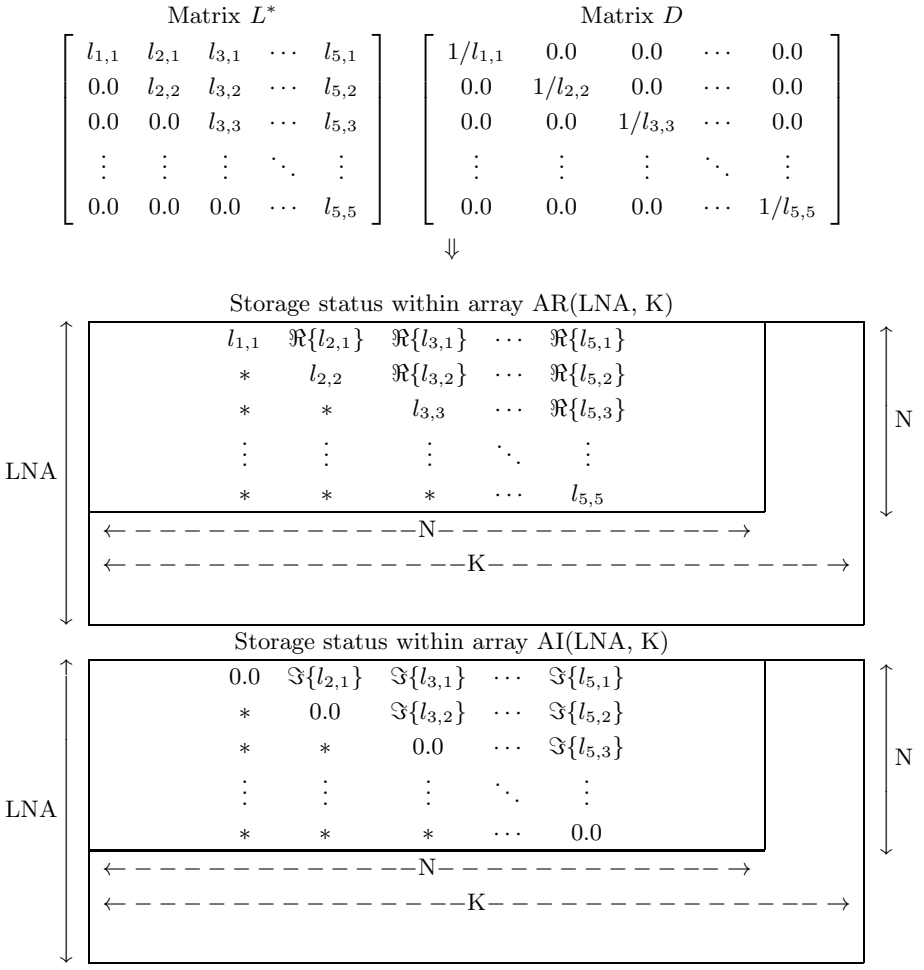
(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1.	Contents of arrays AR and AI are not changed. B(1) ← B(1)/AR(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LDL* decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.9.4 $\left\{\begin{matrix}\text{ZBHPLS}\\ \text{CBHPLS}\end{matrix}\right\}$ the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculation by performing the LDL* decomposition of matrix A only once.

(b) The upper triangular matrix L^* is stored in the upper triangular portions of arrays AR and AI. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays AR and AI. The matrix L is the adjoint matrix of the matrix L^*, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of the matrix L^* as its components. This subroutine uses only the upper triangular portions of arrays AR and AI.



Remarks

a. LNA ≥ N and N ≤ K must hold.

b. Input time values of elements indicated by asterisks (∗) are not guaranteed.

Figure 3−7 Storage Status of Matrix L^* and Contents of Matrix D

(c) This subroutine performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix real part AR and Imaginary part AI, LNA = 11, N = 4 and constant vector B.

(c) Main Program

```
      PROGRAM UBHPSL
!     *** EXAMPLE OF HBHPSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (LNA = 11,LNW = 22,NT = 2)
      DIMENSION AR(LNA,LNA),AI(LNA,LNA),BR(LNA),BI(LNA),W1(LNW)
      DIMENSION IPVT(LNA)
!
      CHARACTER*50      FMT(4)
!
      DATA FMT /'(6X,       4(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
                '(6X,  16X, 3(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
                '(6X,2(16X),2(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
                '(6X,3(16X),  1X,A1,F5.1,1X,A1,F5.1,1X,A1 )'/
!
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
        READ (5,*) (AR(I,J),AI(I,J),J=I,N)
        WRITE (6,FMT(I)) ('(',AR(I,J),',',AI(I,J),')',J=I,N)
   10 CONTINUE
      READ (5,*) (BR(I),BI(I),I=1,N)
      WRITE (6,1100)
      DO 20 I = 1, N
      WRITE (6,1200) BR(I),BI(I)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBHPSL (AR,AI,LNA,N,BR,BI,IPVT,W1,NT,IERR)
      WRITE (6,1400) 'HBHPSL',IERR
      WRITE (6,1600)
      DO 30 I = 1, N
      WRITE (6,1700) I,BR(I),BI(I)
   30 CONTINUE
      STOP
!
 1000 FORMAT (' ',/,/,' ***  HBHPSL  ***',/,2X,'**  INPUT  **',&
             /,6X,'N =',I3,&
             /,6X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (6X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1200 FORMAT (6X,' (',F5.1,' ,',F5.1,' )')
 1300 FORMAT (2X,'**  OUTPUT  **')
 1400 FORMAT (6X,'IERR (',A6,') =',I5)
 1600 FORMAT (6X,'SOLUTION ( REAL, IMAGINARY )')
 1700 FORMAT (10X,'X(',I2,') = (',D18.10,' ,',D18.10,' )')
      END
```

(d) Output results

```
***  HBHPSL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  9.0 ,  0.0 ) (  7.0 ,  3.0 ) (  2.0 ,  5.0 ) (  1.0 ,  1.0 )
                      ( 10.0 ,  0.0 ) (  3.0 ,  2.0 ) (  2.0 ,  4.0 )
                                      (  8.0 ,  0.0 ) (  5.0 ,  1.0 )
                                                      (  6.0 ,  0.0 )
     CONSTANT VECTOR ( REAL, IMAGINARY )
      ( 10.0 ,  6.0 )
      ( 11.0 ,  2.0 )
      (  4.0 ,  6.0 )
      (  4.0 ,  6.0 )
 **  OUTPUT  **
     IERR (HBHPSL) =    0
     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.10000000000D+01 ,  0.00000000000D+00 )
         X( 2) = (  0.10000000000D+01 ,  0.00000000000D+00 )
         X( 3) = ( -0.6628197162D-16 ,  0.10000000000D+01 )
         X( 4) = (  0.2085418925D-16 ,  0.10000000000D+01 )
```

3.7.2 HBHPUD, GBHPUD
LDL* Decomposition of a Hermitian Matrix

(1) **Function**

HBHPUD or GBHPUD uses the modified Cholesky method to perform an LDL* decomposition of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

```
CALL HBHPUD  (AR, AI, LNA, N, IPVT, W1, NT, IERR)
```

Single precision:

```
CALL GBHPUD  (AR, AI, LNA, N, IPVT, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA,N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	AI	{D / R}	LNA,N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	LNA	I	1	Input	Adjustable dimension of array AR and AI
4	N	I	1	Input	Order of matrix A
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
6	W1	{D / R}	$2 \times$ N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 <$ N $\leq$ LNA

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of arrays AR and AI are not changed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LDL* decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays AR and AI. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays AR and AI. This subroutine uses only the upper triangular portions of arrays AR and AI. (See Sections 3.7.1 Figure 3−7.)

(b) This subroutine performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

3.8 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) (NO PIVOTING)

3.8.1 HBHRSL, GBHRSL
Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

HBHRSL or GBHRSL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a Hermitian matrix (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

```
CALL HBHRSL  (AR, AI, LNA, N, BR, BI, W1, NT, IERR)
```

Single precision:

```
CALL GBHRSL  (AR, AI, LNA, N, BR, BI, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA,N	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	AI	{D / R}	LNA,N	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	BR	{D / R}	N	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	BI	{D / R}	N	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	W1	{D / R}	N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

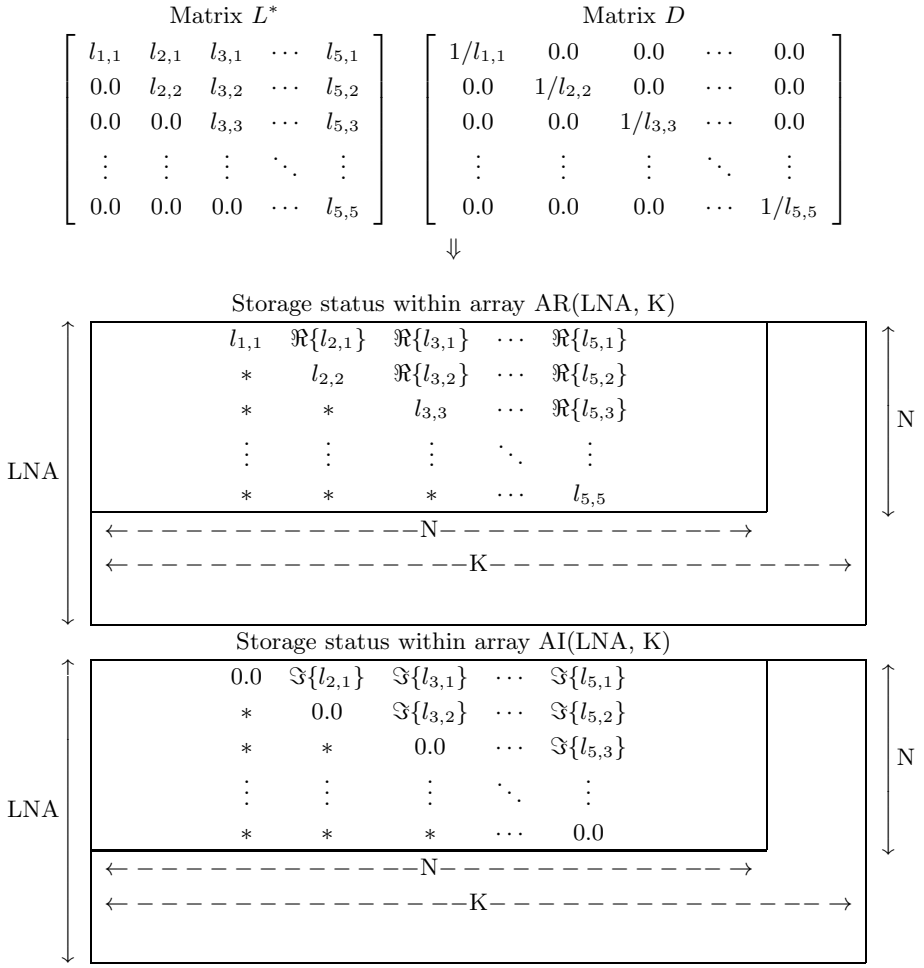
(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1.	Contents of arrays AR and AI are not changed. B(1) ← B(1)/AR(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step of the LDL* decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.10.4 $\left\{\begin{matrix}\text{ZBHRLS}\\ \text{CBHRLS}\end{matrix}\right\}$ the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculation by performing the LDL* decomposition of matrix A only once.

(b) The upper triangular matrix L^* is stored in the upper triangular portions of arrays AR and AI. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays AR and AI. The matrix L is the adjoint matrix of the matrix L^*, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of the matrix L^* as its components. This subroutine uses only the upper triangular portions of arrays AR and AI. (See Fig. 3−8.)



Remarks

a. LNA $\geq$ N and N $\leq$ K must hold.

b. Input time values of elements indicated by asterisks ($*$) are not guaranteed.

Figure 3−8 Storage Status of Matrix L^* and Contents of Matrix D

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix real part AR and Imaginary part AI, LNA = 11, N = 4 and constant vector B.

(c) Main Program

```
      PROGRAM UBHRSL
!     *** EXAMPLE OF HBHRSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NT
      PARAMETER (LNA = 11,LNW = 22,NT = 2)
      DIMENSION AR(LNA,LNA),AI(LNA,LNA),BR(LNA),BI(LNA),W1(LNW)
!
      CHARACTER*50      FMT(4)
```

```
!
      DATA FMT /'(6X,       4(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
               '(6X,  16X, 3(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
               '(6X,2(16X),2(1X,A1,F5.1,1X,A1,F5.1,1X,A1))',&
               '(6X,3(16X),  1X,A1,F5.1,1X,A1,F5.1,1X,A1 )'/
!
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
        READ (5,*) (AR(I,J),AI(I,J),J=I,N)
        WRITE (6,FMT(I)) ('(',AR(I,J),',',AI(I,J),')',J=I,N)
   10 CONTINUE
      READ (5,*) (BR(I),BI(I),I=1,N)
      WRITE (6,1100)
      DO 20 I = 1, N
      WRITE (6,1200) BR(I),BI(I)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBHRSL (AR,AI,LNA,N,BR,BI,W1,NT,IERR)
      WRITE (6,1400) 'HBHRSL',IERR
      WRITE (6,1600)
      DO 30 I = 1, N
      WRITE (6,1700) I,BR(I),BI(I)
   30 CONTINUE
      STOP
!
 1000 FORMAT (' ',/,/,' ***  HBHRSL  ***',/,2X,'**  INPUT  **',&
             /,6X,'N =',I3,&
             /,6X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (6X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1200 FORMAT (6X,' (',F5.1,' ,',F5.1,' )')
 1300 FORMAT (2X,'**  OUTPUT  **')
 1400 FORMAT (6X,'IERR (',A6,') =',I5)
 1600 FORMAT (6X,'SOLUTION ( REAL, IMAGINARY )')
 1700 FORMAT (10X,'X(',I2,') = (',D18.10,' ,',D18.10,' )')
      END
```

(d) Output results

```
***  HBHRSL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  9.0 ,  0.0 ) (  7.0 ,  3.0 ) (  2.0 ,  5.0 ) (  1.0 ,  1.0 )
                      ( 10.0 ,  0.0 ) (  3.0 ,  2.0 ) (  2.0 ,  4.0 )
                                      (  8.0 ,  0.0 ) (  5.0 ,  1.0 )
                                                      (  6.0 ,  0.0 )
     CONSTANT VECTOR ( REAL, IMAGINARY )
      ( 10.0 ,  6.0 )
      ( 11.0 ,  2.0 )
      (  4.0 ,  6.0 )
      (  4.0 ,  6.0 )
 **  OUTPUT  **
     IERR (HBHRSL) =    0
     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.1000000000D+01 ,  0.1480297366D-15 )
         X( 2) = (  0.1000000000D+01 , -0.3903127821D-16 )
         X( 3) = ( -0.1022363649D-15 ,  0.1000000000D+01 )
         X( 4) = (  0.8341675699D-16 ,  0.1000000000D+01 )
```

3.8.2 HBHRUD, GBHRUD
LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

HBHRUD or GBHRUD uses the modified Cholesky method to perform an LDL* decomposition of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

```
CALL HBHRUD  (AR, AI, LNA, N, W1, NT, IERR)
```

Single precision:

```
CALL GBHRUD  (AR, AI, LNA, N, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA,N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	AI	{D / R}	LNA,N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	LNA	I	1	Input	Adjustable dimension of array AR and AI
4	N	I	1	Input	Order of matrix A
5	W1	{D / R}	N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of arrays AR and AI are not changed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays AR and AI. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays AR and AI. This subroutine uses only the upper triangular portions of arrays AR and AI. (See Fig. 3−8 in Section 3.8.1.)

3.9 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE)

3.9.1 HBHFSL, GBHFSL
Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

HBHFSL or GBHFSL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

```
CALL HBHFSL  (A, LNA, N, B, IPVT, W1, NT, IERR)
```

Single precision:

```
CALL GBHFSL  (A, LNA, N, B, IPVT, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	{Z / C}	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	W1	{D / R}	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	$\mathrm{N} = 1$.	Contents of array A are not changed. $\mathrm{B}(1) \leftarrow \mathrm{B}(1)/\mathrm{A}(1,1)$ is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step of the LDL* decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.11.4 $\left\{\begin{matrix}\mathrm{ZBHFLS}\\ \mathrm{CBHFLS}\end{matrix}\right\}$ the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculations by performing the LDL* decomposition of matrix A only once.

(b) The upper triangular matrix L^* is stored in the upper triangular portion of array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array A. The matrix L is the adjoint matrix of the matrix L^*, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of the matrix L^* as its components.

Matrix L^*

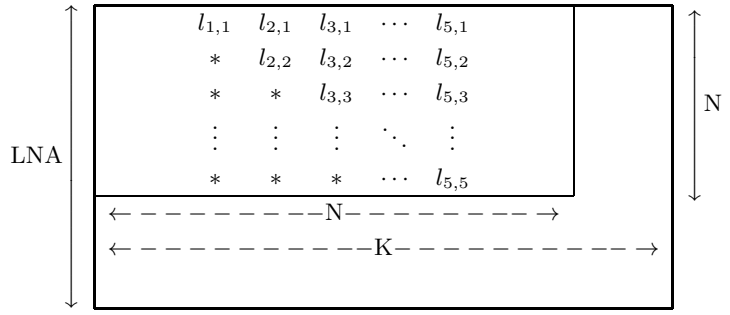
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

Matrix D

$$\begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix}$$

$\Downarrow$

Storage status within array A(LNA, K)



Remarks

a. LNA ≥ N and N ≤ K must hold.

b. Input time values of elements indicated by asterisks (∗) are not guaranteed.

Figure 3−9 Storage Status of Matrix L^* and Contents of Matrix D

(c) This subroutine performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix A, LNA = 11, N = 4 and constant vector $\boldsymbol{b}$.

(c) Main Program

```
      PROGRAM UBHFSL
!     *** EXAMPLE OF HBHFSL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NT
      PARAMETER (LNA = 11,LNW = 22,NT = 2)
      COMPLEX(8) A(LNA,LNA),B(LNA),W1(LNW)
      INTEGER IPVT(LNA)
!
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
        READ (5,*) (A(I,J),J=I,N)
   10 CONTINUE
        WRITE (6,2000) (A(1,J),J=1,N)
        WRITE (6,2100) (A(2,J),J=2,N)
        WRITE (6,2200) (A(3,J),J=3,N)
```

```
        WRITE (6,2300) (A(4,J),J=4,N)
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1100)
      DO 20 I = 1, N
      WRITE (6,1200) B(I)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBHFSL (A,LNA,N,B,IPVT,W1,NT,IERR)
      WRITE (6,1400) 'HBHFSL',IERR
      WRITE (6,1600)
      DO 30 I = 1, N
      WRITE (6,1700) I,B(I)
   30 CONTINUE
      STOP
!
 1000 FORMAT (' ',/,/,' ***  HBHFSL  ***',/,2X,'**  INPUT  **',&
             /,6X,'N =',I3,&
             /,6X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (6X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1200 FORMAT (6X,' (',F5.1,' ,',F5.1,' )')
 1300 FORMAT (2X,'**  OUTPUT  **')
 1400 FORMAT (6X,'IERR (',A6,') =',I5)
 1600 FORMAT (6X,'SOLUTION ( REAL, IMAGINARY )')
 1700 FORMAT (10X,'X(',I2,') = (',D18.10,' ,',D18.10,' )')
 2000 FORMAT (6X,       4(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2100 FORMAT (6X,  16X, 3(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2200 FORMAT (6X,2(16X),2(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2300 FORMAT (6X,3(16X),  1X,'(',F5.1,' ,',F5.1,1X,')' )
      END
```

(d) Output results

```
***  HBHFSL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  9.0 ,  0.0 ) (  7.0 ,  3.0 ) (  2.0 ,  5.0 ) (  1.0 ,  1.0 )
                      ( 10.0 ,  0.0 ) (  3.0 ,  2.0 ) (  2.0 ,  4.0 )
                                      (  8.0 ,  0.0 ) (  5.0 ,  1.0 )
                                                      (  6.0 ,  0.0 )
     CONSTANT VECTOR ( REAL, IMAGINARY )
      ( 10.0 ,  6.0 )
      ( 11.0 ,  2.0 )
      (  4.0 ,  6.0 )
      (  4.0 ,  6.0 )
 **  OUTPUT  **
     IERR (HBHFSL) =    0
     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.1000000000D+01 ,  0.0000000000D+00 )
         X( 2) = (  0.1000000000D+01 ,  0.0000000000D+00 )
         X( 3) = ( -0.6628197162D-16 ,  0.1000000000D+01 )
         X( 4) = (  0.2085418925D-16 ,  0.1000000000D+01 )
```

3.9.2 HBHFUD, GBHFUD
LDL* Decomposition of a Hermitian Matrix

(1) **Function**

HBHFUD or GBHFUD uses the modified Cholesky method to perform an LDL* decomposition of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

CALL HBHFUD (A, LNA, N, IPVT, W1, NT, IERR)

Single precision:

CALL GBHFUD (A, LNA, N, IPVT, W1, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	IPVT	I	N	Output	Pivoting information IPVT(i): Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	W1	{D / R}	N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of array A are not changed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array A. This subroutine uses only the upper triangular portion of array A. (See Fig. 3−9 in Section 3.9.1)

(b) This subroutine performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in IPVT(i). In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) N actually are exchanged at this time.

3.10 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) (NO PIVOTING)

3.10.1 HBHESL, GBHESL
Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

HBHESL or GBHESL uses the modified Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:

CALL HBHESL (A, LNA, N, B, NT, IERR)

Single precision:

CALL GBHESL (A, LNA, N, B, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ Z / C }	LNA , N	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	B	{ Z / C }	N	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	NT	I	1	Input	Number of tasks to be generated
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N = 1.	Contents of array A are not changed. B(1) ← B(1)/A(1, 1) is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step of the LDL* decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this subroutine only once and then call subroutine <Basic Functions Vol. 2> 2.12.4 $\left\{\begin{matrix}\text{ZBHELS}\\\text{CBHELS}\end{matrix}\right\}$ the required number of times varying only the contents of B. This enables you to eliminate unnecessary calculations by performing the LDL* decomposition of matrix A only once.

(b) The upper triangular matrix L^* is stored in the upper triangular portion of array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array A. The matrix L is the adjoint matrix of the matrix L^*, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of the matrix L^* as its components.

Matrix L^*

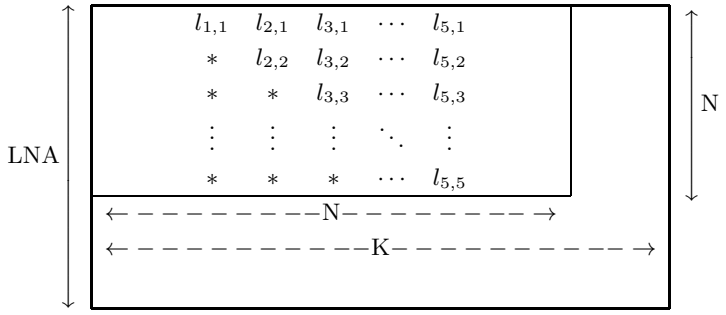
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

Matrix D

$$\begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix}$$

$\Downarrow$

Storage status within array A(LNA, K)



Remarks

a. LNA $\geq$ N and N $\leq$ K must hold.

b. Input time values of elements indicated by asterisks ($*$) are not guaranteed.

Figure 3−10 Storage Status of Matrix L^* and Contents of Matrix D

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix A, LNA $= 11$, N $= 4$ and constant vector $\boldsymbol{b}$.

(c) Main Program

```
      PROGRAM UBHESL
!     *** EXAMPLE OF HBHESL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NT
      PARAMETER (LNA = 11,LNW = 22,NT = 2)
      COMPLEX(8) A(LNA,LNA),B(LNA)
!
      READ (5,*) N
      WRITE (6,1000) N
      DO 10 I = 1, N
        READ (5,*) (A(I,J),J=I,N)
   10 CONTINUE
        WRITE (6,2000) (A(1,J),J=1,N)
        WRITE (6,2100) (A(2,J),J=2,N)
        WRITE (6,2200) (A(3,J),J=3,N)
        WRITE (6,2300) (A(4,J),J=4,N)
      READ (5,*) (B(I),I=1,N)
      WRITE (6,1100)
      DO 20 I = 1, N
      WRITE (6,1200) B(I)
   20 CONTINUE
      WRITE (6,1300)
      CALL HBHESL (A,LNA,N,B,NT,IERR)
      WRITE (6,1400) 'HBHESL',IERR
      WRITE (6,1600)
      DO 30 I = 1, N
      WRITE (6,1700) I,B(I)
```

```
   30 CONTINUE
      STOP
!
 1000 FORMAT (' ',/,/,' ***  HBHESL  ***',/,2X,'**  INPUT  **',&
             /,6X,'N =',I3,&
             /,6X,'COEFFICIENT MATRIX ( REAL, IMAGINARY )')
 1100 FORMAT (6X,'CONSTANT VECTOR ( REAL, IMAGINARY )')
 1200 FORMAT (6X,' (',F5.1,' ,',F5.1,' )')
 1300 FORMAT (2X,'**  OUTPUT  **')
 1400 FORMAT (6X,'IERR (',A6,') =',I5)
 1600 FORMAT (6X,'SOLUTION ( REAL, IMAGINARY )')
 1700 FORMAT (10X,'X(',I2,') = (',D18.10,' ,',D18.10,' )')
 2000 FORMAT (6X,       4(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2100 FORMAT (6X,  16X, 3(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2200 FORMAT (6X,2(16X),2(1X,'(',F5.1,' ,',F5.1,1X,')'))
 2300 FORMAT (6X,3(16X),  1X,'(',F5.1,' ,',F5.1,1X,')' )
      END
```

(d) Output results

```
***  HBHESL  ***
 **  INPUT  **
     N =  4
     COEFFICIENT MATRIX ( REAL, IMAGINARY )
      (  9.0 ,  0.0 ) (  7.0 ,  3.0 ) (  2.0 ,  5.0 ) (  1.0 ,  1.0 )
                      ( 10.0 ,  0.0 ) (  3.0 ,  2.0 ) (  2.0 ,  4.0 )
                                      (  8.0 ,  0.0 ) (  5.0 ,  1.0 )
                                                      (  6.0 ,  0.0 )
     CONSTANT VECTOR ( REAL, IMAGINARY )
      ( 10.0 ,  6.0 )
      ( 11.0 ,  2.0 )
      (  4.0 ,  6.0 )
      (  4.0 ,  6.0 )
 **  OUTPUT  **
     IERR (HBHESL) =    0
     SOLUTION ( REAL, IMAGINARY )
         X( 1) = (  0.1000000000D+01 , -0.9868649108D-16 )
         X( 2) = (  0.1000000000D+01 , -0.6245004514D-16 )
         X( 3) = (  0.5111818243D-16 ,  0.1000000000D+01 )
         X( 4) = ( -0.0000000000D+00 ,  0.1000000000D+01 )
```

3.10.2 HBHEUD, GBHEUD
LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

HBHEUD or GBHEUD uses the modified Cholesky method to perform an LDL* decomposition of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

```
CALL HBHEUD  (A, LNA, N, NT, IERR)
```

Single precision:

```
CALL GBHEUD  (A, LNA, N, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ Z / C }	LNA , N	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	NT	I	1	Input	Number of tasks to be generated
5	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \mathrm{N} \leq \mathrm{LNA}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Contents of array A are not changed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array A. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array A. This subroutine uses only the upper triangular portion of array A. (See Fig. 3−10 in Section 3.10.1)

Chapter 4

SIMULTANEOUS LINEAR EQUATIONS (ITERATIVE METHOD)

4.1 INTRODUCTION

This chapter describes subroutines that use iterative methods to solve simultaneous linear equations having a large dimensional sparse matrix as coefficient matrix.

Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

This library having the functions listed below use either the nonstationary iterative methods as the basic iterative algorithms. In the nonstationary iterative methods are included the methods such as the CG iterative method or the GMRES method.

For the preconditioner, the Scaling method are supported.

Among these functions, one matrix storage format for and two matrix storage formats for random sparse matrices (ELLPACK format) are available.

These functions use the same user interface regardless of the iterative basic method. The users can only have to change the subroutines name which is called in the user's program to try different iterative methods under the the same preconditioning and the same storage format.

Table 4−1 Subroutine for each basic iterative method

BASIC ITERATIVE METHOD	SUBROUTINE NAME
CG (For Symmetric Matrices only) Conjugate Gradient method	QXE010 PXE010
CGS Conjugate Gradient Squared method	QXE020 PXE020
BiCGSTAB Bi-Conjugate Gradient Stabilized method	QXE030 PXE030
GMRES General Minimal Residual method	QXE040 PXE040

4.1.1 Notes

(1) Proper use of iterative methods

The CG method subroutines is primarily used to solve positive definite symmetric large-scale sparse simultaneous linear equations that occur in finite difference approximations or finite element approximations of diffusion equations. The CGS, BiCGSTAB method subroutines are primarily used to solve asymmetric large-scale sparse simultaneous linear equations that occur in finite difference approximations or finite element approximations of advection diffusion equations. Since the memory area required when using an iterative method is smaller than that required when using a direct method, iterative methods are used for solving large-scale problems. However, an iterative method may not converge to the true solution. Therefore, before using these iterative method subroutines, you should test whether you can reduce the maximum number of iterations or the problem size.

In particular, it is generally difficult to solve large-scale asymmetric simultaneous linear equations, and there is no guarantee that the true solution will always be obtained by applying the iterative method algorithms used in this library.

(2) Matrix storage format

The nonzero elements of the coefficient matrix of simultaneous linear equations generated by using the finite difference method to discretize a two-dimensional rectangular area or three-dimensional rectangular parallelepiped area line along straight lines in the direction of the diagonal. This type of matrix is called a regular sparse matrix.

However, the coefficient matrix of simultaneous linear equations generated by using the finite element method to discretize a randomly shaped area generally has no regularity other than the fact that the diagonal elements are nonzero elements. This type of matrix is called an random (irregular) sparse matrix. The subroutines described in this chapter support a storage format for random (irregular) sparse matrices. Each subroutine has a common user interface, with a single exception that the GMRES method subroutines have an additional input argument (GMITR). Therefore, the users can only have to replace the subroutine name in their programs if they want to try different iterative methods with the input values unchanged.(As for trying GMRES method, an input argument (GMITR) should also be added to the user interface.) However, attention must be paid to the fact that the CG method subroutines cannot be applied to problems for which the coefficient matrix is asymmetric, since the CG method is for solving positive definite symmetric simultaneous linear equations.

(3) Preconditioning methods

Preconditioning is a technique for accelerating convergence of a basic iterative method. The simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ are converted to equivalent equations that are easier to solve by using an easily invertible matrix M, which is an approximation matrix for matrix A. The procedure for the basic iterative method is then applied to these equations. In the subroutines described here, the scaling preconditioning is selected.

Scaling preconditioning is the most effective of these preconditioning methods for ordinary problems because the large vectors involved in the iteration calculations of scaling preconditioning enable the multiple parallel pipelines of the Vector Engine to operate efficiently, and also the required memory area is small.

(4) Conditions for terminating iterations

The iteration calculations are interrupted when any of the following conditions occurs:

(a) The residual norm becomes less than or equal to the truncation residual norm.

(b) The iteration count reaches the maximum number of iterations.

(c) An error in the algorithm is detected that makes further calculations impossible.

The truncation residual norm is briefly described below.
If $\boldsymbol{u}^*$ is the approximate solution obtained by an iteration calculation for the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$, then the residual vector at that time is defined by:

$$\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}^*.$$

Here, the term "residual norm" is used to mean the (ordinary) relative norm

$$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}.$$

In addition, the user can determine the meaning of the norm function $\|*\|$ from among L^2 norm, which are defined as follows:

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{n} {r_i}^2\right)^{1/2}.$$

(5) Required memory areas
The subroutines (basic iterative subroutines plus subprograms) described in this chapter require an area for storing the coefficient matrix of the equations, areas for storing the solution vector or right-hand side vector, and work area.

The sizes of these areas differ according to the basic iterative method, matrix storage format, and preconditioning method that are selected. For information about the sizes of these areas, see the notes in the explanations of the basic iterative subroutines and subprograms.

4.1.2 Algorithms Used

4.1.2.1 Nonstationary iterative method (for Symmetric Matrix only)

(1) **the CG method**

Consider the following simultaneous linear equations of order N:

$$A\boldsymbol{u} = \boldsymbol{b} \qquad (1)$$

The CG method is an iterative method for solving the simultaneous linear equations shown in (1) for which the coefficient matrix A is a positive definite symmetric matrix. This method is represented as follows.

$\boldsymbol{u} = \boldsymbol{u}_1$ ($\boldsymbol{u}_1$ is the initial solution),
$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1$,
$\boldsymbol{p}_1 = \boldsymbol{r}_1$.
for $i = 1, 2, \cdots, n$
$a_i = \dfrac{(\boldsymbol{r}_i, \boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)}$,
$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i$,
$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_i A\boldsymbol{p}_i$,
$b_i = \dfrac{(\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, \boldsymbol{r}_i)}$,
$\boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i$.

A great advantage of the CG method is that, theoretically, a strict solution is obtained in N iterative calculations (where n is the order).

4.1.2.2 Nonstationary iterative method (for Asymmetric Matrix)

(1) **the CGS and BiCGSTAB methods**

To explain the CGS and BiCGSTAB methods, we will first explain the BiCG method. The BiCG method considers the simultaneous linear equations of order $2n$ formed by combining the equations shown in (1) with the dual equations as follows:

$$\tilde{A}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{b}},$$
$$\tilde{A} = \begin{bmatrix} A & O \\ O & A^T \end{bmatrix}, \tilde{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{u}^* \end{bmatrix}, \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}^* \end{bmatrix}$$

and uses the conjugate gradient method to obtain the stationary value of:

$$F(\tilde{\boldsymbol{u}}) = < (\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}), \quad A(\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}) > H \quad (\hat{\boldsymbol{u}}; \text{Truesolution})$$

where,

$$< \boldsymbol{u}, \boldsymbol{v} > H \equiv (\boldsymbol{u}, H\boldsymbol{v}) = (H\boldsymbol{u}, \boldsymbol{v}), \quad H = \begin{bmatrix} O & I \\ I & O \end{bmatrix}.$$

Using the initial residual vector $\boldsymbol{r}_0$ the residual vector after k iterations $\boldsymbol{r}_k$ and direction vector after k iterations $\boldsymbol{p}_k$ are represented by:

$$\begin{aligned} \boldsymbol{r}_k &= R_k(A)\boldsymbol{r}_0, \\ \boldsymbol{p}_k &= P_k(A)\boldsymbol{r}_0. \end{aligned}$$

Here, $R_k(A)$, $P_k(A)$ are polynomials generated from A. At this time, the CGS method uses this $R_k(A)$ and $P_k(A)$ to calculate new vectors $\boldsymbol{r}'_k$ and $\boldsymbol{p}'_k$ as follows.

$$\begin{aligned} \boldsymbol{r}'_k &= R_k^2(A)\boldsymbol{r}_0, \\ \boldsymbol{p}'_k &= P_k^2(A)\boldsymbol{r}_0 \end{aligned}$$

Using this calculation method, as

$$\|\boldsymbol{r}_k\| = \|R_k(A)\boldsymbol{r}_0\|$$

gets smaller,

$$\|\boldsymbol{r}'_k\| = \|R_k^2(A)\boldsymbol{r}_0\|$$

is expected to get even smaller, and convergence is expected to be faster than with the BCG method.

The method in which the iterations proceed so that the residual becomes $\boldsymbol{r}'_k$ is the CGS method.

However, the CGS method may exhibit extremely irregular convergence characteristics. In particular, the residual may be extremely large at the beginning of the iterations such as when the iteration initial value is close to the solution, and accurate iteration calculations may not be able to be performed due to the effect of rounding error. The BiCGSTAB method lets $Q_k(A)$ represent the following:

$$Q_k(A) = (1 - \omega_1 A)(1 - \omega_2 A) \cdots (1 - \omega_k A)$$

and obtains the approximate solution $\boldsymbol{u}_k$ so that the residual

$$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{u}_k$$

in each iteration is given by:

$$\boldsymbol{r}_k = Q_k(A)R_k(A)\boldsymbol{r}_0$$

Here, the parameters $\omega_k$that minimize $(\boldsymbol{r}_k, \boldsymbol{r}_k)$ are selected so that the residuals are reduced stably.

(2) **the GMRES method (GMRES(m))**

The GMRES method is an iterative method that for each $j = 1, 2, \cdots$ searches for the vector $\boldsymbol{z}^{(j)}$ that minimizes the L^2norm of

$$\boldsymbol{b} - A(\boldsymbol{u}_0^{(j)} + \boldsymbol{z}^{(j)}) = \boldsymbol{r}^{(j)} - A\boldsymbol{z}^{(j)}$$

from the space spanned by

$$\boldsymbol{r}_0, \quad A\boldsymbol{r}_0, \quad A^2\boldsymbol{r}_0, \quad \cdots, \quad A^i\boldsymbol{r}_0 \quad (i = 1, 2, \cdots, m)$$

when the initial approximate solution vector of the j-th step is $\boldsymbol{u}_0^{(j)}$ and the initial residual vector of the j-th step is

$$\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)}$$

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0$.
for $j = 1, 2, \cdots$
  $\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)}$,
  $\boldsymbol{v}_1 = \boldsymbol{r}^{(j)} / \|\boldsymbol{r}^{(j)}\|_2$.
  for $i = 1, 2, \cdots m$
    $\boldsymbol{w} = A\boldsymbol{v}_i$.
    for $k = 1, 2, \cdots i$
      $\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k$,
    $\boldsymbol{v}_{i+1} = \boldsymbol{w} / \|\boldsymbol{w}\|_2$,
    Determine the parameters $\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$
    so that $\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ takes the least value.
  $\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}$.

As shown in this algorithm, the GMRES method consists of an external iterative calculation (loop related to j) and an internal iterative calculation (loop related to i). Since the GMRES method is an iterative method that depends on the parameter m representing the upper bound on the number of inner iterations, it is referred to, strictly speaking, as the GMRES(m) method.
In this library, the number of iterations in the GMRES(m) algorithm is defined by
(number of internal iterations i) +(number of external iterations j)× m .

4.1.2.3 Preconditioned Iterative Method

Since error is introduced in actual iterative method calculations, convergence characteristics become worse. In particular, if the difference mesh is fine in the difference scheme for partial differential equations, if the coefficient (dispersion or thermal conductivity) values severely fluctuate spatially or if the difference mesh widths or the coefficients described above are anisotropic, then the eigenvalues of matrix A are dispersed, and the iterative method has difficulty in converging under these circumstances. Conversely, convergence will be fast if the eigenvalues are clustered together.
To overcome the difficulties described above, preconditioned iterative methods were developed. A preconditioned iterative method is a technique for improving convergence by preconditioning the original matrix to convert it to a well-conditioned matrix before performing the iterative method calculations. This is represented as follows.
Assume that the matrix $M = M_1M_2$, which approximates $A = (a_{ij})$ in some way, is given.

$$A \sim M = M_1M_2$$

At this time, if the following equation, which is equivalent to equation (1), is created:

$$M_1^{-1}AM_2^{-1}\boldsymbol{u}' = \boldsymbol{b}' \qquad (2)$$
$$(\quad \boldsymbol{u}' = M_2\boldsymbol{u} \quad)$$
$$(\quad \boldsymbol{b}' = M_1^{-1}\boldsymbol{b} \quad)$$

the coefficient matrix of this equation is easier to solve since it is closer to the unit matrix than the original matrix was. Equation (2) is called the preconditioning equation, and matrix M is called the preconditioning matrix. The preconditioned iterative method for equation (1) is defined/derived as the (basic) iterative method for equation (1). The preconditioned iterative method must execute a matrix inversion on M_1 and M_2 or M (left operation on a column vector according to the inverse matrix). This is what is called the preconditioning operation (or simply preconditioning). Usually, a matrix for which the preconditioning operation can easily be performed is used as preconditioning matrix $M = M_1M_2$. The more closely the preconditioning matrix M approximates the original

matrix A, the more the convergence of the preconditioned iterative method will improve. However, there generally is a tendency for the amount of time required for a single preconditioning operation to increase in proportion to the quality of convergence of the preconditioning matrix.

See 4.1.2.4 for the detailed explanation of the preconditioning algorithms that ASL supports.

The algorithms for the preconditioned CG method (PCG method), preconditioned CGS method (PCGS method), preconditioned BiCGSTAB (PBiCGSTAB method), and preconditioned GMRES(m) method (PGMRES(m) method) are shown below.

(1) **the PCG method**

$\boldsymbol{u} = \boldsymbol{u}_1$ ($\boldsymbol{u}_1$ is the initial solution),
$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1$,
$\boldsymbol{p}_1 = M^{-1}\boldsymbol{r}_1$.
for $i = 1, 2, \cdots, n$
$a_i = \dfrac{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)}$,
$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i$,
$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_iA\boldsymbol{p}_i$,
$b_i = \dfrac{(\boldsymbol{r}_{i+1}, M^{-1}\boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)}$,
$\boldsymbol{p}_i = M^{-1}\boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i$.

(2) **the PCGS method** (Bibliography(1))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0$,
$\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$,
$\boldsymbol{e}_0 = \boldsymbol{r}_0$.
for $k = 0, 1, 2, \cdots$
$\alpha_{k+1} = \dfrac{(\boldsymbol{r}_0, \boldsymbol{r}_k)}{(\boldsymbol{r}_0, A\boldsymbol{p}_k)}$,
$\boldsymbol{h}_{k+1} = \boldsymbol{e}_k - \alpha_{k+1}A\boldsymbol{p}_k$,
$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha_{k+1}M^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1})$,
$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_{k+1}AM^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1})$,
$\beta_{k+1} = \dfrac{(\boldsymbol{r}_0, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_0, \boldsymbol{r}_k)}$,
$\boldsymbol{e}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1}$,
$\boldsymbol{p}_{k+1} = M^{-1}(\boldsymbol{e}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1}) + \beta_{k+1}^2\boldsymbol{p}_k$.

(3) **the PBiCGSTAB method** (Bibliography(2))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0$,
for $k = 0, 1, 2, \cdots$
  $\rho_{k+1} = (\boldsymbol{r}_0, \boldsymbol{r}_k)$,
  if $k = 0$ then
    $\boldsymbol{p}_1 = \boldsymbol{r}_0$.
  else
    $\beta = (\frac{\rho_{k+1}}{\rho_k})(\frac{\alpha}{\omega_k})$,
    $\boldsymbol{p}_{k+1} = \boldsymbol{r}_k + \beta(\boldsymbol{p}_k - \omega_k \boldsymbol{v}_k)$.
  end if
  $\boldsymbol{y} = M^{-1}\boldsymbol{p}_{k+1}$,
  $\boldsymbol{v}_{k+1} = A\boldsymbol{y}$,
  $\alpha = \dfrac{\rho_{k+1}}{(\boldsymbol{r}_0, \boldsymbol{v}_{k+1})}$,
  $\boldsymbol{s} = \boldsymbol{r}_k - \alpha \boldsymbol{v}_{k+1}$,
  $\boldsymbol{z} = M^{-1}\boldsymbol{s}$,
  $\boldsymbol{t} = A\boldsymbol{z}$,
  $\omega_{k+1} = \dfrac{(\boldsymbol{t}, \boldsymbol{s})}{(\boldsymbol{t}, \boldsymbol{t})}$,
  $\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha \boldsymbol{y} + \omega_{k+1}\boldsymbol{z}$,
  $\boldsymbol{r}_{k+1} = \boldsymbol{s} - \omega_{k+1}\boldsymbol{t}$.

(4) **the GMRES(m) method** (Bibliography(3))

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0$.
for $j = 1, 2, \cdots$
  $\boldsymbol{r}^{(j)} = M^{-1}(\boldsymbol{b} - A\boldsymbol{u}_0^{(j)})$,
  $\boldsymbol{v}_1 = \dfrac{\boldsymbol{r}^{(j)}}{\|\boldsymbol{r}^{(j)}\|_2}$.
  for $i = 1, 2, \cdots m$
    $\boldsymbol{w} = M^{-1}A\boldsymbol{v_i}$.
    for $k = 1, 2, \cdots i$
      $\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k$.
    $\boldsymbol{v}_{i+1} = \dfrac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$,
    Determine the parameters $\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$
    so that $\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ takes the least value.
  $\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}$

Note: the number of iterations in the GMRES(m) algorithm is defined by
(number of internal iterations i) +(number of external iterations j)× m .

4.1.2.4 Preconditioning Methods

In the following are described the details of those preconditioning algorithms that are implemented in subroutines in this chapter, synopses of which are explained 4.1.2.3.

(1) **Scaling preconditioning**
Scaling preconditioning is a technique that uses the following matrix consisting of the diagonal components of matrix A as the preconditioning matrix M.

$$D = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn}),$$

In particular, the scaling preconditioned CG method is known as the SCG method. On an Vector Engine, the scaling method often is faster than other preconditioning techniques.

4.1.2.5 Advanced Techniques for Improving Performance

(1) **Iterative Improvement Method**

An iterative improvement method is a technique for constructing a higher precision solution based on an approximate solution obtained by some solution method. If a single-precision subroutine was used, then a solution on the order of a double-precision solution will be obtained, and if a double-precision subroutine was used, then a solution on the order of a quadruple-precision solution will be obtained. (However, the calculation time will be approximately two to four times longer.) Another benefit of an iterative improvement method is that the error of the obtained solution can be estimated. If an iterative improvement method is not used, then the error is difficult to estimate even if the residual is known. A problem with the PCG method is that the error cannot be improved beyond a certain degree no matter how close the residual gets to zero. Therefore, if the precision of the solution is particularly important or if a single-precision subroutine has been used, then an iterative improvement of the solution should be performed.

The iterative improvement method can be intuitively derived from the following kind of transformation, where $\boldsymbol{u}$ is the true solution of the equations $A\boldsymbol{u} = \boldsymbol{b}$ and $\boldsymbol{u}'$ is the approximation solution:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{u}' + \boldsymbol{u} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}\boldsymbol{b} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}(\boldsymbol{b} - A\boldsymbol{u}') \end{aligned}$$

From this transformation the following algorithm is obtained:

(a) Use a subroutine from this chapter to solve the given equations $A\boldsymbol{u} = \boldsymbol{b}$ and let $\boldsymbol{u}'$ be the solution that was obtained.

(b) Obtain the residual of the solution with high precision. That is, if a single-precision subroutine was used, calculate $\boldsymbol{r}' = \boldsymbol{b} - A\boldsymbol{u}'$ with double precision, and if a double-precision subroutine was used, calculate $\boldsymbol{r}'$ with quadruple precision.

(c) Return $\boldsymbol{r}'$ to the original precision and assign it to $\boldsymbol{r}$. That is, for single-precision subroutine, assign $\boldsymbol{r}'$ to a single-precision $\boldsymbol{r}$, and for a double-precision subroutine, assign $\boldsymbol{r}'$ to a double-precision $\boldsymbol{r}$.

(d) Use a subroutine from this chapter to solve the equations $A\boldsymbol{v} = \boldsymbol{r}$ and let $\boldsymbol{v}'$ be the solution that was obtained.

(e) Calculate $\boldsymbol{u}' + \boldsymbol{v}'$ and let it be called $\boldsymbol{u}'$.

(f) $\boldsymbol{v}'$ is the estimated error. If $\boldsymbol{v}'$ is sufficiently small, then stop. If not, then return to (b).

The most important part of this algorithm is step (b). The error is improved by calculating the residual with high precision. The iterative improvement method generally can obtain a sufficient degree of precision in several iterations.

(2) **Node numbering in the finite element method**

Although it it known that odd-even ordering or multicolor ordering, for example, should be performed to obtain long vectors, this kind of tricky numbering may significantly degrade convergence in a preconditioned

iterative method. Therefore, care is required(Bibliography(4)). The most stable numbering is numbering that reduces the matrix band width. To automatically generate this kind of numbering, you should use a technique such as the Cuthill-McKee method(Bibliography(5)).

4.1.3 Reference Bibliography

(1) Sonneveld, P. , "CGS, a Fast Lanczos–type Solver for Nonsymmetric Linear Systems", Delft University of Technology, Report No. 84–16, Delft The Netherland (1984).

(2) Van Der Vorst, H, A. , "Bi–CGSTAB:A more smoothly converging variant of CGS for the solution of nonsymmetric linear systems", SIAM J. Sci. Stat. Comput. 13, pp631–644 (1992).

(3) Y. Saad and M. H. SCHULTZ, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM J. Sci. Stat. Comput. , vol. 7, pp856-869 (1986).

(4) Duff, I. S. and Meurant, G. A. , "The Effect of Ordering on Preconditioned Conjugate Gradients", CERFACS, TR88/2, Toulouse, France (1988).

(5) Cuthill, E. and McKee, J. , "Reducing the Bandwidth of Sparse Symmetric Matrices", Proc. of the 24th National Conference of the Association of Computing Machinery, Prandon Press, New Jersey, pp. 157–172 (1969).

4.2 SPARSE MATRIX—NONSTATIONARY ITERATIVE METHODS (BASIC ITERATION METHOD ROUTINES)

4.2.1 QXE010, PXE010
Positive Definite Symmetric Sparse Matrix (ELLPACK Format) (CG method)

(1) **Function**

QXE010 or PXE010 uses the scaling preconditioned CG method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having a sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

```
CALL QXE010  (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)
```

Single precision:

```
CALL PXE010  (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: {INTEGER(4) as for 32bit Integer}

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D, R}	LNA, M	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	LNA	I	1	Input	Adjustable dimension of array A and JA
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	M denotes the column number of both array A and JA (See Note (d))
5	JA	I	LNA, M	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	B	{D, R}	N	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	U	{D, R}	N	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	ITRMAX	I	1	Input	Maximum number of iterations (Default value: N)
9	ITR	I	1	Output	Actual number of iterations
10	EPSMAX	{D, R}	1	Input	Truncation residual norm Default value : $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$
11	EPS	{D, R}	1	Output	Final residual norm
12	ISW	I	1	Input	Processing switch (Default value: 0) (See Note (e)) ISW= 0 : Check whether the values JA are all distinct. ISW= 1 : Does not check whether the values JA are all distinct.
13	NT	I	1	Input	Number of tasks to be generated (Default value: 1)
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}$

(b) $1 \leq \mathrm{M} \leq \mathrm{N}$

(c)
- $\mathrm{JA(i,1)} = \mathrm{i}, \mathrm{JA(i,j)} \neq \mathrm{i} \quad (\mathrm{i} = 1, \cdots, \mathrm{N}; \mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  $1 \leq \mathrm{JA(i,j)} \leq \mathrm{N} \quad (\mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.
- $\mathrm{JA(i, j_i + 1)} = 0$, if $j_i < \mathrm{M}$, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{JA(i,j)} \quad (\mathrm{j} = 1, \cdots, \mathrm{j_i})$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{A(i,1)} \neq 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$

(f) $\mathrm{ITRMAX} \geq 1$

(g) EPSMAX > underflow decision value

(h) $\mathrm{ISW} \in \{0, 1\}$

(i) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with ITRMAX = N.
1200	Restriction (g) was not satisfied.	Processing continues with EPSMAX $= \left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	Processing continues with ISW=0, on the assumption that JA satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with NT = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{U(i)} \leftarrow 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	ISW was equal to 0.	Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.

IERR value	Meaning	Processing
4000	A has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ is less than the underflow decision value.	
4320	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ is greater than the overflow decision value.	
4410	$\lvert(\boldsymbol{p}_i, A\boldsymbol{p}_i)\rvert$ is less than the underflow decision value.	
4420	$\lvert\ (\boldsymbol{p}_i, A\boldsymbol{p}_i)\ \rvert$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL.

Table 4−2 Constants used in ASL

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
Overflow decision value	Maximum value $\times10^{-3}$	
Underflow decision value	Positive minimum value $\times10^{3}$	

(b) The storage method used for the arrays A and JA is as follows.

Figure 4−1 Storage format for Input Data

Matrix A to be stored

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	0.0
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

Storage status of arrays A(LNA , M), JA(LNA , M)

A

$a_{1,1}$	$a_{1,2}$	$a_{1,6}$	$a_{1,4}$	$*$
$a_{2,2}$	$a_{2,1}$	$a_{2,3}$	$a_{2,4}$	$a_{2,7}$
$a_{3,3}$	$a_{3,2}$	$a_{3,6}$	$*$	$*$
⋮	⋮	⋮	⋮	⋮
$a_{6,6}$	$a_{6,1}$	$a_{6,3}$	$a_{6,4}$	$*$
$a_{7,7}$	$a_{7,2}$	$*$	$*$	$*$
$<----$		M		$---->$

JA

1	2	6	4	0
2	1	3	4	7
3	2	6	0	$*$
⋮	⋮	⋮	⋮	⋮
6	1	3	4	0
7	2	0	$*$	$*$
$<----$		M		$---->$

LNA ↕ N ↕

Remarks

a. N is order of Matrix A.

b. LNA $\geq$ N must hold.

c. M is the column number of Array A, which contains the nonzero elements of Matrix A.

d. Array A should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though M-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array JA should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array A.
For those rows in which M − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in JA in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array A and array JA are partially changed in order to optimize performance.

(d) M can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting M as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying ISW = 0 (Performance will degrade remarkably when ISW = 1 is specified). And so, ISW = 1 should be specified when it is assured that JA satisfies Restriction (d). When ISW = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if ISW = 0 was specified but JA doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this subroutine. If the work area could not be successfully allocated, processing will be aborted and IERR returns the value 5000. In this case, the problem cannot be solved by using this subroutine unless the problem size is reduced or the machine

environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & -3 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & -4 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & -5 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & -6 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 3 \\ 6 \\ 9 \\ 11 \\ 13 \\ 15 \\ 17 \\ 27 \\ 30 \\ 23 \end{bmatrix}$$

(b) Input data

Input Arrays A, JA and B,

LNA = 11, N = 10, M = 11, ITRMAX = 100, EPSMAX = 10^{-12}, ISW = 0, NT = 2

(c) Main program

```
      PROGRAM OXE010
! *** EXAMPLE OF QXE010 ***
      IMPLICIT NONE
!
      INTEGER LNA,N,M
      PARAMETER( LNA = 11, N = 10, M = 5 )
      INTEGER JA(LNA,M),ITRMAX,ITR,ISW,NT,IERR
      REAL(8) A(LNA,M),B(N),U(N),EPSMAX,EPS
!
      INTEGER I,J,JCNT(N)
      REAL(8) ORG(LNA,N)
!
      DO 100 J=1,M
      DO 110 I=1,N
         A(I,J) = 0.0D0
  110 CONTINUE
  100 CONTINUE
      READ(5,*) (A(I,1),I=1,N,1)
      READ(5,*)  A(1,2),  A(1,3)
      READ(5,*)  A(2,2),  A(2,3), A(2,4)
      READ(5,*)  A(3,2),  A(3,3), A(3,4)
      READ(5,*)  A(4,2),  A(4,3), A(4,4), A(4,5)
      READ(5,*)  A(5,2),  A(5,3), A(5,4), A(5,5)
      READ(5,*)  A(6,2),  A(6,3), A(6,4), A(6,5)
      READ(5,*)  A(7,2),  A(7,3), A(7,4), A(7,5)
      READ(5,*)  A(8,2),  A(8,3), A(8,4)
      READ(5,*)  A(9,2),  A(9,3), A(9,4)
      READ(5,*) A(10,2), A(10,3)
!
      DO 120 J=1,M
      DO 130 I=1,N
         JA(I,J) = 0
  130 CONTINUE
  120 CONTINUE
      DO 140 I=1,N
         JA(I,1) = I
  140 CONTINUE
      READ(5,*)  JA(1,2),  JA(1,3)
      READ(5,*)  JA(2,2),  JA(2,3), JA(2,4)
      READ(5,*)  JA(3,2),  JA(3,3), JA(3,4)
      READ(5,*)  JA(4,2),  JA(4,3), JA(4,4), JA(4,5)
      READ(5,*)  JA(5,2),  JA(5,3), JA(5,4), JA(5,5)
      READ(5,*)  JA(6,2),  JA(6,3), JA(6,4), JA(6,5)
      READ(5,*)  JA(7,2),  JA(7,3), JA(7,4), JA(7,5)
      READ(5,*)  JA(8,2),  JA(8,3), JA(8,4)
      READ(5,*)  JA(9,2),  JA(9,3), JA(9,4)
      READ(5,*) JA(10,2), JA(10,3)
!
      DO 150 I=1,N
         JCNT(I) = M
  150 CONTINUE
      DO 160 I=1,N
      DO 170 J=1,M
```

```
         IF( JA(I,J) .EQ. 0 ) THEN
            JCNT(I) = J - 1
            GOTO 180
         ENDIF
  170 CONTINUE
  180 CONTINUE
  160 CONTINUE
      DO 190 I=1,N
         B(I) = 0.0D0
  190 CONTINUE
      DO 200 I=1,N
      DO 210 J=1,JCNT(I)
         B(I) = B(I) + A(I,J)
  210 CONTINUE
  200 CONTINUE
!
      DO 220 J=1,N
      DO 230 I=1,LNA
         ORG(I,J) = 0.0D0
  230 CONTINUE
  220 CONTINUE
      DO 240 I=1,N
      DO 250 J=1,JCNT(I)
         ORG(I,JA(I,J)) = A(I,J)
  250 CONTINUE
  240 CONTINUE
!
      ITRMAX = 100
      EPSMAX = 1.0D-12
      ISW    = 1
      NT     = 2
!
      WRITE(6,6000)
      DO 260 I=1,N
         WRITE(6,6010) (ORG(I,J),J=1,N)
  260 CONTINUE
      WRITE(6,6020)
      DO 270 I=1,N
         WRITE(6,6030) B(I)
  270 CONTINUE
      WRITE(6,6040) LNA,N,M,ITRMAX,EPSMAX,NT
!
      CALL QXE010(A,LNA,N,M,JA,B,U,ITRMAX,ITR,EPSMAX,EPS,ISW,NT,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6060) ITR,EPS
      DO 280 I=1,N
         WRITE(6,6070) I,U(I)
  280 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QXE010  ***',/,/,&
             1X,' **  ORIGINAL MATRIX A **',/)
 6010 FORMAT(1X,'  ',10(2X,F5.1))
 6020 FORMAT(/,&
             1X,' **  VECTOR B **',/)
 6030 FORMAT(1X,'    ',F5.1)
 6040 FORMAT(/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA    = ',I5,/,&
             1X,'     N      = ',I5,/,&
             1X,'     M      = ',I5,/,&
             1X,'     ITRMAX = ',I5,/,&
             1X,'     EPSMAX = ',1PD11.3,/,&
             1X,'     NT     = ',I5)
 6050 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I5,/)
 6060 FORMAT(1X,'     ITR  = ',I5,/,&
             1X,'     EPS  = ',1PD11.3,/)
 6070 FORMAT(1X,'     ','U(',I2,') = ',1PD14.6)
!
      END
```

(d) Output results

```
***  QXE010  ***

 **  ORIGINAL MATRIX A **

     3.0    1.0    0.0   -1.0    0.0    0.0    0.0    0.0    0.0    0.0
     1.0    5.0    2.0    0.0   -2.0    0.0    0.0    0.0    0.0    0.0
     0.0    2.0    7.0    3.0    0.0   -3.0    0.0    0.0    0.0    0.0
    -1.0    0.0    3.0    9.0    4.0    0.0   -4.0    0.0    0.0    0.0
     0.0   -2.0    0.0    4.0   11.0    5.0    0.0   -5.0    0.0    0.0
     0.0    0.0   -3.0    0.0    5.0   13.0    6.0    0.0   -6.0    0.0
     0.0    0.0    0.0   -4.0    0.0    6.0   15.0    7.0    0.0   -7.0
     0.0    0.0    0.0    0.0   -5.0    0.0    7.0   17.0    8.0    0.0
     0.0    0.0    0.0    0.0    0.0   -6.0    0.0    8.0   19.0    9.0
     0.0    0.0    0.0    0.0    0.0    0.0   -7.0    0.0    9.0   21.0
```

```
**  VECTOR B **

     3.0
     6.0
     9.0
    11.0
    13.0
    15.0
    17.0
    27.0
    30.0
    23.0

**  INPUT  **

    LNA    =    11
    N      =    10
    M      =     5
    ITRMAX =   100
    EPSMAX =   1.000D-12
    NT     =     2

**  OUTPUT  **

    IERR =     0

    ITR  =    10
    EPS  =   8.131D-16

    U( 1) =   1.000000D+00
    U( 2) =   1.000000D+00
    U( 3) =   1.000000D+00
    U( 4) =   1.000000D+00
    U( 5) =   1.000000D+00
    U( 6) =   1.000000D+00
    U( 7) =   1.000000D+00
    U( 8) =   1.000000D+00
    U( 9) =   1.000000D+00
    U(10) =   1.000000D+00
```

4.2.2 QXE020, PXE020
Asymmetric Sparse Matrix (ELLPACK Format) (CGS method)

(1) **Function**

QXE020 or PXE020 uses the scaling preconditioned CGS method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

```
CALL QXE020  (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)
```

Single precision:

```
CALL PXE020  (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: {INTEGER(4) as for 32bit Integer}

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D, R}	LNA, M	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	LNA	I	1	Input	Adjustable dimension of array A and JA
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	M denotes the column number of both array A and JA (See Note (d))
5	JA	I	LNA, M	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	B	{D, R}	N	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	U	{D, R}	N	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	ITRMAX	I	1	Input	Maximum number of iterations (Default value: N)
9	ITR	I	1	Output	Actual number of iterations
10	EPSMAX	{D, R}	1	Input	Truncation residual norm Default value: $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$
11	EPS	{D, R}	1	Output	Final residual norm
12	ISW	I	1	Input	Processing switch (Default value: 0) (See Note (e)) ISW= 0 : Check whether the values JA are all distinct. ISW= 1 : Does not check whether the values JA are all distinct.
13	NT	I	1	Input	Number of tasks to be generated (Default value: 1)
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}$

(b) $1 \leq \mathrm{M} \leq \mathrm{N}$

(c)
- $\mathrm{JA(i,1)} = \mathrm{i}, \mathrm{JA(i,j)} \neq \mathrm{i} \quad (\mathrm{i} = 1, \cdots, \mathrm{N}; \mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  $1 \leq \mathrm{JA(i,j)} \leq \mathrm{N} \quad (\mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

- $\mathrm{JA(i, j_i + 1)} = 0$, if $j_i < \mathrm{M}$, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{JA(i,j)} \quad (\mathrm{j} = 1, \cdots, \mathrm{j_i})$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{A(i,1)} \neq 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$

(f) $\mathrm{ITRMAX} \geq 1$

(g) $\mathrm{EPSMAX} > underflowdecisionvalue$

(h) $\mathrm{ISW} \in \{0, 1\}$

(i) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with ITRMAX = N.
1200	Restriction (g) was not satisfied.	Processing continues with EPSMAX = $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	ISW has the input value 0. Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with NT = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{U(i)} \leftarrow 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	ISW was equal to 0.	Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	

IERR value	Meaning	Processing
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.
4000	A has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\|(\boldsymbol{r}_0, \boldsymbol{r}_i)\|$ is less than the underflow decision value.	
4320	$\|(\boldsymbol{r}_0, \boldsymbol{r}_i)\|$ is greater than the overflow decision value.	
4410	$\|(\boldsymbol{r}_0, A\boldsymbol{p}_i)\|$ is less than the underflow decision value.	
4420	$\|(\boldsymbol{r}_0, A\boldsymbol{p}_i)\|$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL.

Table 4−3 Constants used in ASL

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
Overflow decision value	Maximum value $\times10^{-3}$	
Underflow decision value	Positive minimum value $\times10^{3}$	

(b) The storage method used for the arrays A and JA is as follows.

Figure 4−2 Storage format for Input Data

Matrix A to be stored

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

Storage status of arrays A(LNA , M), JA(LNA , M)

LNA

A				
$a_{1,1}$	$a_{1,2}$	$a_{1,6}$	$a_{1,4}$	$*$
$a_{2,2}$	$a_{2,1}$	$a_{2,3}$	$a_{2,4}$	$a_{2,7}$
$a_{3,3}$	$a_{3,2}$	$a_{3,6}$	$*$	$*$
⋮	⋮	⋮	⋮	⋮
$a_{6,6}$	$a_{6,1}$	$a_{6,3}$	$a_{6,4}$	$a_{6,7}$
$a_{7,7}$	$a_{7,2}$	$*$	$*$	$*$
$<----$		M		$---->$

JA				
1	2	6	4	0
2	1	3	4	7
3	2	6	0	$*$
⋮	⋮	⋮	⋮	⋮
6	1	3	4	7
7	2	0	$*$	$*$
$<----$		M		$---->$

N

Remarks

a. N is order of Matrix A.

b. LNA $\geq$ N must hold.

c. M is the column number of Array A, which contains the nonzero elements of Matrix A.

d. Array A should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though M-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array JA should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array A.
For those rows in which M − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in JA in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array A and array JA are partially changed in order to optimize performance.

(d) M can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting M as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying ISW = 0 (Performance will degrade remarkably when ISW = 1 is specified). And so, ISW = 1 should be specified when it is assured that JA satisfies Restriction (d). When ISW = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if ISW = 0 was specified but JA doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this subroutine. If the work area could not be suc-

cessfully allocated, processing will be aborted and IERR returns the value 5000. In this case, the problem cannot be solved by using this subroutine unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 15 \\ 18 \\ 21 \\ 17 \\ 27 \\ 30 \\ 23 \end{bmatrix}$$

(b) Input data

Input Arrays A, JA and B,

LNA $= 11$, N $= 10$, M $= 11$, ITRMAX $= 100$, EPSMAX $= 10^{-12}$, ISW $= 0$, NT $= 2$

(c) Main program

```
      PROGRAM OXE020
! *** EXAMPLE OF QXE020 ***
      IMPLICIT NONE
!
      INTEGER LNA,N,M
      PARAMETER( LNA = 11, N = 10, M = 5 )
      INTEGER JA(LNA,M),ITRMAX,ITR,ISW,NT,IERR
      REAL(8) A(LNA,M),B(N),U(N),EPSMAX,EPS
!
      INTEGER I,J,JCNT(N)
      REAL(8) ORG(LNA,N)
!
      DO 100 J=1,M
      DO 110 I=1,N
         A(I,J) = 0.0D0
  110 CONTINUE
  100 CONTINUE
      READ(5,*) (A(I,1),I=1,N,1)
      READ(5,*)  A(1,2)
      READ(5,*)  A(2,2),  A(2,3)
      READ(5,*)  A(3,2),  A(3,3)
      READ(5,*)  A(4,2),  A(4,3), A(4,4)
      READ(5,*)  A(5,2),  A(5,3), A(5,4)
      READ(5,*)  A(6,2),  A(6,3), A(6,4)
      READ(5,*)  A(7,2),  A(7,3), A(7,4), A(7,5)
      READ(5,*)  A(8,2),  A(8,3), A(8,4)
      READ(5,*)  A(9,2),  A(9,3), A(9,4)
      READ(5,*) A(10,2), A(10,3)
!
      DO 120 J=1,M
      DO 130 I=1,N
         JA(I,J) = 0
  130 CONTINUE
  120 CONTINUE
      DO 140 I=1,N
         JA(I,1) = I
  140 CONTINUE
      READ(5,*)  JA(1,2)
      READ(5,*)  JA(2,2),  JA(2,3)
      READ(5,*)  JA(3,2),  JA(3,3)
      READ(5,*)  JA(4,2),  JA(4,3), JA(4,4)
      READ(5,*)  JA(5,2),  JA(5,3), JA(5,4)
      READ(5,*)  JA(6,2),  JA(6,3), JA(6,4)
      READ(5,*)  JA(7,2),  JA(7,3), JA(7,4), JA(7,5)
      READ(5,*)  JA(8,2),  JA(8,3), JA(8,4)
      READ(5,*)  JA(9,2),  JA(9,3), JA(9,4)
      READ(5,*) JA(10,2), JA(10,3)
!
      DO 150 I=1,N
```

```
            JCNT(I) = M
  150 CONTINUE
      DO 160 I=1,N
      DO 170 J=1,M
         IF( JA(I,J) .EQ. 0 ) THEN
            JCNT(I) = J - 1
            GOTO 180
         ENDIF
  170 CONTINUE
  180 CONTINUE
  160 CONTINUE
      DO 190 I=1,N
         B(I) = 0.0D0
  190 CONTINUE
      DO 200 I=1,N
      DO 210 J=1,JCNT(I)
         B(I) = B(I) + A(I,J)
  210 CONTINUE
  200 CONTINUE
!
      DO 220 J=1,N
      DO 230 I=1,LNA
         ORG(I,J) = 0.0D0
  230 CONTINUE
  220 CONTINUE
      DO 240 I=1,N
      DO 250 J=1,JCNT(I)
         ORG(I,JA(I,J)) = A(I,J)
  250 CONTINUE
  240 CONTINUE
!
      ITRMAX = 100
      EPSMAX = 1.0D-12
      ISW    = 1
      NT     = 2
!
      WRITE(6,6000)
      DO 260 I=1,N
         WRITE(6,6010) (ORG(I,J),J=1,N)
  260 CONTINUE
      WRITE(6,6020)
      DO 270 I=1,N
         WRITE(6,6030) B(I)
  270 CONTINUE
      WRITE(6,6040) LNA,N,M,ITRMAX,EPSMAX,NT
!
      CALL QXE020(A,LNA,N,M,JA,B,U,ITRMAX,ITR,EPSMAX,EPS,ISW,NT,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6060) ITR,EPS
      DO 280 I=1,N
         WRITE(6,6070) I,U(I)
  280 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QXE020  ***',/,/,&
             1X,' **  ORIGINAL MATRIX A **',/)
 6010 FORMAT(1X,'  ',10(2X,F5.1))
 6020 FORMAT(/,&
             1X,' **  VECTOR B **',/)
 6030 FORMAT(1X,'    ',F5.1)
 6040 FORMAT(/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA    = ',I5,/,&
             1X,'     N      = ',I5,/,&
             1X,'     M      = ',I5,/,&
             1X,'     ITRMAX = ',I5,/,&
             1X,'     EPSMAX = ',1PD11.3,/,&
             1X,'     NT     = ',I5)
 6050 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I5,/)
 6060 FORMAT(1X,'     ITR  = ',I5,/,&
             1X,'     EPS  = ',1PD11.3,/)
 6070 FORMAT(1X,'     ','U(',I2,') = ',1PD14.6)
!
      END
```

(d) Output results

```
***  QXE020  ***

 **  ORIGINAL MATRIX A **

     3.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
     1.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
     0.0    2.0    7.0    3.0    0.0    0.0    0.0    0.0    0.0    0.0
    -1.0    0.0    3.0    9.0    4.0    0.0    0.0    0.0    0.0    0.0
     0.0   -2.0    0.0    4.0   11.0    5.0    0.0    0.0    0.0    0.0
     0.0    0.0   -3.0    0.0    5.0   13.0    6.0    0.0    0.0    0.0
     0.0    0.0    0.0   -4.0    0.0    6.0   15.0    7.0    0.0   -7.0
```

```
 0.0    0.0    0.0    0.0   -5.0    0.0    7.0   17.0    8.0    0.0
 0.0    0.0    0.0    0.0    0.0   -6.0    0.0    8.0   19.0    9.0
 0.0    0.0    0.0    0.0    0.0    0.0   -7.0    0.0    9.0   21.0

**  VECTOR B **

 4.0
 8.0
12.0
15.0
18.0
21.0
17.0
27.0
30.0
23.0

**  INPUT  **

    LNA    =    11
    N      =    10
    M      =     5
    ITRMAX =   100
    EPSMAX =   1.000D-12
    NT     =     2

**  OUTPUT  **

    IERR =     0

    ITR  =    10
    EPS  =   1.662D-16

    U( 1) =   1.000000D+00
    U( 2) =   1.000000D+00
    U( 3) =   1.000000D+00
    U( 4) =   1.000000D+00
    U( 5) =   1.000000D+00
    U( 6) =   1.000000D+00
    U( 7) =   1.000000D+00
    U( 8) =   1.000000D+00
    U( 9) =   1.000000D+00
    U(10) =   1.000000D+00
```

4.2.3 QXE030, PXE030
Asymmetric Sparse Matrix (ELLPACK Format) (BiCGSTAB method)

(1) **Function**

QXE030 or PXE030 uses the scaling preconditioned BiCGSTAB method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

CALL QXE030 (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)

Single precision:

CALL PXE030 (A,LNA,N,M,JA,B,U,ITRMAX, ITR,EPSMAX, EPS,ISW,NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: {INTEGER(4) as for 32bit Integer}

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D, R}	LNA, M	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	LNA	I	1	Input	Adjustable dimension of array A and JA
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	M denotes the column number of both array A and JA (See Note (d))
5	JA	I	LNA, M	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	B	{D, R}	N	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	U	{D, R}	N	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	ITRMAX	I	1	Input	Maximum number of iterations (Default value: N)
9	ITR	I	1	Output	Actual number of iterations
10	EPSMAX	{D, R}	1	Input	Truncation residual norm Default value: $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$
11	EPS	{D, R}	1	Output	Final residual norm
12	ISW	I	1	Input	Processing switch(Default value: 0) (See Note (e)) ISW= 0 : Check whether the values JA are all distinct. ISW= 1 : Does not check whether the values JA are all distinct.
13	NT	I	1	Input	Number of tasks to be generated (Default value: 1)
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}$

(b) $1 \leq \mathrm{M} \leq \mathrm{N}$

(c)
- $\mathrm{JA(i,1)} = \mathrm{i}, \mathrm{JA(i,j)} \neq \mathrm{i} \quad (\mathrm{i} = 1, \cdots, \mathrm{N}; \mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  $1 \leq \mathrm{JA(i,j)} \leq \mathrm{N} \quad (\mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

- $\mathrm{JA(i, j_i + 1)} = 0$, if $j_i < \mathrm{M}$, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{JA(i,j)} \quad (\mathrm{j} = 1, \cdots, \mathrm{j_i})$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{A(i,1)} \neq 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$

(f) $\mathrm{ITRMAX} \geq 1$

(g) $\mathrm{EPSMAX} > underflowdecisionvalue$

(h) $\mathrm{ISW} \in \{0, 1\}$

(i) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with ITRMAX = N.
1200	Restriction (g) was not satisfied.	Processing continues with EPSMAX = $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	ISW has the input value 0. Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with NT = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{U(i)} \leftarrow 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	ISW was equal to 0.	Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.

IERR value	Meaning	Processing
4000	A has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\lvert\boldsymbol{\rho}_i\rvert$ is less than the underflow decision value.	
4320	$\lvert\boldsymbol{\rho}_i\rvert$ is greater than the overflow decision value.	
4410	$\lvert(\boldsymbol{r}_0, \boldsymbol{v}_i)\rvert$ is less than the underflow decision value.	
4420	$\lvert(\boldsymbol{r}_0, \boldsymbol{v}_i)\rvert$ is greater than the overflow decision value.	
4510	$\lvert(\boldsymbol{t}, \boldsymbol{t})\rvert$ is less than the underflow decision value.	
4520	$\lvert(\boldsymbol{t}, \boldsymbol{t})\rvert$ is greater than the overflow decision value.	
4610	$\lvert\boldsymbol{\omega}_i\rvert$ is less than the underflow decision value.	
4620	$\lvert\boldsymbol{\omega}_i\rvert$ is greater than the overflow decision value.	
4700	$\lvert\boldsymbol{\beta}\rvert$ is greater than the overflow decision value.	
4800	$\lvert\boldsymbol{\alpha}\rvert$ is greater than the overflow decision value.	
4900	$\lvert(\boldsymbol{t}, \boldsymbol{s})\rvert$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL.

Table 4−4 Constants used in ASL

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
Overflow decision value	Maximum value $\times10^{-3}$	
Underflow decision value	Positive minimum value $\times10^{3}$	

(b) The storage method used for the arrays A and JA is as follows.

Figure 4−3 Storage format for Input Data

Matrix A to be stored

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

Storage status of arrays A(LNA , M), JA(LNA , M)

A

$a_{1,1}$	$a_{1,2}$	$a_{1,6}$	$a_{1,4}$	$*$
$a_{2,2}$	$a_{2,1}$	$a_{2,3}$	$a_{2,4}$	$a_{2,7}$
$a_{3,3}$	$a_{3,2}$	$a_{3,6}$	$*$	$*$
⋮	⋮	⋮	⋮	⋮
$a_{6,6}$	$a_{6,1}$	$a_{6,3}$	$a_{6,4}$	$a_{6,7}$
$a_{7,7}$	$a_{7,2}$	$*$	$*$	$*$
$<----$		M		$---->$

JA

1	2	6	4	0
2	1	3	4	7
3	2	6	0	$*$
⋮	⋮	⋮	⋮	⋮
6	1	3	4	7
7	2	0	$*$	$*$
$<----$		M		$---->$

LNA ↕ N

Remarks

a. N is order of Matrix A.

b. LNA $\geq$ N must hold.

c. M is the column number of Array A, which contains the nonzero elements of Matrix A.

d. Array A should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though M-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array JA should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array A.
For those rows in which M − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in JA in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array A and array JA are partially changed in order to optimize performance.

(d) M can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of

nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting M as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying ISW = 0 (Performance will degrade remarkably when ISW = 1 is specified). And so, ISW = 1 should be specified when it is assured that JA satisfies Restriction (d). When ISW = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if ISW = 0 was specified but JA doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this subroutine. If the work area could not be successfully allocated, processing will be aborted and IERR returns the value 5000. In this case, the problem cannot be solved by using this subroutine unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 15 \\ 18 \\ 21 \\ 17 \\ 27 \\ 30 \\ 23 \end{bmatrix}$$

(b) Input data

Input Arrays A, JA and B,

LNA = 11, N = 10, M = 11, ITRMAX = 100, EPSMAX = 10^{-12}, ISW = 0, NT = 2

(c) Main program

```
      PROGRAM OXE030
! *** EXAMPLE OF QXE030 ***
      IMPLICIT NONE
!
      INTEGER LNA,N,M
      PARAMETER( LNA = 11, N = 10, M = 5 )
      INTEGER JA(LNA,M),ITRMAX,ITR,ISW,NT,IERR
      REAL(8) A(LNA,M),B(N),U(N),EPSMAX,EPS
!
      INTEGER I,J,JCNT(N)
      REAL(8) ORG(LNA,N)
!
      DO 100 J=1,M
      DO 110 I=1,N
         A(I,J) = 0.0D0
  110 CONTINUE
  100 CONTINUE
      READ(5,*) (A(I,1),I=1,N,1)
      READ(5,*)  A(1,2)
      READ(5,*)  A(2,2),  A(2,3)
      READ(5,*)  A(3,2),  A(3,3)
      READ(5,*)  A(4,2),  A(4,3), A(4,4)
      READ(5,*)  A(5,2),  A(5,3), A(5,4)
      READ(5,*)  A(6,2),  A(6,3), A(6,4)
      READ(5,*)  A(7,2),  A(7,3), A(7,4), A(7,5)
      READ(5,*)  A(8,2),  A(8,3), A(8,4)
      READ(5,*)  A(9,2),  A(9,3), A(9,4)
      READ(5,*) A(10,2), A(10,3)
!
```

```
      DO 120 J=1,M
      DO 130 I=1,N
         JA(I,J) = 0
  130 CONTINUE
  120 CONTINUE
      DO 140 I=1,N
         JA(I,1) = I
  140 CONTINUE
      READ(5,*)  JA(1,2)
      READ(5,*)  JA(2,2),  JA(2,3)
      READ(5,*)  JA(3,2),  JA(3,3)
      READ(5,*)  JA(4,2),  JA(4,3), JA(4,4)
      READ(5,*)  JA(5,2),  JA(5,3), JA(5,4)
      READ(5,*)  JA(6,2),  JA(6,3), JA(6,4)
      READ(5,*)  JA(7,2),  JA(7,3), JA(7,4), JA(7,5)
      READ(5,*)  JA(8,2),  JA(8,3), JA(8,4)
      READ(5,*)  JA(9,2),  JA(9,3), JA(9,4)
      READ(5,*) JA(10,2), JA(10,3)
!
      DO 150 I=1,N
         JCNT(I) = M
  150 CONTINUE
      DO 160 I=1,N
      DO 170 J=1,M
         IF( JA(I,J) .EQ. 0 ) THEN
            JCNT(I) = J - 1
            GOTO 180
         ENDIF
  170 CONTINUE
  180 CONTINUE
  160 CONTINUE
      DO 190 I=1,N
         B(I) = 0.0D0
  190 CONTINUE
      DO 200 I=1,N
      DO 210 J=1,JCNT(I)
         B(I) = B(I) + A(I,J)
  210 CONTINUE
  200 CONTINUE
!
      DO 220 J=1,N
      DO 230 I=1,LNA
         ORG(I,J) = 0.0D0
  230 CONTINUE
  220 CONTINUE
      DO 240 I=1,N
      DO 250 J=1,JCNT(I)
         ORG(I,JA(I,J)) = A(I,J)
  250 CONTINUE
  240 CONTINUE
!
      ITRMAX = 100
      EPSMAX = 1.0D-12
      ISW    = 1
      NT     = 2
!
      WRITE(6,6000)
      DO 260 I=1,N
         WRITE(6,6010) (ORG(I,J),J=1,N)
  260 CONTINUE
      WRITE(6,6020)
      DO 270 I=1,N
         WRITE(6,6030) B(I)
  270 CONTINUE
      WRITE(6,6040) LNA,N,M,ITRMAX,EPSMAX,NT
!
      CALL QXE030(A,LNA,N,M,JA,B,U,ITRMAX,ITR,EPSMAX,EPS,ISW,NT,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6060) ITR,EPS
      DO 280 I=1,N
         WRITE(6,6070) I,U(I)
  280 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QXE030  ***',/,/,&
             1X,' **  ORIGINAL MATRIX A **',/)
 6010 FORMAT(1X,'  ',10(2X,F5.1))
 6020 FORMAT(/,&
             1X,' **  VECTOR B **',/)
 6030 FORMAT(1X,'    ',F5.1)
 6040 FORMAT(/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA    = ',I5,/,&
             1X,'     N      = ',I5,/,&
             1X,'     M      = ',I5,/,&
             1X,'     ITRMAX = ',I5,/,&
             1X,'     EPSMAX = ',1PD11.3,/,&
             1X,'     NT     = ',I5)
 6050 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I5,/)
```

```
 6060 FORMAT(1X,'     ITR  = ',I5,/,&
             1X,'     EPS  = ',1PD11.3,/)
 6070 FORMAT(1X,'     ','U(',I2,') = ',1PD14.6)
!
      END
```

(d) Output results

```
***  QXE030  ***

 **  ORIGINAL MATRIX A **

      3.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
      1.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
      0.0    2.0    7.0    3.0    0.0    0.0    0.0    0.0    0.0    0.0
     -1.0    0.0    3.0    9.0    4.0    0.0    0.0    0.0    0.0    0.0
      0.0   -2.0    0.0    4.0   11.0    5.0    0.0    0.0    0.0    0.0
      0.0    0.0   -3.0    0.0    5.0   13.0    6.0    0.0    0.0    0.0
      0.0    0.0    0.0   -4.0    0.0    6.0   15.0    7.0    0.0   -7.0
      0.0    0.0    0.0    0.0   -5.0    0.0    7.0   17.0    8.0    0.0
      0.0    0.0    0.0    0.0    0.0   -6.0    0.0    8.0   19.0    9.0
      0.0    0.0    0.0    0.0    0.0    0.0   -7.0    0.0    9.0   21.0

 **  VECTOR B **

      4.0
      8.0
     12.0
     15.0
     18.0
     21.0
     17.0
     27.0
     30.0
     23.0

 **  INPUT  **

     LNA    =    11
     N      =    10
     M      =     5
     ITRMAX =   100
     EPSMAX =   1.000D-12
     NT     =     2

 **  OUTPUT  **

     IERR =     0

     ITR  =    10
     EPS  =   2.421D-16

     U( 1) =   1.000000D+00
     U( 2) =   1.000000D+00
     U( 3) =   1.000000D+00
     U( 4) =   1.000000D+00
     U( 5) =   1.000000D+00
     U( 6) =   1.000000D+00
     U( 7) =   1.000000D+00
     U( 8) =   1.000000D+00
     U( 9) =   1.000000D+00
     U(10) =   1.000000D+00
```

4.2.4 QXE040, PXE040
Asymmetric Sparse Matrix (ELLPACK Format) (GMRES(m) method)

(1) **Function**

QXE040 or PXE040 uses the scaling preconditioned GMRES(m) method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

```
CALL QXE040 (A,LNA,N,M,JA,B,U,ITRMAX,  ITR,MGMRS,EPSMAX,  EPS,ISW,NT,
             IERR)
```

Single precision:

```
CALL PXE040 (A,LNA,N,M,JA,B,U,ITRMAX,  ITR,MGMRS,EPSMAX,  EPS,ISW,NT,
             IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: {INTEGER(4) as for 32bit Integer}

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D, R}	LNA, M	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	LNA	I	1	Input	Adjustable dimension of array A and JA
3	N	I	1	Input	Order of matrix A
4	M	I	1	Input	M denotes the column number of both array A and JA (See Note (d))
5	JA	I	LNA, M	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	B	{D, R}	N	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	U	{D, R}	N	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	ITRMAX	I	1	Input	Maximum number of iterations (Default value: N)
9	ITR	I	1	Output	Actual number of iterations
10	MGMRS	I	1	Input	GMRES(m) parameter m (Default value: 10)
11	EPSMAX	{D, R}	1	Input	Truncation residual norm Default value: { 10^{-12} (doubleprecision) 10^{-6} (singleprecision) }
12	EPS	{D, R}	1	Output	Final residual norm
13	ISW	I	1	Input	Processing switch(Default value: 0) (See Note (e)) ISW= 0 : Check whether the values JA are all distinct. ISW= 1 : Does not check whether the values JA are all distinct.
14	NT	I	1	Input	Number of tasks to be generated (Default value: 1)
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}$

(b) $1 \leq \mathrm{M} \leq \mathrm{N}$

(c)
- $\mathrm{JA(i,1)} = \mathrm{i}, \mathrm{JA(i,j)} \neq \mathrm{i} \quad (\mathrm{i} = 1, \cdots, \mathrm{N}; \mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  $1 \leq \mathrm{JA(i,j)} \leq \mathrm{N} \quad (\mathrm{j} = 2, \cdots, \mathrm{j_i})$,
  where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

- $\mathrm{JA(i, j_i + 1)} = 0$, if $j_i < \mathrm{M}$, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{JA(i,j)} \quad (\mathrm{j} = 1, \cdots, \mathrm{j_i})$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{N})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{A(i,1)} \neq 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$

(f) $\mathrm{ITRMAX} \geq 1$

(g) $\mathrm{MGMRS} \geq 1$

(h) $\mathrm{EPSMAX} > underflowdecisionvalue$

(i) $\mathrm{ISW} \in \{0, 1\}$

(j) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with ITRMAX = N.
1100	Restriction (g) was not satisfied.	Processing continues with MGMRS = 10.
1200	Restriction (h) was not satisfied.	Processing continues with EPSMAX $= \left\{ \begin{array}{ll} 10^{-12} & (\mathrm{doubleprecision}) \\ 10^{-6} & (\mathrm{singleprecision}) \end{array} \right\}$.
1300	Restriction (i) was not satisfied.	ISW has the input value 0. Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
1400	Restriction (j) was not satisfied.	Processing continues with NT = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{U(i)} \leftarrow 0.0 \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	ISW was equal to 0.	Processing continues on the assumption that JA satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	

IERR value	Meaning	Processing
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.
4000	A has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the initial residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The norm of the residual $\boldsymbol{r}$ of the final solution is greater than the overflow decision value.	
4310	$\|AM^{-1}v_i\|_2{}^2$ is less than the underflow decision value. In this case, it is very possible that AM^{-1} is degenerate.	The solution obtained by the (i − 1) iteration is returned, and processing is aborted.
4320	$\|AM^{-1}\boldsymbol{v}_i\|_2{}^2$ is greater than the overflow decision value.	Processing is aborted.
4410	$\|w\|_2{}^2$ after Gram-Schmidt orthogonalization became less than the underflow decision value, and processing could not be continued. This indicates that the iterative solution has already converged. It is very possible that the convergence decision conditions are too strict.	The solution obtained by that time is returned, and processing is aborted.
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL.

Table 4−5 Constants used in ASL

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
Overflow decision value	Maximum value $\times10^{-3}$	
Underflow decision value	Positive minimum value $\times10^{3}$	

(b) The storage method used for the arrays A and JA is as follows.

Figure 4−4 Storage format for Input Data

Matrix A to be stored

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

Storage status of arrays A(LNA , M), JA(LNA , M)

A

$a_{1,1}$	$a_{1,2}$	$a_{1,6}$	$a_{1,4}$	$*$
$a_{2,2}$	$a_{2,1}$	$a_{2,3}$	$a_{2,4}$	$a_{2,7}$
$a_{3,3}$	$a_{3,2}$	$a_{3,6}$	$*$	$*$
⋮	⋮	⋮	⋮	⋮
$a_{6,6}$	$a_{6,1}$	$a_{6,3}$	$a_{6,4}$	$a_{6,7}$
$a_{7,7}$	$a_{7,2}$	$*$	$*$	$*$
< − − − −		M		− − − − >

JA

1	2	6	4	0
2	1	3	4	7
3	2	6	0	$*$
⋮	⋮	⋮	⋮	⋮
6	1	3	4	7
7	2	0	$*$	$*$
< − − − −		M		− − − − >

LNA ↕ N

Remarks

a. N is order of Matrix A.

b. LNA $\geq$ N must hold.

c. M is the column number of Array A, which contains the nonzero elements of Matrix A.

d. Array A should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though M-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array JA should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array A.
For those rows in which M − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in JA in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array A and array JA are partially changed in order to optimize performance.

(d) M can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of

nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting M as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying ISW = 0 (Performance will degrade remarkably when ISW = 1 is specified). And so, ISW = 1 should be specified when it is assured that JA satisfies Restriction (d). When ISW = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if ISW = 0 was specified but JA doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this subroutine. If the work area could not be successfully allocated, processing will be aborted and IERR returns the value 5000. In this case, the problem cannot be solved by using this subroutine unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 15 \\ 18 \\ 21 \\ 17 \\ 27 \\ 30 \\ 23 \end{bmatrix}$$

(b) Input data

Input Arrays A, JA and B,

LNA = 11, N = 10, M = 11, ITRMAX = 100, MGMRS = 5, EPSMAX = 10^{-12}, ISW = 0, NT = 2

(c) Main program

```
      PROGRAM OXE040
! *** EXAMPLE OF QXE040 ***
      IMPLICIT NONE
!
      INTEGER LNA,N,M
      PARAMETER( LNA = 11, N = 10, M = 5 )
      INTEGER JA(LNA,M),ITRMAX,ITR,MGMRS,ISW,NT,IERR
      REAL(8) A(LNA,M),B(N),U(N),EPSMAX,EPS
!
      INTEGER I,J,JCNT(N)
      REAL(8) ORG(LNA,N)
!
      DO 100 J=1,M
      DO 110 I=1,N
         A(I,J) = 0.0D0
  110 CONTINUE
  100 CONTINUE
      READ(5,*) (A(I,1),I=1,N,1)
      READ(5,*)  A(1,2)
      READ(5,*)  A(2,2),  A(2,3)
      READ(5,*)  A(3,2),  A(3,3)
      READ(5,*)  A(4,2),  A(4,3), A(4,4)
      READ(5,*)  A(5,2),  A(5,3), A(5,4)
      READ(5,*)  A(6,2),  A(6,3), A(6,4)
      READ(5,*)  A(7,2),  A(7,3), A(7,4), A(7,5)
      READ(5,*)  A(8,2),  A(8,3), A(8,4)
      READ(5,*)  A(9,2),  A(9,3), A(9,4)
      READ(5,*) A(10,2), A(10,3)
!
```

```
      DO 120 J=1,M
      DO 130 I=1,N
         JA(I,J) = 0
  130 CONTINUE
  120 CONTINUE
      DO 140 I=1,N
         JA(I,1) = I
  140 CONTINUE
      READ(5,*)  JA(1,2)
      READ(5,*)  JA(2,2),  JA(2,3)
      READ(5,*)  JA(3,2),  JA(3,3)
      READ(5,*)  JA(4,2),  JA(4,3), JA(4,4)
      READ(5,*)  JA(5,2),  JA(5,3), JA(5,4)
      READ(5,*)  JA(6,2),  JA(6,3), JA(6,4)
      READ(5,*)  JA(7,2),  JA(7,3), JA(7,4), JA(7,5)
      READ(5,*)  JA(8,2),  JA(8,3), JA(8,4)
      READ(5,*)  JA(9,2),  JA(9,3), JA(9,4)
      READ(5,*) JA(10,2), JA(10,3)
!
      DO 150 I=1,N
         JCNT(I) = M
  150 CONTINUE
      DO 160 I=1,N
      DO 170 J=1,M
         IF( JA(I,J) .EQ. 0 ) THEN
            JCNT(I) = J - 1
            GOTO 180
         ENDIF
  170 CONTINUE
  180 CONTINUE
  160 CONTINUE
      DO 190 I=1,N
         B(I) = 0.0D0
  190 CONTINUE
      DO 200 I=1,N
      DO 210 J=1,JCNT(I)
         B(I) = B(I) + A(I,J)
  210 CONTINUE
  200 CONTINUE
!
      DO 220 J=1,N
      DO 230 I=1,LNA
         ORG(I,J) = 0.0D0
  230 CONTINUE
  220 CONTINUE
      DO 240 I=1,N
      DO 250 J=1,JCNT(I)
         ORG(I,JA(I,J)) = A(I,J)
  250 CONTINUE
  240 CONTINUE
!
      ITRMAX = 100
      MGMRS  = 10
      EPSMAX = 1.0D-12
      ISW    = 1
      NT     = 2
!
      WRITE(6,6000)
      DO 260 I=1,N
         WRITE(6,6010) (ORG(I,J),J=1,N)
  260 CONTINUE
      WRITE(6,6020)
      DO 270 I=1,N
         WRITE(6,6030) B(I)
  270 CONTINUE
      WRITE(6,6040) LNA,N,M,ITRMAX,MGMRS,EPSMAX,NT
!
      CALL QXE040&
      (A,LNA,N,M,JA,B,U,ITRMAX,ITR,MGMRS,EPSMAX,EPS,ISW,NT,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .GE. 3000 ) STOP
!
      WRITE(6,6060) ITR,EPS
      DO 280 I=1,N
         WRITE(6,6070) I,U(I)
  280 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QXE040  ***',/,/,&
             1X,' **  ORIGINAL MATRIX A **',/)
 6010 FORMAT(1X,'  ',10(2X,F5.1))
 6020 FORMAT(/,&
             1X,' **  VECTOR B **',/)
 6030 FORMAT(1X,'    ',F5.1)
 6040 FORMAT(/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     LNA    = ',I5,/,&
             1X,'     N      = ',I5,/,&
             1X,'     M      = ',I5,/,&
             1X,'     ITRMAX = ',I5,/,&
             1X,'     MGMRS  = ',I5,/,&
             1X,'     EPSMAX = ',1PD11.3,/,&
             1X,'     NT     = ',I5)
```

```
 6050 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I5,/)
 6060 FORMAT(1X,'     ITR  = ',I5,/,&
             1X,'     EPS  = ',1PD11.3,/)
 6070 FORMAT(1X,'     ','U(',I2,') = ',1PD14.6)
!
      END
```

(d) Output results

```
***  QXE040  ***

 **  ORIGINAL MATRIX A **

      3.0    1.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
      1.0    5.0    2.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
      0.0    2.0    7.0    3.0    0.0    0.0    0.0    0.0    0.0    0.0
     -1.0    0.0    3.0    9.0    4.0    0.0    0.0    0.0    0.0    0.0
      0.0   -2.0    0.0    4.0   11.0    5.0    0.0    0.0    0.0    0.0
      0.0    0.0   -3.0    0.0    5.0   13.0    6.0    0.0    0.0    0.0
      0.0    0.0    0.0   -4.0    0.0    6.0   15.0    7.0    0.0   -7.0
      0.0    0.0    0.0    0.0   -5.0    0.0    7.0   17.0    8.0    0.0
      0.0    0.0    0.0    0.0    0.0   -6.0    0.0    8.0   19.0    9.0
      0.0    0.0    0.0    0.0    0.0    0.0   -7.0    0.0    9.0   21.0

 **  VECTOR B **

      4.0
      8.0
     12.0
     15.0
     18.0
     21.0
     17.0
     27.0
     30.0
     23.0

 **  INPUT  **

     LNA    =    11
     N      =    10
     M      =     5
     ITRMAX =   100
     MGMRS  =    10
     EPSMAX =   1.000D-12
     NT     =     2

 **  OUTPUT  **

     IERR =     0

     ITR  =    10
     EPS  =   2.142D-16

     U( 1) =   1.000000D+00
     U( 2) =   1.000000D+00
     U( 3) =   1.000000D+00
     U( 4) =   1.000000D+00
     U( 5) =   1.000000D+00
     U( 6) =   1.000000D+00
     U( 7) =   1.000000D+00
     U( 8) =   1.000000D+00
     U( 9) =   1.000000D+00
     U(10) =   1.000000D+00
```

Chapter 5

EIGENVALUES AND EIGENVECTORS

5.1 INTRODUCTION

This chapter describes subroutines that obtain eigenvalues and eigenvectors of symmetric matrices.
Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

In the standard eigenvalue problem, obtain the value λ and corresponding vector $\boldsymbol{x}$ which satisfy the following equation for a given matrix A:

$A\boldsymbol{x} = \lambda\boldsymbol{x}$.

The value λ and the corresponding vector $\boldsymbol{x}$ are called an eigenvalue and the corresponding eigenvector, respectively.

In generalized eigenvalue problem, obtain the value λ and corresponding vector $\boldsymbol{x}$ which satisfy one of the following equations for given matrices A and B:

$A\boldsymbol{x} = \lambda B\boldsymbol{x}$

or

$AB\boldsymbol{x} = \lambda\boldsymbol{x}$ (both A and B are Hermitian, B is positive definite)

or

$BA\boldsymbol{x} = \lambda\boldsymbol{x}$ (both A and B are Hermitian, B is positive definite).

These λ and $\boldsymbol{x}$ are also called an eigenvalue and an eigenvector. If both A and B are Hermitian and B is positive definite, all the eigenvalues are real and the eigenvectors for different eigenvalues are orthogonal for each other.

The subroutines contained in this chapter provide functions corresponding to the following four categories.

All eigenvalues and all eigenvectors: Obtain all eigenvalues and the corresponding eigenvectors.

All eigenvalues: Obtain all eigenvalues only.

Eigenvalues and eigenvectors: Obtain a number of the largest or smallest eigenvalues and the corresponding eigenvectors.

Eigenvalues: Obtain a number of the largest or smallest eigenvalues.

5.1.1 Notes

(1) In general, functions corresponding to 'All eigenvalues and all eigenvectors' or 'Eigenvalues and eigenvectors' require more processing time and memory than functions corresponding to 'All eigenvalues' or 'Eigenvalues' respectively.

(2) In general, it is more efficient to use functions corresponding to 'Eigenvalues and eigenvectors' or 'Eigenvalues' if you want to obtain no more than 20% of the total number of eigenvalues. To obtain more than 20% of the total number of eigenvalues, functions corresponding to 'All eigenvalues and all eigenvectors' or 'All eigenvalues' require less processing time.

(3) In this library, the subroutines of the generalized eigenvalue require the condition that B is positive definite. In the following cases, however, the eigenvalues and the eigenvectors can be obtained even if B is not positive definite.

(a) Matrix B is not positive definite but matrix A is positive definite

$$Bv = \lambda^{-1} Av$$

gives non-zero eigenvalues.

(b) Both of A and B are not positive definite but $A + B$ is positive definite

$$Av = \frac{\lambda}{1+\lambda}(A+B)v$$

gives the eigenvalues which are not -1.

(4) If the input matrix is a symmetric matrix or Hermitian matrix, the use of the exclusive subroutines requires less processing time.

(5) Since the parallel processing overhead significantly affects the computation cost if the order of the matrix is small, performance may be worse than when a non-parallel subroutine is used.

5.1.2 Algorithms Used

5.1.2.1 Transforming a real symmetric matrix to a real symmetric tridiagonal matrix

The Householder method is used to transform an $n \times n$ real symmetric matrix A to a real symmetric tridiagonal matrix T. That is, $A_1 = A$ is assumed, and for $k = 1, 2, \cdots, n-2$, a vector u_k is taken so that:

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

and all elements below the subdiagonal component in column k of:

$$A_{k+1} = P_k A_k P_k$$

become 0. A_{n-1} becomes the obtained real symmetric tridiagonal matrix. In addition, the transformation matrix P_k is an orthogonal symmetric matrix.

5.1.2.2 Transforming a Hermitian matrix to a real symmetric tridiagonal matrix

First, the Householder method is used to transform an $n \times n$ Hermitian matrix A to a Hermitian tridiagonal matrix S.

$$S = P_{n-2} \cdots P_2 P_1 A P_1 P_2 \cdots P_{n-2}$$

Then a regular complex diagonal matrix D is used (similarity transformation) to transform matrix S to a real symmetric tridiagonal matrix T.

$$T = D^* S D$$

5.1.2.3 The Householder transformation by block algorithm

As for the Householder transformation, the block algorithm is used. This method simplifies the update of a matrix by applying the lump sum of a rank-one matrix update that transforms the original real symmetric matrix into a real symmetric tridiagonal matrix. Let A_{k+1} be the symmetric matrix that is generated after similarity transformations are performed for k times on the original symmetric matrix A. Then it holds that:

$$A_{k+1} = P_k A_k P_k = A_k - \boldsymbol{u}_k \boldsymbol{v}_k^T - \boldsymbol{v}_k \boldsymbol{u}_k^T$$

where P_k is an orthogonal matrix. Also the following relationship holds:

$$A_1 = A$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$\boldsymbol{y}_k = A_k \boldsymbol{u}_k$$

$$\boldsymbol{v}_k = \frac{(\boldsymbol{y}_k - \frac{(\boldsymbol{u}_k^T \boldsymbol{y}_k)\boldsymbol{u}_k}{2H_k})}{H_k}$$

The Householder transformation updates the symmetric matrix twice for one similarity transformation.
The A_{k+1} can be expressed without using A_k explicitly.

$$A_{k+1} = A_{k-1} - \boldsymbol{u}_{k-1}\boldsymbol{v}_{k-1}{}^T - \boldsymbol{v}_{k-1}\boldsymbol{u}_{k-1}{}^T - \boldsymbol{u}_k\boldsymbol{v}_k{}^T - \boldsymbol{v}_k\boldsymbol{u}_k{}^T$$

Similarly, matrix A_{p+1} after the p times of mirror transformation becomes:

$$A_{p+1} = A_1 - \sum_{i=1}^{p}(\boldsymbol{u}_i\boldsymbol{v}_i^T + \boldsymbol{v}_i\boldsymbol{u}_i^T)$$

Therefore, matrix A_{p+1} can be computed at a higher speed if matrix A_1 is updated in the lump, once per $2p$ matrices. If the original matrix is Hermitian, the transpose notation "^{T}" should be replaced with the Hermite conjugate notation "*". For details, refer to (3) and (4) in the reference bibliography.

5.1.2.4 QR method

For a tridiagonal matrix T, there is a unitary matrix Q and an upper triangular matrix R (for which all diagonal components are positive) such that T is uniquely decomposed into $T = QR$. $T_k = T$ is assumed, T_k is decomposed into $T_k = Q_kR_k$, and these are multiplied in the reverse order to form:

$$T_{k+1} \leftarrow R_kQ_k = Q_k^*T_kQ_k(Q_k^* \text{ is the adjoint matrix of } Q_k)(k = 1, \cdots)$$

If $T_1, T_2, \cdots, T_k, T_{k+1}$ are created, they are all tridiagonal matrices. As $k \to \infty, T_k$ converges to a diagonal matrix having the eigenvalues of T as its diagonal elements.
To accelerate convergence in the actual QR method, the values μ_k are taken as approximations of the eigenvalues, $T_k - \mu_kI$ are created in place of T_k by performing an origin shift, and these are decomposed into:

$$T_k - \mu_kI = Q_kR_k$$

To calculate the approximation of an eigenvalue, consider the case when there is an adjacent eigenvalue (or an eigenvalue having a close absolute value). Let the eigenvalue μ_k of the lower-right corner submatrix obtained by the root-free QR method. If $T_{k+1} = R_kQ_k + \mu_kI$ is created, then T_{k+1} becomes:

$$T_{k+1} = Q_k^*T_kQ_k$$

After this operation is iterated until the sequence of matrices converges, the values adjusted by the cumulative amount the origin was shifted become the eigenvalues.
The eigenvectors of the original matrix before tridiagonalization are obtained by the following procedures. First, sequentially accumulate the transformation matrices used when obtaining the trigonal matrix T according to the Householder transformation method. Next, accumulate the transformation matrices $Q_1, Q_2, \cdots, Q_k$ obtained according to the QR method.

5.1.2.5 root-free QR method

The root-free QR method, which eliminates the square root calculations of the QR method, is faster when only seeking the eigenvalues of a real symmetric tridiagonal matrix. Let $\alpha_1, \cdots, \alpha_n$ be the diagonal elements and $\beta_1, \cdots, \beta_{n-1}$ be the subdiagonal elements. Let one of the components of the transformation matrix during the calculation be $P^{(i)}$ and let S_i and C_i be $\sin\theta$ and $\cos\theta$ within $P^{(i)}$. Although square roots must be computed in the calculations:

$$P_i = \alpha_iC_{i-1} - \beta_{i-1}S_{i-1}C_{i-2}$$

$$S_i = \frac{\beta_i}{\sqrt{P_i^2 + \beta_i^2}}$$

$$C_i = \frac{P_i}{\sqrt{P_i^2 + \beta_i^2}}$$

New $\alpha_{i-1} = \alpha_i + P_{i-1}C_{i-2} - P_iC_{i-1}$

New $\beta_{i-2} = S_{i-2}\sqrt{P_{i-1}^2 + \beta_{i-1}^2}$

If these calculations are performed using the squared formats of each of P_i, β_i, S_i, and C_i, and if the following values are assumed: $C_0 = 1, S_0 = 0, r_1 = \alpha_1, P_1^2 = \alpha_1^2, \alpha_{n+1} = \beta_{n+1} = 0$ then the equations can be expressed as follows:

$$t_i^2 = P_i^2 + \beta_{i+1}^2$$

New $\beta_i^2 = S_{i-1}^2 t_i^2$

$$S_i^2 = \frac{\beta_{i+1}^2}{t_i^2}, C_i^2 = \frac{P_i^2}{t_i^2}$$

$$P_{i+1}^2 = \alpha_{i+1}^2 C_i^2 - 2\alpha_i + S_i^2\gamma_i + \beta_{i+1}^2 S_i^2 C_{i-1}^2$$

$$\gamma_{i+1} = \alpha_{i+1}C_i^2 = S_i^2\gamma_i$$

New $\alpha_i = \alpha_{i+1} + \gamma_i - \gamma_{i+1}$

and the calculations can be performed without computing any square roots.
For details, refer to (9) in the reference bibliography.

5.1.2.6 Bisection method

The bisection method obtains several of the largest or smallest eigenvalues of a real symmetric tridiagonal matrix T. If we let $d_1, d_2, \cdots, d_n$ be the diagonal components of T, let $s_1, s_2, \cdots, s_{n-1}$ be the subdiagonal components, let λ be a variable, and create the sequence of functions:

$$f_0(\lambda) = 1$$

$$f_1(\lambda) = d_1 - \lambda$$

$$f_i(\lambda) = (d_i - \lambda)f_{i-1}(\lambda) - s_{i-1}^2 f_{i-2}(\lambda) \quad (i = 2, \cdots, n)$$

then $f_0(\lambda), f_1(\lambda), \cdots, f_m(\lambda)$ is the Sturm sequence. That is, if we let $L(\lambda)$ be the number of non-matching signs for the successive sequence of functions for a given λ, then this $L(\lambda)$ is equal to the number of eigenvalues less than λ. To prevent overflow or underflow, if $g_i(\lambda)$ is actually defined as:

$$g_i(\lambda) = \frac{f_i(\lambda)}{f_{i-1}(\lambda)} \quad (i = 1, 2, \cdots, n)$$

$L(\lambda)$ becomes the number of $g_i(\lambda)$ that are negative. Furthermore, $g_i(\lambda)$ satisfies the following:

$$g_1(\lambda) = d_1 - \lambda$$

$$g_i = (d_i - \lambda) - \frac{s_{i-1}^2}{g_{i-1}(\lambda)} \quad (i = 2, \cdots, n)$$

If $g_{i-1}(\lambda)$=0, then $g_i(\lambda)$ is assumed to be:

$$g_i(\lambda) = (d_i - \lambda) - \frac{|s_{i-1}|}{\varepsilon} \quad (\varepsilon : \text{Units for determining error})$$

Assume that the eigenvalues of T satisfy:

$$\lambda_1 \leq \lambda_2 \leq \cdots\cdots \leq \lambda_m$$

From the Gerschgorin theorem, the lower limit ($x_{\min}$) and upper limit ($x_{\max}$) of all the eigenvalues are given by:

$$x_{\max} = \mathrm{MAX}(d_i + (|s_{i-1}| + |s_i|)) \quad (1 \leq i \leq n)$$

$$x_{\min} = \mathrm{MIN}(d_i - (|s_{i-1}| + |s_i|)) \quad (1 \leq i \leq n)$$

where, $s_0 = s_n = 0$ is assumed.
We continue to make the eigenvalue existence range smaller by repeatedly subdividing the interval while counting the number of eigenvalues as described above, based on $x_{\min}$ and $x_{\max}$. In this way, both ends of the infinitesimal interval can be made to converge to a given eigenvalue.
For information about the Sturm sequence of functions and the Gerschgorin theorem, refer to entries (2) and (5) of the bibliography.

5.1.2.7 Accumulation of similarity (unitary) transformation by block algorithm

In seeking the eigenvectors of a real symmetric matrix using the QR method or the inverse iteration method, it is necessary to calculate the accumulation of the similarity (unitary) matrices that had already been computed in the preceding Householder transformation. It is very effective to apply the block algorithm to this accumulating procedure for getting better performance.
Let P_k be a transformation matrix that is obtained in Householder transformation which transform the real symmetric matrix to a tridiagonal matrix.

$$P_k = \boldsymbol{I} - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

The accumulation of the transformation matrix P_k becomes to:

$$P_1 P_2 \cdots P_{n-2} = \boldsymbol{I} - \Sigma_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T$$

where $\boldsymbol{w}_i^T$ is expressed by the following recurrence formula.

$$\boldsymbol{w}_{n-2}^T = \frac{\boldsymbol{u}_{n-2}^T}{H_{n-2}}$$

$$\boldsymbol{w}_i^T = \frac{\boldsymbol{u}_i^T - \sum_{j=i-1}^{n-2} (\boldsymbol{u}_i^T \boldsymbol{u}_j)\boldsymbol{w}_j^T}{H_i}$$

If we let $\boldsymbol{V}$ be the eigenvectors of a real symmetric tridiagonal matrix which are obtained with the QR method or the inverse iteration method. The eigenvectors $\boldsymbol{X}$ of the original matrix is obtained by the following.

$$\begin{aligned} \boldsymbol{X} &= P_1 \cdots P_{n-2} \boldsymbol{V} \\ &= \boldsymbol{V} - \sum_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T \boldsymbol{V} \end{aligned}$$

A product of a similarity (unitary) matrix P_k and the eigenvectors $\boldsymbol{V}$ is a rank-one update. Therefore, the accumulation of the transformation matrices can be obtained at a higher speed if the matrix updates are performed in the lump. If the original matrix is Hermitian, the transpose notation "^{T}" should be replaced with the Hermite conjugate notation "*".

5.1.2.8 Inverse iteration method

The inverse iteration method is used to obtain the eigenvector corresponding to the eigenvalues obtained by the root-free QR method or bisection method.
Assume the approximate value μ_k of a given eigenvalue λ_k of the real symmetric tridiagonal matrix T has been obtained. If a suitable initial vector $\boldsymbol{v_0}$ has been chosen at this time and the linear simultaneous equations:

$$(T - \mu_k I)\boldsymbol{v}_i = \boldsymbol{v}_{i-1} \quad (i = 1, 2, \cdots)$$

are iteratively solved, then if $\boldsymbol{v_i}$ satisfy the convergence conditions, they converge to the eigenvector.
To solve the simultaneous linear equations, partial pivoting is performed while using the Gauss method to perform an LU decomposition. Then forward elimination and back substitution are performed.

5.1.2.9 Generalized eigenvalue problem

A Cholesky decomposition of B is performed:

$$B = LL^*$$

in the generalized eigenvalue problem for a Hermitian matrix:

$$A\boldsymbol{x} = \lambda B\boldsymbol{x} \quad (A: \text{ Hermitian}, B: \text{ Positive Hermitian})$$

yielding:

$$(L^{-1}A(L^*)^{-1})(L^*\boldsymbol{x}) = \lambda(L^*\boldsymbol{x})$$

If we set:

$$P = L^{-1}A(L^*)^{-1}$$

$$L^*\boldsymbol{x} = \boldsymbol{y}$$

then the generalized eigenvalue problem is transformed to a standard eigenvalue problem for the Hermitian matrix P.

$$P\boldsymbol{y} = \lambda\boldsymbol{y}$$

The eigenvector of matrix A is given by:

$$\boldsymbol{x} = (L^*)^{-1}\boldsymbol{y}.$$

Generalized eigenvalue problems for Hermitian matrix other than $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (B : Positive Hermitian) are classified into two cases by the position of positive Hermitian Matrix B as:

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

and

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

The eigenvalues λ and the eigenvectors x of these equations can be obtained by reducing them to standard eigenvalue problems using the following procedure:

① Apply the Cholesky decomposition to the positive matrix B as $B = L^*L$. (Where L is a lower triangle matrix.)

② $AB\boldsymbol{x} = \lambda\boldsymbol{x}$ is reduced to the eigenvalue problem for $C = LAL^*$, and the eigenvectors are obtained by multiplying the inverse of L.

③ $BA\boldsymbol{x} = \lambda\boldsymbol{x}$ is reduced to the eigenvalue problem for $C = LAL^*$, and the eigenvectors are obtained by multiplying L^*.

5.1.3 Reference Bibliography

(1) Wilkinson, J. H. and Reinsch, C. , "Handbook for Automatic Computation, Vol. II, Linear Algebra", Springer-Verlag, (1971).

(2) Wilkinson, J. H. , "The Algebraic Eigenvalue Problem", Clarendon Press, Oxford, (1965).

(3) Dongarra J. J., Sorensen D. C., and Hammarling A. J., "Block reduction of matrices to condensed forms for eigenvalue computations", Journal of Computational and Applied Mathematics, Vol.27, PP.215-227 (1989).

(4) Dongarra J. J. and van de Geijn R. A. , "Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures", LAPACK Working Note 30, PP.1-12(1991).

(5) Francis, J. G. F. , "The QR transformation, I, II", Comput. J. 4, pp. 265-271, pp.332-345(1961, 1962).

(6) Cuppen, J. J. M., "A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem", Numer. Math. 36, pp. 177-195(1981).

(7) Gu, M. and Eisenstat, S. C., "A Stable and Efficient Algorithm for the rank-1 modification of the symmetric eigenproblem", SIAM J. Matrix Anal. Appl. 15, pp. 1266-1276(1994).

(8) Gu, M. and Eisenstat, S. C., "A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem", SIAM J. Matrix Anal. Appl. 16, pp. 172-191(1995).

(9) Y. Beppu and I. Ninomiya, "HQRII—A Fast Diagonalization Subroutine", Computers and Chemistry Vol.6(1982).

5.2 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)

5.2.1 QCSMAA, PCSMAA
All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix

(1) **Function**

QCSMAA or PCSMAA uses the Householder method and QR method to obtain all eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:

CALL QCSMAA (A, LNA, N, E, W1, NT, IERR)

Single precision:

CALL PCSMAA (A, LNA, N, E, W1, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vectors) corresponding to each eigenvalue
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	E	{D / R}	N	Output	Eigenvalues
5	W1	{D / R}	N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1) and A(1, 1) ← 1.0 are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalues obtained. ($1 \leq i \leq$ N)	Eigenvalues correctly obtained by this time are entered in $E(1), \cdots, E(i-1)$ and eigenvectors corresponding to them are entered in A. (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array A.

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal system.

(d) If eigenvectors are not required, use 5.2.2 $\left\{\begin{matrix}\text{QCSMAN}\\ \text{PCSMAN}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, LNA=11, N=4 and NT=2.

(c) Main program

```
      PROGRAM OCSMAA
! *** EXAMPLE OF QCSMAA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( LNA = 11 )
      DIMENSION A(LNA,LNA), E(LNA), W1(LNA)
!
      READ(5,*) N,NT
      DO 10 I=1, N
         READ(5,*) (A(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N,NT
      DO 20 I=1, N
         WRITE(6,1100) (A(J,I), J=1, I-1), (A(I,J), J=I, N)
   20 CONTINUE
!
      CALL QCSMAA(A,LNA,N,E,W1,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 K=1, N-3, 4
         WRITE(6,1300) ('EIGENVALUE ', I=1, 4)
         WRITE(6,1400) (E(I), I=K, K+3)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 4)
```

```
         DO 30 J=1, N
            WRITE(6,1500) (A(J,I), I=K, K+3)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(N,4).NE.0) THEN
         WRITE(6,1300) ('EIGENVALUE ', I=N/4*4+1, N)
         WRITE(6,1400) (E(I), I=N/4*4+1, N)
         WRITE(6,1300) ('EIGENVECTOR', I=N/4*4+1, N)
         DO 50 J=1, N
            WRITE(6,1500) (A(J,I), I=N/4*4+1, N)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  QCSMAA  ***',/,/,&
             '  **  INPUT  **',/,/,&
             '       N = ', I2,/,/,&
             '       NT= ', I2,/,/,&
             '       INPUT MATRIX A',/)
 1100 FORMAT(7X, 11(F7.1))
 1200 FORMAT(' ',/,/,&
             '  **  OUTPUT  **',/,/,&
             '       IERR = ', I4)
 1300 FORMAT(' ',/,1X, 4(5X, A11, 2X))
 1400 FORMAT(3X, 4(2X, 1PD14.7, 2X))
 1500 FORMAT(2X, 4(F14.8, 4X))
      END
```

(d) Output results

```
***  QCSMAA  ***

 **  INPUT  **

     N =  4

     NT=  2

     INPUT MATRIX A

          6.0    4.0    4.0    1.0
          4.0    6.0    1.0    4.0
          4.0    1.0    6.0    4.0
          1.0    4.0    4.0    6.0


 **  OUTPUT  **

     IERR =    0

     EIGENVALUE         EIGENVALUE         EIGENVALUE         EIGENVALUE
    -1.0000000D+00     5.0000000D+00      5.0000000D+00      1.5000000D+01

     EIGENVECTOR        EIGENVECTOR        EIGENVECTOR        EIGENVECTOR
     0.50000000         0.70710678         0.00000000         0.50000000
    -0.50000000         0.00000000        -0.70710678         0.50000000
    -0.50000000        -0.00000000         0.70710678         0.50000000
     0.50000000        -0.70710678         0.00000000         0.50000000
```

5.2.2 QCSMAN, PCSMAN
All Eigenvalues of a Real Symmetric Matrix

(1) **Function**

QCSMAN or PCSMAN uses the Householder method and root-free QR method to obtain all eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

CALL QCSMAN (A, LNA, N, E, W1, NT, IERR)

Single precision:

CALL PCSMAN (A, LNA, N, E, W1, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	E	{D / R}	N	Output	Eigenvalues
5	W1	{D / R}	N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)←A(1, 1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalues obtained. ($1 \leq i \leq$ N)	Eigenvalues correctly obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$. (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array A.

(b) Eigenvalues are stored in ascending order.

5.2.3 QCSMSS, PCSMSS
Eigenvalues and Eigenvectors of a Real Symmetric Matrix

(1) **Function**

QCSMSS or PCSMSS uses the Householder method, root free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type) and the inverse iteration method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

```
CALL QCSMSS  (A, LNA, N, EPS, E, M, VE, LNV, ISW, IW1, W1, NT, IERR)
```

Single precision:

```
CALL PCSMSS  (A, LNA, N, EPS, E, M, VE, LNV, ISW, IW1, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalues convergence test. (See Note (d))
5	E	{D / R}	M	Output	Eigenvalues
6	M	I	1	Input	The number m of eigenvalues to be obtained.
7	VE	{D / R}	LNV,M	Output	Eigenvectors (column vector) corresponding to each eigenvalue.
8	LNV	I	1	Input	Adjustable dimension of array VE
9	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
10	IW1	I	M	Output	Eigenvectors flag (See Note (e))
11	W1	{D / R}	8×N	Work	Work area
12	NT	I	1	Input	Number of tasks to be generated
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNV

(b) 0<M≤N

(c) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1) and VE(1, 1) ← 1.0 are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array A.

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (IERR = 2000 is output), the following processing is performed.
If IW1(i) = 0: The i-th eigenvector calculation is normally terminated.
If IW1(i) ≠ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for IW1(i).
If processing is normally terminated (IERR = 0 is output), IW1(i) = 0 is set.

(f) The eigenvectors are an orthonormal system.

(g) If eigenvectors are not required, use 5.2.4 $\left\{\begin{matrix}\text{QCSMSN}\\ \text{PCSMSN}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain the three smallest eigenvalues of the matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, LNA=11, N=6, EPS=−1.0, M=3, LNV=11, ISW=−1 and NT=2.

(c) Main program

```
      PROGRAM OCSMSS
! *** EXAMPLE OF DCSMSS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( LNA = 11, LNV = 11 )
      DIMENSION A(LNA,LNA), E(LNA), VE(LNV,LNV), IW1(LNA), W1(8*LNA)
!
      READ(5,*) N, M, NT
      DO 10 I=1, N
         READ(5,*) (A(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N, M, NT
      DO 20 I=1, N
         WRITE(6,1100) (A(J,I), J=1, I-1), (A(I,J), J=I, N)
   20 CONTINUE
!
      ISW = -1
      EPS = -1.0D0
!
      CALL QCSMSS(A,LNA,N,EPS,E,M,VE,LNV,ISW,IW1,W1,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 K=1, M-3, 4
         WRITE(6,1300) ('EIGENVALUE ', I=1, 4)
         WRITE(6,1400) (E(I), I=K, K+3)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 4)
         DO 30 J=1, N
            WRITE(6,1500) (VE(J,I), I=K, K+3)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(M,4).NE.0) THEN
         WRITE(6,1300) ('EIGENVALUE ', I=M/4*4+1, M)
         WRITE(6,1400) (E(I), I=M/4*4+1, M)
         WRITE(6,1300) ('EIGENVECTOR', I=M/4*4+1, M)
         DO 50 J=1, N
            WRITE(6,1500) (VE(J,I), I=M/4*4+1, M)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  QCSMSS  ***',/,/,&
             '  **  INPUT  **',/,/,&
             '       N = ', I2,/,/,&
             '       M = ', I2,/,/,&
             '       NT= ', I2,/,/,&
             '       INPUT MATRIX A',/)
 1100 FORMAT(7X, 11(F7.1))
 1200 FORMAT(' ',/,/,&
             '  **  OUTPUT  **',/,/,&
             '       IERR = ', I4)
 1300 FORMAT(' ',/,1X, 4(5X, A11, 2X))
 1400 FORMAT(3X, 4(2X, 1PD14.7, 2X))
 1500 FORMAT(2X, 4(F14.8, 4X))
      END
```

(d) Output results

```
***  QCSMSS  ***

 **  INPUT  **
```

```
     N =  6

     M =  3

     NT=  2

     INPUT MATRIX A

          0.0    1.0    0.0    0.0    0.0    1.0
          1.0    0.0    1.0    0.0    0.0    0.0
          0.0    1.0    0.0    1.0    0.0    0.0
          0.0    0.0    1.0    0.0    1.0    0.0
          0.0    0.0    0.0    1.0    0.0    1.0
          1.0    0.0    0.0    0.0    1.0    0.0


**   OUTPUT  **

     IERR =    0

     EIGENVALUE         EIGENVALUE         EIGENVALUE
    -2.0000000D+00     -1.0000000D+00     -1.0000000D+00

     EIGENVECTOR        EIGENVECTOR        EIGENVECTOR
     0.40824829         0.00000000         0.57735027
    -0.40824829        -0.50000000        -0.28867513
     0.40824829         0.50000000        -0.28867513
    -0.40824829        -0.00000000         0.57735027
     0.40824829        -0.50000000        -0.28867513
    -0.40824829         0.50000000        -0.28867513
```

5.2.4 QCSMSN, PCSMSN
Eigenvalues of a Real Symmetric Matrix

(1) **Function**

QCSMSN or PCSMSN uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

```
CALL QCSMSN  (A, LNA, N, EPS, E, M, ISW, W1, NT, IERR)
```

Single precision:

```
CALL PCSMSN  (A, LNA, N, EPS, E, M, ISW, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA,N	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalues convergence test. (See Note (d))
5	E	{D / R}	M	Output	Eigenvalues
6	M	I	1	Input	The number m of eigenvalues to be obtained.
7	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
8	W1	{D / R}	5×N	Work	Work area
9	NT	I	1	Input	Number of tasks to be generated
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $0<M\leq N$

(c) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)←A(1, 1) is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array A.

(b) If $ISW\geq 0$, the eigenvalues are stored in descending order. If $ISW<0$, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If $EPS \leq 0$, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

5.3 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.3.1 HCHRAA, GCHRAA
All Eigenvalues and All Eigenvectors of a Hermitian Matrix

(1) **Function**

HCHRAA or GCHRAA uses the Householder method and QR method to obtain all eigenvalues of the Hermitian matrix A=(AR, AI) (two-dimensional array type) (upper triangular type) (real argument type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:

```
CALL HCHRAA  (AR, AI, LNA, N, E, VR, VI, LNV, W1, NT, IERR)
```

Single precision:

```
CALL GCHRAA  (AR, AI, LNA, N, E, VR, VI, LNV, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	E	{D / R}	N	Output	Eigenvalues
6	VR	{D / R}	LNV , N	Output	Real part (column vector) of eigenvectors corresponding to each eigenvalue
7	VI	{D / R}	LNV , N	Output	Imaginary part (column vectors) of eigenvectors corresponding to each eigenvalue
8	LNV	I	1	Input	Adjustable dimension of arrays VR and VI
9	W1	{D / R}	3×N	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNV

(b) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1), VR(1, 1) ← 1.0 and VI(1, 1) ← 0.0 are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$. (However, the order is irregular.) Not eigenvector is obtained at this time.

(6) **Notes**

(a) Real and imaginary parts of the Hermitian matrix are stored only in the upper triangular portions of arrays AR and AI respectively. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal set.

(d) If eigenvectors are not required, use 5.3.2 $\left\{\begin{matrix}\text{HCHRAN}\\\text{GCHRAN}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Real part AR and imaginary part AI of matrix A, LNA=11, N=4, LNV=10 and NT=2.

(c) Main program

```
      PROGRAM UCHRAA
! *** EXAMPLE OF HCHRAA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      CHARACTER*80 FMT
      PARAMETER ( LNA = 11, LNV = 11 )
      DIMENSION AR(LNA,LNA), AI(LNA,LNA), E(LNA),&
                VR(LNV,LNV), VI(LNV,LNV), W1(3*LNA)
!
      READ(5,*) N,NT
      DO 10 I=1, N
         READ(5,*) (AR(I,J), AI(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N,NT
      DO 20 I=1, N
         WRITE(FMT,1100) N
         WRITE(6,FMT) (AR(J,I), -AI(J,I), J=1, I-1),&
                      (AR(I,J),  AI(I,J), J=I, N)
   20 CONTINUE
!
      CALL HCHRAA(AR,AI,LNA,N,E,VR,VI,LNV,W1,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 J=1, N-1, 2
         WRITE(6,1300) ('EIGENVALUE ', I=1, 2)
         WRITE(6,1400) E(J), E(J+1)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 2)
         DO 30 I=1, N
            WRITE(6,1500) VR(I,J), VI(I,J), VR(I,J+1), VI(I,J+1)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(N,2).NE.0) THEN
         WRITE(6,1300) 'EIGENVALUE '
         WRITE(6,1400) E(N)
         WRITE(6,1300) 'EIGENVECTOR'
         DO 50 I=1, N
            WRITE(6,1500) VR(I,N), VI(I,N)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(1X,/,/,&
             1X,'***  HCHRAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ', I4,/,/,&
             1X,'     NT= ', I4,/,/,&
             1X,'     INPUT MATRIX A ( REAL,IMAGINARY )',/)
 1100 FORMAT('(1X,5X,', I2,'(''('',F5.1,'' ,'',F5.1,'')'')')')
 1200 FORMAT(1X,/,/,&
```

```
            1X,' **  OUTPUT  **',/,/,&
            1X,'     IERR = ', I4)
 1300 FORMAT(1X,/, 2(14X, A11, 8X))
 1400 FORMAT(1X, 2(12X, 1PD14.7, 7X))
 1500 FORMAT(1X, 2(5X, F12.8, ' ,', F12.8, 2X))
      END
```

(d) Output results

```
***  HCHRAA  ***

 **  INPUT  **

     N =    4

     NT=    2

     INPUT MATRIX A ( REAL,IMAGINARY )

     (  7.0 ,  0.0) (  3.0 ,  0.0) (  1.0 ,  2.0) ( -1.0 ,  2.0)
     (  3.0 ,  0.0) (  7.0 ,  0.0) (  1.0 , -2.0) ( -1.0 , -2.0)
     (  1.0 , -2.0) (  1.0 ,  2.0) (  7.0 ,  0.0) ( -3.0 ,  0.0)
     ( -1.0 , -2.0) ( -1.0 ,  2.0) ( -3.0 ,  0.0) (  7.0 ,  0.0)


 **  OUTPUT  **

     IERR =    0

            EIGENVALUE                      EIGENVALUE
            0.0000000D+00                   8.0000000D+00

            EIGENVECTOR                     EIGENVECTOR
      0.50000000 ,  0.00000000       -0.70710678 , -0.00000000
     -0.50000000 ,  0.00000000        0.00000000 , -0.00000000
      0.00000000 ,  0.50000000        0.35355339 ,  0.35355339
     -0.00000000 ,  0.50000000       -0.35355339 ,  0.35355339

            EIGENVALUE                      EIGENVALUE
            8.0000000D+00                   1.2000000D+01

            EIGENVECTOR                     EIGENVECTOR
      0.00000000 ,  0.00000000        0.50000000 ,  0.00000000
     -0.09987868 ,  0.70001732        0.50000000 ,  0.00000000
     -0.30006932 , -0.39994800        0.50000000 , -0.00000000
     -0.39994800 ,  0.30006932       -0.50000000 , -0.00000000
```

5.3.2 HCHRAN, GCHRAN
All Eigenvalues of a Hermitian Matrix

(1) **Function**

HCHRAN or GCHRAN uses the Householder method and root-free QR method to obtain all eigenvalues of the Hermitian matrix A=(AR, AI) (two-dimensional array type) (upper triangular type) (real argument type).

(2) **Usage**

Double precision:

CALL HCHRAN (AR, AI, LNA, N, E, W1, NT, IERR)

Single precision:

CALL GCHRAN (AR, AI, LNA, N, E, W1, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	E	{D / R}	N	Output	Eigenvalues
6	W1	{D / R}	3×N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)←AR(1, 1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$. (However, the order is irregular.)

(6) **Notes**

(a) Real and imaginary parts of the Hermitian matrix are stored only in the upper triangular portions of arrays AR and AI respectively. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

5.3.3 HCHRSS, GCHRSS
Eigenvalues and Eigenvectors of a Hermitian Matrix

(1) **Function**

HCHRSS or GCHRSS uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A=(AR, AI) (two-dimensional array type) (upper triangular type) (real argument type) and the inverse iteration method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

CALL HCHRSS (AR, AI, LNA, N, EPS, E, M, VR, VI, LNV, ISW, IW1, W1, NT, IERR)

Single precision:

CALL GCHRSS (AR, AI, LNA, N, EPS, E, M, VR, VI, LNV, ISW, IW1, W1, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
6	E	{D / R}	M	Output	Eigenvalues
7	M	I	1	Input	The number of m of eigenvalues to be obtained.
8	VR	{D / R}	LNV , M	Output	Real part (column vector) of eigenvectors corresponding to each eigenvalue
9	VI	{D / R}	LNV , M	Output	Imaginary parts (column vectors) of eigenvectors corresponding to each eigenvalue
10	LNV	I	1	Input	Adjustable dimension of arrays VR and VI
11	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
12	IW1	I	M	Output	Eigenvector flag (See Note (e))
13	W1	{D / R}	10×N	Work	Work area
14	NT	I	1	Input	Number of tasks to be generated
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNV

(b) 0<M≤N

(c) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← AR(1, 1), VR(1, 1) ← 1.0 and VI(1, 1) ← 0.0 are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The real and imaginary parts of the Hermitian matrix should be stored only in the upper triangular portions of arrays AR and AI respectively (See Appendix A).

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (IERR = 2000 is output), the following processing is performed.
If IW1(i) = 0: The i-th eigenvector calculation is normally terminated.
If IW1(i) ≠ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for IW1(i).
If processing is normally terminated (IERR = 0 is output), IW1(i) = 0 is set.

(f) The eigenvectors are an orthonormal set.

(g) If eigenvectors are not required, use 5.3.4 $\begin{Bmatrix}\text{HCHRSN}\\ \text{GCHRSN}\end{Bmatrix}$.

(7) **Example**

(a) Problem
Obtain the three largest eigenvalues of the Hermitian matrix A.

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data
Real part AR and imaginary part AI of matrix A, LNA=11, N=4, EPS=−1.0, M=3, LNV=11, ISW=1 and NT=2.

(c) Main program

```
      PROGRAM UCHRSS
! *** EXAMPLE OF HCHRSS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      CHARACTER*80 FMT
      PARAMETER ( LNA = 11, LNV = 11 )
      DIMENSION AR(LNA,LNA), AI(LNA,LNA), E(LNA),&
                VR(LNV,LNV), VI(LNV,LNV), IW1(LNA), W1(10*LNA)
!
      READ(5,*) N,M,NT
      DO 10 I=1, N
         READ(5,*) (AR(I,J), AI(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N,M,NT
      DO 20 I=1, N
         WRITE(FMT,1100) N
         WRITE(6,FMT) (AR(J,I), -AI(J,I), J=1, I-1),&
                      (AR(I,J),  AI(I,J), J=I, N)
   20 CONTINUE
!
      ISW = 1
      EPS = -1.0D0
!
      CALL HCHRSS(AR,AI,LNA,N,EPS,E,M,VR,VI,LNV,ISW,IW1,W1,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 J=1, M-1, 2
         WRITE(6,1300) ('EIGENVALUE ', I=1, 2)
         WRITE(6,1400) E(J), E(J+1)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 2)
         DO 30 I=1, N
            WRITE(6,1500) VR(I,J), VI(I,J), VR(I,J+1), VI(I,J+1)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(M,2).NE.0) THEN
         WRITE(6,1300) 'EIGENVALUE '
         WRITE(6,1400) E(M)
         WRITE(6,1300) 'EIGENVECTOR'
         DO 50 I=1, N
            WRITE(6,1500) VR(I,M), VI(I,M)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(1X,/,/,&
             1X,'***  HCHRSS  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ', I4,/,/,&
             1X,'     M = ', I4,/,/,&
             1X,'     NT= ', I4,/,/,&
             1X,'     INPUT MATRIX A ( REAL,IMAGINARY )',/)
 1100 FORMAT('(1X,5X,', I2,'('''(''',F5.1,'' ,''',F5.1,''') ''))')
 1200 FORMAT(1X,/,/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ', I4)
 1300 FORMAT(1X,/, 2(14X, A11, 8X))
 1400 FORMAT(1X, 2(12X, 1PD14.7, 7X))
 1500 FORMAT(1X, 2(5X, F12.8, ' ,', F12.8, 2X))
      END
```

(d) Output results

```
***  HCHRSS  ***

 **  INPUT  **

     N =    4

     M =    3

     NT=    2

     INPUT MATRIX A ( REAL,IMAGINARY )

     (  7.0 ,  0.0) (  3.0 ,  0.0) (  1.0 ,  2.0) ( -1.0 ,  2.0)
     (  3.0 ,  0.0) (  7.0 ,  0.0) (  1.0 , -2.0) ( -1.0 , -2.0)
     (  1.0 , -2.0) (  1.0 ,  2.0) (  7.0 ,  0.0) ( -3.0 ,  0.0)
     ( -1.0 , -2.0) ( -1.0 ,  2.0) ( -3.0 ,  0.0) (  7.0 ,  0.0)


 **  OUTPUT  **

     IERR =    0

            EIGENVALUE                      EIGENVALUE
            1.2000000D+01                   8.0000000D+00

            EIGENVECTOR                     EIGENVECTOR
       0.50000000 ,  0.00000000         0.00000000 ,  0.00000000
       0.50000000 ,  0.00000000        -0.09987868 ,  0.70001732
```

```
 0.50000000 , -0.00000000        -0.30006932 , -0.39994800
-0.50000000 , -0.00000000        -0.39994800 ,  0.30006932

       EIGENVALUE
       8.0000000D+00

       EIGENVECTOR
 0.70710678 ,  0.00000000
-0.00000000 , -0.00000000
-0.35355339 , -0.35355339
 0.35355339 , -0.35355339
```

5.3.4 HCHRSN, GCHRSN
Eigenvalues of a Hermitian Matrix

(1) **Function**

HCHRSN or GCHRSN uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A=(AR, AI) (two-dimensional array type) (upper triangular type) (real argument type).

(2) **Usage**

Double precision:

```
CALL HCHRSN  (AR, AI, LNA, N, EPS, E, M, ISW, W1, NT, IERR)
```

Single precision:

```
CALL GCHRSN  (AR, AI, LNA, N, EPS, E, M, ISW, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrix A
5	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
6	E	{D / R}	M	Output	Eigenvalues
7	M	I	1	Input	The number of m of eigenvalues to be obtained.
8	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
9	W1	{D / R}	5×N	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA

(b) 0<M≤N

(c) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← AR(1, 1), is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The real and imaginary parts of the Hermitian matrix should be stored only in the upper triangular portions of arrays AR and AI respectively (See Appendix A).

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS $\leq$ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

5.4 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE)

5.4.1 HCHEAA, GCHEAA
All Eigenvalues and All Eigenvectors of a Hermitian Matrix

(1) **Function**

HCHEAA or GCHEAA uses the Householder method or QR method to obtain all eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:

CALL HCHEAA (A, LNA, N, E, W1, W2, NT, IERR)

Single precision:

CALL GCHEAA (A, LNA, N, E, W1, W2, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vector) corresponding to each eigenvalue
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	E	{D / R}	N	Output	Eigenvalues
5	W1	{D / R}	N	Work	Work area
6	W2	{Z / C}	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA

(b) NT≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1) and A(1, 1) ←(1.0, 0.0) are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$ (However, the order is irregular). No eigenvector is obtained at this time.

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array A. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal set.

(d) If eigenvectors are not required, use 5.4.2 $\left\{\begin{matrix}\mathrm{HCHEAN}\\ \mathrm{GCHEAN}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, LNA=11 and N=4.

(c) Main program

```
      PROGRAM UCHEAA
! *** EXAMPLE OF HCHEAA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      CHARACTER*80 FMT
      COMPLEX(8) A,W2
      INTEGER NT
      PARAMETER ( LNA = 11, NT = 2 )
      DIMENSION A(LNA,LNA), E(LNA), W1(LNA), W2(LNA)
!
      READ(5,*) N
      DO 10 I=1, N
         READ(5,*) (A(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N
      DO 20 I=1, N
         WRITE(FMT,1100) N
         WRITE(6,FMT) (DCONJG(A(J,I)), J=1, I-1), (A(I,J), J=I, N)
   20 CONTINUE
!
      CALL HCHEAA(A,LNA,N,E,W1,W2,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 J=1, N-1, 2
         WRITE(6,1300) ('EIGENVALUE ', I=1, 2)
         WRITE(6,1400) E(J), E(J+1)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 2)
         DO 30 I=1, N
            WRITE(6,1500) A(I,J), A(I,J+1)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(N,2).NE.0) THEN
         WRITE(6,1300) 'EIGENVALUE '
         WRITE(6,1400) E(N)
         WRITE(6,1300) 'EIGENVECTOR'
         DO 50 I=1, N
            WRITE(6,1500) A(I,N)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  HCHEAA  ***',/,/,&
             '  **  INPUT  **',/,/,&
             '       N = ', I4,/,/,&
             '       INPUT MATRIX A ( REAL,IMAGINARY )',/)
 1100 FORMAT('( ,5X,', I2,'(''('',F5.1,'' ,'',F5.1,'')'')'')')
 1200 FORMAT(' ',/,/,&
             '  **  OUTPUT  **',/,/,&
             '       IERR = ', I4)
 1300 FORMAT(' ',/, 2(14X, A11, 8X))
 1400 FORMAT(' ', 2(12X, 1PD14.7, 7X))
 1500 FORMAT(' ', 2(5X, F12.8, ' ,', F12.8, 2X))
      END
```

(d) Output results

```
***  HCHEAA  ***

 **  INPUT  **

     N =    4

     INPUT MATRIX A ( REAL,IMAGINARY )

    (  7.0 ,  0.0) (  3.0 ,  0.0) (  1.0 ,  2.0) ( -1.0 ,  2.0)
```

```
(  3.0 ,  0.0) (  7.0 ,  0.0) (  1.0 , -2.0) ( -1.0 , -2.0)
(  1.0 , -2.0) (  1.0 ,  2.0) (  7.0 ,  0.0) ( -3.0 ,  0.0)
( -1.0 , -2.0) ( -1.0 ,  2.0) ( -3.0 ,  0.0) (  7.0 ,  0.0)

**  OUTPUT  **

    IERR =    0

           EIGENVALUE                    EIGENVALUE
           0.0000000D+00                 8.0000000D+00

           EIGENVECTOR                   EIGENVECTOR
     0.50000000 ,  0.00000000      -0.70710678 , -0.00000000
    -0.50000000 ,  0.00000000       0.00000000 , -0.00000000
     0.00000000 ,  0.50000000       0.35355339 ,  0.35355339
    -0.00000000 ,  0.50000000      -0.35355339 ,  0.35355339

           EIGENVALUE                    EIGENVALUE
           8.0000000D+00                 1.2000000D+01

           EIGENVECTOR                   EIGENVECTOR
     0.00000000 ,  0.00000000       0.50000000 ,  0.00000000
    -0.09987868 ,  0.70001732       0.50000000 ,  0.00000000
    -0.30006932 , -0.39994800       0.50000000 , -0.00000000
    -0.39994800 ,  0.30006932      -0.50000000 , -0.00000000
```

5.4.2 HCHEAN, GCHEAN
All Eigenvalues of a Hermitian Matrix

(1) **Function**

HCHEAN or GCHEAN uses the Householder method or root-free QR method to obtain all eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type).

(2) **Usage**

Double precision:

CALL HCHEAN (A, LNA, N, E, W1, W2, NT, IERR)

Single precision:

CALL GCHEAN (A, LNA, N, E, W1, W2, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A
4	E	{D / R}	N	Output	Eigenvalues
5	W1	{D / R}	N	Work	Work area
6	W2	{Z / C}	N	Work	Work area
7	NT	I	1	Input	Number of tasks to be generated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	$\mathrm{E}(1) \leftarrow \mathrm{A}(1,1)$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$. (However, the order is irregular.)

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array A. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

5.4.3 HCHESS, GCHESS
Eigenvalues and Eigenvectors of a Hermitian Matrix

(1) **Function**

HCHESS or GCHESS uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type) and the inverse iterative method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

CALL HCHESS (A, LNA, N, EPS, E, M, VE, LNV, ISW, IW1, W1, W2, NT, IERR)

Single precision:

CALL GCHESS (A, LNA, N, EPS, E, M, VE, LNV, ISW, IW1, W1, W2, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A.
3	N	I	1	Input	Order of matrix A
4	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
5	E	{D / R}	M	Output	Eigenvalues
6	M	I	1	Input	The number of m of eigenvalues to be obtained.
7	VE	{Z / C}	LNV , M	Output	Eigenvectors (column vector) corresponding to each eigenvalue
8	LNV	I	1	Input	Adjustable dimension of array VE
9	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
10	IW1	I	M	Output	Eigenvector flag (See Note (e))
11	W1	{D / R}	8×N	Work	Work area
12	W2	{Z / C}	N	Work	Work area
13	NT	I	1	Input	Number of tasks to be generated
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNV

(b) 0<M≤N

(c) NT≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1) and VE(1, 1) ← (1.0, 0.0) are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array A. (See Appendix A)

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW < 0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (IERR = 2000 is output), the following processing is performed.

If IW1(i) = 0: The i-th eigenvector calculation is normally terminated.

If IW1(i) $\neq$ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for IW1(i).

If processing is normally terminated (IERR = 0 is output), IW1(i) = 0 is set.

(f) The eigenvectors are an orthonormal set.

(g) If eigenvectors are not required, use 5.4.4 $\begin{Bmatrix}\text{HCHESN}\\ \text{GCHESN}\end{Bmatrix}$.

(7) **Example**

(a) Problem

Obtain the three largest eigenvalues of the following Hermitian matrix A:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, LNA=11, N=4, EPS=−1.0, M=3, LNV=11 and ISW=1.

(c) Main program

```
      PROGRAM UCHESS
! *** EXAMPLE OF HCHESS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      CHARACTER*80 FMT
      COMPLEX(8) A,VE,W2
      INTEGER NT
      PARAMETER ( LNA = 11, LNV = 11, NT = 2 )
      DIMENSION A(LNA,LNA), E(LNA), VE(LNV,LNV),&
                IW1(LNA), W1(8*LNA), W2(LNA)
!
      READ(5,*) N, M
      DO 10 I=1, N
         READ(5,*) (A(I,J), J=I, N)
   10 CONTINUE
!
      WRITE(6,1000) N, M
      DO 20 I=1, N
         WRITE(FMT,1100) N
         WRITE(6,FMT) (DCONJG(A(J,I)), J=1, I-1), (A(I,J), J=I, N)
   20 CONTINUE
!
      ISW = 1
      EPS = -1.0D0
!
      CALL HCHESS(A,LNA,N,EPS,E,M,VE,LNV,ISW,IW1,W1,W2,NT,IERR)
!
      WRITE(6,1200) IERR
!
      DO 40 J=1, M-1, 2
         WRITE(6,1300) ('EIGENVALUE ', I=1, 2)
         WRITE(6,1400) E(J), E(J+1)
         WRITE(6,1300) ('EIGENVECTOR', I=1, 2)
         DO 30 I=1, N
            WRITE(6,1500) VE(I,J), VE(I,J+1)
   30    CONTINUE
   40 CONTINUE
      IF(MOD(M,2).NE.0) THEN
         WRITE(6,1300) 'EIGENVALUE '
         WRITE(6,1400) E(M)
         WRITE(6,1300) 'EIGENVECTOR'
         DO 50 I=1, N
            WRITE(6,1500) VE(I,M)
   50    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(' ',/,/,&
             ' ***  HCHESS  ***',/,/,&
             '  **  INPUT  **',/,/,&
             '       N = ', I4,/,/,&
             '       M = ', I4,/,/,&
             '       INPUT MATRIX A ( REAL,IMAGINARY )',/)
 1100 FORMAT('( ,5X,', I2,'(''('',F5.1,'' ,'',F5.1,'')''))')
 1200 FORMAT(' ',/,/,&
             '  **  OUTPUT  **',/,/,&
             '       IERR = ', I4)
 1300 FORMAT(' ',/, 2(14X, A11, 8X))
 1400 FORMAT(' ', 2(12X, 1PD14.7, 7X))
 1500 FORMAT(' ', 2(5X, F12.8, ' ,', F12.8, 2X))
      END
```

(d) Output results

```
***  HCHESS  ***

 **  INPUT  **

     N =    4

     M =    3

     INPUT MATRIX A ( REAL,IMAGINARY )

    (  7.0 ,  0.0) (  3.0 ,  0.0) (  1.0 ,  2.0) ( -1.0 ,  2.0)
    (  3.0 ,  0.0) (  7.0 ,  0.0) (  1.0 , -2.0) ( -1.0 , -2.0)
    (  1.0 , -2.0) (  1.0 ,  2.0) (  7.0 ,  0.0) ( -3.0 ,  0.0)
    ( -1.0 , -2.0) ( -1.0 ,  2.0) ( -3.0 ,  0.0) (  7.0 ,  0.0)


 **  OUTPUT  **

     IERR =    0

            EIGENVALUE                    EIGENVALUE
            1.2000000D+01                 8.0000000D+00

            EIGENVECTOR                   EIGENVECTOR
       0.50000000 ,  0.00000000       0.00000000 ,  0.00000000
       0.50000000 ,  0.00000000      -0.09987868 ,  0.70001732
       0.50000000 , -0.00000000      -0.30006932 , -0.39994800
      -0.50000000 , -0.00000000      -0.39994800 ,  0.30006932
```

```
       EIGENVALUE
       8.0000000D+00

       EIGENVECTOR
 0.70710678 ,  0.00000000
-0.00000000 ,  0.00000000
-0.35355339 , -0.35355339
 0.35355339 , -0.35355339
```

5.4.4 HCHESN, GCHESN
Eigenvalues of a Hermitian Matrix

(1) **Function**

HCHESN or GCHESN uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type).

(2) **Usage**

Double precision:

```
CALL HCHESN  (A, LNA, N, EPS, E, M, ISW, W1, W2, NT, IERR)
```

Single precision:

```
CALL GCHESN  (A, LNA, N, EPS, E, M, ISW, W1, W2, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{Z / C}	LNA , N	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A.
3	N	I	1	Input	Order of matrix A
4	EPS	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
5	E	{D / R}	M	Output	Eigenvalues
6	M	I	1	Input	The number of m of eigenvalues to be obtained.
7	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
8	W1	{D / R}	2×N	Work	Work area
9	W2	{Z / C}	N	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA$

(b) $0<M\leq N$

(c) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	$E(1) \leftarrow A(1,1)$ is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array A. (See Appendix A)

(b) If $ISW\geq 0$, the eigenvalues are stored in descending order. If $ISW<0$, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If $EPS \leq 0$, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

5.5 GENERALIZED EIGENVALUE PROBLEM FOR A REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($Ax = \lambda Bx$)

5.5.1 QCGSAA, PCGSAA

All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

QCGSAA or PCGSAA uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and QR method to obtain all eigenvalues and corresponding all eigenvectors.

(2) **Usage**

Double precision:

```
CALL QCGSAA  (A, LNA, N, B, LNB, E, W1, NT, IERR)
```

Single precision:

```
CALL PCGSAA  (A, LNA, N, B, LNB, E, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vector) corresponding to each eigenvalue
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A and B
4	B	{D / R}	LNB , N	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues
7	W1	{D / R}	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0<N\leq LNA, LNB$

(b) $NT\geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1)/B(1, 1) and A(1, 1) ← $1.0/\sqrt{\mathrm{B}(1,1)}$ are performed.
2100	B has a diagonal element very close to zero.	Some eigenvectors may be obtained with low precision, and processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained (1≤i≤N).	Eigenvalues obtained by this time are entered in E(1), ⋯, E(i−1). (However, the order is irregular.) No eigenvector is obtained at this time.

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays A and B.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(d) If eigenvectors are not required, use 5.5.2 $\left\{\begin{matrix}\mathrm{QCGSAN}\\ \mathrm{PCGSAN}\end{matrix}\right\}$.

(7) **Example**

(a) Problem
Obtain all eigenvalues of $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ and their corresponding eigenvectors, where matrices A and B are as follows:

$$A = \begin{bmatrix} 2 & 1 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 153 & 31 & 58 & -58 \\ 31 & 153 & -53 & 58 \\ 58 & -58 & 153 & 31 \\ -58 & 58 & 31 & 153 \end{bmatrix}$$

(b) Input data

Matrix A, LNA=11, N=4, matrix B, LNB=11 and NT=2.

(c) Main program

```
      PROGRAM OCGSAA
! *** EXAMPLE OF QCGSAA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( LNA = 11, LNB = 11 )
      DIMENSION A(LNA,LNA), B(LNB,LNB), E(LNA), W1(2*LNA)
!
      READ(5,*) N,NT
      DO 10 J=1, N
         READ(5,*) (A(J,I), I=J, N)
   10 CONTINUE
      DO 20 J=1, N
         READ(5,*) (B(J,I), I=J, N)
   20 CONTINUE
!
      WRITE(6,1000) N,NT
      WRITE(6,1100) 'A'
      DO 30 J=1, N
         WRITE(6,1200) (A(I,J), I=1, J-1), (A(J,I), I=J, N)
   30 CONTINUE
      WRITE(6,1100) 'B'
      DO 40 J=1, N
         WRITE(6,1200) (B(I,J), I=1, J-1), (B(J,I), I=J, N)
   40 CONTINUE
!
      CALL QCGSAA(A,LNA,N,B,LNB,E,W1,NT,IERR)
!
      WRITE(6,1300) IERR
!
      DO 60 K=1, N-3, 4
         WRITE(6,1400) ('EIGENVALUE ', I=1, 4)
         WRITE(6,1500) (E(I), I=K, K+3)
         WRITE(6,1400) ('EIGENVECTOR', I=1, 4)
         DO 50 J=1, N
            WRITE(6,1500) (A(J,I), I=K, K+3)
   50    CONTINUE
   60 CONTINUE
      IF(MOD(N,4).NE.0) THEN
         WRITE(6,1400) ('EIGENVALUE ', I=N/4*4+1, N)
         WRITE(6,1500) (E(I), I=N/4*4+1, N)
         WRITE(6,1400) ('EIGENVECTOR', I=N/4*4+1, N)
         DO 70 J=1, N
            WRITE(6,1500) (A(J,I), I=N/4*4+1, N)
   70    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(1X,/,/,&
             1X,'***  QCGSAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ', I2,/,/,&
             1X,'     NT= ', I2)
 1100 FORMAT(1X,/,&
             1X,'     INPUT MATRIX ',A1,/)
 1200 FORMAT(1X, 6X, 11(F8.1))
 1300 FORMAT(1X,/,/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ', I4)
 1400 FORMAT(1X,/,1X, 4(5X, A11, 2X))
 1500 FORMAT(1X, 4(2X, 1PD14.7, 2X))
      END
```

(d) Output results

```
***  QCGSAA  ***

 **  INPUT  **

     N =  4
```

```
    NT=  2

    INPUT MATRIX A

          2.0     1.0     1.0     2.0
          1.0     1.0     1.0     1.0
          1.0     1.0     2.0     2.0
          2.0     1.0     2.0     4.0

    INPUT MATRIX B

        153.0    31.0    58.0   -58.0
         31.0   153.0   -58.0    58.0
         58.0   -58.0   153.0    31.0
        -58.0    58.0    31.0   153.0


**  OUTPUT  **

    IERR =    0

    EIGENVALUE        EIGENVALUE        EIGENVALUE        EIGENVALUE
  6.4779811D-04     5.3688291D-03     2.7447086D-02     2.1668091D-01

    EIGENVECTOR       EIGENVECTOR       EIGENVECTOR       EIGENVECTOR
 2.9405960D-02     4.9839235D-02    -1.6115522D-02     2.0451432D-01
-4.6877602D-02     3.7709049D-02     6.8634504D-02    -1.9262477D-01
 3.1083549D-02    -1.9357438D-02     8.5979167D-02    -1.9157548D-01
-1.9570768D-02    -3.3245027D-02     1.4513123D-03     2.0962854D-01
```

5.5.2 QCGSAN, PCGSAN
All Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

QCGSAN or PCGSAN uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and QR method to obtain all eigenvalues.

(2) **Usage**

Double precision:

```
CALL QCGSAN  (A, LNA, N, B, LNB, E, W1, NT, IERR)
```

Single precision:

```
CALL PCGSAN  (A, LNA, N, B, LNB, E, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A and B
4	B	{D / R}	LNB , N	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues
7	W1	{D / R}	N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNB

(b) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1)/B(1, 1) is performed.
2100	B has a diagonal element very close to zero.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained. (1≤i≤N)	Eigenvalues obtained by this time are entered in E(1), ⋯, E(i−1). (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays A and B.

(b) Eigenvalues are stored in ascending order.

5.5.3 QCGSSS, PCGSSS
Eigenvalues and Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

QCGSSS or PCGSSS uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and the root-free QR method or Bisection method to obtain the m largest eigenvalues or m smallest eigenvalues, and uses the reverse iterative method to obtain the eigenvectors.

(2) **Usage**

Double precision:

```
CALL QCGSSS  (A, LNA, N, B, LNB, EPS, E, M, VE, LNV, ISW, IW1, W1, NT, IERR)
```

Single precision:

```
CALL PCGSSS  (A, LNA, N, B, LNB, EPS, E, M, VE, LNV, ISW, IW1, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D/R}	LNA , N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A and B
4	B	{D/R}	LNB , N	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	The strict upper triangular portion is not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	EPS	{D/R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
7	E	{D/R}	M	Output	Eigenvalues
8	M	I	1	Input	The number of m of eigenvalues to be obtained.
9	VE	{D/R}	LNV , M	Output	Eigenvectors (column vector) corresponding to each eigenvalue
10	LNV	I	1	Input	Adjustable dimension of array VE
11	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
12	IW1	I	M	Output	Eigenvector flag (See Note (a))
13	W1	{D/R}	9×N	Work	Work area
14	NT	I	1	Input	Number of tasks to be generated
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNB, LNV

(b) 0<M≤N

(c) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1)/B(1, 1) and VE(1, 1) ← $1.0/\sqrt{\mathrm{B}(1,1)}$ are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
2100	B has a diagonal element very close to zero.	Some eigenvectors may be obtained with low precision, and processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
4000	B was not positive definite.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays A and B.

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (IERR = 2000 is output), the following processing is performed.
If IW1(i) = 0: The i-th eigenvector calculation is normally terminated.
If IW1(i) $\neq$ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for IW1(i).
If processing is normally terminated (IERR = 0 is output), IW1(i) = 0 is set.

(f) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(g) If eigenvectors are not required, use 5.5.4 $\left\{\begin{matrix}\mathrm{QCGSSN}\\ \mathrm{PCGSSN}\end{matrix}\right\}$.

(7) **Example**

(a) ProblemObtain all eigenvalues of $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ and their corresponding eigenvectors, where matrices A and B are as follows:

$$A = \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

$$B = \begin{bmatrix} 170 & 18 & 33 & -21 & -17 & 13 & 25 & -36 \\ 18 & 171 & -21 & 22 & 13 & -17 & -36 & 25 \\ 33 & -21 & 171 & 18 & 25 & -36 & -17 & 13 \\ -21 & 22 & 18 & 171 & -36 & 25 & 13 & -17 \\ -17 & 13 & 25 & -36 & 171 & 18 & 33 & -21 \\ 13 & -17 & -36 & 25 & 18 & 171 & -21 & -3 \\ 25 & -36 & -17 & 13 & 33 & -21 & 171 & 18 \\ -36 & 25 & 13 & -17 & -21 & -3 & 18 & 171 \end{bmatrix}$$

(b) Input data

Matrix A, LNA=11, N=8, matrix B, LNB=11, EPS=-1.0, M=2, LNV=10 and ISW=-1.

(c) Main program

```
      PROGRAM OCGSSS
! *** EXAMPLE OF QCGSSS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( LNA = 11, LNB = 11, LNV = 10 )
      DIMENSION A(LNA,LNA), B(LNB,LNB), E(LNA), VE(LNV,LNV),&
                IW1(LNA), W1(9*LNA)
!
      READ(5,*) N,M,NT
      DO 10 J=1, N
         READ(5,*) (A(J,I), I=J, N)
   10 CONTINUE
      DO 20 J=1, N
         READ(5,*) (B(J,I), I=J, N)
   20 CONTINUE
!
      WRITE(6,1000) N,M,NT
      WRITE(6,1100) 'A'
      DO 30 J=1, N
         WRITE(6,1200) (A(I,J), I=1, J-1), (A(J,I), I=J, N)
   30 CONTINUE
      WRITE(6,1100) 'B'
      DO 40 J=1, N
         WRITE(6,1200) (B(I,J), I=1, J-1), (B(J,I), I=J, N)
   40 CONTINUE
!
      ISW = -1
      EPS = -1.0D0
!
      CALL QCGSSS(A,LNA,N,B,LNB,EPS,E,M,VE,LNV,ISW,IW1,W1,NT,IERR)
!
      WRITE(6,1300) IERR
!
      DO 60 K=1, M-3, 4
         WRITE(6,1400) ('EIGENVALUE ', I=1, 4)
         WRITE(6,1500) (E(I), I=K, K+3)
         WRITE(6,1400) ('EIGENVECTOR', I=1, 4)
         DO 50 J=1, N
            WRITE(6,1500) (VE(J,I), I=K, K+3)
   50    CONTINUE
   60 CONTINUE
      IF(MOD(M,4).NE.0) THEN
         WRITE(6,1400) ('EIGENVALUE ', I=M/4*4+1, M)
         WRITE(6,1500) (E(I), I=M/4*4+1, M)
         WRITE(6,1400) ('EIGENVECTOR', I=M/4*4+1, M)
         DO 70 J=1, N
            WRITE(6,1500) (VE(J,I), I=M/4*4+1, M)
   70    CONTINUE
      ENDIF
      STOP
!
 1000 FORMAT(1X,/,/,&
             1X,'***  QCGSSS  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ', I2,/,/,&
             1X,'     M = ', I2,/,/,&
             1X,'     NT= ', I2)
 1100 FORMAT(1X,/,&
             1X,'     INPUT MATRIX ',A1,/)
 1200 FORMAT(1X, 6X, 11(F8.1))
 1300 FORMAT(1X,/,/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ', I4)
 1400 FORMAT(1X,/,1X, 4(5X, A11, 2X))
 1500 FORMAT(1X, 4(2X, 1PD14.7, 2X))
      END
```

(d) Output results

```
***  QCGSSS  ***

 **  INPUT  **

     N =  8

     M =  2

     NT=  2

     INPUT MATRIX A

         611.0   196.0  -192.0   407.0    -8.0   -52.0   -49.0    29.0
         196.0   899.0   113.0  -192.0   -71.0   -43.0    -8.0   -44.0
        -192.0   113.0   899.0   196.0    61.0    49.0     8.0    52.0
         407.0  -192.0   196.0   611.0     8.0    44.0    59.0   -23.0
          -8.0   -71.0    61.0     8.0   411.0  -599.0   208.0   208.0
         -52.0   -43.0    49.0    44.0  -599.0   411.0   208.0   208.0
         -49.0    -8.0     8.0    59.0   208.0   208.0    99.0  -911.0
          29.0   -44.0    52.0   -23.0   208.0   208.0  -911.0    99.0

     INPUT MATRIX B

         170.0    18.0    33.0   -21.0   -17.0    13.0    25.0   -36.0
          18.0   171.0   -21.0    22.0    13.0   -17.0   -36.0    25.0
          33.0   -21.0   171.0    18.0    25.0   -36.0   -17.0    13.0
         -21.0    22.0    18.0   171.0   -36.0    25.0    13.0   -17.0
         -17.0    13.0    25.0   -36.0   171.0    18.0    33.0   -21.0
          13.0   -17.0   -36.0    25.0    18.0   171.0   -21.0    -3.0
          25.0   -36.0   -17.0    13.0    33.0   -21.0   171.0    18.0
         -36.0    25.0    13.0   -17.0   -21.0    -3.0    18.0   171.0


 **  OUTPUT  **

     IERR =    0

     EIGENVALUE          EIGENVALUE
  -5.3020806D+00     -1.0369304D-15

     EIGENVECTOR         EIGENVECTOR
   7.8865043D-04     -3.2898475D-03
   1.4571715D-03     -6.5796950D-03
   6.2438782D-04      6.5796950D-03
  -1.6786293D-03      3.2898475D-03
  -2.4707363D-02     -4.6057865D-02
  -1.9017760D-02     -4.6057865D-02
   4.7852016D-02     -2.3028932D-02
   4.4548878D-02     -2.3028932D-02
```

5.5.4 QCGSSN, PCGSSN
Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

QCGSSN or PCGSSN uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and the root-free QR method or Bisection method to obtain the m largest eigenvalues or m smallest eigenvalues.

(2) **Usage**

Double precision:

```
CALL QCGSSN  (A, LNA, N, B, LNB, EPS, E, M, ISW, W1, NT, IERR)
```

Single precision:

```
CALL PCGSSN  (A, LNA, N, B, LNB, EPS, E, M, ISW, W1, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D R}	LNA , N	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrix A and B
4	B	{D R}	LNB , N	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	EPS	{D R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
7	E	{D R}	M	Output	Eigenvalues
8	M	I	1	Input	The number of m of eigenvalues to be obtained.
9	ISW	I	1	Input	Processing switch ISW≥0: Obtain M eigenvalues from the largest one. ISW<0: Obtain M eigenvalues from the smallest one.
10	W1	{D R}	5×N	Work	Work area
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) 0<N≤LNA, LNB

(b) 0<M≤N

(c) NT≥1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1) ← A(1, 1)/B(1, 1) is performed.
2100	B has a diagonal element very close to zero.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
4000	B was not positive definite.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays A and B.

(b) If ISW≥0, the eigenvalues are stored in descending order. If ISW<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If EPS ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. EPS is used to obtain eigenvalues by using the Bisection method.

5.6 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($ABx = \lambda x$)

5.6.1 QCGJAA, PCGJAA

All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

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

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{x}$.

(2) **Usage**

Double precision:

```
CALL QCGJAA  (A, LNA, N, B, LNB, E, WORK, NT, IERR)
```

Single precision:

```
CALL PCGJAA  (A, LNA, N, B, LNB, E, WORK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A
				Output	Eigenvectors $\boldsymbol{x}$
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrices A and B
4	B	{D / R}	LNB , N	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues λ
7	WORK	{D / R}	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}, \mathrm{LNB}$

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	$\mathrm{E}(1) \leftarrow \mathrm{A}(1,1) * \mathrm{B}(1,1)$ and $\mathrm{A}(1,1) \leftarrow \dfrac{1.0}{\sqrt{\mathrm{B}(1,1)}}$ are performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq \mathrm{N}$)	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$. (However, the order is irregular.) No eigenvector is obtained at this time.

(6) **Notes**

(a) Arrays A and B should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.6.2 $\left\{\begin{matrix}\mathrm{QCGJAN}\\ \mathrm{PCGJAN}\end{matrix}\right\}$ should be used if the eigenvectors are not needed.

(e) 5.7.1 $\left\{\begin{matrix}\mathrm{QCGKAA}\\ \mathrm{PCGKAA}\end{matrix}\right\}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

Obtain all eigenvalues and their corresponding eigenvectors non-symmetric matrix AB when A and B are positive symmetric matrices.

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

Note The input data A and B are defined as

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

where

$$P(a^2, \mathrm{N})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt \quad (1 \le i \le \mathrm{N}; 1 \le j \le \mathrm{N}).$$

All eigenvalues of the each matrix exist within a finit interval which is independent of N.

(b) Input data

N = 4, LNA = LNB = 4, NT = 2 and symmetric matrices A, B.

(c) Main program

```
      PROGRAM OCGJAA
! *** EXAMPLE OF QCGJAA ***
      IMPLICIT NONE
!
      INTEGER N,LNA,LNB
      PARAMETER( N = 4, LNA = 4, LNB = 4 )
      INTEGER IERR,I,J,L,NT
      REAL(8) A(LNA,N),B(LNB,N)
      REAL(8) E(N),WORK(2*N)
      REAL(8) ONE,TRE,FIV
      PARAMETER( ONE = 1.D0, TRE = 3.D0, FIV = 5.D0 )
!
      NT = 2
!
      WRITE(6,6000) N, LNA, LNB, NT
      DO 100 I=1,N
      DO 110 J=1,N
         A(I,J)= ONE/(ONE+TRE*DBLE(I+J)**2)+ONE/(ONE+TRE*DBLE(I-J)**2)
         B(I,J)= ONE/(ONE+FIV*DBLE(I+J)**2)+ONE/(ONE+FIV*DBLE(I-J)**2)
  110 CONTINUE
  100 CONTINUE
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) A(I,1),A(I,2),A(I,3),A(I,4)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 I=1,N
         WRITE(6,6020) B(I,1),B(I,2),B(I,3),B(I,4)
  130 CONTINUE
!
      CALL QCGJAA(A, LNA, N, B, LNB, E, WORK, NT, IERR)
!
      WRITE(6,6040) IERR
      DO 140 I=1,N,2
         WRITE(6,6050) (' EIGENVALUE',L=1,2)
         WRITE(6,6060) E(I),E(I+1)
         WRITE(6,6050) ('EIGENVECTOR',L=1,2)
         WRITE(6,6060) A(1,I),A(1,I+1)
         WRITE(6,6060) A(2,I),A(2,I+1)
         WRITE(6,6060) A(3,I),A(3,I+1)
         WRITE(6,6060) A(4,I),A(4,I+1)
  140 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QCGJAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ',I4, '     LNA = ',I4,'  LNB = ',I4,&
                '   NT = ',I4,/)
 6010 FORMAT(/,&
             1X,'     INPUT MATRIX A',/)
 6020 FORMAT(1X,3X,4(2X,F9.5))
 6030 FORMAT(/,&
             1X,'     INPUT MATRIX B',/)
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6050 FORMAT(/,&
             1X,7X,A11,22X,A11)
 6060 FORMAT(1X,5X,1PD14.7,19X,1PD14.7)
      END
```

(d) Output results

```
***  QCGJAA  ***

 **  INPUT  **

     N =    4    LNA =    4  LNB =    4   NT =    2


     INPUT MATRIX A

       1.07692    0.28571    0.09733    0.04887
```

```
      0.28571    1.02041    0.26316    0.08610
      0.09733    0.26316    1.00917    0.25676
      0.04887    0.08610    0.25676    1.00518

    INPUT MATRIX B

      1.04762    0.18841    0.05996    0.02968
      0.18841    1.01235    0.17460    0.05314
      0.05996    0.17460    1.00552    0.17073
      0.02968    0.05314    0.17073    1.00312

**  OUTPUT  **

    IERR =    0


       EIGENVALUE                        EIGENVALUE
     5.0334859D-01                     7.0649951D-01

       EIGENVECTOR                       EIGENVECTOR
    -3.5988845D-01                    -5.7276742D-01
     7.0243936D-01                     4.9740733D-01
    -7.1756352D-01                     4.1982158D-01
     4.0575850D-01                    -6.2631989D-01

       EIGENVALUE                        EIGENVALUE
     1.1519432D+00                     2.1334368D+00

       EIGENVECTOR                       EIGENVECTOR
     5.9964094D-01                     4.1404961D-01
     2.5727788D-01                     4.9417207D-01
    -3.8926246D-01                     4.5974369D-01
    -6.0442734D-01                     3.2426391D-01
```

5.6.2 QCGJAN, PCGJAN
All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

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

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

CALL QCGJAN (A, LNA, N, B, LNB, E, WORK, NT, IERR)

Single precision:

CALL PCGJAN (A, LNA, N, B, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrices A and B
4	B	{D / R}	LNB , N	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues λ
7	WORK	{D / R}	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA, LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) is performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E}(1), \cdots, \mathrm{E}(\mathrm{i}-1)$ (However, the order is irregular).

(6) **Notes**

(a) Arrays A and B should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.6.1 $\left\{\begin{matrix}\mathrm{QCGJAA}\\ \mathrm{PCGJAA}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

(d) 5.7.2 $\left\{\begin{matrix}\mathrm{QCGKAN}\\ \mathrm{PCGKAN}\end{matrix}\right\}$ should be used if matrix A is only positive.

5.7 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($BAx = \lambda x$)

5.7.1 QCGKAA, PCGKAA

All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

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

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{x}$.

(2) **Usage**

Double precision:

```
CALL QCGKAA  (A, LNA, N, B, LNB, E, WORK, NT, IERR)
```

Single precision:

```
CALL PCGKAA  (A, LNA, N, B, LNB, E, WORK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A
				Output	Eigenvectors $\boldsymbol{x}$
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrices A and B
4	B	{D / R}	LNB , N	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues λ
7	WORK	{D / R}	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA, LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) and A(1, 1)← $\sqrt{\text{B}(1,1)}$ are performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in E(1), $\cdots$, E(i − 1) (However, the order is irregular). No eigenvector is obtained at this time.

(6) **Notes**

(a) Arrays A and B should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B^{-1} \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.7.2 $\left\{\begin{matrix}\text{QCGKAN}\\ \text{PCGKAN}\end{matrix}\right\}$ should be used if the eigenvectors are not needed.

(e) 5.6.1 $\left\{\begin{matrix}\text{QCGJAA}\\ \text{PCGJAA}\end{matrix}\right\}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

Obtain all eigenvalues and their corresponding eigenvectors non-symmetric matrix AB when A and B are positive symmetric matrices.

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

Note The input data A and B are defined as

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

where

$$P(a^2, \mathrm{N})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt (1 \le i \le \mathrm{N}; 1 \le j \le \mathrm{N}).$$

All eigenvalues of the each matrix exist within a finit interval which is independent of N.

(b) Input data

N=4, LNA=LNB=4, NT=2 and symmetric matrices A, B.

(c) Main program

```
      PROGRAM OCGKAA
! *** EXAMPLE OF QCGKAA ***
      IMPLICIT NONE
!
      INTEGER N,LNA,LNB
      PARAMETER( N = 4, LNA = 4, LNB = 4 )
      INTEGER IERR,I,J,L,NT
      REAL(8) A(LNA,N),B(LNB,N)
      REAL(8) E(N),WORK(2*N)
      REAL(8) ONE,TRE,FIV
      PARAMETER( ONE = 1.D0, TRE = 3.D0, FIV = 5.D0 )
!
      NT = 2
!
      WRITE(6,6000) N, LNA, LNB, NT
      DO 100 I=1,N
      DO 110 J=1,N
         A(I,J)= ONE/(ONE+TRE*DBLE(I+J)**2)+ONE/(ONE+TRE*DBLE(I-J)**2)
         B(I,J)= ONE/(ONE+FIV*DBLE(I+J)**2)+ONE/(ONE+FIV*DBLE(I-J)**2)
  110 CONTINUE
  100 CONTINUE
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) A(I,1),A(I,2),A(I,3),A(I,4)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 I=1,N
         WRITE(6,6020) B(I,1),B(I,2),B(I,3),B(I,4)
  130 CONTINUE
!
      CALL QCGKAA(A, LNA, N, B, LNB, E, WORK, NT, IERR)
!
      WRITE(6,6040) IERR
      DO 140 I=1,N,2
         WRITE(6,6050) (' EIGENVALUE',L=1,2)
         WRITE(6,6060) E(I),E(I+1)
         WRITE(6,6050) ('EIGENVECTOR',L=1,2)
         WRITE(6,6060) A(1,I),A(1,I+1)
         WRITE(6,6060) A(2,I),A(2,I+1)
         WRITE(6,6060) A(3,I),A(3,I+1)
         WRITE(6,6060) A(4,I),A(4,I+1)
  140 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QCGKAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ',I4,'     LNA = ',I4,'  LNB = ',I4,&
                '   NT = ',I4,/)
 6010 FORMAT(/,&
             1X,'     INPUT MATRIX A',/)
 6020 FORMAT(1X,3X,4(2X,F9.5))
 6030 FORMAT(/,&
             1X,'     INPUT MATRIX B',/)
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6050 FORMAT(/,&
             1X,7X,A11,22X,A11)
 6060 FORMAT(1X,5X,1PD14.7,19X,1PD14.7)
      END
```

(d) Output results

```
***  QCGKAA  ***

 **  INPUT  **

     N =    4    LNA =    4  LNB =    4   NT =    2


     INPUT MATRIX A

       1.07692    0.28571    0.09733    0.04887
```

```
      0.28571    1.02041    0.26316    0.08610
      0.09733    0.26316    1.00917    0.25676
      0.04887    0.08610    0.25676    1.00518

    INPUT MATRIX B

      1.04762    0.18841    0.05996    0.02968
      0.18841    1.01235    0.17460    0.05314
      0.05996    0.17460    1.00552    0.17073
      0.02968    0.05314    0.17073    1.00312

**  OUTPUT  **

    IERR =    0


       EIGENVALUE                        EIGENVALUE
     5.0334859D-01                     7.0649951D-01

       EIGENVECTOR                       EIGENVECTOR
    -2.7566971D-01                    -4.9973959D-01
     5.3958111D-01                     4.3565256D-01
    -5.5118459D-01                     3.6771142D-01
     3.1116215D-01                    -5.4715726D-01

       EIGENVALUE                        EIGENVALUE
     1.1519432D+00                     2.1334368D+00

       EIGENVECTOR                       EIGENVECTOR
     6.3538913D-01                     5.6406228D-01
     2.7334189D-01                     6.7578767D-01
    -4.1372916D-01                     6.2875822D-01
    -6.4130226D-01                     4.4231632D-01
```

5.7.2 QCGKAN, PCGKAN
All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

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

(A : Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

CALL QCGKAN (A, LNA, N, B, LNB, E, WORK, NT, IERR)

Single precision:

CALL PCGKAN (A, LNA, N, B, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	LNA , N	Input	Real symmetric matrix A
				Output	Input-time contents are not retained.
2	LNA	I	1	Input	Adjustable dimension of array A
3	N	I	1	Input	Order of matrices A and B
4	B	{D / R}	LNB , N	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	LNB	I	1	Input	Adjustable dimension of array B
6	E	{D / R}	N	Output	Eigenvalues λ
7	WORK	{D / R}	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA, LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) is performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ N).	Eigenvalues obtained by this time are entered in $\mathrm{E(1)}, \cdots, \mathrm{E(i-1)}$. (However, the order is irregular.)

(6) **Notes**

(a) Arrays A and B should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.7.1 $\left\{\begin{matrix}\mathrm{QCGKAA}\\ \mathrm{PCGKAA}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

(d) 5.6.2 $\left\{\begin{matrix}\mathrm{QCGJAN}\\ \mathrm{PCGJAN}\end{matrix}\right\}$ should be used if matrix A is only positive.

5.8 GENERALIZED EIGENVALUE PROBLEM ($Az = \lambda Bz$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.8.1 HCGRAA, GCGRAA

All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$A\boldsymbol{z} = \lambda B\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

CALL HCGRAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGRAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D / R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D / R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D / R}	N	Output	Eigenvalues λ
9	WORK	{D / R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA , LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)$\leftarrow \dfrac{\mathrm{A}(1,1)}{\mathrm{B}(1,1)}$ and A(1, 1)$\leftarrow \dfrac{1.0}{\sqrt{\mathrm{B}(1,1)}}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.8.2 $\left\{\begin{matrix}\mathrm{HCGRAN}\\ \mathrm{GCGRAN}\end{matrix}\right\}$ should be used if the eigenvectors are not needed.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem.

(b) Input data

N=4, LNA=4, matrix A, LNB=4, NT=2 and matrix B.

(c) Main program

```
      PROGRAM UCGRAA
! *** EXAMPLE OF HCGRAA ***
      IMPLICIT NONE
!
      INTEGER IERR,I,J,L,NT
      INTEGER N,LNA,LNB
      PARAMETER( N = 4, LNA = 4, LNB = 4 )
      REAL(8) AR(LNA,N),BR(LNB,N),AI(LNA,N),BI(LNB,N)
      REAL(8) E(N), WORK(4*N)
      COMPLEX(8) KEEP(LNA,N)
!
      NT = 2
!
```

```
      KEEP(1,1)=( 8.D0, 0.D0)
      KEEP(2,2)=( 9.D0, 0.D0)
      KEEP(3,3)=( 10.D0, 0.D0)
      KEEP(4,4)=( 11.D0, 0.D0)
      KEEP(1,2)=( 3.D0, 0.D0)
      KEEP(1,3)=( 1.D0, 2.D0)
      KEEP(1,4)=(-1.D0, 2.D0)
      KEEP(2,3)=( 1.D0,-2.D0)
      KEEP(2,4)=(-1.D0,-2.D0)
      KEEP(3,4)=(-3.D0, 0.D0)
      WRITE(6,6000) N, LNA, LNB, NT
      DO 100 I=1,N
      DO 110 J=I,N
         BR(I,J)=DBLE (KEEP(I,J))
         BI(I,J)=DIMAG(KEEP(I,J))
         KEEP(J,I)=CONJG(KEEP(I,J))
         AR(I,J)=BR(I,J)
         AI(I,J)=-BI(I,J)
         AR(J,I)=AR(I,J)
         AI(J,I)=-AI(I,J)
         BR(J,I)=BR(I,J)
         BI(J,I)=-BI(I,J)
  110 CONTINUE
  100 CONTINUE
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) AR(I,1),AI(I,1),AR(I,2),AI(I,2),&
                       AR(I,3),AI(I,3),AR(I,4),AI(I,4)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 I=1,N
         WRITE(6,6020) BR(I,1),BI(I,1),BR(I,2),BI(I,2),&
                       BR(I,3),BI(I,3),BR(I,4),BI(I,4)
  130 CONTINUE
!
      CALL HCGRAA(AR,AI, LNA, N, BR,BI, LNB, E, WORK, NT, IERR)
!
      WRITE(6,6040) IERR
      DO 140 I=1,N,2
         WRITE(6,6050) (' EIGENVALUE',L=1,2)
         WRITE(6,6060) E(I),E(I+1)
         WRITE(6,6050) ('EIGENVECTOR',L=1,2)
         WRITE(6,6070) AR(1,I),AI(1,I),AR(1,I+1),AI(1,I+1)
         WRITE(6,6070) AR(2,I),AI(2,I),AR(2,I+1),AI(2,I+1)
         WRITE(6,6070) AR(3,I),AI(3,I),AR(3,I+1),AI(3,I+1)
         WRITE(6,6070) AR(4,I),AI(4,I),AR(4,I+1),AI(4,I+1)
  140 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  HCGRAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ',I4,'     LNA = ',I4,'   LNB = ',I4,&
                '    NT = ',I4,/)
 6010 FORMAT(/,&
             1X,'     INPUT MATRIX A ( REAL , IMAGINARY )',/)
 6020 FORMAT(1X,5X,4('(',F5.1,',',F5.1,')'))
 6030 FORMAT(/,&
             1X,'     INPUT MATRIX B ( REAL , IMAGINARY )',/)
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6050 FORMAT(/,&
             1X,14X,A11,22X,A11)
 6060 FORMAT(1X,12X,1PD14.7,19X,1PD14.7)
 6070 FORMAT(1X,5X,F12.8,' ,',F12.8,7X,F12.8,' ,',F12.8)
 6080 FORMAT(1X,' RE(A) = ',D10.2,' IM(A) = ',D10.2,&
              ' RE(B) = ',D10.2,' IM(B) = ',D10.2,&
               ' I,J = ',I2,3X,I2)
      END
```

(d) Output results

```
***  HCGRAA  ***

 **  INPUT  **

     N =    4    LNA =    4   LNB =    4    NT =    2


     INPUT MATRIX A ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  1.0,  2.0)(  1.0, -2.0)( 10.0,  0.0)( -3.0,  0.0)
     ( -1.0,  2.0)( -1.0, -2.0)( -3.0,  0.0)( 11.0,  0.0)

     INPUT MATRIX B ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  1.0, -2.0)(  1.0,  2.0)( 10.0,  0.0)( -3.0,  0.0)
     ( -1.0, -2.0)( -1.0,  2.0)( -3.0,  0.0)( 11.0,  0.0)
```

```
**  OUTPUT  **

    IERR =    0


          EIGENVALUE                    EIGENVALUE
        2.3087618D-01                 1.0000000D+00

         EIGENVECTOR                   EIGENVECTOR
  0.17507856 ,  0.00000000      0.00000000 ,  0.00000000
 -0.16010855 ,  0.00181530     -0.00000000 , -0.00000000
 -0.00199113 , -0.14885444     -0.19242638 ,  0.00302079
  0.00028967 , -0.13796265      0.19242638 , -0.00302079

          EIGENVALUE                    EIGENVALUE
        1.0000000D+00                 4.3313260D+00

         EIGENVECTOR                   EIGENVECTOR
  0.21125364 ,  0.00000000      0.36437067 ,  0.00000000
  0.21125364 , -0.00000000     -0.33321532 , -0.00377797
 -0.03129684 , -0.00000000     -0.00414390 ,  0.30979344
  0.03129684 , -0.00000000      0.00060286 ,  0.28712563
```

5.8.2 HCGRAN, GCGRAN
All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$A\boldsymbol{z} = \lambda B\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

CALL HCGRAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGRAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	AI	{D R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D R}	N	Output	Eigenvalues λ
9	WORK	{D R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA , LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)$\leftarrow \dfrac{\mathrm{A}(1,1)}{\mathrm{B}(1,1)}$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.8.1 $\left\{\begin{matrix}\mathrm{HCGRAA}\\ \mathrm{GCGRAA}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

5.9 GENERALIZED EIGENVALUE PROBLEM ($ABz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.9.1 HCGJAA, GCGJAA

All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{z} = \lambda\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

CALL HCGJAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGJAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D / R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D / R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D / R}	N	Output	Eigenvalues λ
9	WORK	{D / R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{LNA}$, LNB

(b) $\mathrm{NT} \geq 1$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) and A(1, 1)← $\frac{1.0}{\sqrt{\mathrm{B}(1,1)}}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.9.2 $\begin{Bmatrix}\text{HCGJAN}\\ \text{GCGJAN}\end{Bmatrix}$ should be used if the eigenvectors are not needed.

(e) 5.10.1 $\begin{Bmatrix}\text{HCGKAA}\\ \text{GCGKAA}\end{Bmatrix}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem $ABx = \lambda x$.

(b) Input data

N=4, LNA=4, matrix A, LNB=4, NT=2, matrix B.

(c) Main program

```
      PROGRAM UCGJAA
! *** EXAMPLE OF HCGJAA ***
      IMPLICIT NONE
!
      INTEGER N,LNA,LNB
      PARAMETER( N = 4, LNA = 4, LNB = 4 )
      INTEGER IERR,I,J,L,NT
      REAL(8) AR(LNA,N),BR(LNB,N),AI(LNA,N),BI(LNB,N)
      REAL(8) E(N), WORK(4*N)
      COMPLEX(8) KEEP(LNA,N)
!
      NT = 2
!
      KEEP(1,1)=( 8.D0, 0.D0)
      KEEP(2,2)=( 9.D0, 0.D0)
      KEEP(3,3)=( 10.D0, 0.D0)
      KEEP(4,4)=( 11.D0, 0.D0)
      KEEP(1,2)=( 3.D0, 0.D0)
      KEEP(1,3)=( 1.D0, 2.D0)
      KEEP(1,4)=(-1.D0, 2.D0)
      KEEP(2,3)=( 1.D0,-2.D0)
      KEEP(2,4)=(-1.D0,-2.D0)
      KEEP(3,4)=(-3.D0, 0.D0)
      WRITE(6,6000) N, LNA, LNB, NT
      DO 100 I=1,N
      DO 110 J=I,N
         BR(I,J)=DBLE (KEEP(I,J))
         BI(I,J)=DIMAG(KEEP(I,J))
         KEEP(J,I)=CONJG(KEEP(I,J))
         AR(I,J)=BR(I,J)
         AI(I,J)=-BI(I,J)
         AR(J,I)=AR(I,J)
         AI(J,I)=-AI(I,J)
         BR(J,I)=BR(I,J)
         BI(J,I)=-BI(I,J)
```

```
  110 CONTINUE
  100 CONTINUE
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) AR(I,1),AI(I,1),AR(I,2),AI(I,2),&
                       AR(I,3),AI(I,3),AR(I,4),AI(I,4)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 I=1,N
         WRITE(6,6020) BR(I,1),BI(I,1),BR(I,2),BI(I,2),&
                       BR(I,3),BI(I,3),BR(I,4),BI(I,4)
  130 CONTINUE
!
      CALL HCGJAA(AR,AI, LNA, N, BR,BI, LNB, E, WORK, NT, IERR)
!
      WRITE(6,6040) IERR
      DO 140 I=1,N,2
         WRITE(6,6050) (' EIGENVALUE',L=1,2)
         WRITE(6,6060) E(I),E(I+1)
         WRITE(6,6050) ('EIGENVECTOR',L=1,2)
         WRITE(6,6070) AR(1,I),AI(1,I),AR(1,I+1),AI(1,I+1)
         WRITE(6,6070) AR(2,I),AI(2,I),AR(2,I+1),AI(2,I+1)
         WRITE(6,6070) AR(3,I),AI(3,I),AR(3,I+1),AI(3,I+1)
         WRITE(6,6070) AR(4,I),AI(4,I),AR(4,I+1),AI(4,I+1)
  140 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  HCGJAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ',I4, '    LNA = ',I4, '   LNB = ',I4,&
                '    NT = ',I4,/)
 6010 FORMAT(/,&
             1X,'     INPUT MATRIX A ( REAL , IMAGINARY )',/)
 6020 FORMAT(1X,5X,4('(',F5.1,',',F5.1,')'))
 6030 FORMAT(/,&
             1X,'     INPUT MATRIX B ( REAL , IMAGINARY )',/)
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6050 FORMAT(/,&
             1X,14X,A11,22X,A11)
 6060 FORMAT(1X,12X,1PD14.7,19X,1PD14.7)
 6070 FORMAT(1X,5X,F12.8,' ,',F12.8,7X,F12.8,' ,',F12.8)
 6080 FORMAT(1X,' RE(A) = ',D10.2,' IM(A) = ',D10.2,&
                ' RE(B) = ',D10.2,' IM(B) = ',D10.2,&
                ' I,J = ',I2,3X,I2)
      END
```

(d) Output results

```
***  HCGJAA  ***

 **  INPUT  **

     N =    4    LNA =    4   LNB =    4    NT =    2


     INPUT MATRIX A ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  1.0,  2.0)(  1.0, -2.0)( 10.0,  0.0)( -3.0,  0.0)
     ( -1.0,  2.0)( -1.0, -2.0)( -3.0,  0.0)( 11.0,  0.0)

     INPUT MATRIX B ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  1.0, -2.0)(  1.0,  2.0)( 10.0,  0.0)( -3.0,  0.0)
     ( -1.0, -2.0)( -1.0,  2.0)( -3.0,  0.0)( 11.0,  0.0)

 **  OUTPUT  **

     IERR =    0


              EIGENVALUE                      EIGENVALUE
            1.6653903D+01                   3.6956492D+01

             EIGENVECTOR                     EIGENVECTOR
      -0.39854861 ,  0.00000000       -0.10853896 , -0.00000000
       0.34532468 ,  0.00093582        0.10159604 ,  0.01599199
       0.00725931 , -0.13233869        0.01872983 , -0.32676633
      -0.00420696 , -0.12491356        0.01561597 , -0.28141982

              EIGENVALUE                      EIGENVALUE
            1.0637116D+02                   2.1801844D+02

             EIGENVECTOR                     EIGENVECTOR
       0.17035278 ,  0.00000000        0.09167780 ,  0.00000000
       0.20284574 ,  0.00833634        0.10125790 , -0.00403758
      -0.09968579 , -0.00601150        0.14657602 , -0.00401165
       0.12984924 , -0.00064446       -0.16601583 ,  0.00268875
```

5.9.2 HCGJAN, GCGJAN
All Eigenvalues of Hermitian Matrices
(Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{z} = \lambda \boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

CALL HCGJAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGJAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D / R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D / R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D / R}	N	Output	Eigenvalues λ
9	WORK	{D / R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA , LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.9.1 $\left\{\begin{matrix}\text{HCGJAA}\\\text{GCGJAA}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

(d) 5.10.2 $\left\{\begin{matrix}\text{HCGKAN}\\\text{GCGKAN}\end{matrix}\right\}$ should be used if matrix A is only positive.

5.10 GENERALIZED EIGENVALUE PROBLEM ($BAz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.10.1 HCGKAA, GCGKAA

All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$BA\boldsymbol{z} = \lambda\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

CALL HCGKAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGKAA (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D / R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D / R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D / R}	N	Output	Eigenvalues λ
9	WORK	{D / R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA , LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) and A(1, 1)← $\sqrt{\mathrm{B}(1,1)}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B^{-1} \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.10.2 $\begin{Bmatrix}\text{HCGKAN}\\ \text{GCGKAN}\end{Bmatrix}$ should be used if the eigenvectors are not needed.

(e) 5.9.1 $\begin{Bmatrix}\text{HCGJAA}\\ \text{GCGJAA}\end{Bmatrix}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem $BAx = \lambda x$.

(b) Input data

N=4, LNA=4, matrix A, LNB=4, NT=2 and matrix B.

(c) Main program

```
      PROGRAM UCGKAA
! *** EXAMPLE OF HCGKAA ***
      IMPLICIT NONE
!
      INTEGER N,LNA,LNB
      PARAMETER( N = 4, LNA = 4, LNB = 4 )
      INTEGER IERR,I,J,L,NT
      REAL(8) AR(LNA,N),BR(LNB,N),AI(LNA,N),BI(LNB,N)
      REAL(8) E(N),WORK(4*N)
      COMPLEX(8) KEEP(LNA,N)
!
```

```
      NT = 2
!
      KEEP(1,1)=( 8.D0, 0.D0)
      KEEP(2,2)=( 9.D0, 0.D0)
      KEEP(3,3)=( 10.D0, 0.D0)
      KEEP(4,4)=( 11.D0, 0.D0)
      KEEP(1,2)=( 3.D0, 0.D0)
      KEEP(1,3)=( 1.D0, 2.D0)
      KEEP(1,4)=(-1.D0, 2.D0)
      KEEP(2,3)=( 1.D0,-2.D0)
      KEEP(2,4)=(-1.D0,-2.D0)
      KEEP(3,4)=(-3.D0, 0.D0)
      WRITE(6,6000) N, LNA, LNB, NT
      DO 100 I=1,N
      DO 110 J=I,N
         BR(I,J)=DBLE (KEEP(I,J))
         BI(I,J)=DIMAG(KEEP(I,J))
         KEEP(J,I)=CONJG(KEEP(I,J))
         AR(I,J)=BR(I,J)
         AI(I,J)=-BI(I,J)
         AR(J,I)=AR(I,J)
         AI(J,I)=-AI(I,J)
         BR(J,I)=BR(I,J)
         BI(J,I)=-BI(I,J)
  110 CONTINUE
  100 CONTINUE
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) AR(I,1),AI(I,1),AR(I,2),AI(I,2),&
                       AR(I,3),AI(I,3),AR(I,4),AI(I,4)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 I=1,N
         WRITE(6,6020) BR(I,1),BI(I,1),BR(I,2),BI(I,2),&
                       BR(I,3),BI(I,3),BR(I,4),BI(I,4)
  130 CONTINUE
!
      CALL HCGKAA(AR,AI, LNA, N, BR,BI, LNB, E, WORK, NT, IERR)
!
      WRITE(6,6040) IERR
      DO 140 I=1,N,2
         WRITE(6,6050) (' EIGENVALUE',L=1,2)
         WRITE(6,6060) E(I),E(I+1)
         WRITE(6,6050) ('EIGENVECTOR',L=1,2)
         WRITE(6,6070) AR(1,I),AI(1,I),AR(1,I+1),AI(1,I+1)
         WRITE(6,6070) AR(2,I),AI(2,I),AR(2,I+1),AI(2,I+1)
         WRITE(6,6070) AR(3,I),AI(3,I),AR(3,I+1),AI(3,I+1)
         WRITE(6,6070) AR(4,I),AI(4,I),AR(4,I+1),AI(4,I+1)
  140 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  HCGKAA  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N = ',I4, '     LNA = ',I4, '   LNB = ',I4,&
                '    NT = ',I4,/)
 6010 FORMAT(/,&
             1X,'     INPUT MATRIX A ( REAL , IMAGINARY )',/)
 6020 FORMAT(1X,5X,4('(',F5.1,',',F5.1,')'))
 6030 FORMAT(/,&
             1X,'     INPUT MATRIX B ( REAL , IMAGINARY )',/)
 6040 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6050 FORMAT(/,&
             1X,14X,A11,22X,A11)
 6060 FORMAT(1X,12X,1PD14.7,19X,1PD14.7)
 6070 FORMAT(1X,5X,F12.8,' ,',F12.8,7X,F12.8,' ,',F12.8)
 6080 FORMAT(1X,' RE(A) = ',D10.2,' IM(A) = ',D10.2,&
               ' RE(B) = ',D10.2,' IM(B) = ',D10.2,&
                ' I,J = ',I2,3X,I2)
      END
```

(d) Output results

```
***  HCGKAA  ***

 **  INPUT  **

     N =    4    LNA =    4   LNB =    4    NT =    2


     INPUT MATRIX A ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  1.0,  2.0)(  1.0, -2.0)( 10.0,  0.0)( -3.0,  0.0)
     ( -1.0,  2.0)( -1.0, -2.0)( -3.0,  0.0)( 11.0,  0.0)

     INPUT MATRIX B ( REAL , IMAGINARY )

     (  8.0,  0.0)(  3.0,  0.0)(  1.0,  2.0)( -1.0,  2.0)
     (  3.0,  0.0)(  9.0,  0.0)(  1.0, -2.0)( -1.0, -2.0)
     (  1.0, -2.0)(  1.0,  2.0)( 10.0,  0.0)( -3.0,  0.0)
```

```
    ( -1.0, -2.0)( -1.0,  2.0)( -3.0,  0.0)( 11.0,  0.0)

**  OUTPUT  **

    IERR =    0

          EIGENVALUE                    EIGENVALUE
        1.6653903D+01                 3.6956492D+01

          EIGENVECTOR                   EIGENVECTOR
 -1.62644403 ,  0.00148705     0.65596257 ,  0.07132105
  1.40923809 , -0.00510746    -0.62451092 ,  0.02988983
  0.03011844 ,  0.54003620     0.10152348 , -1.98714181
 -0.01670217 ,  0.50977765     0.09054508 , -1.71104099

          EIGENVALUE                    EIGENVALUE
        1.0637116D+02                 2.1801844D+02

          EIGENVECTOR                   EIGENVECTOR
  1.75513631 ,  0.07996887     1.35243375 , -0.05769277
  2.09382301 ,  0.00933315     1.49630058 , -0.00415901
 -1.02987983 ,  0.01514061     2.16481859 , -0.03306016
  1.33752782 ,  0.06759506    -2.45076277 ,  0.06480904
```

5.10.2 HCGKAN, GCGKAN
All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$BA\boldsymbol{z} = \lambda\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

CALL HCGKAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

Single precision:

CALL GCGKAN (AR, AI, LNA, N, BR, BI, LNB, E, WORK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	AR	{D / R}	LNA , N	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	AI	{D / R}	LNA , N	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	LNA	I	1	Input	Adjustable dimension of arrays AR and AI
4	N	I	1	Input	Order of matrices A and B
5	BR	{D / R}	LNB , N	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	BI	{D / R}	LNB , N	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	LNB	I	1	Input	Adjustable dimension of arrays BR and BI
8	E	{D / R}	N	Output	Eigenvalues λ
9	WORK	{D / R}	4×N	Work	Work area
10	NT	I	1	Input	Number of tasks
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq$ N $\leq$ LNA , LNB

(b) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	E(1)← A(1, 1)*B(1, 1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays AR, AI, BR and BI should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.10.1 $\left\{\begin{matrix}\text{HCGKAA}\\\text{GCGKAA}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

(d) 5.9.2 $\left\{\begin{matrix}\text{HCGJAN}\\\text{GCGJAN}\end{matrix}\right\}$ should be used if matrix A is only positive.

Chapter 6

FOURIER TRANSFORMS AND THEIR APPLICATIONS

6.1 INTRODUCTION

This chapter describes subroutines that perform fast Fourier transforms, convolutions, correlations, power spectrum analysis.

Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

The following subroutines are provided for computing the discrete Fourier transform according to the data characteristics. You can perform processing efficiently if your data satisfies any of the characteristics described below.

(1) Multiple one-dimensional complex Fourier transform
Data values are complex numbers and are multiple one-dimensional data.

(2) Multiple one-dimensional real Fourier transform
Data values are real numbers and are multiple one-dimensional data.

(3) Two-dimensional complex Fourier transform
Data values are complex numbers and are two-dimensional data.

(4) Two-dimensional real Fourier transform
Data values are real numbers and are two-dimensional data.

(5) Three-dimensional complex Fourier transform
Data values are complex numbers and are three-dimensional data.

(6) Three-dimensional real Fourier transform
Data values are real numbers and are three-dimensional data.

In addition, although the number of equal divisions n of the input data can be transformed by the fast Fourier transforms handled in this chapter no matter what prime number is used as radix, calculation efficiency decreases for a sequence formed from a large prime number. Therefore, the number of equal divisions n should be able to be factored into small radices (2, 3, 5, 7).

The subroutines, which handles the following data, are provided for performing the convolutions, correlations, power spectrum analysis.

(1) Two-dimensional data

(2) Three-dimensional data

6.1.1 Notes

(1) For a two-dimensional complex Fourier transform or two-dimensional real Fourier transform, the number of sample points NX, NY (NX, NY, NZ for a three-dimensional complex Fourier transform or three-dimensional real Fourier transform) corresponds to a sample on a single period (0, 2π).

(2) You first must call a transform subroutine that includes initialization. This subroutine performs such tasks as creating a trigonometric function table. If you plan to continue computing Fourier transforms for the same number of data N, processing will be more efficient if you use the post-initialization transform subroutine thereafter since the contents of work areas are retained.

6.1.2 Algorithms Used

6.1.2.1 Two-dimensional complex Fourier transform

The complex Fourier forward transform C_{j_x,j_y} for one period c_{k_x,k_y} $(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$ of complex multiperiodic data $\hat{c}_{k_x,k_y}$ satisfying $\hat{c}_{k_x,k_y} = \hat{c}_{k_x+n_x,k_y+n_y}$ for arbitrary integers k_x and k_y is defined by the following equation.

$$\begin{aligned} C_{j_x,j_y} &= \frac{1}{\alpha}\sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\ &\quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1) \end{aligned}$$

α is an arbitrary constant for which 1 or $n_x n_y$ normally is selected. At this time, the complex data after the transform C_{j_x,j_y} $(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$ also corresponds to one period of complex multiperiodic data $\hat{C}_{j_x,j_y}$ satisfying $\hat{C}_{j_x,j_y} = \hat{C}_{j_x+n_x,j_y+n_y}$ for an arbitrary integers j_x and j_y. The corresponding backward transform is as follows:

$$\begin{aligned} c_{k_x,k_y} &= \frac{1}{n_x n_y}\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1} (\alpha C_{j_x,j_y}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1) \end{aligned}$$

The subroutines in this manual obtain $\alpha C_{j_x,j_y}$ and $n_x n_y c_{k_x,k_y}$ except for a constant factor, from the normal Fourier transform definition.

6.1.2.2 Two-dimensional real Fourier transform

When the data for which the two-dimensional Fourier forward transform is to be calculated is real, the following relationship is satisfied.

$$\begin{aligned} \alpha C^*_{n_x-j_x,n_y-j_y} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c^*_{k_x,k_y} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}\}} \\ &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}\}} \\ &= \alpha C_{j_x,j_y} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. In particular, $C_{0,0}$ is real and when n_x and n_y are even, then $C_{\frac{n_x}{2},\frac{n_y}{2}}$ is also real.
Similarly, the following relationship is satisfied.

$$\begin{aligned} \alpha C^*_{n_x-j_x,j_y} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c^*_{k_x,k_y} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{j_y k_y}{n_y}\}} \\ &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}\}} \\ &= \alpha C_{j_x,n_y-j_y} \end{aligned}$$

Therefore, the calculations for a Fourier transform can be executed using half the data for the case of general complex data (for c_{k_x,k_y}, only the real part, and for C_{j_x,j_y}, half of its period for either j_x or j_y).

6.1.2.3 Three-dimensional complex Fourier transform

The complex Fourier forward transform C_{j_x,j_y,j_z} for one period c_{k_x,k_y,k_z} ($k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1$) of complex multiperiodic data $\hat{c}_{k_x,k_y,k_z}$ satisfying $\hat{c}_{k_x,k_y,k_z} = \hat{c}_{k_x+n_x,k_y+n_y,k_z+n_z}$ for arbitrary integers k_x, k_y and k_z is defined by the following equation.

$$\begin{aligned} C_{j_x,j_y,j_z} &= \frac{1}{\alpha}\sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1) \end{aligned}$$

α is an arbitrary constant for which 1 or $n_x n_y n_z$ normally is selected. At this time, the complex data after the transform C_{j_x,j_y,j_z} ($j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1$) also corresponds to one period of complex multiperiodic data $\hat{C}_{j_x,j_y,j_z}$ satisfying $\hat{C}_{j_x,j_y,j_z} = \hat{C}_{j_x+n_x,j_y+n_y,j_z+n_z}$ for an arbitrary integers j_x, j_y and j_z. The corresponding backward transform is as follows:

$$\begin{aligned} c_{k_x,k_y,j_z} &= \frac{1}{n_x n_y n_z}\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} (\alpha C_{j_x,j_y,j_z}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1) \end{aligned}$$

The subroutines in this manual obtain $\alpha C_{j_x,j_y,j_z}$ and $n_x n_y n_z c_{k_x,k_y,k_z}$ except for a constant factor, from the normal Fourier transform definition.

6.1.2.4 Three-dimensional real Fourier transform

When the data for which the three-dimensional Fourier forward transform is to be calculated is real, the following relationship is satisfied.

$$\begin{aligned} \alpha C^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c^*_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}+\frac{(n_z-j_z)k_z}{n_z}\}} \\ &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z}\}} \\ &= \alpha C_{j_x,j_y,j_z} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. In particular, $C_{0,0,0}$ is real and when n_x, n_y and n_z are all even, then $C_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}$ is also real.
Similarly, the following kinds of relationships are satisfied.

$$C^*_{n_x-j_x,j_y,j_z} = C_{j_x,n_y-j_y,n_z-j_z}$$

That is, the complex number after substitutions are made in the subscripts using the following correspondence relationships for all of j_x, j_y and j_z is mutually related as a complex conjugate to the complex number before the substitutions are made.

$$\begin{aligned} j_x &\leftrightarrow n_x - j_x \\ j_y &\leftrightarrow n_y - j_y \\ j_z &\leftrightarrow n_z - j_z \end{aligned}$$

Therefore, the calculations for a Fourier transform can be executed using half the data for the case of general complex data (for c_{k_x,k_y,k_z}, only the real part, and for C_{j_x,j_y,j_z}, half of its period for either j_x, j_y, or j_z).

6.1.3 Reference Bibliography

(1) Pease, M. C, "An Adaption of the Fast Fourier Transform for Parallel Processing", J. Assn. Comput. Mach., 15, 252(1968).

(2) Stockham, T. G. , "High Speed Convolution and Correlation", AFIPS Conf. Proc. , 28, 229(1966).

(3) Swarztrauber, P. N. , "Vectorizing the FFTs", Parallel Computations, 51(1982).

(4) Singleton, R. C. , "An Algorithm for Computing the Mixed Radix Fast Fourier Transform", IEEE Trans. Audio and Electroacoust. , AU-17, 93(1969).

(5) Singleton, R. C. , "ALGOL Procedures for the Fast Fourier Transform", Commun. ACM, 11, 773(1968).

(6) Petersen, W. P. , "Vector Fortran for Numerical Problems on CRAY-1", Commun. ACM, 26, 1008(1983).

(7) Brigham, E. O. , "The Fast Fourier Transform", Prentice-Hall Inc. , 1974.

(8) Temperton, C. , "Implementation of a Self-Sorting In-Place Prime-Factor FFT Algorithm", J. Comp. Phys., 58, 283(1985).

(9) Temperton, C. , "Self-Sorting Mixed-Radix Fast Fourier Transform", J. Comp. Phys., 52, 1(1983).

(10) Temperton, C. , "Fast Mixed-Radix Real Fourier Transforms", J. Comp. Phys., 52, 340(1983)

(11) Willemstein, T. , "Two-dimensional Fourier Transforms on the Cray-1S", Supercomputer, 10(1985)

6.2 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.2.1 [DEPRECATED]QFCMFB, PFCMFB Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

QFCMFB or PFCMFB computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

QFCMFB or PFCMFB computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

CALL QFCMFB (N, M, CR, CI, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

Single precision:

CALL PFCMFB (N, M, CR, CI, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	CR	{D / R}	See Contents	Input	Real part of input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Real part of output data $d_{j,l}$ (See Notes (b) and (c))
4	CI	{D / R}	See Contents	Input	Imaginary part of input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Imaginary part of output data $d_{j,l}$ (See Notes (b) and (c))
5	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
6	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
7	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
8	IFAX	I	20	Output	Factorization results and number of factors (See Note (d))
9	TRIGS	{D / R}	2 × N	Output	Trigonometric function table (See Note (d))
10	WK	{D / R}	2 × M × N	Work	Work area
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 1, M $\geq$ 1

(b) INCN $\geq$ 1, INCM $\geq$ 1

(c) INCN $\geq$ M $\times$ gcm (INCN, INCM) or
INCM $\geq$ N $\times$ gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW $\in \{0, 1, -1\}$

(e) NT $\geq$ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of transformed data N can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting N = 289(=17^2), it is usually more efficient to set N = 300(=$2^2 \times 3 \times 5^2$), N = 320(=$2^6 \times 5$), N = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data $c_{k,l}$ ($k = 0, \cdots, n-1;\ l = 1, \cdots, m$) be $\Re\{c_{k,l}\}$ and $\Im\{c_{k,l}\}$, respectively, the $c_{k,l}$ and elements of arrays CR and CI are associated as follows.

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CR}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$
$$\Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CI}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CR}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1)), \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CI}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CR}(\mathrm{l} + \mathrm{m} * \mathrm{k}), \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{CI}(\mathrm{l} + \mathrm{m} * \mathrm{k})$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l} (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of arrays CR and CI is not stored do not change when this subroutine is called.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform

immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data N, you should call this subroutine once, and then use the after-initialization transform 6.2.2 $\begin{Bmatrix}\text{QFCMBF}\\ \text{PFCMBF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.2.2 $\begin{Bmatrix}\text{QFCMBF}\\ \text{PFCMBF}\end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for arrays CR and CI.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.2.2 (7).

6.2.2 [DEPRECATED]QFCMBF, PFCMBF Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

QFCMBF or PFCMBF computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

QFCMBF or PFCMBF computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

CALL QFCMBF (N, M, CR, CI, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

Single precision:

CALL PFCMBF (N, M, CR, CI, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	CR	{D / R}	See Contents	Input	Real part of input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Real part of output data $d_{j,l}$ (See Notes (b) and (c))
4	CI	{D / R}	See Contents	Input	Imaginary part of input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Imaginary part of output data $d_{j,l}$ (See Notes (b) and (c))
5	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
6	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
7	ISW	I	1	Input	Processing switch ISW= 1:Forward transform ISW=−1:Backward transform
8	IFAX	I	20	Input	Factorization results and number of factors (See Note (a))
9	TRIGS	{D / R}	2 × N	Input	Trigonometric function table (See Note (a))
10	WK	{D / R}	2×M×N	Work	Work area
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 1, M ≥ 1

(b) INCN ≥ 1, INCM ≥ 1

(c) INCN ≥ M × gcm (INCN, INCM) or
INCM ≥ N × gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW ∈ {1, −1}

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of transformed data N after the including-initialization subroutine 6.2.1 $\left\{\begin{matrix}\text{QFCMFB}\\\text{PFCMFB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) If we let the real and imaginary parts of the complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be $\Re\{c_{k,l}\}$ and $\Im\{c_{k,l}\}$, respectively, the $c_{k,l}$ and elements of arrays CR and CI are associated as follows.

$$\begin{aligned}\Re\{c_{k,l}\} &\leftrightarrow \text{CR}(1 + \text{INCN} * \text{k} + \text{INCM} * (\text{l} - 1))\\ \Im\{c_{k,l}\} &\leftrightarrow \text{CI}(1 + \text{INCN} * \text{k} + \text{INCM} * (\text{l} - 1))\end{aligned}$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \text{CR}((\text{k} + 1) + \text{n} * (\text{l} - 1)),\quad \Im\{c_{k,l}\} \leftrightarrow \text{CI}((\text{k} + 1) + \text{n} * (\text{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \text{CR}(\text{l} + \text{m} * \text{k}),\quad \Im\{c_{k,l}\} \leftrightarrow \text{CI}(\text{l} + \text{m} * \text{k})$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of arrays CR and CI is not stored do not change when this subroutine is called.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l}\quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that

is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional complex Fourier transform using the following sequence of numbers as input data.

CR(1)= 1.000 CI(1)=4.000
CR(2)= 2.000 CI(2)=3.000
CR(3)= 3.000 CI(3)=2.000
CR(4)= 4.000 CI(4)=1.000
CR(5)= 4.000 CI(5)=1.000
CR(6)= 3.000 CI(6)=2.000
CR(7)= 2.000 CI(7)=3.000
CR(8)= 1.000 CI(8)=4.000
CR(10)= 1.000 CI(10)=2.000
CR(11)= 1.000 CI(11)=2.000
CR(12)= 2.000 CI(12)=1.000
CR(13)= 2.000 CI(13)=1.000
CR(14)= 2.000 CI(14)=1.000
CR(15)= 2.000 CI(15)=1.000
CR(16)= 1.000 CI(16)=2.000
CR(17)= 1.000 CI(17)=2.000
CR(19)= 1.000 CI(19)=2.000
CR(20)= 1.000 CI(20)=2.000
CR(21)= 1.000 CI(21)=2.000
CR(22)= 1.000 CI(22)=2.000
CR(23)= 2.000 CI(23)=1.000
CR(24)= 2.000 CI(24)=1.000
CR(25)= 2.000 CI(25)=1.000
CR(26)= 2.000 CI(26)=1.000
CR(28)= 1.000 CI(28)=1.000
CR(29)= 1.000 CI(29)=1.000
CR(30)= 1.000 CI(30)=1.000
CR(31)= 1.000 CI(31)=1.000
CR(32)= 1.000 CI(32)=1.000
CR(33)= 1.000 CI(33)=1.000
CR(34)= 1.000 CI(34)=1.000
CR(35)= 1.000 CI(35)=1.000

(b) Input data

Array CR and CI, N=8, M=4, INCN=1, INCM=9, ISW=1 (forward transform), ISW=−1 (backward

transform) and NT=2.

(c) Main program

```
      PROGRAM OFCMBF
! *** EXAMPLE OF QFCMFB AND QFCMBF ***
      IMPLICIT  REAL(8)(A-H,O-Z)
      PARAMETER (NN=9,MM=4)
      DIMENSION CR(NN*MM),CI(NN*MM),TRIGS(2*NN),WK(2*NN*MM)
      DIMENSION IFAX(20)
      COMMON CR,CI,TRIGS,WK,IFAX
!**** INPUT ****
      READ(5,*) N,M,INCN,INCM,NT
      WRITE(6,1000) N,M,INCN,INCM,NT
      DO 20 J=1,M
        DO 10 I=1,N
          READ(5,*) CR(1+(I-1)*INCN+(J-1)*INCM),&
                    CI(1+(I-1)*INCN+(J-1)*INCM)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((CR(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((CI(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
   30 CONTINUE
!**** OUTPUT ****
      WRITE(6,1010)
!**** FORWARD TRANSFORM ****
      ISW = 1
      CALL QFCMFB(N,M,CR,CI,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 50 J=1,M
        DO 40 I=1,N
          CR(1+(I-1)*INCN+(J-1)*INCM)=&
            CR(1+(I-1)*INCN+(J-1)*INCM)/DBLE(N)
          CI(1+(I-1)*INCN+(J-1)*INCM)=&
            CI(1+(I-1)*INCN+(J-1)*INCM)/DBLE(N)
   40   CONTINUE
   50 CONTINUE
      WRITE(6,1020) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((CR(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((CI(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
!**** BACKWARD TRANSFORM ****
      ISW = -1
      CALL QFCMBF(N,M,CR,CI,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1030) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((CR(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((CI(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFCMFB AND QFCMBF ***',/,/,&
             1X,' ** INPUT **',/,/,&
             1X,'       N = ',I3,'       M = ',I3,/,&
             1X,'    INCN = ',I3,'   INCM = ',I3,/,&
             1X,'      NT = ',I3,/)
 1010 FORMAT(1X,/,&
             1X,' ** OUTPUT **',/)
 1020 FORMAT(1X,'  ( FORWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 1030 FORMAT(1X,'  ( BACKWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 2000 FORMAT(1X,4X,A)
 2010 FORMAT(1X,4X,8F8.4,/)
      END
```

(d) Output results

```
*** QFCMFB AND QFCMBF ***

 ** INPUT **

       N =   8      M =   4
    INCN =   1   INCM =   9
      NT =   2

    REAL PART
      1.0000  2.0000  3.0000  4.0000  4.0000  3.0000  2.0000  1.0000

      1.0000  1.0000  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000

      1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000

      1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000

    IMAGINARY PART
      4.0000  3.0000  2.0000  1.0000  1.0000  2.0000  3.0000  4.0000

      2.0000  2.0000  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000

      2.0000  2.0000  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000
```

```
     1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000

** OUTPUT **
 ( FORWARD TRANSFORM )
   IERR =    0
   REAL PART
     2.5000 -1.0303  0.0000 -0.0732  0.0000  0.0303  0.0000 -0.4268
     1.5000 -0.4268  0.0000  0.1768  0.0000 -0.0732  0.0000 -0.1768
     1.5000  0.1768  0.0000 -0.0732  0.0000 -0.1768  0.0000 -0.4268
     1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   IMAGINARY PART
     2.5000  0.4268  0.0000 -0.0303  0.0000  0.0732  0.0000  1.0303
     1.5000  0.1768  0.0000  0.0732  0.0000 -0.1768  0.0000  0.4268
     1.5000  0.4268  0.0000  0.1768  0.0000  0.0732  0.0000 -0.1768
     1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 ( BACKWARD TRANSFORM )
   IERR =    0
   REAL PART
     1.0000  2.0000  3.0000  4.0000  4.0000  3.0000  2.0000  1.0000
     1.0000  1.0000  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000
     1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000
     1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
   IMAGINARY PART
     4.0000  3.0000  2.0000  1.0000  1.0000  2.0000  3.0000  4.0000
     2.0000  2.0000  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000
     2.0000  2.0000  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000
     1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

6.3 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.3.1 [DEPRECATED]HFCMFB, GFCMFB
Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

HFCMFB or GFCMFB computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

HFCMFB or GFCMFB computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

```
CALL HFCMFB  (N, M, C, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL GFCMFB  (N, M, C, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	C	{Z / C}	See Contents	Input	Input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Output data $d_{j,l}$ (See Notes (b) and (c))
4	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	20	Output	Factorization results and number of factors (See Note (d))
8	TRIGS	{D / R}	2 × N	Output	Trigonometric function table (See Note (d))
9	WK	{Z / C}	M×N	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 1, M ≥ 1

(b) INCN ≥ 1, INCM ≥ 1

(c) INCN ≥ M × gcm (INCN, INCM) or
INCM ≥ N × gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW ∈ {0, 1, −1}

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of transformed data N can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting N = 289(=17^2), it is usually more efficient to set N = 300(=$2^2 \times 3 \times 5^2$), N = 320(=$2^6 \times 5$), N = 384(=$2^7 \times 3$) or the like.

(b) The complex data $c_{k,l}$ ($k = 0, \cdots, n-1;\ l = 1, \cdots, m$) and elements of array C are associated as follows.

$$c_{k,l} \leftrightarrow \mathrm{C}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{C}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{C}(\mathrm{l} + \mathrm{m} * \mathrm{k})$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of array C is not stored do not change when this subroutine is called.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = n c_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data N, you should call this subroutine once, and then use the after-initialization transform 6.3.2 $\begin{Bmatrix}\mathrm{HFCMBF}\\ \mathrm{GFCMBF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays

IFAX and TRIGS so they can be used as input to the subroutine 6.3.2 $\left\{\begin{matrix}\text{HFCMBF}\\ \text{GFCMBF}\end{matrix}\right\}$.
To perform initialization only by setting ISW=0, you need not set input data for array C.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.3.2 (7).

6.3.2 [DEPRECATED]HFCMBF, GFCMBF Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

HFCMBF or GFCMBF computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

HFCMBF or GFCMBF computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

CALL HFCMBF (N, M, C, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

Single precision:

CALL GFCMBF (N, M, C, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	C	{Z / C}	See Contents	Input	Input data $c_{k,l}$ (See Note (b)) **Size:** INCN × (N − 1) + INCM × (M − 1) + 1
				Output	Output data $d_{j,l}$ (See Notes (b) and (c))
4	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	ISW	I	1	Input	Processing switch ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	20	Input	Factorization results and number of factors (See Note (a))
8	TRIGS	{D / R}	2 × N	Input	Trigonometric function table (See Note (a))
9	WK	{Z / C}	M×N	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 1, M ≥ 1

(b) INCN ≥ 1, INCM ≥ 1

(c) INCN ≥ M × gcm (INCN, INCM) or
INCM ≥ N × gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW ∈ {1, −1}

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of transformed data N after the including-initialization subroutine 6.3.1 $\left\{\begin{matrix}\text{HFCMFB}\\\text{GFCMFB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array C are associated as follows.

$$c_{k,l} \leftrightarrow \mathrm{C}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{C}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{C}(\mathrm{l} + \mathrm{m} * \mathrm{k})$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of array C is not stored do not change when this subroutine is called.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that

is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional complex Fourier transform using the following sequence of numbers as input data.

C(1)= (1.000, 4.000)
C(2)= (2.000, 3.000)
C(3)= (3.000, 2.000)
C(4)= (4.000, 1.000)
C(5)= (4.000, 1.000)
C(6)= (3.000, 2.000)
C(7)= (2.000, 3.000)
C(8)= (1.000, 4.000)
C(10)= (1.000, 2.000)
C(11)= (1.000, 2.000)
C(12)= (2.000, 1.000)
C(13)= (2.000, 1.000)
C(14)= (2.000, 1.000)
C(15)= (2.000, 1.000)
C(16)= (1.000, 2.000)
C(17)= (1.000, 2.000)
C(19)= (1.000, 2.000)
C(20)= (1.000, 2.000)
C(21)= (1.000, 2.000)
C(22)= (1.000, 2.000)
C(23)= (2.000, 1.000)
C(24)= (2.000, 1.000)
C(25)= (2.000, 1.000)
C(26)= (2.000, 1.000)
C(28)= (1.000, 1.000)
C(29)= (1.000, 1.000)
C(30)= (1.000, 1.000)
C(31)= (1.000, 1.000)
C(32)= (1.000, 1.000)
C(33)= (1.000, 1.000)
C(34)= (1.000, 1.000)
C(35)= (1.000, 1.000)

(b) Input data

Array C, N=8, M=4, INCN=1, INCM=9, ISW=1 (forward transform), ISW=−1 (backward transform)

and NT=2.

(c) Main program

```
      PROGRAM UFCMBF
! *** EXAMPLE OF HFCMFB AND HFCMBF ***
      IMPLICIT  REAL(8)(A-H,O-Z)
      PARAMETER (NN=9,MM=4)
      COMPLEX(8) C(NN*MM),WK(NN*MM)
      DIMENSION TRIGS(2*NN)
      DIMENSION IFAX(20)
      COMMON C,WK,TRIGS,IFAX
!**** INPUT ****
      READ(5,*) N,M,INCN,INCM,NT
      WRITE(6,1000) N,M,INCN,INCM,NT
      DO 20 J=1,M
        DO 10 I=1,N
          READ(5,*) C(1+(I-1)*INCN+(J-1)*INCM)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((DBLE(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((DIMAG(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
   30 CONTINUE
!**** OUTPUT ****
      WRITE(6,1010)
!**** FORWARD TRANSFORM ****
      ISW = 1
      CALL HFCMFB(N,M,C,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 50 J=1,M
        DO 40 I=1,N
          C(1+(I-1)*INCN+(J-1)*INCM)=&
            C(1+(I-1)*INCN+(J-1)*INCM)/DBLE(N)
   40   CONTINUE
   50 CONTINUE
      WRITE(6,1020) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((DBLE(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((DIMAG(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
!**** BACKWARD TRANSFORM ****
      ISW = -1
      CALL HFCMBF(N,M,C,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1030) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((DBLE(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2010) ((DIMAG(C(1+(I-1)*INCN+(J-1)*INCM)),I=1,N),J=1,M)
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** HFCMFB AND HFCMBF ***',/,/,&
             1X,' ** INPUT **',/,/,&
             1X,'       N = ',I3,'      M = ',I3,/,&
             1X,'    INCN = ',I3,'   INCM = ',I3,/,&
             1X,'      NT = ',I3,/)
 1010 FORMAT(1X,/,&
             1X,' ** OUTPUT **',/)
 1020 FORMAT(1X,'  ( FORWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 1030 FORMAT(1X,'  ( BACKWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 2000 FORMAT(1X,4X,A)
 2010 FORMAT(1X,4X,8F8.4,/)
      END
```

(d) Output results

```
*** HFCMFB AND HFCMBF ***

 ** INPUT **

       N =   8      M =   4
    INCN =   1   INCM =   9
      NT =   2

    REAL PART
      1.0000  2.0000  3.0000  4.0000  4.0000  3.0000  2.0000  1.0000

      1.0000  1.0000  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000

      1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000

      1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000

    IMAGINARY PART
      4.0000  3.0000  2.0000  1.0000  1.0000  2.0000  3.0000  4.0000

      2.0000  2.0000  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000

      2.0000  2.0000  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000

      1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

```
** OUTPUT **
 ( FORWARD TRANSFORM )
   IERR =    0
   REAL PART
     2.5000 -1.0303  0.0000 -0.0732  0.0000  0.0303  0.0000 -0.4268
     1.5000 -0.4268  0.0000  0.1768  0.0000 -0.0732  0.0000 -0.1768
     1.5000  0.1768  0.0000 -0.0732  0.0000 -0.1768  0.0000 -0.4268
     1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   IMAGINARY PART
     2.5000  0.4268  0.0000 -0.0303  0.0000  0.0732  0.0000  1.0303
     1.5000  0.1768  0.0000  0.0732  0.0000 -0.1768  0.0000  0.4268
     1.5000  0.4268  0.0000  0.1768  0.0000  0.0732  0.0000 -0.1768
     1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 ( BACKWARD TRANSFORM )
   IERR =    0
   REAL PART
     1.0000  2.0000  3.0000  4.0000  4.0000  3.0000  2.0000  1.0000
     1.0000  1.0000  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000
     1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000
     1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
   IMAGINARY PART
     4.0000  3.0000  2.0000  1.0000  1.0000  2.0000  3.0000  4.0000
     2.0000  2.0000  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000
     2.0000  2.0000  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000
     1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

6.4 MULTIPLE ONE-DIMENSIONAL REAL FOURIER TRANSFORM

6.4.1 [DEPRECATED]QFRMFB, PFRMFB Multiple One-Dimensional Real Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

QFRMFB or PFRMFB obtains a half period of the m-fold one-dimensional Fourier forward transform (arbitrary radix) for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$c_{j,l} = \sum_{k=0}^{n-1} r_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j,l} = c_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j,l}(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ for n complex data groups $c_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ satisfying $c^*_{n-j,l} = c_{j,l}$, QFRMFB or PFRMFB obtains the m-fold one-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k,l} &= \sum_{j=0}^{n-1} c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[\Re\{c_{j,l}\} \cos(2\pi\frac{jk}{n}) - \Im\{c_{j,l}\} \sin(2\pi\frac{jk}{n}) \right] \\ &\quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ and $\Im\{z\}$ represent the real and imaginary parts of the complex number z, respectively. Also, when n is odd, $\hat{c}_{\frac{n}{2},l} = 0$, and when n is even, $\hat{c}_{\frac{n}{2},l} = c_{\frac{n}{2},l}$.

(2) **Usage**

Double precision:

```
CALL QFRMFB  (N, M, R, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFRMFB  (N, M, R, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	R	{D / R}	See Contents	Input	Input data $r_{k,l}$ (Forward transform) or $c_{j,l}$ (Backward transform) (See Note (b)). **Size:** INCN × (N) + INCM × (M − 1) + 1, where N is an odd, or INCN × (N + 1) + INCM × (M − 1) + 1, where N is an even.
				Output	Output results $c_{j,l}$ (Forward transform), or $r_{k,l}$ (Backward transform) (See Notes (b) and (c))
4	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	20	Output	Factorization results and number of factors (See Note (d))
8	TRIGS	{D / R}	N	Output	Trigonometric function table (See Note (d))
9	WK	{D / R}	See Contents	Work	Work area **Size:** (N+1)×M, where N is an odd, or (N+2)×M, where N is an even.
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1, M≥1

(b) INCN ≥ 1, INCM ≥ 1

(c) INCN ≥ M × gcm (INCN, INCM) or:
In the case where N is an odd:
INCM ≥ (N+1) × gcm (INCN, INCM)
or if N is an even:
INCM ≥ (N+2) × gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW ∈ {0, 1, −1}

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Input-time contents are output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of data N can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting N = 289(=17^2), it is usually more efficient to set N = 300(=$2^2 \times 3 \times 5^2$), N = 320(=$2^6 \times 5$), N = 384(=$2^7 \times 3$) or the like.

(b) The real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array R are associated as follows.

$$r_{k,l} \leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{R}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{R}(\mathrm{l} + \mathrm{m} * \mathrm{k})$$

and the data is stored so that it is packed consecutively for subscript l. When computing the backward transform, if N(=n) is odd, then R(1 + INCN ∗ N + INCM ∗ (l − 1)) = 0, and when N is even, then R(1 + INCN ∗ N + INCM ∗ (l − 1)) = R(1 + INCN ∗ (N + 1) + INCM ∗ (l − 1)) = 0. If we let the real and imaginary parts of the complex data $c_{j,l}$ $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ be $\Re\{c_{j,l}\}$ and

$\Im\{c_{j,l}\}$, respectively, the $c_{j,l}$ and elements of array R are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned} \Re\{c_{j,l}\} &\leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * (2\mathrm{j}) + \mathrm{INCM} * (\mathrm{l} - 1)) \\ \Im\{c_{j,l}\} &\leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * (2\mathrm{j} + 1) + \mathrm{INCM} * (\mathrm{l} - 1)) \end{aligned}$$

From the properties of a real Fourier transform, when N is odd, $\Im\{c_{0,l}\} = 0$, and when N is even, $\Im\{c_{0,l}\} = \Im\{c_{\frac{n}{2},l}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements $c_{j,l}$ $(j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ can be obtained according to the following relationship from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c_{n-j,l} = c^*_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{r}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{r}_{k,l} = n r_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data N, you should call this subroutine once, and then use the after-initialization transform 6.4.2 $\begin{Bmatrix} \mathrm{QFRMBF} \\ \mathrm{PFRMBF} \end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.4.2 $\begin{Bmatrix} \mathrm{QFRMBF} \\ \mathrm{PFRMBF} \end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for array R.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.4.2 (7).

6.4.2 [DEPRECATED]QFRMBF, PFRMBF Multiple One-Dimensional Real Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

QFRMBF or PFRMBF obtains a half period of the m-fold one-dimensional Fourier forward transform (arbitrary radix) for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$c_{j,l} = \sum_{k=0}^{n-1} r_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j,l} = c_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j,l}(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ for n complex data groups $c_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ satisfying $c^*_{n-j,l} = c_{j,l}$, QFRMBF or PFRMBF obtains the m-fold one-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k,l} &= \sum_{j=0}^{n-1} c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[\Re\{c_{j,l}\} \cos(2\pi\frac{jk}{n}) - \Im\{c_{j,l}\} \sin(2\pi\frac{jk}{n}) \right] \\ &\quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ and $\Im\{z\}$ represent the real and imaginary parts of the complex number z, respectively. Also, when n is odd, $\hat{c}_{\frac{n}{2},l} = 0$, and when n is even, $\hat{c}_{\frac{n}{2},l} = c_{\frac{n}{2},l}$.

(2) **Usage**

Double precision:

CALL QFRMBF (N, M, R, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

Single precision:

CALL PFRMBF (N, M, R, INCN, INCM, ISW, IFAX, TRIGS, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	N	I	1	Input	Number of transformed data values n (See Note (a))
2	M	I	1	Input	Multiplicity m
3	R	{D / R}	See Contents	Input	Input data $r_{k,l}$ (Forward transform) or $c_{j,l}$ (Backward transform) (See Note (b)). **Size:** INCN × (N) + INCM × (M − 1) + 1, where N is an odd, or INCN × (N + 1) + INCM × (M − 1) + 1, where N is an even.
				Output	Output results $c_{j,l}$ (Forward transform), or $r_{k,l}$ (Backward transform) (See Notes (b) and (c))
4	INCN	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	INCM	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	ISW	I	1	Input	Processing switch ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	20	Input	Factorization results and number of factors (See Note (a))
8	TRIGS	{D / R}	N	Input	Trigonometric function table (See Note (a))
9	WK	{D / R}	See Contents	Work	Work area **Size:** (N+1)×M, where N is an odd, or (N+2)×M, where N is an even.
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1, M≥1

(b) INCN ≥ 1, INCM ≥ 1

(c) INCN ≥ M × gcm (INCN, INCM) or
In the case where N is an odd:
INCM ≥ (N+1) × gcm (INCN, INCM)
or if N is an even:
INCM ≥ (N+2) × gcm (INCN, INCM)
where, gcm(i, j) is the greatest common measure between i and j.

(d) ISW ∈ {1, −1}

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	N was equal to 1.	Input-time contents are output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of transformed data N after the including-initialization subroutine 6.4.1 $\left\{\begin{matrix}\text{QFRMFB}\\ \text{PFRMFB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array R are associated as follows.

$$r_{k,l} \leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * \mathrm{k} + \mathrm{INCM} * (\mathrm{l} - 1))$$

For example, if we let INCN=1 and INCM=n, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{R}((\mathrm{k} + 1) + \mathrm{n} * (\mathrm{l} - 1))$$

and the data is stored so that it is packed consecutively for subscript k. If we let INCN=m and INCM=1, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{R}(\mathrm{l} + \mathrm{m} * \mathrm{k})$$

and the data is stored so that it is packed consecutively for subscript l. When computing the backward transform, if N(=n) is odd, then $\mathrm{R}(1 + \mathrm{INCN} * \mathrm{N} + \mathrm{INCM} * (\mathrm{l} - 1)) = 0$, and when N is even, then $\mathrm{R}(1 + \mathrm{INCN} * \mathrm{N} + \mathrm{INCM} * (\mathrm{l} - 1)) = \mathrm{R}(1 + \mathrm{INCN} * (\mathrm{N} + 1) + \mathrm{INCM} * (\mathrm{l} - 1)) = 0$. If we let the real and imaginary parts of the complex data $c_{j,l}$ $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ be $\Re\{c_{j,l}\}$ and

$\Im\{c_{j,l}\}$, respectively, the $c_{j,l}$ and elements of array R are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned} \Re\{c_{j,l}\} &\leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * (2\mathrm{j}) + \mathrm{INCM} * (\mathrm{l} - 1)) \\ \Im\{c_{j,l}\} &\leftrightarrow \mathrm{R}(1 + \mathrm{INCN} * (2\mathrm{j} + 1) + \mathrm{INCM} * (\mathrm{l} - 1)) \end{aligned}$$

From the properties of a real Fourier transform, when N is odd, $\Im\{c_{0,l}\} = 0$, and when N is even, $\Im\{c_{0,l}\} = \Im\{c_{\frac{n}{2},l}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements $c_{j,l}$ $(j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ can be obtained according to the following relationship from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c_{n-j,l} = c_{j,l}^*$$

Here, z^* represents the conjugate complex number of the complex number z.

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{r}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{r}_{k,l} = n r_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional real Fourier forward and backward transforms using the following sequence of numbers as input data.

R(1)= 1.000
R(2)= 2.000
R(3)= 3.000
R(4)= 4.000
R(5)= 5.000
R(6)= 6.000
R(7)= 7.000
R(8)= 8.000
R(13)= 1.000
R(14)= 1.000
R(15)= 2.000
R(16)= 2.000
R(17)= 3.000
R(18)= 3.000
R(19)= 4.000
R(20)= 4.000
R(25)= 1.000
R(26)= 1.000
R(27)= 1.000
R(28)= 1.000
R(29)= 2.000
R(30)= 2.000
R(31)= 2.000
R(32)= 2.000
R(37)= 1.000
R(38)= 1.000
R(39)= 1.000
R(40)= 1.000
R(41)= 1.000
R(42)= 1.000
R(43)= 1.000
R(44)= 1.000

(b) Input data

Array R, N=8, M=4, INCN=1, INCM=12, ISW=1(Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM OFRMBF
! *** EXAMPLE OF QFRMFB AND QFRMBF ***
      IMPLICIT  REAL(8)(A-H,O-Z)
      PARAMETER (NN=12,MM=4)
      DIMENSION R(NN*MM),TRIGS(NN),WK(NN*MM)
      DIMENSION IFAX(20)
      COMMON R,TRIGS,WK,IFAX
!**** INPUT ****
      READ(5,*) N,M,INCN,INCM,NT
```

```
      WRITE(6,1000) N,M,INCN,INCM,NT
      DO 20 J=1,M
        DO 10 I=1,N
          READ(5,*) R(1+(I-1)*INCN+(J-1)*INCM)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((R(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
   30 CONTINUE
!**** OUTPUT ****
      WRITE(6,1010)
!**** FORWARD TRANSFORM ****
      ISW = 1
      CALL QFRMFB(N,M,R,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 50 J=1,M
        DO 40 I=1,N+2
          R(1+(I-1)*INCN+(J-1)*INCM)=&
            R(1+(I-1)*INCN+(J-1)*INCM)/DBLE(N)
   40   CONTINUE
   50 CONTINUE
      WRITE(6,1020) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2020) ((R(1+(2*I-2)*INCN+(J-1)*INCM),&
                     I=1,(N+2)/2),J=1,M)
      WRITE(6,2000) 'IMAGINARY PART'
      WRITE(6,2020) ((R(1+(2*I-1)*INCN+(J-1)*INCM),&
                     I=1,(N+2)/2),J=1,M)
!**** BACKWARD TRANSFORM ****
      ISW = -1
      CALL QFRMBF(N,M,R,INCN,INCM,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1030) IERR
      WRITE(6,2000) 'REAL PART'
      WRITE(6,2010) ((R(1+(I-1)*INCN+(J-1)*INCM),I=1,N),J=1,M)
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFRMFB AND QFRMBF ***',/,/,&
             1X,' ** INPUT **',/,/,&
             1X,'       N = ',I3,'      M = ',I3,/,&
             1X,'    INCN = ',I3,'   INCM = ',I3,/,&
             1X,'      NT = ',I3,/)
 1010 FORMAT(1X,/,&
             1X,' ** OUTPUT **',/)
 1020 FORMAT(1X,'  ( FORWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 1030 FORMAT(1X,'  ( BACKWARD TRANSFORM )',/,/,&
             1X,'    IERR = ', I4,/)
 2000 FORMAT(1X,4X,A)
 2010 FORMAT(1X,4X,8F8.4,/)
 2020 FORMAT(1X,4X,5F8.4,/)
      END
```

(d) Output results

```
*** QFRMFB AND QFRMBF ***

 ** INPUT **

       N =   8      M =   4
    INCN =   1   INCM =  12
      NT =   2

    REAL PART
      1.0000  2.0000  3.0000  4.0000  5.0000  6.0000  7.0000  8.0000

      1.0000  1.0000  2.0000  2.0000  3.0000  3.0000  4.0000  4.0000

      1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000

      1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000


 ** OUTPUT **

  ( FORWARD TRANSFORM )

    IERR =    0

    REAL PART
      4.5000 -0.5000 -0.5000 -0.5000 -0.5000

      2.5000 -0.2500 -0.2500 -0.2500  0.0000

      1.5000 -0.1250  0.0000 -0.1250  0.0000

      1.0000  0.0000  0.0000  0.0000  0.0000

    IMAGINARY PART
      0.0000  1.2071  0.5000  0.2071  0.0000

      0.0000  0.6036  0.2500  0.1036  0.0000

      0.0000  0.3018  0.0000  0.0518  0.0000
```

```
    0.0000 -0.0000  0.0000  0.0000  0.0000

( BACKWARD TRANSFORM )

  IERR =    0

  REAL PART
    1.0000  2.0000  3.0000  4.0000  5.0000  6.0000  7.0000  8.0000

    1.0000  1.0000  2.0000  2.0000  3.0000  3.0000  4.0000  4.0000

    1.0000  1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000

    1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

6.5 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.5.1 [DEPRECATED]QFC2FB, PFC2FB Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

QFC2FB or PFC2FB computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

QFC2FB or PFC2FB computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:

```
CALL QFC2FB  (NX, NY, CR, CI, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFC2FB  (NX, NY, CR, CI, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	CR	{D / R}	LX, LY	Input	Real part of input data c_{k_x,k_y} (See Note (b))
				Output	Real part of output data d_{j_x,j_y} (See Notes (b) and (c))
4	CI	{D / R}	LX, LY	Input	Imaginary part of input data c_{k_x,k_y} (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y} (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array CR and CI (See Note (b))
6	LY	I	1	Input	Second dimension of array CR and CI (See Note (b))
7	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
8	IFAX	I	40	Output	Factorization results and number of factors (See Note (d))
9	TRIGS	{D / R}	2 × (NX + NY)	Output	Trigonometric function table (See Note (d))
10	WK	{D / R}	2 × LX × LY	Work	Work area
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) NX≤LX, NY≤LY

(c) ISW $\in \{0, 1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data NX or NY can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300 (=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y} ($k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1$) be $\Re\{c_{k_x,k_y}\}$ and $\Im\{c_{k_x,k_y}\}$, respectively, the c_{k_x,k_y} and elements of arrays CR and CI are associated as follows.

$$\Re\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{CR}(\mathrm{k_x}+1, \mathrm{k_y}+1)$$
$$\Im\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{CI}(\mathrm{k_x}+1, \mathrm{k_y}+1)$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions LX **and** LY **of arrays** CR **and** CI **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is even,** LX=NX+1 **is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY), you should call this subroutine once, and then use the after-initialization transform 6.5.2 $\left\{\begin{matrix}\mathrm{QFC2BF}\\ \mathrm{PFC2BF}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.5.2 $\left\{\begin{matrix}\mathrm{QFC2BF}\\ \mathrm{PFC2BF}\end{matrix}\right\}$.
To perform initialization only by setting ISW=0, you need not set input data for arrays CR and CI.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period,

the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.5.2 (7).

6.5.2 [DEPRECATED]QFC2BF, PFC2BF Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

QFC2BF or PFC2BF computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

QFC2BF or PFC2BF computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:

```
CALL QFC2BF  (NX, NY, CR, CI, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFC2BF  (NX, NY, CR, CI, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	CR	{D / R}	LX, LY	Input	Real part of input data c_{k_x,k_y} (See Note (b))
				Output	Real part of output data d_{j_x,j_y} (See Notes (b) and (c))
4	CI	{D / R}	LX, LY	Input	Imaginary part of input data c_{k_x,k_y}
				Output	Imaginary part of output results d_{j_x,j_y} (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array CR and CI (See Note (b))
6	LY	I	1	Input	Second dimension of array CR and CI (See Note (b))
7	ISW	I	1	Input	Processing switch ISW= 1:Forward transform ISW=−1:Backward transform
8	IFAX	I	40	Input	Factorization results and number of factors (See Note (a))
9	TRIGS	{D / R}	2 × (NX + NY)	Input	Trigonometric function table (See Note (a))
10	WK	{D / R}	2 × LX × LY	Work	Work area
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) NX≤LX, NY≤LY

(c) ISW ∈ {1, −1}

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY) after the including-initialization subroutine 6.5.1 $\left\{\begin{matrix}\text{QFC2FB}\\ \text{PFC2FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be $\Re\{c_{k_x,k_y}\}$ and $\Im\{c_{k_x,k_y}\}$, respectively, the c_{k_x,k_y} and elements of arrays CR and CI are associated as follows.

$$\begin{aligned}\Re\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{CR}(\mathrm{k_x}+1, \mathrm{k_y}+1)\\ \Im\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{CI}(\mathrm{k_x}+1, \mathrm{k_y}+1)\end{aligned}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions LX **and** LY **of arrays** CR **and** CI **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is even,** LX=NX+1 **is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the two-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y} = (k_x+1) + (k_y+1) + \sqrt{-1}\frac{(k_x+1)(k_y+1)}{n_x n_y}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

as input data.

(b) Input data

Array CR and CI, NX=5, NY=4, LX=5, LY=5, ISW=1 (forward transform), ISW=−1 (backward transform) and NT=2.

(c) Main program

```
      PROGRAM OFC2BF
! *** EXAMPLE OF QFC2FB AND QFC2BF ***
      PARAMETER (NX=5,NY=4,LX=5,LY=5,NT=2)
      REAL(8) CR(LX,LY),CI(LX,LY)
      REAL(8) TRIGS(2*(NX+NY)),WK(2*LX*LY)
      INTEGER IFAX(40)
      COMMON CR,CI,TRIGS,WK,IFAX
!**** INPUT ****
      DO 20 J=1,NY
        DO 10 I=1,NX
          CR(I,J)= DBLE(I+J)
          CI(I,J)= DBLE(I*J)/(NX*NY)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,LX,LY,NT
      WRITE(6,1020)
      WRITE(6,1030) ((CR(I,J),CI(I,J),J=1,NY),I=1,NX)
!**** OUTPUT ****
      WRITE(6,1040)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL QFC2FB(NX,NY,CR,CI,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 40 J=1,NY
        DO 30 I=1,NX
          CR(I,J)=CR(I,J)/DBLE(NX*NY)
          CI(I,J)=CI(I,J)/DBLE(NX*NY)
   30   CONTINUE
   40 CONTINUE
      WRITE(6,1050) IERR
      WRITE(6,1020)
      WRITE(6,1030) ((CR(I,J),CI(I,J),J=1,NY),I=1,NX)
!**** BACKWARD TRANSFORM ****
      ISW=-1
      CALL QFC2BF(NX,NY,CR,CI,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1060) IERR
      WRITE(6,1020)
      WRITE(6,1030) ((CR(I,J),CI(I,J),J=1,NY),I=1,NX)
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFC2FB AND QFC2BF ***',/,/,&
             1X,' ** INPUT **',/)
 1010 FORMAT(1X,'    NX =',I3,'   NY =',I3,/,&
             1X,'    LX =',I3,'   LY =',I3,/,&
             1X,'    NT =',I3,/)
 1020 FORMAT(1X,'  ( CR(IX,IY) , CI(IX,IY) )')
 1030 FORMAT(4(1X,'  (',F6.3,',',F6.3,')'))
 1040 FORMAT(/,1X,' ** OUTPUT **')
 1050 FORMAT(/,1X,'  ( FORWARD  TRANSFORM )',/,&
             /,1X,'    IERR =',I4,/)
 1060 FORMAT(/,1X,'  ( BACKWARD TRANSFORM )',/,&
             /,1X,'    IERR =',I4,/)
      END
```

(d) Output results

```
*** QFC2FB AND QFC2BF ***

 ** INPUT **

    NX =  5   NY =  4
    LX =  5   LY =  5
    NT =  2

  ( CR(IX,IY) , CI(IX,IY) )
  ( 2.000, 0.050)   ( 3.000, 0.100)   ( 4.000, 0.150)   ( 5.000, 0.200)
  ( 3.000, 0.100)   ( 4.000, 0.200)   ( 5.000, 0.300)   ( 6.000, 0.400)
  ( 4.000, 0.150)   ( 5.000, 0.300)   ( 6.000, 0.450)   ( 7.000, 0.600)
```

```
 ( 5.000, 0.200)   ( 6.000, 0.400)   ( 7.000, 0.600)   ( 8.000, 0.800)
 ( 6.000, 0.250)   ( 7.000, 0.500)   ( 8.000, 0.750)   ( 9.000, 1.000)

** OUTPUT **

 ( FORWARD  TRANSFORM )

   IERR =   0

 ( CR(IX,IY) , CI(IX,IY) )
 ( 5.500, 0.375)   (-0.575, 0.425)   (-0.500,-0.075)   (-0.425,-0.575)
 (-0.586, 0.626)   ( 0.030,-0.005)   ( 0.017, 0.013)   ( 0.005, 0.030)
 (-0.520, 0.100)   ( 0.017, 0.008)   ( 0.004, 0.012)   (-0.008, 0.017)
 (-0.480,-0.225)   ( 0.008, 0.017)   (-0.004, 0.012)   (-0.017, 0.008)
 (-0.414,-0.751)   (-0.005, 0.030)   (-0.017, 0.013)   (-0.030,-0.005)

 ( BACKWARD TRANSFORM )

   IERR =   0

 ( CR(IX,IY) , CI(IX,IY) )
 ( 2.000, 0.050)   ( 3.000, 0.100)   ( 4.000, 0.150)   ( 5.000, 0.200)
 ( 3.000, 0.100)   ( 4.000, 0.200)   ( 5.000, 0.300)   ( 6.000, 0.400)
 ( 4.000, 0.150)   ( 5.000, 0.300)   ( 6.000, 0.450)   ( 7.000, 0.600)
 ( 5.000, 0.200)   ( 6.000, 0.400)   ( 7.000, 0.600)   ( 8.000, 0.800)
 ( 6.000, 0.250)   ( 7.000, 0.500)   ( 8.000, 0.750)   ( 9.000, 1.000)
```

6.6 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.6.1 [DEPRECATED]HFC2FB, GFC2FB Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

HFC2FB or GFC2FB computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

HFC2FB or GFC2FB computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:

```
CALL HFC2FB  (NX, NY, C, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL GFC2FB  (NX, NY, C, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	C	{Z / C}	LX, LY	Input	Input data c_{k_x,k_y} (See Note (b))
				Output	Output results d_{j_x,j_y} (See Notes (b) and (c))
4	LX	I	1	Input	Adjustable dimension of array C (See Note (b))
5	LY	I	1	Input	Second dimension of array C (See Note (b))
6	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	40	Output	Factorization results and number of factors (See Note (d))
8	TRIGS	{D / R}	2 × (NX + NY)	Output	Trigonometric function table (See Note (d))
9	WK	{Z / C}	LX × LY	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) NX≤LX, NY≤LY

(c) ISW $\in \{0, 1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data NX or NY can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300 (=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like.

(b) The complex data c_{k_x,k_y} ($k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1$) and elements of array C are associated as follows.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{C(k_x + 1, k_y + 1)}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions LX **and** LY **of array** C **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is even,** LX=NX**+1 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY), you should call this subroutine once, and then use the after-initialization transform 6.6.2 $\begin{Bmatrix}\mathrm{HFC2BF}\\ \mathrm{GFC2BF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.6.2 $\begin{Bmatrix}\mathrm{HFC2BF}\\ \mathrm{GFC2BF}\end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for array C.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.6.2 (7).

6.6.2 [DEPRECATED]HFC2BF, GFC2BF Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

HFC2BF or GFC2BF computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

HFC2BF or GFC2BF computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:

```
CALL HFC2BF  (NX, NY, C, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL GFC2BF  (NX, NY, C, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	C	{Z / C}	LX, LY	Input	Input data c_{k_x,k_y} (See Note (b))
				Output	Output results d_{j_x,j_y} (See Notes (b) and (c))
4	LX	I	1	Input	Adjustable dimension of array C (See Note (b))
5	LY	I	1	Input	Second dimension of array C (See Note (b))
6	ISW	I	1	Input	Processing switch ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	40	Input	Factorization results and number of factors (See Note (a))
8	TRIGS	{D / R}	2 × (NX + NY)	Input	Trigonometric function table (See Note (a))
9	WK	{Z / C}	LX × LY	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) NX≤LX, NY≤LY

(c) ISW ∈ {1, −1}

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY) after the including-initialization subroutine 6.6.1 $\left\{\begin{matrix}\text{HFC2FB}\\ \text{GFC2FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array C are associated as follows.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{C(k_x + 1, k_y + 1)}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions LX **and** LY **of array** C **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is even,** LX=NX**+1 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the two-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y} = (k_x+1) + (k_y+1) + \sqrt{-1}\frac{(k_x+1)(k_y+1)}{n_x n_y}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

as input data.

(b) Input data

Array C, NX=5, NY=4, LX=5, LY=5, ISW=1(Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM UFC2BF
! *** EXAMPLE OF HFC2FB AND HFC2BF ***
      PARAMETER (NX=5,NY=4,LX=5,LY=5,NT=2)
      COMPLEX(8) C(LX,LY),WK(LX*LY)
      REAL(8) TRIGS(2*(NX+NY))
      INTEGER IFAX(40)
      COMMON C,WK,TRIGS,IFAX
!**** INPUT ****
      DO 20 J=1,NY
        DO 10 I=1,NX
          C(I,J)= CMPLX(DBLE(I+J), DBLE(I*J)/DBLE(NX*NY), KIND=8)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,LX,LY,NT
      WRITE(6,1020)
      WRITE(6,1030) ((C(I,J),J=1,NY),I=1,NX)
!**** OUTPUT ****
      WRITE(6,1040)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL HFC2FB(NX,NY,C,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 40 J=1,NY
        DO 30 I=1,NX
          C(I,J)=C(I,J)/DBLE(NX*NY)
   30   CONTINUE
   40 CONTINUE
      WRITE(6,1050) IERR
      WRITE(6,1020)
      WRITE(6,1030) ((C(I,J),J=1,NY),I=1,NX)
!**** BACKWARD TRANSFORM ****
      ISW=-1
      CALL HFC2BF(NX,NY,C,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1060) IERR
      WRITE(6,1020)
      WRITE(6,1030) ((C(I,J),J=1,NY),I=1,NX)
      STOP
!
!**** FORMAT ****
!
 1000 FORMAT(1X,'*** HFC2FB AND HFC2BF ***',/,/,&
             1X,' ** INPUT **',/)
 1010 FORMAT(1X,'    NX =',I3,'   NY =',I3,/,&
             1X,'    LX =',I3,'   LY =',I3,/,&
             1X,'    NT =',I3,/)
 1020 FORMAT(1X,'  C(IX,IY)')
 1030 FORMAT(4(1X,'  (',F6.3,',',F6.3,')'))
 1040 FORMAT(/,1X,' ** OUTPUT **')
 1050 FORMAT(/,1X,'  ( FORWARD  TRANSFORM )',/,&
             /,1X,'    IERR =',I4,/)
 1060 FORMAT(/,1X,'  ( BACKWARD TRANSFORM )',/,&
             /,1X,'    IERR =',I4,/)
      END
```

(d) Output results

```
*** HFC2FB AND HFC2BF ***

 ** INPUT **

    NX =  5   NY =  4
    LX =  5   LY =  5
    NT =  2

  C(IX,IY)
  ( 2.000, 0.050)   ( 3.000, 0.100)   ( 4.000, 0.150)   ( 5.000, 0.200)
  ( 3.000, 0.100)   ( 4.000, 0.200)   ( 5.000, 0.300)   ( 6.000, 0.400)
  ( 4.000, 0.150)   ( 5.000, 0.300)   ( 6.000, 0.450)   ( 7.000, 0.600)
```

```
 ( 5.000, 0.200)   ( 6.000, 0.400)   ( 7.000, 0.600)   ( 8.000, 0.800)
 ( 6.000, 0.250)   ( 7.000, 0.500)   ( 8.000, 0.750)   ( 9.000, 1.000)

** OUTPUT **

 ( FORWARD  TRANSFORM )

   IERR =   0

 C(IX,IY)
 ( 5.500, 0.375)   (-0.575, 0.425)   (-0.500,-0.075)   (-0.425,-0.575)
 (-0.586, 0.626)   ( 0.030,-0.005)   ( 0.017, 0.013)   ( 0.005, 0.030)
 (-0.520, 0.100)   ( 0.017, 0.008)   ( 0.004, 0.012)   (-0.008, 0.017)
 (-0.480,-0.225)   ( 0.008, 0.017)   (-0.004, 0.012)   (-0.017, 0.008)
 (-0.414,-0.751)   (-0.005, 0.030)   (-0.017, 0.013)   (-0.030,-0.005)

 ( BACKWARD TRANSFORM )

   IERR =   0

 C(IX,IY)
 ( 2.000, 0.050)   ( 3.000, 0.100)   ( 4.000, 0.150)   ( 5.000, 0.200)
 ( 3.000, 0.100)   ( 4.000, 0.200)   ( 5.000, 0.300)   ( 6.000, 0.400)
 ( 4.000, 0.150)   ( 5.000, 0.300)   ( 6.000, 0.450)   ( 7.000, 0.600)
 ( 5.000, 0.200)   ( 6.000, 0.400)   ( 7.000, 0.600)   ( 8.000, 0.800)
 ( 6.000, 0.250)   ( 7.000, 0.500)   ( 8.000, 0.750)   ( 9.000, 1.000)
```

6.7 TWO-DIMENSIONAL REAL FOURIER TRANSFORM

6.7.1 [DEPRECATED]QFR2FB, PFR2FB Two-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

QFR2FB or PFR2FB obtains a half period of the two-dimensional Fourier forward transform (arbitrary radix) for the two-dimensional real data $r_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$.

$$c_{j_x,j_y} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} r_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \quad (j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1)$$

Here, $\lfloor x\rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y}(j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1)$ for $n_x n_y$ complex data $c_{j_x,j_y}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$ satisfying $c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$ and $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$, QFR2FB or PFR2FB obtains the two-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1}\{c_{0,j_y}+(-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y}\}e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})}\} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1) \end{aligned}$$

Here, $\lceil x\rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$.

(2) **Usage**

Double precision:

```
CALL QFR2FB  (NX, NY, R, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFR2FB  (NX, NY, R, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	R	{D / R}	LX, LY	Input	Input data r_{k_x,k_y} (Forward transform), or c_{j_x,j_y} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y} (Forward transform), or r_{k_x,k_y} (Backward transform) (See Notes (b) and (c))
4	LX	I	1	Input	Adjustable dimension of array R (See Note (b))
5	LY	I	1	Input	Second dimension of array R (See Note (b))
6	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
7	IFAX	I	40	Output	Factorization results and number of factors (See Note (d))
8	TRIGS	{D / R}	NX+2×NY	Output	Trigonometric function table (See Note (d))
9	WK	{D / R}	LX × LY	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) In the case where NX is an odd, NX+1≤LX, NY≤LY
or if NX is an even, NX+2≤LX, NY≤LY

(c) ISW $\in \{0, 1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data NX or NY can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300 (=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like.

(b) The real data r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array R are associated as follows.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{R(k_x + 1, k_y + 1)}$$

When computing the backward transform, if NX(=n_x) is odd, then $\mathrm{R(NX + 1, k_y + 1)} = 0$, and when NX is even, then $\mathrm{R(NX + 1, k_y + 1)} = \mathrm{R(NX + 2, k_y + 1)} = 0$. Also, when entering the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ into array R, the corresponding zeros mentioned above need not be specifically stored.

If we let the real and imaginary parts of the complex data $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ be $\Re\{c_{j_x,j_y}\}$ and $\Im\{c_{j_x,j_y}\}$, respectively, the c_{j_x,j_y} and elements of array R are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned} \Re\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 1, j_y + 1)} \\ \Im\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 2, j_y + 1)} \end{aligned}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0}\} = 0$, and when NX and NY are both even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$\begin{aligned} c^*_{n_x - j_x, n_y - j_y} &= c_{j_x,j_y} \\ c^*_{n_x - j_x, j_y} &= c_{j_x, n_y - j_y} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** R **should be set so that** LX**/2 and** LY **are odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX=NX**+4 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY), you should call this subroutine once, and then use the after-initialization transform 6.7.2 $\begin{Bmatrix}\text{QFR2BF}\\ \text{PFR2BF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.7.2 $\begin{Bmatrix}\text{QFR2BF}\\ \text{PFR2BF}\end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for array R.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.7.2 (7).

6.7.2 [DEPRECATED]QFR2BF, PFR2BF Two-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

QFR2BF or PFR2BF obtains a half period of the two-dimensional Fourier forward transform (arbitrary radix) for the two-dimensional real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$c_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} r_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ for $n_x n_y$ complex data $c_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ satisfying $c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$ and $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$, QFR2BF or PFR2BF obtains the two-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1} \{c_{0,j_y} + (-1)^{k_x} \hat{c}_{\frac{n_x}{2},j_y}\} e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2 \sum_{j_y=0}^{n_y-1} \sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil - 1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})}\} \\ &\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$.

(2) **Usage**

Double precision:

```
CALL QFR2BF  (NX, NY, R, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFR2BF  (NX, NY, R, LX, LY, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	R	{D / R}	LX, LY	Input	Input data r_{k_x,k_y} (Forward transform), or c_{j_x,j_y} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y} (Forward transform), or r_{k_x,k_y} (Backward transform) (See Notes (b) and (c))
4	LX	I	1	Input	Adjustable dimension of array R (See Note (b))
5	LY	I	1	Input	Second dimension of array R (See Note (b))
6	ISW	I	1	Input	Processing switch ISW= 1: Forward transform ISW=−1: Backward transform
7	IFAX	I	40	Input	Factorization results and number of factors (See Note (a))
8	TRIGS	{D / R}	NX + 2 × NY	Input	Trigonometric function table (See Note (a))
9	WK	{D / R}	LX × LY	Work	Work area
10	NT	I	1	Input	Number of tasks to be generated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2

(b) In the case where NX is an odd, NX+1≤LX, NY≤LY
or if NX is an even, NX+2≤LX, NY≤LY

(c) ISW $\in \{1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY) after the including-initialization subroutine 6.7.1 $\left\{\begin{matrix}\text{QFR2FB}\\ \text{PFR2FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The real data r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array R are associated as follows.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{R(k_x + 1, k_y + 1)}$$

When computing the backward transform, if NX($=n_x$) is odd, then $\mathrm{R(NX + 1, k_y + 1)} = 0$, and when NX is even, then $\mathrm{R(NX + 1, k_y + 1)} = \mathrm{R(NX + 2, k_y + 1)} = 0$. Also, when entering the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ into array R, the corresponding zeros mentioned above need not be specifically stored.

If we let the real and imaginary parts of the complex data $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ be $\Re\{c_{j_x,j_y}\}$ and $\Im\{c_{j_x,j_y}\}$, respectively, the c_{j_x,j_y} and elements of array R are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned}\Re\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 1, j_y + 1)}\\ \Im\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 2, j_y + 1)}\end{aligned}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0}\} = 0$, and when NX and NY are both even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$\begin{aligned}c^*_{n_x-j_x,n_y-j_y} &= c_{j_x,j_y}\\ c^*_{n_x-j_x,j_y} &= c_{j_x,n_y-j_y}\end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** R **should be set so that** LX**/2 and** LY **are odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX=NX**+4 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem
Compute the two-dimensional real Fourier forward and backward transforms using

$$r_{k_x,k_y} = \frac{n_x + n_y}{(k_x + 1) + (k_y + 1)}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

as input data.

(b) Input data
Array R, NX=6, NY=4, LX=10, LY=5, ISW=1(Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM OFR2BF
! *** EXAMPLE OF QFR2FB AND QFR2BF ***
      PARAMETER (NX=6,NY=4,LX=10,LY=5,NT=2)
      REAL(8) R(LX,LY),WK(LX*LY),TRIGS(NX+2*NY)
      INTEGER IFAX(40)
      COMMON R,WK,TRIGS,IFAX
      COMPLEX(8) C(LX/2,LY)
      POINTER (CP, C)
      CP=LOC(R)
!**** INPUT ****
      DO 20 J=1,NY
        DO 10 I=1,NX
          R(I,J)= DBLE(NX+NY)/DBLE(I+J)
   10   CONTINUE
   20 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,LX,LY,NT
      WRITE(6,1020) 'R(I,J)'
      WRITE(6,1030) ((R(I,J),J=1,NY),I=1,NX)
!**** OUTPUT ****
      WRITE(6,1040)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL QFR2FB(NX,NY,R,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 40 J=1,NY
        DO 30 I=1,(NX+2)/2
          C(I,J)=C(I,J)/DBLE(NX*NY)
   30   CONTINUE
   40 CONTINUE
      WRITE(6,1050) IERR
      WRITE(6,1020) 'C(I,J)'
      DO 50 I=1,(NX+2)/2
        WRITE(6,1060) (C(I,J),J=1,NY)
   50 CONTINUE
!**** BACKWARD TRANSFORM ****
      ISW=-1
```

```
      CALL QFR2BF(NX,NY,R,LX,LY,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1070) IERR
      WRITE(6,1020) 'R(I,J)'
      WRITE(6,1030) ((R(I,J),J=1,NY),I=1,NX+2)
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFR2FB AND QFR2BF ***',/,/,&
             1X,' ** INPUT **',/)
 1010 FORMAT(1X,'    NX =',I3,4X,'NY =',I3,/,&
             1X,'    LX =',I3,4X,'LY =',I3,/,&
             1X,'    NT =',I3,/)
 1020 FORMAT(1X,4X,A)
 1030 FORMAT(4(4X,F7.4))
 1040 FORMAT(/,1X,' ** OUTPUT **')
 1050 FORMAT(/,1X,' ( FORWARD  TRANSFORM )',/,/,&
             4X,'IERR =',I5,/)
 1060 FORMAT(3X,4(' (',F6.3,',',F6.3,')'))
 1070 FORMAT(/,1X,' ( BACKWARD TRANSFORM )',/,/,&
             4X,'IERR =',I5,/)
      END
```

(d) Output results

```
*** QFR2FB AND QFR2BF ***

 ** INPUT **

    NX =  6    NY =  4
    LX = 10    LY =  5
    NT =  2

    R(I,J)
    5.0000     3.3333     2.5000     2.0000
    3.3333     2.5000     2.0000     1.6667
    2.5000     2.0000     1.6667     1.4286
    2.0000     1.6667     1.4286     1.2500
    1.6667     1.4286     1.2500     1.1111
    1.4286     1.2500     1.1111     1.0000

 ** OUTPUT **

 ( FORWARD  TRANSFORM )

   IERR =    0

    C(I,J)
   ( 1.938, 0.000) ( 0.249,-0.155) ( 0.219, 0.000) ( 0.249, 0.155)
   ( 0.296,-0.247) ( 0.058,-0.094) ( 0.076,-0.045) ( 0.119,-0.009)
   ( 0.229,-0.093) ( 0.056,-0.053) ( 0.058,-0.019) ( 0.079, 0.010)
   ( 0.219, 0.000) ( 0.064,-0.030) ( 0.055, 0.000) ( 0.064, 0.030)

 ( BACKWARD TRANSFORM )

   IERR =    0

    R(I,J)
    5.0000     3.3333     2.5000     2.0000
    3.3333     2.5000     2.0000     1.6667
    2.5000     2.0000     1.6667     1.4286
    2.0000     1.6667     1.4286     1.2500
    1.6667     1.4286     1.2500     1.1111
    1.4286     1.2500     1.1111     1.0000
    0.0000     0.0000     0.0000     0.0000
    0.0000     0.0000     0.0000     0.0000
```

6.8 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.8.1 [DEPRECATED]QFC3FB, PFC3FB Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

QFC3FB or PFC3FB computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Backward transform

QFC3FB or PFC3FB computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

(2) **Usage**

Double precision:

```
CALL QFC3FB  (NX, NY, NZ, CR, CI, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFC3FB  (NX, NY, NZ, CR, CI, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	CR	{D / R}	LX, LY, LZ	Input	Real part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Real part of output data d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	CI	{D / R}	LX, LY, LZ	Input	Imaginary part of input data $c_{k,s,u}$ (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
6	LX	I	1	Input	Adjustable dimension of array CR and CI (See Note (b))
7	LY	I	1	Input	Second dimension of array CR and CI (See Note (b))
8	LZ	I	1	Input	Third dimension of array CR and CI (See Note (b))
9	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
10	IFAX	I	60	Output	Factorization results and number of factors (See Note (d))
11	TRIGS	{D / R}	2 × (NX + NY + NZ)	Output	Trigonometric function table (See Note (d))
12	WK	{D / R}	2 × LX × LY × LZ	Work	Work area
13	NT	I	1	Input	Number of tasks to be generated
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2 or NZ≥2

(b) NX≤LX, NY≤LY, or NZ≤LZ

(c) ISW $\in \{0, 1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data NX, NY or NZ can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300(=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y,k_z} ($k_x = 0, \cdots, n_x-1$; $k_y = 0, \cdots, n_y-1$; $k_z = 0, \cdots, n_z-1$) be $\Re\{c_{k_x,k_y,k_z}\}$ and $\Im\{c_{k_x,k_y,k_z}\}$, respectively, the c_{k_x,k_y,k_z} and elements of arrays CR and CI are associated as follows.

$$\begin{aligned} \Re\{c_{k_x,k_y,k_z}\} &\leftrightarrow \mathrm{CR}(\mathrm{k_x}+1, \mathrm{k_y}+1, \mathrm{k_z}+1) \\ \Im\{c_{k_x,k_y,k_z}\} &\leftrightarrow \mathrm{CI}(\mathrm{k_x}+1, \mathrm{k_y}+1, \mathrm{k_z}+1) \end{aligned}$$

Similarly, for the complex data d_{j_x,j_y,j_z}($j_x = 0, \cdots, n_x-1$; $j_y = 0, \cdots, n_y-1$; $j_z = 0, \cdots, n_z-1$).
The adjustable dimensions LX, LY, **and** LZ **of arrays** CR **and** CI **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** CR **and** CI**. Usually, when** NX**, for example, is even,** LX=NX+1 **is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data c_{k_x,k_y,k_z}($k_x = 0, \cdots, n_x-1$; $k_y = 0, \cdots, n_y-1$; $k_z = 0, \cdots, n_z-1$) be represented by $\hat{c}_{k_x,k_y,k_z}$($k_x = 0, \cdots, n_x-1$; $k_y = 0, \cdots, n_y-1$; $k_z = 0, \cdots, n_z-1$), then the following relationship holds.

$$\begin{aligned} &\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z} \\ &\quad (k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1,\ k_z = 0, \cdots, n_z-1) \end{aligned}$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY, NZ), you should call this subroutine once, and then use the after-initialization transform 6.8.2 $\left\{\begin{matrix}\mathrm{QFC3BF}\\ \mathrm{PFC3BF}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.8.2 $\left\{\begin{matrix}\mathrm{QFC3BF}\\ \mathrm{PFC3BF}\end{matrix}\right\}$.
To perform initialization only by setting ISW=0, you need not set input data for arrays CR and CI.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.8.2 (7).

6.8.2 [DEPRECATED]QFC3BF, PFC3BF Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

QFC3BF or PFC3BF computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Backward transform

QFC3BF or PFC3BF computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

(2) **Usage**

Double precision:

```
CALL QFC3BF  (NX, NY, NZ, CR, CI, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFC3BF  (NX, NY, NZ, CR, CI, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	CR	{D / R}	LX, LY, LZ	Input	Real part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Real part of output data d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	CI	{D / R}	LX, LY, LZ	Input	Imaginary part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
6	LX	I	1	Input	Adjustable dimension of array CR and CI (See Note (b))
7	LY	I	1	Input	Second dimension of array CR and CI (See Note (b))
8	LZ	I	1	Input	Third dimension of array CR and CI (See Note (b))
9	ISW	I	1	Input	Processing switch ISW= 1 :Forward transform ISW=−1 :Backward transform
10	IFAX	I	60	Input	Factorization results and number of factors (See Note (d))
11	TRIGS	{D / R}	2 × (NX + NY + NZ)	Input	Trigonometric function table (See Note (d))
12	WK	{D / R}	2 × LX × LY × LZ	Work	Work area
13	NT	I	1	Input	Number of tasks to be generated
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2, NZ≥2

(b) NX≤LX, NY≤LY, NZ≤LZ

(c) ISW ∈ {1, −1}

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY, NZ) after the including-initialization subroutine 6.8.1 $\left\{\begin{matrix}\text{QFC3FB}\\ \text{PFC3FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be $\Re\{c_{k_x,k_y,k_z}\}$ and $\Im\{c_{k_x,k_y,k_z}\}$, respectively, the c_{k_x,k_y,k_z} and elements of arrays CR and CI are associated as follows.

$$\begin{aligned}\Re\{c_{k_x,k_y,k_z}\} &\leftrightarrow \mathrm{CR(k_x+1, k_y+1, k_z+1)}\\ \Im\{c_{k_x,k_y,k_z}\} &\leftrightarrow \mathrm{CI(k_x+1, k_y+1, k_z+1)}\end{aligned}$$

Similarly, for the complex data $d_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$.
The adjustable dimensions LX, LY, **and** LZ **of arrays** CR **and** CI **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** CR **and** CI**. Usually, when** NX**, for example, is even,** LX=NX+1 **is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\begin{aligned}&\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}\\ &\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)\end{aligned}$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$,

then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the three-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y,k_z} = \frac{n_x + n_y + n_z}{(k_x + 1) + (k_y + 1) + (k_z + 1)} + \sqrt{-1}\frac{(k_x + 1)(k_y + 1)(k_z + 1)}{n_x n_y n_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

as input data.

(b) Input data

Array CR and CI, NX=5, NY=4, NZ=3, LX=5, LY=5, LZ=3, ISW=1 (Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM OFC3BF
! *** EXAMPLE OF QFC3FB AND QFC3BF ***
      PARAMETER (NX=5,NY=4,NZ=3,LX=5,LY=5,LZ=3,NT=2)
      REAL(8) CR(LX,LY,LZ),CI(LX,LY,LZ)
      REAL(8) TRIGS(2*(NX+NY+NZ)),WK(2*LX*LY*LZ)
      INTEGER IFAX(60)
      COMMON CR,CI,TRIGS,WK,IFAX
!**** INPUT ****
      DO 30 K=1,NZ
        DO 20 J=1,NY
          DO 10 I=1,NX
            CR(I,J,K)=DBLE(NX+NY+NZ)/DBLE(I+J+K)
            CI(I,J,K)=DBLE(I*J*K)/DBLE(NX*NY*NZ)
   10     CONTINUE
   20   CONTINUE
   30 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,NZ,LX,LY,LZ,NT
      DO 100 K=1,NZ
        WRITE(6,1020) K,K,((CR(I,J,K),CI(I,J,K),J=1,NY),I=1,NX)
  100 CONTINUE
!**** OUTPUT ****
      WRITE(6,1030)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL QFC3FB(NX,NY,NZ,CR,CI,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 60 K=1,NZ
        DO 50 J=1,NY
          DO 40 I=1,NX
            CR(I,J,K)=CR(I,J,K)/DBLE(NX*NY*NZ)
            CI(I,J,K)=CI(I,J,K)/DBLE(NX*NY*NZ)
   40     CONTINUE
   50   CONTINUE
   60 CONTINUE
      WRITE(6,1040) IERR
      DO 200 K=1,NZ
        WRITE(6,1020) K,K,((CR(I,J,K),CI(I,J,K),J=1,NY),I=1,NX)
  200 CONTINUE
!**** BACKWARD TRANSFORM ****
      ISW=-1
      CALL QFC3BF(NX,NY,NZ,CR,CI,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1050) IERR
      DO 300 K=1,NZ
        WRITE(6,1020) K,K,((CR(I,J,K),CI(I,J,K),J=1,NY),I=1,NX)
  300 CONTINUE
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFC3FB AND QFC3BF ***',/,/,&
             1X,'  ** INPUT **',/)
 1010 FORMAT(1X,'     NX =',I3,'  NY =',I3,'  NZ =',I3,/,&
             1X,'     LY =',I3,'  LY =',I3,'  LZ =',I3,/,&
             1X,'     NT =',I3,/)
 1020 FORMAT(/,4X,'( CR(IX,IY,',I2,') , CI(IX,IY,',I2,') )',&
           /,4(4X,'(',F6.3,',',F6.3,')'))
 1030 FORMAT(/,1X,'** OUTPUT **')
```

```
 1040 FORMAT(/,1X,'  ( FORWARD  TRANSFORM )',/,/,6X,'IERR =',I6)
 1050 FORMAT(/,1X,'  ( BACKWARD TRANSFORM )',/,/,6X,'IERR =',I6)
      END
```

(d) Output results

```
*** QFC3FB AND QFC3BF ***

  ** INPUT **

     NX =  5  NY =  4  NZ =  3
     LY =  5  LY =  5  LZ =  3
     NT =  2


   ( CR(IX,IY, 1) , CI(IX,IY, 1) )
   ( 4.000, 0.017)    ( 3.000, 0.033)    ( 2.400, 0.050)    ( 2.000, 0.067)
   ( 3.000, 0.033)    ( 2.400, 0.067)    ( 2.000, 0.100)    ( 1.714, 0.133)
   ( 2.400, 0.050)    ( 2.000, 0.100)    ( 1.714, 0.150)    ( 1.500, 0.200)
   ( 2.000, 0.067)    ( 1.714, 0.133)    ( 1.500, 0.200)    ( 1.333, 0.267)
   ( 1.714, 0.083)    ( 1.500, 0.167)    ( 1.333, 0.250)    ( 1.200, 0.333)

   ( CR(IX,IY, 2) , CI(IX,IY, 2) )
   ( 3.000, 0.033)    ( 2.400, 0.067)    ( 2.000, 0.100)    ( 1.714, 0.133)
   ( 2.400, 0.067)    ( 2.000, 0.133)    ( 1.714, 0.200)    ( 1.500, 0.267)
   ( 2.000, 0.100)    ( 1.714, 0.200)    ( 1.500, 0.300)    ( 1.333, 0.400)
   ( 1.714, 0.133)    ( 1.500, 0.267)    ( 1.333, 0.400)    ( 1.200, 0.533)
   ( 1.500, 0.167)    ( 1.333, 0.333)    ( 1.200, 0.500)    ( 1.091, 0.667)

   ( CR(IX,IY, 3) , CI(IX,IY, 3) )
   ( 2.400, 0.050)    ( 2.000, 0.100)    ( 1.714, 0.150)    ( 1.500, 0.200)
   ( 2.000, 0.100)    ( 1.714, 0.200)    ( 1.500, 0.300)    ( 1.333, 0.400)
   ( 1.714, 0.150)    ( 1.500, 0.300)    ( 1.333, 0.450)    ( 1.200, 0.600)
   ( 1.500, 0.200)    ( 1.333, 0.400)    ( 1.200, 0.600)    ( 1.091, 0.800)
   ( 1.333, 0.250)    ( 1.200, 0.500)    ( 1.091, 0.750)    ( 1.000, 1.000)

** OUTPUT **

  ( FORWARD  TRANSFORM )

     IERR =     0

   ( CR(IX,IY, 1) , CI(IX,IY, 1) )
   ( 1.737, 0.250)    ( 0.102,-0.160)    ( 0.137,-0.050)    ( 0.202, 0.060)
   ( 0.108,-0.189)    ( 0.038,-0.047)    ( 0.041,-0.012)    ( 0.052, 0.016)
   ( 0.125,-0.078)    ( 0.034,-0.017)    ( 0.026, 0.003)    ( 0.025, 0.021)
   ( 0.152,-0.005)    ( 0.037, 0.001)    ( 0.021, 0.014)    ( 0.012, 0.028)
   ( 0.223, 0.106)    ( 0.046, 0.024)    ( 0.018, 0.029)    (-0.002, 0.041)

   ( CR(IX,IY, 2) , CI(IX,IY, 2) )
   ( 0.106,-0.127)    ( 0.041,-0.022)    ( 0.031, 0.003)    ( 0.030, 0.025)
   ( 0.042,-0.032)    (-0.002,-0.009)    ( 0.002,-0.008)    ( 0.009,-0.010)
   ( 0.032,-0.007)    ( 0.001,-0.007)    ( 0.004,-0.005)    ( 0.009,-0.004)
   ( 0.030, 0.008)    ( 0.005,-0.007)    ( 0.007,-0.003)    ( 0.011, 0.000)
   ( 0.032, 0.029)    ( 0.011,-0.009)    ( 0.012,-0.002)    ( 0.016, 0.006)

   ( CR(IX,IY, 3) , CI(IX,IY, 3) )
   ( 0.178, 0.002)    ( 0.040, 0.014)    ( 0.017, 0.022)    ( 0.001, 0.033)
   ( 0.048, 0.009)    ( 0.005,-0.010)    ( 0.008,-0.006)    ( 0.015,-0.002)
   ( 0.024, 0.016)    ( 0.007,-0.006)    ( 0.008,-0.002)    ( 0.011, 0.003)
   ( 0.013, 0.024)    ( 0.010,-0.003)    ( 0.008, 0.002)    ( 0.008, 0.008)
   ( 0.001, 0.036)    ( 0.016, 0.001)    ( 0.011, 0.007)    ( 0.007, 0.016)

  ( BACKWARD TRANSFORM )

     IERR =     0

   ( CR(IX,IY, 1) , CI(IX,IY, 1) )
   ( 4.000, 0.017)    ( 3.000, 0.033)    ( 2.400, 0.050)    ( 2.000, 0.067)
   ( 3.000, 0.033)    ( 2.400, 0.067)    ( 2.000, 0.100)    ( 1.714, 0.133)
   ( 2.400, 0.050)    ( 2.000, 0.100)    ( 1.714, 0.150)    ( 1.500, 0.200)
   ( 2.000, 0.067)    ( 1.714, 0.133)    ( 1.500, 0.200)    ( 1.333, 0.267)
   ( 1.714, 0.083)    ( 1.500, 0.167)    ( 1.333, 0.250)    ( 1.200, 0.333)

   ( CR(IX,IY, 2) , CI(IX,IY, 2) )
   ( 3.000, 0.033)    ( 2.400, 0.067)    ( 2.000, 0.100)    ( 1.714, 0.133)
   ( 2.400, 0.067)    ( 2.000, 0.133)    ( 1.714, 0.200)    ( 1.500, 0.267)
   ( 2.000, 0.100)    ( 1.714, 0.200)    ( 1.500, 0.300)    ( 1.333, 0.400)
   ( 1.714, 0.133)    ( 1.500, 0.267)    ( 1.333, 0.400)    ( 1.200, 0.533)
   ( 1.500, 0.167)    ( 1.333, 0.333)    ( 1.200, 0.500)    ( 1.091, 0.667)

   ( CR(IX,IY, 3) , CI(IX,IY, 3) )
   ( 2.400, 0.050)    ( 2.000, 0.100)    ( 1.714, 0.150)    ( 1.500, 0.200)
   ( 2.000, 0.100)    ( 1.714, 0.200)    ( 1.500, 0.300)    ( 1.333, 0.400)
   ( 1.714, 0.150)    ( 1.500, 0.300)    ( 1.333, 0.450)    ( 1.200, 0.600)
   ( 1.500, 0.200)    ( 1.333, 0.400)    ( 1.200, 0.600)    ( 1.091, 0.800)
   ( 1.333, 0.250)    ( 1.200, 0.500)    ( 1.091, 0.750)    ( 1.000, 1.000)
```

6.9 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.9.1 [DEPRECATED]HFC3FB, GFC3FB Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

HFC3FB or GFC3FB computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Backward transform

HFC3FB or GFC3FB computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

(2) **Usage**

Double precision:

```
CALL HFC3FB  (NX, NY, NZ, C, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL GFC3FB  (NX, NY, NZ, C, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	C	{Z / C}	LX, LY, LZ	Input	Input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array C (See Note (b))
6	LY	I	1	Input	Second dimension of array C (See Note (b))
7	LZ	I	1	Input	Third dimension of array C (See Note (b))
8	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
9	IFAX	I	60	Output	Factorization results and number of factors (See Note (d))
10	TRIGS	{D / R}	2 × (NX + NY + NZ)	Output	Trigonometric function table (See Note (d))
11	WK	{Z / C}	LX×LY×LZ	Work	Work area
12	NT	I	1	Input	Number of tasks to be generated
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2, NZ≥2

(b) NX≤LX, NY≤LY, NZ≤LZ

(c) ISW $\in \{0, 1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data NX, NY or NZ can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = $289(=17^2)$, it is usually more efficient to set NX = $300(=2^2 \times 3 \times 5^2)$, NX = $320(=2^6 \times 5)$, NX = $384(=2^7 \times 3)$ or the like.

(b) The complex data c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array C are associated as follows.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{C}(\mathrm{k_x}+1, \mathrm{k_y}+1, \mathrm{k_z}+1)$$

Similarly, for the complex data $d_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$. **The adjustable dimensions** LX**,** LY**, and** LZ **of array** C **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** C**. Usually, when** NX**, for example, is even,** LX=NX**+1 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY, NZ), you should call this subroutine once, and then use the after-initialization transform 6.9.2 $\begin{Bmatrix}\mathrm{HFC3BF}\\ \mathrm{GFC3BF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.9.2 $\begin{Bmatrix}\mathrm{HFC3BF}\\ \mathrm{GFC3BF}\end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for array C.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period,

the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.9.2 (7).

6.9.2 [DEPRECATED]HFC3BF, GFC3BF Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

HFC3BF or GFC3BF computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Backward transform

HFC3BF or GFC3BF computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

(2) **Usage**

Double precision:

```
CALL HFC3BF  (NX, NY, NZ, C, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL GFC3BF  (NX, NY, NZ, C, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	C	{ Z / C }	LX, LY, LZ	Input	Input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array C (See Note (b))
6	LY	I	1	Input	Second dimension of array C (See Note (b))
7	LZ	I	1	Input	Third dimension of array C (See Note (b))
8	ISW	I	1	Input	Processing switch ISW= 1 :Forward transform ISW=−1 :Backward transform
9	IFAX	I	60	Input	Factorization results and number of factors (See Note (a))
10	TRIGS	{ D / R }	2 × (NX + NY + NZ)	Input	Trigonometric function table (See Note (a))
11	WK	{ Z / C }	LX×LY×LZ	Work	Work area
12	NT	I	1	Input	Number of tasks to be generated
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2, NZ≥2

(b) NX≤LX, NY≤LY, NZ≤LZ

(c) ISW $\in \{1, -1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY, NZ) after the including-initialization subroutine 6.9.1 $\left\{\begin{matrix}\text{HFC3FB}\\ \text{GFC3FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The complex data c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array C are associated as follows.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{C(k_x + 1, k_y + 1, k_z + 1)}$$

The adjustable dimensions LX**,** LY**, and** LZ **of array** C **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** C**. Usually, when** NX**, for example, is even,** LX=NX+1 **is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the three-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y,k_z} = \frac{n_x + n_y + n_z}{(k_x+1)+(k_y+1)+(k_z+1)} + \sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

as input data.

(b) Input data

Array CR and CI, NX=5, NY=4, NZ=3, LX=5, LY=5, LZ=3, ISW=1 (Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM UFC3BF
! *** EXAMPLE OF HFC3FB AND HFC3BF ***
      PARAMETER (NX=5,NY=4,NZ=3,LX=5,LY=5,LZ=3,NT=2)
      COMPLEX(8) C(LX,LY,LZ),WK(LX*LY*LZ)
      REAL(8) TRIGS(2*(NX+NY+NZ))
      INTEGER IFAX(60)
      COMMON C,WK,TRIGS,IFAX
!**** INPUT ****
      DO 30 K=1,NZ
        DO 20 J=1,NY
          DO 10 I=1,NX
            C(I,J,K)=CMPLX( DBLE(NX+NY+NZ)/DBLE(I+J+K),&
                            DBLE(I*J*K)/DBLE(NX*NY*NZ),&
                            KIND=8 )
   10     CONTINUE
   20   CONTINUE
   30 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,NZ,LX,LY,LZ,NT
      DO 100 K=1,NZ
        WRITE(6,1020) K,((C(I,J,K),J=1,NY),I=1,NX)
  100 CONTINUE
!**** OUTPUT ****
      WRITE(6,1030)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL HFC3FB(NX,NY,NZ,C,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 60 K=1,NZ
        DO 50 J=1,NY
          DO 40 I=1,NX
            C(I,J,K)=C(I,J,K)/DBLE(NX*NY*NZ)
   40     CONTINUE
   50   CONTINUE
   60 CONTINUE
      WRITE(6,1040) IERR
      DO 200 K=1,NZ
        WRITE(6,1020) K,((C(I,J,K),J=1,NY),I=1,NX)
  200 CONTINUE
!**** BACKWARD TRANSFORM ****
      ISW=-1
      CALL HFC3BF(NX,NY,NZ,C,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1050) IERR
      DO 300 K=1,NZ
        WRITE(6,1020) K,((C(I,J,K),J=1,NY),I=1,NX)
  300 CONTINUE
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** HFC3FB AND HFC3BF ***',/,/,&
             1X,' ** INPUT **',/)
 1010 FORMAT(1X,'    NX =',I3,'  NY =',I3,'  NZ =',I3,/,&
             1X,'    LX =',I3,'  LY =',I3,'  LZ =',I3,/,&
             1X,'    NT =',I3,/)
 1020 FORMAT(/,1X,'  C(IX,IY,',I2,')',&
           /,4(1X,'  (',F6.3,',',F6.3,')'))
 1030 FORMAT(/,1X,' ** OUTPUT **')
 1040 FORMAT(/,1X,'  ( FORWARD  TRANSFORM )',/,/,4X,'IERR =',I6)
 1050 FORMAT(/,1X,'  ( BACKWARD TRANSFORM )',/,/,4X,'IERR =',I6)
      END
```

(d) Output results

```
*** HFC3FB AND HFC3BF ***
```

```
** INPUT **

   NX =  5  NY =  4  NZ =  3
   LX =  5  LY =  5  LZ =  3
   NT =  2


 C(IX,IY, 1)
 ( 4.000, 0.017)   ( 3.000, 0.033)   ( 2.400, 0.050)   ( 2.000, 0.067)
 ( 3.000, 0.033)   ( 2.400, 0.067)   ( 2.000, 0.100)   ( 1.714, 0.133)
 ( 2.400, 0.050)   ( 2.000, 0.100)   ( 1.714, 0.150)   ( 1.500, 0.200)
 ( 2.000, 0.067)   ( 1.714, 0.133)   ( 1.500, 0.200)   ( 1.333, 0.267)
 ( 1.714, 0.083)   ( 1.500, 0.167)   ( 1.333, 0.250)   ( 1.200, 0.333)

 C(IX,IY, 2)
 ( 3.000, 0.033)   ( 2.400, 0.067)   ( 2.000, 0.100)   ( 1.714, 0.133)
 ( 2.400, 0.067)   ( 2.000, 0.133)   ( 1.714, 0.200)   ( 1.500, 0.267)
 ( 2.000, 0.100)   ( 1.714, 0.200)   ( 1.500, 0.300)   ( 1.333, 0.400)
 ( 1.714, 0.133)   ( 1.500, 0.267)   ( 1.333, 0.400)   ( 1.200, 0.533)
 ( 1.500, 0.167)   ( 1.333, 0.333)   ( 1.200, 0.500)   ( 1.091, 0.667)

 C(IX,IY, 3)
 ( 2.400, 0.050)   ( 2.000, 0.100)   ( 1.714, 0.150)   ( 1.500, 0.200)
 ( 2.000, 0.100)   ( 1.714, 0.200)   ( 1.500, 0.300)   ( 1.333, 0.400)
 ( 1.714, 0.150)   ( 1.500, 0.300)   ( 1.333, 0.450)   ( 1.200, 0.600)
 ( 1.500, 0.200)   ( 1.333, 0.400)   ( 1.200, 0.600)   ( 1.091, 0.800)
 ( 1.333, 0.250)   ( 1.200, 0.500)   ( 1.091, 0.750)   ( 1.000, 1.000)

** OUTPUT **

 ( FORWARD  TRANSFORM )

  IERR =     0

 C(IX,IY, 1)
 ( 1.737, 0.250)   ( 0.102,-0.160)   ( 0.137,-0.050)   ( 0.202, 0.060)
 ( 0.108,-0.189)   ( 0.038,-0.047)   ( 0.041,-0.012)   ( 0.052, 0.016)
 ( 0.125,-0.078)   ( 0.034,-0.017)   ( 0.026, 0.003)   ( 0.025, 0.021)
 ( 0.152,-0.005)   ( 0.037, 0.001)   ( 0.021, 0.014)   ( 0.012, 0.028)
 ( 0.223, 0.106)   ( 0.046, 0.024)   ( 0.018, 0.029)   (-0.002, 0.041)

 C(IX,IY, 2)
 ( 0.106,-0.127)   ( 0.041,-0.022)   ( 0.031, 0.003)   ( 0.030, 0.025)
 ( 0.042,-0.032)   (-0.002,-0.009)   ( 0.002,-0.008)   ( 0.009,-0.010)
 ( 0.032,-0.007)   ( 0.001,-0.007)   ( 0.004,-0.005)   ( 0.009,-0.004)
 ( 0.030, 0.008)   ( 0.005,-0.007)   ( 0.007,-0.003)   ( 0.011, 0.000)
 ( 0.032, 0.029)   ( 0.011,-0.009)   ( 0.012,-0.002)   ( 0.016, 0.006)

 C(IX,IY, 3)
 ( 0.178, 0.002)   ( 0.040, 0.014)   ( 0.017, 0.022)   ( 0.001, 0.033)
 ( 0.048, 0.009)   ( 0.005,-0.010)   ( 0.008,-0.006)   ( 0.015,-0.002)
 ( 0.024, 0.016)   ( 0.007,-0.006)   ( 0.008,-0.002)   ( 0.011, 0.003)
 ( 0.013, 0.024)   ( 0.010,-0.003)   ( 0.008, 0.002)   ( 0.008, 0.008)
 ( 0.001, 0.036)   ( 0.016, 0.001)   ( 0.011, 0.007)   ( 0.007, 0.016)

 ( BACKWARD TRANSFORM )

  IERR =     0

 C(IX,IY, 1)
 ( 4.000, 0.017)   ( 3.000, 0.033)   ( 2.400, 0.050)   ( 2.000, 0.067)
 ( 3.000, 0.033)   ( 2.400, 0.067)   ( 2.000, 0.100)   ( 1.714, 0.133)
 ( 2.400, 0.050)   ( 2.000, 0.100)   ( 1.714, 0.150)   ( 1.500, 0.200)
 ( 2.000, 0.067)   ( 1.714, 0.133)   ( 1.500, 0.200)   ( 1.333, 0.267)
 ( 1.714, 0.083)   ( 1.500, 0.167)   ( 1.333, 0.250)   ( 1.200, 0.333)

 C(IX,IY, 2)
 ( 3.000, 0.033)   ( 2.400, 0.067)   ( 2.000, 0.100)   ( 1.714, 0.133)
 ( 2.400, 0.067)   ( 2.000, 0.133)   ( 1.714, 0.200)   ( 1.500, 0.267)
 ( 2.000, 0.100)   ( 1.714, 0.200)   ( 1.500, 0.300)   ( 1.333, 0.400)
 ( 1.714, 0.133)   ( 1.500, 0.267)   ( 1.333, 0.400)   ( 1.200, 0.533)
 ( 1.500, 0.167)   ( 1.333, 0.333)   ( 1.200, 0.500)   ( 1.091, 0.667)

 C(IX,IY, 3)
 ( 2.400, 0.050)   ( 2.000, 0.100)   ( 1.714, 0.150)   ( 1.500, 0.200)
 ( 2.000, 0.100)   ( 1.714, 0.200)   ( 1.500, 0.300)   ( 1.333, 0.400)
 ( 1.714, 0.150)   ( 1.500, 0.300)   ( 1.333, 0.450)   ( 1.200, 0.600)
 ( 1.500, 0.200)   ( 1.333, 0.400)   ( 1.200, 0.600)   ( 1.091, 0.800)
 ( 1.333, 0.250)   ( 1.200, 0.500)   ( 1.091, 0.750)   ( 1.000, 1.000)
```

6.10 THREE-DIMENSIONAL REAL FOURIER TRANSFORM

6.10.1 [DEPRECATED]QFR3FB, PFR3FB Three-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

QFR3FB or PFR3FB obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$\begin{aligned} c_{j_x,j_y,j_z} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1) \end{aligned}$$

Here, $\lfloor x\rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y,j_z}(j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ for $n_xn_yn_z$ complex data $c_{j_x,j_y,j_z}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ satisfying $c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$, and $c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$, QFR3FB or PFR3FB obtains the three-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &= \sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\{c_{0,j_y,j_z} + (-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y,j_z}\} e^{2\pi\sqrt{-1}(\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad + 2\sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}\} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1) \end{aligned}$$

Here, $\lceil x\rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = c_{\frac{n_x}{2},j_y,j_z}$.

(2) **Usage**

Double precision:

```
CALL QFR3FB  (NX, NY, NZ, R, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFR3FB  (NX, NY, NZ, R, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	R	{D / R}	LX, LY, LZ	Input	Input data r_{k_x,k_y,k_z} (Forward transform), or c_{j_x,j_y,j_z} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y,j_z} (Forward transform), or r_{k_x,k_y,k_z} (Backward transform) (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array R (See Note (b))
6	LY	I	1	Input	Second dimension of array R (See Note (b))
7	LZ	I	1	Input	Third dimension of array R (See Note (b))
8	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0:Initialization only ISW= 1:Forward transform ISW=−1:Backward transform
9	IFAX	I	60	Output	Factorization results and number of factors (See Note (d))
10	TRIGS	{D / R}	NX + 2 × (NY + NZ)	Output	Trigonometric function table (See Note (d))
11	WK	{D / R}	LX×LY×LZ	Work	Work area
12	NT	I	1	Input	Number of tasks to be generated
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2, NZ≥2

(b) In the case where NX is an odd, NX+1≤LX, NY≤LY, NZ≤LZ
or if NX is an even, NX+2≤LX, NY≤LY, NZ≤LZ

(c) LX should be even number.

(d) ISW ∈ $\{0, 1, -1\}$

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of data NX, NY or NZ can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300(=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like.

(b) The real data r_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array R are associated as follows.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{R(k_x + 1, k_y + 1, k_z + 1)}$$

When computing the backward transform, if NX(=n_x) is odd, then $\mathrm{R(NX + 1, k_y + 1, k_z + 1) = 0}$, and when NX is even, then $\mathrm{R(NX + 1, k_y + 1, k_z + 1) = R(NX + 2, k_y + 1, k_z + 1) = 0}$. Also, when entering the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ into array R, the corresponding zeros mentioned above need not be specifically stored.
If we let the real and imaginary parts of the complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ be $\Re\{c_{j_x,j_y,j_z}\}$ and $\Im\{c_{j_x,j_y,j_z}\}$, respectively, the c_{j_x,j_y,j_z} and elements of array R are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned} \Re\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 1, j_y + 1, j_z + 1)} \\ \Im\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{R(2 * j_x + 2, j_y + 1, j_z + 1)} \end{aligned}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0,0}\} = 0$, and when NX, NY and NZ are all even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y,j_z} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$\begin{aligned} c^*_{n_x - j_x, n_y - j_y, n_z - j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x - j_x, j_y, j_z} &= c_{j_x, n_y - j_y, n_z - j_z} \\ c^*_{n_x - j_x, n_y - j_y, j_z} &= c_{j_x, j_y, n_z - j_z} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** R **should be set so that** LX**/2,** LY**, and** LZ **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** R**. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX=NX**+4 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of

data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{r}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (NX, NY, NZ), you should call this subroutine once, and then use the after-initialization transform 6.10.2 $\begin{Bmatrix}\text{QFR3BF}\\ \text{PFR3BF}\end{Bmatrix}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input to the subroutine 6.10.2 $\begin{Bmatrix}\text{QFR3BF}\\ \text{PFR3BF}\end{Bmatrix}$.
To perform initialization only by setting ISW=0, you need not set input data for array R.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.10.2 (7).

6.10.2 [DEPRECATED]QFR3BF, PFR3BF Three-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

QFR3BF or PFR3BF obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1)$.

$$\begin{aligned} c_{j_x,j_y,j_z} &= \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y-1;\ j_z = 0, \cdots, n_z-1) \end{aligned}$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y-1;\ j_z = 0, \cdots, n_z-1)$ for $n_x n_y n_z$ complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x-1;\ j_y = 0, \cdots, n_y-1;\ j_z = 0, \cdots, n_z-1)$ satisfying $c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$, and $c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$, QFR3BF or PFR3BF obtains the three-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &= \sum_{j_z=0}^{n_z-1} \sum_{j_y=0}^{n_y-1} \{c_{0,j_y,j_z} + (-1)^{k_x} \hat{c}_{\frac{n_x}{2},j_y,j_z}\} e^{2\pi\sqrt{-1}(\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad + 2 \sum_{j_z=0}^{n_z-1} \sum_{j_y=0}^{n_y-1} \sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil-1} \Re\{c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}\} \\ &\quad (k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = c_{\frac{n_x}{2},j_y,j_z}$.

(2) **Usage**

Double precision:

```
CALL QFR3BF  (NX, NY, NZ, R, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

Single precision:

```
CALL PFR3BF  (NX, NY, NZ, R, LX, LY, LZ, ISW, IFAX, TRIGS, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	NY	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	NZ	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	R	{D / R}	LX, LY, LZ	Input	Input data r_{k_x,k_y,k_z} (Forward transform), or c_{j_x,j_y,j_z} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y,j_z} (Forward transform), or r_{k_x,k_y,k_z} (Backward transform) (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array R (See Note (b))
6	LY	I	1	Input	Second dimension of array R (See Note (b))
7	LZ	I	1	Input	Third dimension of array R (See Note (b))
8	ISW	I	1	Input	Processing switch ISW= 1: Forward transform ISW=−1: Backward transform
9	IFAX	I	60	Input	Factorization results and number of factors (See Note (a))
10	TRIGS	{D / R}	NX + 2 × (NY + NZ)	Input	Trigonometric function table (See Note (a))
11	WK	{D / R}	LX×LY×LZ	Work	Work area
12	NT	I	1	Input	Number of tasks to be generated
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX≥2, NY≥2, NZ≥2

(b) In the case where NX is an odd, NX+1≤LX, NY≤LY, NZ≤LZ
or if NX is an even, NX+2≤LX, NY≤LY, NZ≤LZ

(c) LX should be even number.

(d) ISW $\in \{1, -1\}$

(e) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This subroutine can be used to repeatedly compute the transform for the same number of data (NX, NY, NZ) after the including-initialization subroutine 6.10.1 $\left\{\begin{matrix}\text{QFR3FB}\\ \text{PFR3FB}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays IFAX and TRIGS so they can be used as input in this subroutine.

(b) The real data r_{k_x,k_y,k_z} $(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ and elements of array R are associated as follows.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{R(k_x+1, k_y+1, k_z+1)}$$

When computing the backward transform, if NX$(=n_x)$ is odd, then $\mathrm{R(NX+1, k_y+1, k_z+1)} = 0$, and when NX is even, then $\mathrm{R(NX+1, k_y+1, k_z+1)} = \mathrm{R(NX+2, k_y+1, k_z+1)} = 0$. Also, when entering the real data $r_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ into array R, the corresponding zeros mentioned above need not be specifically stored.
If we let the real and imaginary parts of the complex data $c_{j_x,j_y,j_z}(j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ be $\Re\{c_{j_x,j_y,j_z}\}$ and $\Im\{c_{j_x,j_y,j_z}\}$, respectively, the c_{j_x,j_y,j_z} and elements of array R are associated as follows. Here, $\lfloor x\rfloor$ represents the maximum integer that does not exceed x.

$$\begin{aligned}\Re\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{R(2*j_x+1, j_y+1, j_z+1)}\\ \Im\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{R(2*j_x+2, j_y+1, j_z+1)}\end{aligned}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0,0}\} = 0$, and when NX, NY and NZ are all even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array R, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y,j_z} $(j_x = \lfloor\frac{n_x}{2}\rfloor+1,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$\begin{aligned}&c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}\\ &c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}\\ &c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}\end{aligned}$$

The adjustable dimensions of array R **should be set so that** LX**/2,** LY**, and** LZ **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** R**. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX=NX**+4 is set.**

(c) When this subroutine is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of

data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{r}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This subroutine will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the three-dimensional real Fourier forward and backward transforms using

$$r_{k_x,k_y,k_z} = \frac{(k_x + 1)(k_y + 1)(k_z + 1)}{n_x n_y n_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

as input data.

(b) Input data

Array R, NX=6, NY=4, NZ=3, LX=10, LY=5, LZ=3, ISW=1 (Forward transform), ISW=−1 (Backward transform) and NT=2.

(c) Main program

```
      PROGRAM OFR3BF
! *** EXAMPLE OF QFR3FB AND QFR3BF ***
      PARAMETER (NX=6,NY=4,NZ=3,LX=10,LY=5,LZ=3,NT=2)
      REAL(8) R(LX,LY,LZ)
      REAL(8) TRIGS(NX+2*(NY+NZ)),WK(LX*LY*LZ)
      INTEGER IFAX(60)
      COMMON R,TRIGS,WK,IFAX
      COMPLEX(8) C(LX/2,LY,LZ)
      POINTER (CP, C)
      CP=LOC(R)
!**** INPUT ****
      DO 30 K=1,NZ
        DO 20 J=1,NY
          DO 10 I=1,NX
            R(I,J,K)=DBLE(I*J*K)/DBLE(NX*NY*NZ)
   10     CONTINUE
   20   CONTINUE
   30 CONTINUE
      WRITE(6,1000)
      WRITE(6,1010) NX,NY,NZ,LX,LY,LZ,NT
      DO 100 K=1,NZ
        WRITE(6,1020) K,((R(I,J,K),J=1,NY),I=1,NX)
  100 CONTINUE
```

```
!**** OUTPUT ****
      WRITE(6,1030)
!**** INITIALIZATION + FORWARD TRANSFORM ****
      ISW= 1
      CALL QFR3FB(NX,NY,NZ,R,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
!**** NORMALIZATION ****
      DO 60 K=1,NZ
        DO 50 J=1,NY
          DO 40 I=1,(NX+2)/2
            C(I,J,K)=C(I,J,K)/DBLE(NX*NY*NZ)
   40     CONTINUE
   50   CONTINUE
   60 CONTINUE
      WRITE(6,1040) IERR
      DO 220 K=1,NZ
        WRITE(6,1050) K
        DO 210 I=1,(NX+2)/2
          WRITE(6,1060) (C(I,J,K),J=1,NY)
  210   CONTINUE
  220 CONTINUE
!**** BACKWARD TRANSFORM ****
      ISW=-1
      CALL QFR3BF(NX,NY,NZ,R,LX,LY,LZ,ISW,IFAX,TRIGS,WK,NT,IERR)
      WRITE(6,1070) IERR
      DO 300 K=1,NZ
        WRITE(6,1020) K,((R(I,J,K),J=1,NY),I=1,NX+2)
  300 CONTINUE
      STOP
!**** FORMAT ****
 1000 FORMAT(1X,'*** QFR3FB AND QFR3BF ***',/,/,&
             1X,' ** INPUT **',/)
 1010 FORMAT(1X,'  NX =',I3,'  NY =',I3,'  NZ =',I3,/,&
             1X,'  LX =',I3,'  LY =',I3,'  LZ =',I3,/,&
             1X,'  NT =',I3,/)
 1020 FORMAT(/,3X,'R(IX,IY,',I2,') ',&
             /,4(4X,F7.4))
 1030 FORMAT(/,1X,' ** OUTPUT **')
 1040 FORMAT(/,1X,'  ( FORWARD  TRANSFORM )',/,/,4X,'IERR =',I6)
 1050 FORMAT(/,3X,'C(IX,IY,',I2,') ')
 1060 FORMAT(3X,4(' (',F6.3,',',F6.3,')'))
 1070 FORMAT(/,1X,'  ( BACKWARD TRANSFORM )',/,/,4X,'IERR =',I6)
      END
```

(d) Output results

```
*** QFR3FB AND QFR3BF ***

 ** INPUT **

  NX =  6  NY =  4  NZ =  3
  LX = 10  LY =  5  LZ =  3
  NT =  2


  R(IX,IY, 1)
    0.0139     0.0278     0.0417     0.0556
    0.0278     0.0556     0.0833     0.1111
    0.0417     0.0833     0.1250     0.1667
    0.0556     0.1111     0.1667     0.2222
    0.0694     0.1389     0.2083     0.2778
    0.0833     0.1667     0.2500     0.3333

  R(IX,IY, 2)
    0.0278     0.0556     0.0833     0.1111
    0.0556     0.1111     0.1667     0.2222
    0.0833     0.1667     0.2500     0.3333
    0.1111     0.2222     0.3333     0.4444
    0.1389     0.2778     0.4167     0.5556
    0.1667     0.3333     0.5000     0.6667

  R(IX,IY, 3)
    0.0417     0.0833     0.1250     0.1667
    0.0833     0.1667     0.2500     0.3333
    0.1250     0.2500     0.3750     0.5000
    0.1667     0.3333     0.5000     0.6667
    0.2083     0.4167     0.6250     0.8333
    0.2500     0.5000     0.7500     1.0000

 ** OUTPUT **

  ( FORWARD  TRANSFORM )

   IERR =     0

  C(IX,IY, 1)
   ( 0.243, 0.000) (-0.049, 0.049) (-0.049, 0.000) (-0.049,-0.049)
   (-0.035, 0.060) (-0.005,-0.019) ( 0.007,-0.012) ( 0.019,-0.005)
   (-0.035, 0.020) ( 0.003,-0.011) ( 0.007,-0.004) ( 0.011, 0.003)
   (-0.035, 0.000) ( 0.007,-0.007) ( 0.007, 0.000) ( 0.007, 0.007)

  C(IX,IY, 2)
   (-0.061, 0.035) ( 0.005,-0.019) ( 0.012,-0.007) ( 0.019, 0.005)
   ( 0.000,-0.020) ( 0.004, 0.004) (-0.000, 0.004) (-0.004, 0.004)
   ( 0.006,-0.010) ( 0.001, 0.003) (-0.001, 0.002) (-0.003, 0.001)
```

```
 ( 0.009,-0.005) (-0.001, 0.003) (-0.002, 0.001) (-0.003,-0.001)

C(IX,IY, 3)
 (-0.061,-0.035) ( 0.019,-0.005) ( 0.012, 0.007) ( 0.005, 0.019)
 ( 0.017,-0.010) (-0.001, 0.005) (-0.003, 0.002) (-0.005,-0.001)
 ( 0.012, 0.000) (-0.002, 0.002) (-0.002, 0.000) (-0.002,-0.002)
 ( 0.009, 0.005) (-0.003, 0.001) (-0.002,-0.001) (-0.001,-0.003)

( BACKWARD TRANSFORM )

 IERR =     0

R(IX,IY, 1)
  0.0139     0.0278     0.0417     0.0556
  0.0278     0.0556     0.0833     0.1111
  0.0417     0.0833     0.1250     0.1667
  0.0556     0.1111     0.1667     0.2222
  0.0694     0.1389     0.2083     0.2778
  0.0833     0.1667     0.2500     0.3333
  0.0000     0.0000     0.0000     0.0000
  0.0000     0.0000     0.0000     0.0000

R(IX,IY, 2)
  0.0278     0.0556     0.0833     0.1111
  0.0556     0.1111     0.1667     0.2222
  0.0833     0.1667     0.2500     0.3333
  0.1111     0.2222     0.3333     0.4444
  0.1389     0.2778     0.4167     0.5556
  0.1667     0.3333     0.5000     0.6667
  0.0000     0.0000     0.0000     0.0000
  0.0000     0.0000     0.0000     0.0000

R(IX,IY, 3)
  0.0417     0.0833     0.1250     0.1667
  0.0833     0.1667     0.2500     0.3333
  0.1250     0.2500     0.3750     0.5000
  0.1667     0.3333     0.5000     0.6667
  0.2083     0.4167     0.6250     0.8333
  0.2500     0.5000     0.7500     1.0000
  0.0000     0.0000     0.0000     0.0000
  0.0000     0.0000     0.0000     0.0000
```

6.11 CONVOLUTIONS

6.11.1 QFCN2D, PFCN2D
Two-Dimensional Convolutions

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ of period (m_x, m_y) satisfying:

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

for arbitrary integers L_x and L_y take nonzero values within their basic periods only for $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$ and $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$. Here, $[0, a] \times [0, b]$ is the direct product region (region contained in the square for which the point (0, 0) and the point (a, b) are diagonal points) on the plane in which the plane coordinates (i, j) lie. At this time, QFCN2D or PFCN2D calculates the discrete convolution $p(k_x, k_y)$ defined as follows:

$$\begin{aligned} p(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x - i_x, k_y - i_y) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} g(j_x, j_y) f(k_x - j_x, k_y - j_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$ and $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ and M_x and M_y are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$ and $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, respectively. The two-dimensional real Fourier transform of $p(k_x, k_y)$ can also be obtained.

(2) **Usage**

Double precision:

```
CALL QFCN2D  (NX1, NY1, NX2, NY2, R1, LX1, LY1, R2, LX2, LY2, MX, MY, ISW, IWK,
              WK, NT, IERR)
```

Single precision:

```
CALL PFCN2D  (NX1, NY1, NX2, NY2, R1, LX1, LY1, R2, LX2, LY2, MX, MY, ISW, IWK,
              WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y)$
2	NY1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y)$
3	NX2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y)$
4	NY2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y)$
5	R1	{D / R}	LX1, LY1	Input	Values of discrete function $f(i_x, i_y)$ (See Note (a))
				Output	When ISW ≥ 1, result of two-dimensional real Fourier transform of discrete function $f(i_x, i_y)$ (period (M_x, M_y))
6	LX1	I	1	Input	Adjustable dimension of array R1
7	LY1	I	1	Input	Second dimension of array R1
8	R2	{D / R}	LX2, LY2	Input	Values of discrete function $g(j_x, j_y)$ (See Note (a))
				Output	Value of discrete function $p(k_x, k_y)$ or its two-dimensional real Fourier transform (See Note (b))
9	LX2	I	1	Input	Adjustable dimension of array R2
10	LY2	I	1	Input	Second dimension of array R2
11	MX	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $p(k_x, k_y)$ (See Note (c))
12	MY	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $p(k_x, k_y)$ (See Note (c))
13	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0: Calculate the convolution according to the definition. ISW= 1: Calculate the convolution according to the FFT method. ISW= 2: Calculate the real Fourier transform of the convolution.

No.	Argument	Type	Size	Input/ Output	Contents
14	IWK	I	See Contents	Work	Work area **Size**: 0 (When ISW= 0) 40 (When ISW $\geq$ 1)
15	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: NX2 $\times$ NY2 ((When ISW= 0 and NX2 is odd) (NX2 + 1) $\times$ NY2 (When ISW= 0 and NX2 is even) MX + 2 $\times$ MY + MAX(LX1 $\times$ LY1, LX2 $\times$ LY2) (When ISW $\geq$ 1)
16	NT	I	1	Input	Number of tasks to be generated
17	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, 1, 2\}$

(b) NX1 > 1 and NY1 > 1

(c) NX2 > 1 and NY2 > 1

(d) MX $\geq$ MAX(NX1, NX2) and MY $\geq$ MAX(NY1, NY2)

(e) LX1 $\geq$ NX1 and LY1 $\geq$ NY1 (When ISW=0)
LX1 $\geq$ MX + 1 and LY1 $\geq$ MY (When ISW $\geq$ 1 and MX is odd)
LX1 $\geq$ MX + 2 and LY1 $\geq$ MY (When ISW $\geq$ 1 and MX is even)

(f) LX2 $\geq$ MX and LY2 $\geq$ MY (When ISW=0)
LX2 $\geq$ MX + 1 and LY2 $\geq$ MY (When ISW $\geq$ 1 and MX is odd)
LX2 $\geq$ MX + 2 and LY2 $\geq$ MY (When ISW $\geq$ 1 and MX is even)

(g) NT $\geq$ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	MX $<$ NX1 + NX2 $-$ 1 or MY $<$ NY1 + NY2 $-$ 1	Overlapping occurred during the convolution calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ and the elements of arrays R1 and R2 are associated as follows.

$$\begin{aligned} f(i_x, i_y) &\leftrightarrow \mathrm{R1(i_x + 1, i_y + 1)} \\ g(j_x, j_y) &\leftrightarrow \mathrm{R2(j_x + 1, j_y + 1)} \end{aligned}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** R1 **and** R2 **should be set so that** LX1**/2,** LY1**,** LX2**/2, and** LY2 **are odd numbers to avoid bank conflict of main memory. Usually, when** MX**, for example, is a multiple of 4,** LX1=MX**+3 is set.**

(b) The values of the discrete convolution $p(k_x, k_y)$ and the elements of array R2 are associated as follows.

$$p(k_x, k_y) \leftrightarrow \mathrm{R2(k_x + 1, k_y + 1)}$$

Here, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$. When ISW=2 is set to obtain the two-dimensional real Fourier transform $P(j_x, j_y)$ of the discrete convolution $p(k_x, k_y)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$\begin{aligned} P(j_x, j_y) &= \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} p(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor) \end{aligned}$$

the following associations are made:

$$\begin{aligned} \Re\{P(j_x, j_y)\} &\leftrightarrow \mathrm{R2(2 * j_x + 1, j_y + 1)} \\ \Im\{P(j_x, j_y)\} &\leftrightarrow \mathrm{R2(2 * j_x + 2, j_y + 1)} \end{aligned}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$\begin{aligned} P(M_x - j_x, M_y - j_y)^* &= P(j_x, j_y) \\ P(M_x - j_x, j_y)^* &= P(j_x, M_y - j_y) \end{aligned}$$

(Here, z^* represents the conjugate complex number of the complex number z.)

(c) If MX $\geq$ NX1 + NX2 $-$ 1 and MY $\geq$ NY1 + NY2 $-$ 1 are set, the convolution can be calculated without causing an overlap with the convolution of the next period. When MX $>$ NX1 + NX2 $-$ 1 or MY $>$ NY1 + NY2 $-$ 1, the following correspondences are made:

$$p(k_x, k_y) \leftrightarrow \mathrm{R2(k_x + 1, k_y + 1)}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x =$ NX1 + NX2 $-$ 1, $\cdots$, MX $-$ 1; $k_y = 0, \cdots,$ MY $-$ 1 or $k_x = 0, \cdots,$ MX $-$ 1; $k_y =$ NY1 + NY2 $-$ 1, $\cdots$, MY $-$ 1. When ISW=0, MX = NX1 + NX2 $-$ 1 and MY = NY1 + NY2 $-$ 1 should be set. When ISW $\geq$ 1, the calculations can be performed more efficiently by setting a value for MX or MY for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if NX1=NX2=145, then when ISW=0, MX = 289($=17^2$) should be set. However, when ISW $\geq$ 1, it is usually more efficient to set MX = 300($=2^2 \times 3 \times 5^2$), MX = 320($=2^6 \times 5$), MX = 384($=2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** ISW**=1 to calculate the FFT convolution.** However, to conserve work area or if there is a restriction on the method of selecting the parameter MX or MY, the calculations should be performed by setting ISW=0.

(e) To calculate the convolution of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$ be defined as follows:

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

and apply this subroutine to $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$. Let $\hat{p}(k_x, k_y)$ represent the result that was obtained, and the convolution $p(k_x, k_y)$ of the original functions $f(i_x, i_y)$ and $g(j_x, j_y)$ is given as follows:

$$p(k_x, k_y) = \hat{p}(k_x + (i_0 + j_0), k_y)$$

That is, the desired results are obtained if you shift $f(i_x, i_y)$ and $g(j_x, j_y)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete convolution, and then shift the calculated value of the convolution after applying this subroutine by $i_0 + j_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y.

(f) The sampling interval squared multiplied by the discrete convolution calculated by this subroutine is the square approximation (or approximation by using the trapezoidal formula) of the continuous convolution integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous convolution, it is easiest to let $p(n_x^{(f)} + n_x^{(g)} - 1, k_y) = 0$ and $p(k_x, n_y^{(f)} + n_y^{(g)} - 1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ data of $p(k_x, k_y)$ $(k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1;\ k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1)$. In this case, the coordinate (0, 0) element usually is associated with $p(0, 0)$. However,
When ISW=0,
then LX1 = NX1, LY1 = NY1, LX2 = MX, LY2 = MY, and
NWK = NX2 × NY2(when NX2 is odd) or
NWK = (NX2 + 1) × NY2(when NX2 is even)
When ISW ≥ 1,
then LX1=LX2=MX+1 (when MX is odd) or
LX1=LX2=MX+2 (when MX is even),
LY1=LY2=MY, and NWK = MX + (LX1 + 2) × MY.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete convolution.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x, y) = \begin{cases} x_g - x & ((x, y) \in [0, x_g] \times [0, y_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data
Sampling data
$\mathrm{R1}(\mathrm{i_x} + 1, \mathrm{i_y} + 1) = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{NX1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{NY1} - 1)$,
$\mathrm{R2}(\mathrm{j_x} + 1, \mathrm{j_y} + 1) = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{NX2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{NY2} - 1)$.

Here, $\Delta = 0.5$.

NX1, NY1, NX2, NY2, MX, MY and ISW.

(c) Main program

```
      PROGRAM OFCN2D
! *** EXAMPLE OF QFCN2D ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER I,J
      INTEGER NT
      INTEGER ISW,IERR,IWK(40)
      INTEGER NX1,NX2,LX1,LX2,MX
      INTEGER NY1,NY2,LY1,LY2,MY
      INTEGER M0
      PARAMETER (M0 = 8)
      PARAMETER (LX1 = M0+2)
      PARAMETER (LY1 = M0)
      PARAMETER (LX2 = M0+2)
      PARAMETER (LY2 = M0)
      REAL(8) R1(LX1,LY1),R2(LX2,LY2),WK(3*M0+LX2*M0)
      REAL(8) T
      REAL(8) XF,YF,XG,YG,DT
      PARAMETER (DT = 0.5D0)
      PARAMETER (XF = 2.0D0,YF=2.0D0)
      PARAMETER (XG = 2.0D0,YG=2.0D0)
      COMMON IWK,R1,R2,WK
!
      NT=2
      ISW=1
      NX1=XF/DT
      NY1=YF/DT
      NX2=XG/DT
      NY2=YG/DT
      MX=M0
      MY=M0
      WRITE (6,1000) ISW,NX1,NY1,NX2,NY2,MX,MY
      DO 100 J=1,NY1
      DO 101 I=1,NX1
        T=DBLE(I-1)*DT
        R1(I,J)=T
  101 CONTINUE
  100 CONTINUE
      DO 200 J=1,NY2
      DO 201 I=1,NX2
        T=DBLE(I-1)*DT
        R2(I,J)=XG-T
  201 CONTINUE
  200 CONTINUE
      WRITE (6,1100) (I,(R1(I,J),J=1,NY1),I=1,NX1)
      WRITE (6,1150) (I,(R2(I,J),J=1,NY2),I=1,NX2)
      CALL QFCN2D(NX1,NY1,NX2,NY2,R1,LX1,LY1,&
       R2,LX2,LY2,MX,MY,ISW,IWK,WK,NT,IERR)
      WRITE (6,1300)
      WRITE (6,1400) IERR
      WRITE (6,1200)&
       (I,(R2(I,J),J=1,MY),I=1,MX)
 1000 FORMAT(' ',/,/,&
             ' ***  QFCN2D  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'ISW =',I3,/,&
             6X,'(NX1,NY1) =(',I3,',',I3,')',/,&
             6X,'(NX2,NY2) =(',I3,',',I3,')',/,&
             6X,'(MX, MY ) =(',I3,',',I3,')')
 1100 FORMAT(12X,'DATA R1(I,J)',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1150 FORMAT(12X,'DATA R2(I,J)',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR =',I5)
 1200 FORMAT(17X,'CONVOLUTION R2(I,J)',/,&
             10X,'I/J   1     2     3     4      5',&
                          ,      6     7      8',/,&
             10X,'-----------------------------------',&
                   ,'-------------------',/,&
             8(8X,I3,8F7.2,/))
      END
```

(d) Output results

```
***  QFCN2D  ***
 **  INPUT  **
     ISW =  1
     (NX1,NY1) =(  4,  4)
     (NX2,NY2) =(  4,  4)
     (MX, MY ) =(  8,  8)
          DATA R1(I,J)
        I/J   1        2        3        4
        -----------------------------------
```

```
        1   0.0000   0.0000   0.0000   0.0000
        2   0.5000   0.5000   0.5000   0.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   1.5000   1.5000   1.5000   1.5000

          DATA R2(I,J)
        I/J   1        2        3        4
        ------------------------------------
        1   2.0000   2.0000   2.0000   2.0000
        2   1.5000   1.5000   1.5000   1.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   0.5000   0.5000   0.5000   0.5000

**  OUTPUT  **
    IERR =    0
              CONVOLUTION R2(I,J)
        I/J   1      2      3      4      5      6      7      8
        ---------------------------------------------------------
        1   0.00   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
        2   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
        3   2.75   5.50   8.25  11.00   8.25   5.50   2.75  -0.00
        4   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
        5   3.50   7.00  10.50  14.00  10.50   7.00   3.50  -0.00
        6   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
        7   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
        8  -0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00  -0.00
```

6.11.2 QFCN3D, PFCN3D
Three-Dimensional Convolutions

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ of period (m_x, m_y, m_z) satisfying:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ & \quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

for arbitrary integers L_x, L_y, and L_z take nonzero values within their basic periods only for $(i_x, i_y, i_z) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1] \times [0, n_z^{(f)} - 1]$ and $(j_x, j_y, j_z) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1] \times [0, n_z^{(g)} - 1]$. Here, $[0, a] \times [0, b] \times [0, c]$ is the direct product region (region contained in the cube for which the point (0, 0, 0) and the point (a, b, c) are diagonal points) in the space in which the space coordinates (i, j, k) lie. At this time, QFCN3D or PFCN3D calculates the discrete convolution $p(k_x, k_y, k_z)$ defined as follows:

$$\begin{aligned} p(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x - i_x, k_y - i_y, k_z - i_z) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} \sum_{j_z=0}^{m_z-1} g(j_x, j_y, j_z) f(k_x - j_x, k_y - j_y, k_z - j_z) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$, and $m_z = \min(n_z^{(f)} + n_z^{(g)} - 1, M_z)$ and M_x, M_y, and M_z are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, and $M_z \geq \max(n_z^{(f)}, n_z^{(g)})$, respectively. The three-dimensional real Fourier transform of $p(k_x, k_y, k_z)$ can also be obtained.

(2) **Usage**

Double precision:

```
CALL QFCN3D (NX1, NY1, NZ1, NX2, NY2, NZ2, R1, LX1, LY1, LZ1, R2, LX2, LY2, LZ2,
             MX, MY, MZ, ISW, IWK, WK, NT, IERR)
```

Single precision:

```
CALL PFCN3D (NX1, NY1, NZ1, NX2, NY2, NZ2, R1, LX1, LY1, LZ1, R2, LX2, LY2, LZ2,
             MX, MY, MZ, ISW, IWK, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
2	NY1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
3	NZ1	I	1	Input	Number of effective data in i_z direction $n_z^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
4	NX2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
5	NY2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
6	NZ2	I	1	Input	Number of effective data in j_z direction $n_z^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
7	R1	{ D / R }	LX1, LY1, LZ1	Input	Values of discrete function $f(i_x, i_y, i_z)$ (See Note (a))
				Output	When ISW $\geq$ 1, result of three-dimensional real Fourier transform of discrete function $f(i_x, i_y, i_z)$ (period (M_x, M_y, M_z))
8	LX1	I	1	Input	Adjustable dimension of array R1
9	LY1	I	1	Input	Second dimension of array R1
10	LZ1	I	1	Input	Third dimension of array R1
11	R2	{ D / R }	LX2, LY2, LZ2	Input	Values of discrete function $g(j_x, j_y, j_z)$ (See Note (a))
				Output	Value of discrete function $p(k_x, k_y, k_z)$ or its three-dimensional real Fourier transform (See Note (b))
12	LX2	I	1	Input	Adjustable dimension of array R2
13	LY2	I	1	Input	Second dimension of array R2
14	LZ2	I	1	Input	Third dimension of array R2
15	MX	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))
16	MY	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))
17	MZ	I	1	Input	Parameter M_z corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))

No.	Argument	Type	Size	Input/ Output	Contents
18	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0: Calculate the convolution according to the definition. ISW= 1: Calculate the convolution according to the FFT method. ISW= 2: Calculate the real Fourier transform of the convolution.
19	IWK	I	See Contents	Work	Work area **Size**: 0 (When ISW= 0) 60 (When ISW $\geq$ 1)
20	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: (NX2 + 1) $\times$ (NY2 + 1) $\times$ NZ2 (When ISW= 0, NX2 is even and NY2 is even) NX2 $\times$ (NY2 + 1) $\times$ NZ2 (When ISW= 0, NX2 is odd and NY2 is even) (NX2 + 1) $\times$ NY2 $\times$ NZ2 (When ISW= 0, NX2 is even and NY2 is odd) NX2 $\times$ NY2 $\times$ NZ2 (When ISW= 0, NX2 is odd and NY2 is odd) MX + 2 $\times$ (MY + MZ) + MAX(LX1 $\times$ LY1 $\times$ LZ1, LX2 $\times$ LY2 $\times$ LZ2) (When ISW $\geq$ 1)
21	NT	I	1	Input	Number of tasks to be generated
22	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, 1, 2\}$

(b) NX1 > 1 and NY1 > 1 and NZ1 > 1

(c) NX2 > 1 and NY2 > 1 and NZ2 > 1

(d) MX $\geq$ MAX(NX1, NX2) and MY $\geq$ MAX(NY1, NY2) and MZ $\geq$ MAX(NZ1, NZ2)

(e) LX1 $\geq$ NX1 and LY1 $\geq$ NY1 and LZ1 $\geq$ NZ1 (when ISW=0)
LX1 $\geq$ MX + 1 and LX1 is even and LY1 $\geq$ MY and LZ1 $\geq$ MZ (when ISW $\geq$ 1 and MX is odd)
LX1 $\geq$ MX + 2 and LX1 is even and LY1 $\geq$ MY and LZ1 $\geq$ MZ (when ISW $\geq$ 1 and MX is even)

(f) LX2 $\geq$ MX and LY2 $\geq$ NY2 and LZ2 $\geq$ NZ2 (when ISW=0)
LX2 $\geq$ MX + 1 and LX2 is even and LY2 $\geq$ MY and LZ2 $\geq$ MZ (when ISW $\geq$ 1 and MX is odd)
LX2 $\geq$ MX + 2 and LX2 is even and LY2 $\geq$ MY and LZ2 $\geq$ MZ (when ISW $\geq$ 1 and MX is even)

(g) NT $\geq$ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	MX $<$ NX1 + NX2 $-$ 1 or MY $<$ NY1 + NY2 $-$ 1 or MZ $<$ NZ1 + NZ2 $-$ 1	Overlapping occurred during the convolution calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ and the elements of arrays R1 and R2 are associated as follows.

$$\begin{aligned} f(i_x, i_y, i_z) &\leftrightarrow \mathrm{R1(i_x + 1, i_y + 1, i_z + 1)} \\ g(j_x, j_y, j_z) &\leftrightarrow \mathrm{R2(j_x + 1, j_y + 1, j_z + 1)} \end{aligned}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$; $i_z = 0, \cdots, n_z^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$; $j_z = 0, \cdots, n_z^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** R1 **and** R2 **should be set so that** LX1**/2,** LY1**,** LZ1**,** LX2**/2,** LY2**, and** LZ2 **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** R1 **and** R2**. Usually, when** MX**, for example, is (a multiple of 4)+2,** LX1=MX**+4 is set.**

(b) The values of the discrete convolution $p(k_x, k_y, k_z)$ and the elements of array R2 are associated as follows.

$$p(k_x, k_y, k_z) \leftrightarrow \mathrm{R2(k_x + 1, k_y + 1, k_z + 1)}$$

Here, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$; $k_z = 0, \cdots, M_z - 1$. When ISW=2 is set to obtain the three-dimensional real Fourier transform $P(j_x, j_y, j_z)$ of the discrete convolution $p(k_x, k_y, k_z)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$\begin{aligned} P(j_x, j_y, j_z) &= \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} p(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor) \end{aligned}$$

the following associations are made:

$$\begin{aligned} \Re\{P(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{R2(2 * j_x + 1, j_y + 1, j_z + 1)} \\ \Im\{P(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{R2(2 * j_x + 2, j_y + 1, j_z + 1)} \end{aligned}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$\begin{aligned} P(M_x - j_x, M_y - j_y, M_z - j_z)^* &= P(j_x, j_y, j_z) \\ P(M_x - j_x, j_y, j_z)^* &= P(j_x, M_y - j_y, M_z - j_z) \\ P(M_x - j_x, M_y - j_y, j_z)^* &= P(j_x, j_y, M_z - j_z) \end{aligned}$$

(Here, z^* represents the conjugate complex number of the complex number z.)

(c) If MX $\geq$ NX1 + NX2 − 1 and MY $\geq$ NY1 + NY2 − 1 and MZ $\geq$ NZ1 + NZ2 − 1 are set, the convolution can be calculated without causing an overlap with the convolution of the next period. When MX > NX1 + NX2 − 1 or MY > NY1 + NY2 − 1 or MZ > NZ1 + NZ2 − 1, the following correspondences are made:

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{R2}(\mathrm{k_x}+1, \mathrm{k_y}+1, \mathrm{k_z}+1)$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{NX1}+\mathrm{NX2}-1, \cdots, \mathrm{MX}-1$; $k_y = 0, \cdots, \mathrm{MY}-1$; $k_z = 0, \cdots, \mathrm{MZ}-1$ or $k_x = 0, \cdots, \mathrm{MX}-1$; $k_y = \mathrm{NY1}+\mathrm{NY2}-1, \cdots, \mathrm{MY}-1$; $k_z = 0, \cdots, \mathrm{MZ}-1$ or $k_x = 0, \cdots, \mathrm{MX}-1$; $k_y = 0, \cdots, \mathrm{MY}-1$; $k_z = \mathrm{NZ1}+\mathrm{NZ2}-1, \cdots, \mathrm{MZ}-1$. When ISW=0, MX = NX1 + NX2 − 1, MY = NY1 + NY2 − 1, and MZ = NZ1 + NZ2 − 1 should be set. When ISW $\geq$ 1, the calculations can be performed more efficiently by setting a value for MX, MY or MZ for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if NX1=NX2=145, then when ISW=0, MX = 289(=17^2) should be set. However, when ISW $\geq$ 1, it is usually more efficient to set MX = 300(=$2^2 \times 3 \times 5^2$), MX = 320(=$2^6 \times 5$), MX = 384(=$2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting ISW=1 to calculate the FFT convolution.** However, to conserve work area or if there is a restriction on the method of selecting the parameter MX, MY or MZ, the calculations should be performed by setting ISW=0.

(e) To calculate the convolution of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$ be defined as follows:

$$\hat{f}(i_x, i_y, i_z) = f(i_x - i_0, i_y, i_z), \quad \hat{g}(j_x, j_y, j_z) = g(j_x - j_0, j_y, j_z)$$

and apply this subroutine to $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$. Let $\hat{p}(k_x, k_y, k_z)$ represent the result that was obtained, and the convolution $p(k_x, k_y, k_z)$ of the original functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ is given as follows:

$$p(k_x, k_y, k_z) = \hat{p}(k_x + (i_0 + j_0), k_y, k_z)$$

That is, the desired results are obtained if you shift $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete convolution, and then shift the calculated value of the convolution after applying this subroutine by $i_0 + j_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y and i_z, j_z, and k_z.

(f) The sampling interval cubed multiplied by the discrete convolution calculated by this subroutine is the square approximation (or approximation by using the trapezoidal formula) of the continuous convolution integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous convolution, it is easiest to let $p(n_x^{(f)} + n_x^{(g)} - 1, k_y, k_z) = 0$, $p(k_x, n_y^{(f)} + n_y^{(g)} - 1, k_z) = 0$, and $p(k_x, k_y, n_z^{(f)} + n_z^{(g)} - 1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})(n_z^{(f)} + n_z^{(g)})$ data of $p(k_x, k_y, k_z)$ ($k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1$; $k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1$; $k_z = 0, 1, \cdots, n_z^{(f)} + n_z^{(g)} - 1$). In this case, the coordinate (0, 0, 0) element usually is associated with $p(0, 0, 0)$. However,
When ISW=0,
then LX1 = NX1, LY1 = NY1, LZ1 = NZ1, LX2 = MX, LY2 = MY, LZ2 = MZ, and
NWK = (NX2 + 1) × (NY2 + 1) × NZ2 (when NX2 is even and NY2 is even) or

NWK = NX2 × (NY2 + 1) × NZ2 (when NX2 is odd and NY2 is even) or
NWK = (NX2 + 1) × NY2 × NZ2 (when NX2 is even and NY2 is odd) or
NWK = NX2 × NY2 × NZ2 (when NX2 is odd and NY2 is odd)
When ISW ≥ 1
LX1=LX2=MX+1 (when MX is odd) or
LX1=LX2=MX+2 (when MX is even),
LY1=LY2=MY, LZ1=LZ2=MZ, and NWK = MX + 2 × (MY + MZ) + LX1 × MY × MZ.

(7) **Example**

(a) Problem

Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete convolution.

$$f(x,y,z) = \begin{cases} x & ((x,y,z) \in [0,x_f] \times [0,y_f] \times [0,z_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x,y,z) = \begin{cases} x_g - x & ((x,y,z) \in [0,x_g] \times [0,y_g] \times [0,z_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data

Sampling data
R1($i_x + 1, i_y + 1, i_z + 1$) = f($i_x\Delta, i_y\Delta, i_z\Delta$) ($i_x = 0, 1, \cdots, NX1 - 1$; $i_y = 0, 1, \cdots, NY1 - 1$; $i_z = 0, 1, \cdots, NZ1 - 1$),
R2($j_x + 1, j_y + 1, j_z + 1$) = g($j_x\Delta, j_y\Delta, j_z\Delta$) ($j_x = 0, 1, \cdots, NX2 - 1$; $j_y = 0, 1, \cdots, NY2 - 1$; $j_z = 0, 1, \cdots, NZ2 - 1$).
Here, $\Delta = 0.5$.
NX1, NY1, NZ1, NX2, NY2, NZ2, MX, MY, MZ and ISW.

(c) Main program

```
      PROGRAM OFCN3D
! *** EXAMPLE OF QFCN3D ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER I,J,K
      INTEGER NT
      INTEGER ISW,IERR,IWK(60)
      INTEGER NX1,NX2,LX1,LX2,MX
      INTEGER NY1,NY2,LY1,LY2,MY
      INTEGER NZ1,NZ2,LZ1,LZ2,MZ
      INTEGER M0
      PARAMETER (M0 = 8)
      PARAMETER (LX1 = (M0+2)/2*2)
      PARAMETER (LY1 = M0)
      PARAMETER (LZ1 = M0)
      PARAMETER (LX2 = LX1)
      PARAMETER (LY2 = LY1)
      PARAMETER (LZ2 = LZ1)
      REAL(8) R1(LX1,LY1,LZ1),R2(LX2,LY2,LZ2)
      REAL(8) WK(5*M0+LX1*M0*M0)
      REAL(8) T
      REAL(8) XF,YF,ZF,XG,YG,ZG,DT
      PARAMETER (DT = 0.5D0)
      PARAMETER (XF = 2.0D0,YF=2.0D0,ZF=2.0D0)
      PARAMETER (XG = 2.0D0,YG=2.0D0,ZG=2.0D0)
      COMMON IWK,R1,R2,WK
!
      NT=2
      ISW=1
      NX1=XF/DT
      NY1=YF/DT
      NZ1=ZF/DT
      NX2=XG/DT
      NY2=YG/DT
      NZ2=ZG/DT
      MX=M0
      MY=M0
      MZ=M0
      WRITE (6,1000) ISW,NX1,NY1,NZ1,NX2,NY2,NZ2,MX,MY,MZ
      DO 100 K=1,NZ1
```

```
      DO 101 J=1,NY1
      DO 102 I=1,NX1
        T=DBLE(I-1)*DT
        R1(I,J,K)=T
  102 CONTINUE
  101 CONTINUE
  100 CONTINUE
      DO 200 K=1,NZ2
      DO 201 J=1,NY2
      DO 202 I=1,NX2
        T=DBLE(I-1)*DT
        R2(I,J,K)=XG-T
  202 CONTINUE
  201 CONTINUE
  200 CONTINUE
      DO 300 K=1,NZ1
      WRITE (6,1100) K,(I,(R1(I,J,K),J=1,NY1),I=1,NX1)
  300 CONTINUE
      DO 400 K=1,NZ2
      WRITE (6,1150) K,(I,(R2(I,J,K),J=1,NY2),I=1,NX2)
  400 CONTINUE
      CALL QFCN3D(NX1,NY1,NZ1,NX2,NY2,NZ2,R1,LX1,LY1,LZ1,&
       R2,LX2,LY2,LZ2,MX,MY,MZ,ISW,IWK,WK,NT,IERR)
      WRITE (6,1300)
      WRITE (6,1400) IERR
      DO 500 K=1,MZ
      WRITE (6,1200) K,(I,(R2(I,J,K),J=1,MY),I=1,MX)
  500 CONTINUE
 1000 FORMAT(' ',/,/,&
             ' ***  QFCN3D  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'ISW =',I3,/,&
             6X,'(NX1,NY1,NZ1) =(',I3,',',I3,',',I3,')',/,&
             6X,'(NX2,NY2,NZ2) =(',I3,',',I3,',',I3,')',/,&
             6X,'(MX ,MY ,MZ ) =(',I3,',',I3,',',I3,')')
 1100 FORMAT(12X,'DATA R1(I,J,',I3,')',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1150 FORMAT(12X,'DATA R2(I,J,',I3,')',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR =',I5)
 1200 FORMAT(17X,'CONVOLUTION R2(I,J,',I3,')',/,&
             10X,'I/J   1      2      3      4      5',&
                       ',      6      7      8',/,&
             10X,'-----------------------------------',&
                   '--------------------',/,&
             8(8X,I3,8F7.2,/))
      END
```

(d) Output results

```
***  QFCN3D  ***
 **  INPUT  **
     ISW =  1
     (NX1,NY1,NZ1) =(  4,  4,  4)
     (NX2,NY2,NZ2) =(  4,  4,  4)
     (MX ,MY ,MZ ) =(  8,  8,  8)
          DATA R1(I,J,  1)
        I/J   1        2        3        4
        -----------------------------------
        1   0.0000   0.0000   0.0000   0.0000
        2   0.5000   0.5000   0.5000   0.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   1.5000   1.5000   1.5000   1.5000

          DATA R1(I,J,  2)
        I/J   1        2        3        4
        -----------------------------------
        1   0.0000   0.0000   0.0000   0.0000
        2   0.5000   0.5000   0.5000   0.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   1.5000   1.5000   1.5000   1.5000

          DATA R1(I,J,  3)
        I/J   1        2        3        4
        -----------------------------------
        1   0.0000   0.0000   0.0000   0.0000
        2   0.5000   0.5000   0.5000   0.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   1.5000   1.5000   1.5000   1.5000

          DATA R1(I,J,  4)
        I/J   1        2        3        4
        -----------------------------------
        1   0.0000   0.0000   0.0000   0.0000
        2   0.5000   0.5000   0.5000   0.5000
        3   1.0000   1.0000   1.0000   1.0000
        4   1.5000   1.5000   1.5000   1.5000

          DATA R2(I,J,  1)
```

```
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

  DATA R2(I,J,  2)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

  DATA R2(I,J,  3)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

  DATA R2(I,J,  4)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

**  OUTPUT  **
    IERR =    0
          CONVOLUTION R2(I,J,  1)
I/J   1      2      3      4      5      6      7      8
-----------------------------------------------------------
1   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00   0.00
2   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
3   2.75   5.50   8.25  11.00   8.25   5.50   2.75   0.00
4   5.00  10.00  15.00  20.00  15.00  10.00   5.00   0.00
5   3.50   7.00  10.50  14.00  10.50   7.00   3.50   0.00
6   2.00   4.00   6.00   8.00   6.00   4.00   2.00   0.00
7   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
8   0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00   0.00

          CONVOLUTION R2(I,J,  2)
I/J   1      2      3      4      5      6      7      8
-----------------------------------------------------------
1   0.00   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
2   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
3   5.50  11.00  16.50  22.00  16.50  11.00   5.50  -0.00
4  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
5   7.00  14.00  21.00  28.00  21.00  14.00   7.00  -0.00
6   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
7   1.50   3.00   4.50   6.00   4.50   3.00   1.50  -0.00
8   0.00  -0.00  -0.00   0.00  -0.00  -0.00  -0.00  -0.00

          CONVOLUTION R2(I,J,  3)
I/J   1      2      3      4      5      6      7      8
-----------------------------------------------------------
1   0.00   0.00   0.00   0.00   0.00   0.00   0.00  -0.00
2   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
3   8.25  16.50  24.75  33.00  24.75  16.50   8.25  -0.00
4  15.00  30.00  45.00  60.00  45.00  30.00  15.00  -0.00
5  10.50  21.00  31.50  42.00  31.50  21.00  10.50  -0.00
6   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
7   2.25   4.50   6.75   9.00   6.75   4.50   2.25  -0.00
8   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00  -0.00

          CONVOLUTION R2(I,J,  4)
I/J   1      2      3      4      5      6      7      8
-----------------------------------------------------------
1   0.00   0.00   0.00   0.00   0.00   0.00  -0.00   0.00
2   4.00   8.00  12.00  16.00  12.00   8.00   4.00   0.00
3  11.00  22.00  33.00  44.00  33.00  22.00  11.00  -0.00
4  20.00  40.00  60.00  80.00  60.00  40.00  20.00  -0.00
5  14.00  28.00  42.00  56.00  42.00  28.00  14.00  -0.00
6   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
7   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
8  -0.00  -0.00   0.00  -0.00  -0.00   0.00  -0.00   0.00

          CONVOLUTION R2(I,J,  5)
I/J   1      2      3      4      5      6      7      8
-----------------------------------------------------------
1   0.00  -0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
2   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
3   8.25  16.50  24.75  33.00  24.75  16.50   8.25  -0.00
4  15.00  30.00  45.00  60.00  45.00  30.00  15.00  -0.00
5  10.50  21.00  31.50  42.00  31.50  21.00  10.50   0.00
6   6.00  12.00  18.00  24.00  18.00  12.00   6.00   0.00
7   2.25   4.50   6.75   9.00   6.75   4.50   2.25  -0.00
8  -0.00  -0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00

          CONVOLUTION R2(I,J,  6)
I/J   1      2      3      4      5      6      7      8
```

1	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	0.00
2	2.00	4.00	6.00	8.00	6.00	4.00	2.00	0.00
3	5.50	11.00	16.50	22.00	16.50	11.00	5.50	-0.00
4	10.00	20.00	30.00	40.00	30.00	20.00	10.00	-0.00
5	7.00	14.00	21.00	28.00	21.00	14.00	7.00	-0.00
6	4.00	8.00	12.00	16.00	12.00	8.00	4.00	-0.00
7	1.50	3.00	4.50	6.00	4.50	3.00	1.50	-0.00
8	0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	0.00

CONVOLUTION R2(I,J, 7)

I/J	1	2	3	4	5	6	7	8
1	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
2	1.00	2.00	3.00	4.00	3.00	2.00	1.00	-0.00
3	2.75	5.50	8.25	11.00	8.25	5.50	2.75	-0.00
4	5.00	10.00	15.00	20.00	15.00	10.00	5.00	-0.00
5	3.50	7.00	10.50	14.00	10.50	7.00	3.50	-0.00
6	2.00	4.00	6.00	8.00	6.00	4.00	2.00	-0.00
7	0.75	1.50	2.25	3.00	2.25	1.50	0.75	-0.00
8	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00

CONVOLUTION R2(I,J, 8)

I/J	1	2	3	4	5	6	7	8
1	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
2	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
3	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
4	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
5	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
6	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
7	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
8	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00

6.12 CORRELATIONS

6.12.1 QFCR2D, PFCR2D
Two-Dimensional Correlations

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ of period (m_x, m_y) satisfying:

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

for arbitrary integers L_x and L_y take nonzero values within their basic periods only for $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$ and $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$. Here, $[0, a] \times [0, b]$ is the direct product region (region contained in the square for which the point (0, 0) and the point (a, b) are diagonal points) on the plane in which the plane coordinates (i, j) lie. At this time, QFCR2D or PFCR2D calculates the quantity $\tilde{q}(k_x, k_y)$ obtained by shifting the discrete correlation $q(k_x, k_y)$, which is defined as follows:

$$\begin{aligned} q(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x + i_x, k_y + i_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

by $(n_x^{(f)} - 1, n_y^{(f)} - 1)$ in the positive direction for (k_x, k_y), respectively. $\tilde{q}(k_x, k_y)$ is defined as follows:

$$\begin{aligned} \tilde{q}(k_x, k_y) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1)) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$ and $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ and M_x and M_y are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$ and $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, respectively. The two-dimensional real Fourier transform of $q(k_x, k_y)$ can also be obtained.

(2) **Usage**

Double precision:

```
CALL QFCR2D  (NX1, NY1, NX2, NY2, R1, LX1, LY1, R2, LX2, LY2, MX, MY, ISW, IWK,
              WK, NT, IERR)
```

Single precision:

```
CALL PFCR2D  (NX1, NY1, NX2, NY2, R1, LX1, LY1, R2, LX2, LY2, MX, MY, ISW, IWK,
              WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y)$
2	NY1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y)$
3	NX2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y)$
4	NY2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y)$
5	R1	{D / R}	LX1, LY1	Input	Values of discrete function $f(i_x, i_y)$ (See Note (a))
				Output	When ISW ≥ 1, result of two-dimensional real Fourier transform of discrete function $f(i_x, i_y)$ (period (M_x, M_y))
6	LX1	I	1	Input	Adjustable dimension of array R1
7	LY1	I	1	Input	Second dimension of array R1
8	R2	{D / R}	LX2, LY2	Input	Values of discrete function $g(j_x, j_y)$ (See Note (a))
				Output	Value of discrete function $\tilde{q}(k_x, k_y)$ or the two-dimensional real Fourier transform of $q(k_x, k_y)$ (See Note (b))
9	LX2	I	1	Input	Adjustable dimension of array R2
10	LY2	I	1	Input	Second dimension of array R2
11	MX	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $\tilde{q}(k_x, k_y)$ (See Note (c))
12	MY	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $\tilde{q}(k_x, k_y)$ (See Note (c))
13	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0: Calculate the correlation according to the definition. ISW= 1: Calculate the correlation according to the FFT method. ISW= 2: Calculate the real Fourier transform of the correlation.

No.	Argument	Type	Size	Input/ Output	Contents
14	IWK	I	See Contents	Work	Work area **Size**: 0 (When ISW= 0) 40 (When ISW $\geq$ 1)
15	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: NX2 $\times$ NY2 ((When ISW= 0 and NX2 is odd) (NX2 + 1) $\times$ NY2 (When ISW= 0 and NX2 is even) MX + 2 $\times$ MY + MAX(LX1 $\times$ LY1, LX2 $\times$ LY2) (When ISW $\geq$ 1)
16	NT	I	1	Input	Number of tasks to be generated
17	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, 1, 2\}$

(b) NX1 > 1 and NY1 > 1

(c) NX2 > 1 and NY2 > 1

(d) MX $\geq$ MAX(NX1, NX2) and MY $\geq$ MAX(NY1, NY2)

(e) LX1 $\geq$ NX1 and LY1 $\geq$ NY1 (When ISW=0)
LX1 $\geq$ MX + 1 and LY1 $\geq$ MY (When ISW $\geq$ 1 and MX is odd)
LX1 $\geq$ MX + 2 and LY1 $\geq$ MY (When ISW $\geq$ 1 and MX is even)

(f) LX2 $\geq$ MX and LY2 $\geq$ MY (When ISW=0)
LX2 $\geq$ MX + 1 and LY2 $\geq$ MY (When ISW $\geq$ 1 and MX is odd)
LX2 $\geq$ MX + 2 and LY2 $\geq$ MY (When ISW $\geq$ 1 and MX is even)

(g) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	MX $<$ NX1 + NX2 $-$ 1 or MY $<$ NY1 + NY2 $-$ 1	Overlapping occurred during the correlation calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ and the elements of arrays R1 and R2 are associated as follows.

$$\begin{aligned} f(i_x, i_y) &\leftrightarrow \mathrm{R1(i_x + 1, i_y + 1)} \\ g(j_x, j_y) &\leftrightarrow \mathrm{R2(j_x + 1, j_y + 1)} \end{aligned}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** R1 **and** R2 **should be set so that** LX1**/2,** LY1**,** LX2**/2, and** LY2 **are odd numbers to avoid bank conflict of main memory. Usually, when** MX**, for example, is a multiple of 4,** LX1=MX**+3 is set.**

(b) The values of the discrete correlation $\tilde{q}(k_x, k_y)$ and the elements of array R2 are associated as follows.

$$\tilde{q}(k_x, k_y) \leftrightarrow \mathrm{R2(k_x + 1, k_y + 1)}$$

Here, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$. When ISW=2 is set to obtain the two-dimensional real Fourier transform $Q(j_x, j_y)$ of the discrete correlation $q(k_x, k_y)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$\begin{aligned} Q(j_x, j_y) &= \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} q(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor) \end{aligned}$$

the following associations are made:

$$\begin{aligned} \Re\{Q(j_x, j_y)\} &\leftrightarrow \mathrm{R2(2 * j_x + 1, j_y + 1)} \\ \Im\{Q(j_x, j_y)\} &\leftrightarrow \mathrm{R2(2 * j_x + 2, j_y + 1)} \end{aligned}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$\begin{aligned} Q(M_x - j_x, M_y - j_y)^* &= Q(j_x, j_y) \\ Q(M_x - j_x, j_y)^* &= Q(j_x, M_y - j_y) \end{aligned}$$

(Here, z^* represents the conjugate complex number of the complex number z.) Now, $Q(j_x, j_y)$ can be thought of as an estimate of the cross spectrum of the original two functions for which the correlation is to be calculated. In this case, M_x and M_y should be thought of as $M_x = n_x^{(f)} + n_x^{(g)}$ and $M_y = n_y^{(f)} + n_y^{(g)}$. In particular, if the original two functions for which the correlation is to be calculated are the same function, $Q(j_x, j_y)$ corresponds to the raw Fourier periodogram (estimate of the power spectrum), and $Q(j_x, j_y)$ is a real number.

(c) If MX $\geq$ NX1 + NX2 $-$ 1 and MY $\geq$ NY1 + NY2 $-$ 1 are set, the correlation can be calculated without causing an overlap with the correlation of the next period. When MX $>$ NX1 + NX2 $-$ 1 or MY $>$ NY1 + NY2 $-$ 1, the following correspondences are made:

$$\tilde{q}(k_x, k_y) \leftrightarrow \mathrm{R2(k_x + 1, k_y + 1)}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{NX1 + NX2 - 1}, \cdots, \mathrm{MX - 1}$; $k_y = 0, \cdots, \mathrm{MY - 1}$ or $k_x = 0, \cdots, \mathrm{MX - 1}$; $k_y = \mathrm{NY1 + NY2 - 1}, \cdots, \mathrm{MY - 1}$. When ISW=0, MX = NX1 + NX2 $-$ 1 and MY = NY1 + NY2 $-$ 1 should be set. When ISW $\geq$ 1, the calculations can be performed more efficiently by setting a value for MX or MY for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if NX1=NX2=145, then when ISW=0, MX = 289($=17^2$) should be set. However, when ISW $\geq$ 1, it is usually more efficient to set MX = 300($=2^2 \times 3 \times 5^2$), MX = 320($=2^6 \times 5$), MX = 384($=2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** ISW=**1 to calculate the FFT correlation.** However, to conserve work area or if there is a restriction on the method of selecting the parameter MX or MY, the calculations should be performed by setting ISW=0.

(e) To calculate the correlation of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$ be defined as follows:

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

and apply this subroutine to $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$. Let $\tilde{q}(k_x, k_y)$ represent the result that was obtained, and the correlation $q(k_x, k_y)$ of the original functions $f(i_x, i_y)$ and $g(j_x, j_y)$ is given as follows:

$$q(k_x, k_y) = \tilde{q}(k_x - (j_0 - i_0) + (n_x^{(f)} - 1), k_y)$$

Therefore, even when $i_0 = j_0 = 0$, to consider the correlation $q(k_x, k_y)$ that conforms to the normal definition, you must consider shifting the result by $n_x^{(f)} - 1$ in the negative direction of k_x after applying this subroutine or if you shift $f(i_x, i_y)$ and $g(j_x, j_y)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete correlation, you must then shift the calculation result again by $j_0 - i_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y.

(f) The sampling interval squared multiplied by the discrete correlation calculated by this subroutine is the square approximation (or approximation by using the trapezoidal formula) of the continuous correlation integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous correlation, it is easiest to let $q(-n_x^{(f)}, k_y) = \tilde{q}(-1, k_y) = 0$ and $q(k_x, -n_y^{(f)}) = \tilde{q}(k_x, -1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ data of $q(k_x, k_y)$ ($k_x = -n_x^{(f)}, \cdots, -1, 0, 1, \cdots, n_x^{(g)} - 1$; $k_y = -n_y^{(f)}, \cdots, -1, 0, 1, \cdots, n_y^{(g)} - 1$). Of course, this is the same as letting $q(n_x^{(f)} + n_x^{(g)}, k_y) = \tilde{q}(n_x^{(g)}, k_y) = 0$ and $q(k_x, n_y^{(f)} + n_y^{(g)}) = \tilde{q}(k_x, n_y^{(g)}) = 0$ and considering $q(k_x, k_y)$ ($k_x = -(n_x^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_x^{(g)}$; $k_y = -(n_y^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_y^{(g)}$). In this case, the coordinate (0, 0) element usually is associated with $q(0, 0)$. However,
When ISW=0,
then LX1 = NX1, LY1 = NY1, LX2 = MX, LY2 = MY, and
NWK = NX2 × NY2(when NX2 is odd) or
NWK = (NX2 + 1) × NY2(when NX2 is even)
When ISW ≥ 1,
then LX1=LX2=MX+1 (when MX is odd) or
LX1=LX2=MX+2 (when MX is even),
LY1=LY2=MY, and NWK = MX + (LX1 + 2) × MY.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete correlation.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x,y) = \begin{cases} x_g - x & ((x,y) \in [0,x_g] \times [0,y_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data

Sampling data

$\mathrm{R1}(\mathrm{i_x}+1, \mathrm{i_y}+1) = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{NX1}-1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{NY1}-1)$,

$\mathrm{R2}(\mathrm{j_x}+1, \mathrm{j_y}+1) = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{NX2}-1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{NY2}-1)$.

Here, $\Delta = 0.5$.

NX1, NY1, NX2, NY2, MX, MY and ISW.

(c) Main program

```
      PROGRAM OFCR2D
! *** EXAMPLE OF QFCR2D ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER I,J
      INTEGER NT
      INTEGER ISW,IERR,IWK(40)
      INTEGER NX1,NX2,LX1,LX2,MX
      INTEGER NY1,NY2,LY1,LY2,MY
      INTEGER M0
      PARAMETER (M0 = 8)
      PARAMETER (LX1 = M0+2)
      PARAMETER (LY1 = M0)
      PARAMETER (LX2 = M0+2)
      PARAMETER (LY2 = M0)
      REAL(8) R1(LX1,LY1),R2(LX2,LY2),WK(3*M0+LX2*M0)
      REAL(8) T
      REAL(8) XF,YF,XG,YG,DT
      PARAMETER (DT = 0.5D0)
      PARAMETER (XF = 2.0D0,YF=2.0D0)
      PARAMETER (XG = 2.0D0,YG=2.0D0)
      COMMON IWK,R1,R2,WK
!
      NT=2
      ISW=1
      NX1=XF/DT
      NY1=YF/DT
      NX2=XG/DT
      NY2=YG/DT
      MX=M0
      MY=M0
      WRITE (6,1000) ISW,NX1,NY1,NX2,NY2,MX,MY
      DO 100 J=1,NY1
      DO 101 I=1,NX1
        T=DBLE(I-1)*DT
        R1(I,J)=T
  101 CONTINUE
  100 CONTINUE
      DO 200 J=1,NY2
      DO 201 I=1,NX2
        T=DBLE(I-1)*DT
        R2(I,J)=XG-T
  201 CONTINUE
  200 CONTINUE
      WRITE (6,1100) (I,(R1(I,J),J=1,NY1),I=1,NX1)
      WRITE (6,1150) (I,(R2(I,J),J=1,NY2),I=1,NX2)
      CALL QFCR2D(NX1,NY1,NX2,NY2,R1,LX1,LY1,&
       R2,LX2,LY2,MX,MY,ISW,IWK,WK,NT,IERR)
      WRITE (6,1300)
      WRITE (6,1400) IERR
      WRITE (6,1200)&
       (I,(R2(I,J),J=1,MY),I=1,MX)
 1000 FORMAT(' ',/,/,&
             ' ***  QFCR2D  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'ISW =',I3,/,&
             6X,'(NX1,NY1) =(',I3,',',I3,')',/,&
             6X,'(NX2,NY2) =(',I3,',',I3,')',/,&
             6X,'(MX, MY ) =(',I3,',',I3,')')
 1100 FORMAT(12X,'DATA R1(I,J)',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1150 FORMAT(12X,'DATA R2(I,J)',/,&
             10X,'I/J   1        2        3        4',/,&
             10X,'-----------------------------------',/,&
             6(8X,I3,4F9.4,/))
 1300 FORMAT(2X,'**  OUTPUT  **')
 1400 FORMAT(6X,'IERR =',I5)
 1200 FORMAT(17X,'CORRELATION R2(I,J)',/,&
             10X,'I/J   1      2      3      4      5',&
                           '      6      7      8',/,&
             10X,'-----------------------------------',&
                    '--------------------',/,&
             8(8X,I3,8F7.2,/))
      END
```

(d) Output results

```
***  QFCR2D  ***
 **  INPUT  **
     ISW =  1
     (NX1,NY1) =(  4,  4)
     (NX2,NY2) =(  4,  4)
     (MX, MY ) =(  8,  8)
           DATA R1(I,J)
         I/J   1        2        3        4
         ------------------------------------
         1   0.0000   0.0000   0.0000   0.0000
         2   0.5000   0.5000   0.5000   0.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   1.5000   1.5000   1.5000   1.5000

           DATA R2(I,J)
         I/J   1        2        3        4
         ------------------------------------
         1   2.0000   2.0000   2.0000   2.0000
         2   1.5000   1.5000   1.5000   1.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   0.5000   0.5000   0.5000   0.5000

 **  OUTPUT  **
     IERR =    0
              CORRELATION R2(I,J)
         I/J   1      2      3      4      5      6      7      8
         ---------------------------------------------------------
         1   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
         2   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
         3   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
         4   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
         5   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
         6   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
         7   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
         8   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
```

6.12.2 QFCR3D, PFCR3D
Three-Dimensional Correlations

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ of period (m_x, m_y, m_z) satisfying:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ & \quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

for arbitrary integers L_x, L_y, and L_z take nonzero values within their basic periods only for $(i_x, i_y, i_z) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1] \times [0, n_z^{(f)} - 1]$ and $(j_x, j_y, j_z) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1] \times [0, n_z^{(g)} - 1]$. Here, $[0, a] \times [0, b] \times [0, c]$ is the direct product region (region contained in the cube for which the point (0, 0, 0) and the point (a, b, c) are diagonal points) in the space in which the space coordinates (i, j, k) lie. At this time, QFCR3D or PFCR3D calculates the quantity $\tilde{q}(k_x, k_y, k_z)$ obtained by shifting the discrete correlation $q(k_x, k_y, k_z)$, which is defined as follows:

$$\begin{aligned} q(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x + i_x, k_y + i_y, k_z + i_z) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

by $(n_x^{(f)} - 1, n_y^{(f)} - 1, n_z^{(f)} - 1)$ in the positive direction for (k_x, k_y, k_z), respectively. $\tilde{q}(k_x, k_y, k_z)$ is defined as follows:

$$\begin{aligned} \tilde{q}(k_x, k_y, k_z) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1), k_z - (n_z^{(f)} - 1)) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$, and $m_z = \min(n_z^{(f)} + n_z^{(g)} - 1, M_z)$ and M_x, M_y, and M_z are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, and $M_z \geq \max(n_z^{(f)}, n_z^{(g)})$, respectively. The three-dimensional real Fourier transform of $q(k_x, k_y, k_z)$ can also be obtained.

(2) **Usage**

Double precision:

```
CALL QFCR3D  (NX1, NY1, NZ1, NX2, NY2, NZ2, R1, LX1, LY1, LZ1, R2, LX2, LY2, LZ2,
              MX, MY, MZ, ISW, IWK, WK, NT, IERR)
```

Single precision:

```
CALL PFCR3D  (NX1, NY1, NZ1, NX2, NY2, NZ2, R1, LX1, LY1, LZ1, R2, LX2, LY2, LZ2,
              MX, MY, MZ, ISW, IWK, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
2	NY1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
3	NZ1	I	1	Input	Number of effective data in i_z direction $n_z^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
4	NX2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
5	NY2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
6	NZ2	I	1	Input	Number of effective data in j_z direction $n_z^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
7	R1	{D / R}	LX1, LY1, LZ1	Input	Values of discrete function $f(i_x, i_y, i_z)$ (See Note (a))
				Output	When ISW $\geq$ 1, result of three-dimensional real Fourier transform of discrete function $f(i_x, i_y, i_z)$ (period (M_x, M_y, M_z))
8	LX1	I	1	Input	Adjustable dimension of array R1
9	LY1	I	1	Input	Second dimension of array R1
10	LZ1	I	1	Input	Third dimension of array R1
11	R2	{D / R}	LX2, LY2, LZ2	Input	Values of discrete function $g(j_x, j_y, j_z)$ (See Note (a))
				Output	Value of discrete function $\tilde{q}(k_x, k_y, k_z)$ or the three-dimensional real Fourier transform of $q(k_x, k_y, k_z)$ (See Note (b))
12	LX2	I	1	Input	Adjustable dimension of array R2
13	LY2	I	1	Input	Second dimension of array R2
14	LZ2	I	1	Input	Third dimension of array R2
15	MX	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))

No.	Argument	Type	Size	Input/ Output	Contents
16	MY	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))
17	MZ	I	1	Input	Parameter M_z corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))
18	ISW	I	1	Input	Processing switch (See Note (d)) ISW= 0: Calculate the correlation according to the definition. ISW= 1: Calculate the correlation according to the FFT method. ISW= 2: Calculate the real Fourier transform of the correlation.
19	IWK	I	See Contents	Work	Work area **Size**: 0 (When ISW= 0) 60 (When ISW $\geq$ 1)
20	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: (NX2 + 1) × (NY2 + 1) × NZ2 (When ISW= 0, NX2 is even and NY2 is even) NX2 × (NY2 + 1) × NZ2 (When ISW= 0, NX2 is odd and NY2 is even) (NX2 + 1) × NY2 × NZ2 (When ISW= 0, NX2 is even and NY2 is odd) NX2 × NY2 × NZ2 (When ISW= 0, NX2 is odd and NY2 is odd) MX + 2 × (MY + MZ) + MAX(LX1 × LY1 × LZ1, LX2 × LY2 × LZ2) (When ISW $\geq$ 1)
21	NT	I	1	Input	Number of tasks to be generated
22	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, 1, 2\}$

(b) NX1 > 1 and NY1 > 1 and NZ1 > 1

(c) NX2 > 1 and NY2 > 1 and NZ2 > 1

(d) MX $\geq$ MAX(NX1, NX2) and MY $\geq$ MAX(NY1, NY2) and MZ $\geq$ MAX(NZ1, NZ2)

(e) LX1 $\geq$ NX1 and LY1 $\geq$ NY1 and LZ1 $\geq$ NZ1 (when ISW=0)
LX1 $\geq$ MX $+ 1$ and LX1 is even and LY1 $\geq$ MY and LZ1 $\geq$ MZ (when ISW ≥ 1 and MX is odd)
LX1 $\geq$ MX $+ 2$ and LX1 is even and LY1 $\geq$ MY and LZ1 $\geq$ MZ (when ISW ≥ 1 and MX is even)

(f) LX2 $\geq$ MX and LY2 $\geq$ NY2 and LZ2 $\geq$ NZ2 (when ISW=0)
LX2 $\geq$ MX $+ 1$ and LX2 is even and LY2 $\geq$ MY and LZ2 $\geq$ MZ (when ISW ≥ 1 and MX is odd)
LX2 $\geq$ MX $+ 2$ and LX2 is even and LY2 $\geq$ MY and LZ2 $\geq$ MZ (when ISW ≥ 1 and MX is even)

(g) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	MX $<$ NX1 $+$ NX2 $- 1$ or MY $<$ NY1 $+$ NY2 $- 1$ or MZ $<$ NZ1 $+$ NZ2 $- 1$	Overlapping occurred during the correlation calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ and the elements of arrays R1 and R2 are associated as follows.

$$f(i_x, i_y, i_z) \quad \leftrightarrow \quad \mathrm{R1(i_x + 1, i_y + 1, i_z + 1)}$$
$$g(j_x, j_y, j_z) \quad \leftrightarrow \quad \mathrm{R2(j_x + 1, j_y + 1, j_z + 1)}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$; $i_z = 0, \cdots, n_z^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$; $j_z = 0, \cdots, n_z^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** R1 **and** R2 **should be set so that** LX1**/2,** LY1**,** LZ1**,** LX2**/2,** LY2**, and** LZ2 **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** R1 **and** R2**. Usually, when** MX**, for example, is (a multiple of 4)+2,** LX1=MX**+4 is set.**

(b) The values of the discrete convolution $\tilde{q}(k_x, k_y, k_z)$ and the elements of array R2 are associated as follows.

$$\tilde{q}(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{R2(k_x + 1, k_y + 1, k_z + 1)}$$

Here, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$; $k_z = 0, \cdots, M_z - 1$. When ISW=2 is set to obtain the three-dimensional real Fourier transform $Q(j_x, j_y, j_z)$ of the discrete correlation $q(k_x, k_y, k_z)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$Q(j_x, j_y, j_z) = \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} q(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor;\ j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor)$$

the following associations are made:

$$\Re\{Q(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{R2(2 * j_x + 1, j_y + 1, j_z + 1)}$$
$$\Im\{Q(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{R2(2 * j_x + 2, j_y + 1, j_z + 1)}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$Q(M_x - j_x, M_y - j_y, M_z - j_z)^* = Q(j_x, j_y, j_z)$$
$$Q(M_x - j_x, j_y, j_z)^* = Q(j_x, M_y - j_y, M_z - j_z)$$
$$Q(M_x - j_x, M_y - j_y, j_z)^* = Q(j_x, j_y, M_z - j_z)$$

(Here, z^* represents the conjugate complex number of the complex number z.) Now, $Q(j_x, j_y, j_z)$ can be thought of as an estimate of the cross spectrum of the original two functions for which the correlation is to be calculated. In this case, M_x, M_y, and M_z should be thought of as $M_x = n_x^{(f)} + n_x^{(g)}$, $M_y = n_y^{(f)} + n_y^{(g)}$, and $M_z = n_z^{(f)} + n_z^{(g)}$. In particular, if the original two functions for which the correlation is to be calculated are the same function, $Q(j_x, j_y, j_z)$ corresponds to the raw Fourier periodogram (estimate of the power spectrum), and $Q(j_x, j_y, j_z)$ is a real number.

(c) If MX $\geq$ NX1 + NX2 $-$ 1 and MY $\geq$ NY1 + NY2 $-$ 1 and MZ $\geq$ NZ1 + NZ2 $-$ 1 are set, the correlation can be calculated without causing an overlap with the correlation of the next period. When MX $>$ NX1+NX2$-$1 or MY $>$ NY1+NY2$-$1 or MZ $>$ NZ1+NZ2$-$1, the following correspondences are made:

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{R2(k_x + 1, k_y + 1, k_z + 1)}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x =$ NX1 + NX2 $-$ 1, $\cdots$, MX $-$ 1; $k_y = 0, \cdots,$ MY $-$ 1; $k_z = 0, \cdots,$ MZ $-$ 1 or $k_x = 0, \cdots,$ MX $-$ 1; $k_y =$

$\mathrm{NY1}+\mathrm{NY2}-1,\cdots,\mathrm{MY}-1$; $k_z=0,\cdots,\mathrm{MZ}-1$ or $k_x=0,\cdots,\mathrm{MX}-1$; $k_y=0,\cdots,\mathrm{MY}-1$; $k_z=\mathrm{NZ1}+\mathrm{NZ2}-1,\cdots,\mathrm{MZ}-1$. When ISW=0, $\mathrm{MX}=\mathrm{NX1}+\mathrm{NX2}-1$, $\mathrm{MY}=\mathrm{NY1}+\mathrm{NY2}-1$, and $\mathrm{MZ}=\mathrm{NZ1}+\mathrm{NZ2}-1$ should be set. When ISW ≥ 1, the calculations can be performed more efficiently by setting a value for MX, MY or MZ for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if NX1=NX2=145, then when ISW=0, MX = 289(=17^2) should be set. However, when ISW ≥ 1, it is usually more efficient to set MX = 300(=$2^2\times 3\times 5^2$), MX = 320(=$2^6\times 5$), MX = 384(=$2^7\times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting ISW=1 to calculate the FFT correlation.** However, to conserve work area or if there is a restriction on the method of selecting the parameter MX, MY or MZ, the calculations should be performed by setting ISW=0.

(e) To calculate the correlation of discrete functions the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x,i_y,i_z)$ and $g(j_x,j_y,j_z)$ are the intervals $[i_0,i_0+n_x^{(f)}-1]$ and $[j_0,j_0+n_x^{(g)}-1]$ for i_x and j_x, respectively, let $\hat{f}(i_x,i_y,i_z)$ and $\hat{g}(j_x,j_y,j_z)$ be defined as follows:

$$\hat{f}(i_x,i_y,i_z)=f(i_x-i_0,i_y,i_z),\quad \hat{g}(j_x,j_y,j_z)=g(j_x-j_0,j_y,j_z)$$

and apply this subroutine to $\hat{f}(i_x,i_y,i_z)$ and $\hat{g}(j_x,j_y,j_z)$. Let $\tilde{q}(k_x,k_y,k_z)$ represent the result that was obtained, and the correlation $q(k_x,k_y,k_z)$ of the original functions $f(i_x,i_y,i_z)$ and $g(j_x,j_y,j_z)$ is given as follows:

$$q(k_x,k_y,k_z)=\tilde{q}(k_x-(j_0-i_0)+(n_x^{(f)}-1),k_y,k_z)$$

Therefore, even when $i_0=j_0=0$, to consider the correlation $q(k_x,k_y,k_z)$ that conforms to the normal definition, you must consider shifting the result by $n_x^{(f)}-1$ in the negative direction of k_x after applying this subroutine or if you shift $f(i_x,i_y,i_z)$ and $g(j_x,j_y,j_z)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete correlation, you must then shift the calculation result again by j_0-i_0 in the positive direction of k_x. Similarly, for i_y, j_y, and k_y and i_z, j_z, and k_z.

(f) The sampling interval cubed multiplied by the discrete correlation calculated by this subroutine is the square approximation (or approximation by using the trapezoidal formula) of the continuous correlation integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous correlation it is easiest to let $q(-n_x^{(f)},k_y,k_z)=\tilde{q}(-1,k_y,k_z)=0$, $q(k_x,-n_y^{(f)},k_z)=\tilde{q}(k_x,-1,k_z)=0$, and $q(k_x,k_y,-n_z^{(f)})=\tilde{q}(k_x,k_y,-1)=0$ and consider $(n_x^{(f)}+n_x^{(g)})(n_y^{(f)}+n_y^{(g)})(n_z^{(f)}+n_z^{(g)})$ data of $q(k_x,k_y,k_z)$ ($k_x=-n_x^{(f)},\cdots,-1,0,1,\cdots,n_x^{(g)}-1$; $k_y=-n_y^{(f)},\cdots,-1,0,1,\cdots,n_y^{(g)}-1$; $k_z=-n_z^{(f)},\cdots,-1,0,1,\cdots,n_z^{(g)}-1$). Of course, this is the same as letting $q(n_x^{(f)}+n_x^{(g)},k_y,k_z)=\tilde{q}(n_x^{(g)},k_y,k_z)=0$, $q(k_x,n_y^{(f)}+n_y^{(g)},k_z)=\tilde{q}(k_x,n_y^{(g)},k_z)=0$, and $q(k_x,k_y,n_z^{(f)}+n_z^{(g)})=\tilde{q}(k_x,k_y,n_y^{(g)})=0$ and considering $q(k_x,k_y,k_z)$ ($k_x=-(n_x^{(f)}-1),\cdots,-1,0,1,\cdots,n_x^{(g)}$; $k_y=-(n_y^{(f)}-1),\cdots,-1,0,1,\cdots,n_y^{(g)}$; $k_z=-(n_z^{(f)}-1),\cdots,-1,0,1,\cdots,n_z^{(g)}$). In this case, the coordinate (0, 0, 0) element usually is associated with $q(0,0,0)$. However,

When ISW=0,

then $\mathrm{LX1}=\mathrm{NX1},\mathrm{LY1}=\mathrm{NY1},\mathrm{LZ1}=\mathrm{NZ1},\mathrm{LX2}=\mathrm{MX},\mathrm{LY2}=\mathrm{MY},\mathrm{LZ2}=\mathrm{MZ}$, and

$\mathrm{NWK}=(\mathrm{NX2}+1)\times(\mathrm{NY2}+1)\times\mathrm{NZ2}$ (when NX2 is even and NY2 is even) or

$\mathrm{NWK}=\mathrm{NX2}\times(\mathrm{NY2}+1)\times\mathrm{NZ2}$ (when NX2 is odd and NY2 is even) or

$\mathrm{NWK}=(\mathrm{NX2}+1)\times\mathrm{NY2}\times\mathrm{NZ2}$ (when NX2 is even and NY2 is odd) or

$\mathrm{NWK}=\mathrm{NX2}\times\mathrm{NY2}\times\mathrm{NZ2}$ (when NX2 is odd and NY2 is odd)

When ISW ≥ 1
LX1=LX2=MX+1 (when MX is odd) or
LX1=LX2=MX+2 (when MX is even),
LY1=LY2=MY, LZ1=LZ2=MZ, and NWK = MX + 2 × (MY + MZ) + LX1 × MY × MZ.

(7) **Example**

(a) Problem

Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete correlation.

$$f(x,y,z) = \begin{cases} x & ((x,y,z) \in [0,x_f] \times [0,y_f] \times [0,z_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x,y,z) = \begin{cases} x_g - x & ((x,y,z) \in [0,x_g] \times [0,y_g] \times [0,z_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data

Sampling data

$R1(i_x+1, i_y+1, i_z+1) = f(i_x\Delta, i_y\Delta, i_z\Delta)$ $(i_x = 0,1,\cdots,NX1-1;\ i_y = 0,1,\cdots,NY1-1;\ i_z = 0,1,\cdots,NZ1-1)$,

$R2(j_x+1, j_y+1, j_z+1) = g(j_x\Delta, j_y\Delta, j_z\Delta)$ $(j_x = 0,1,\cdots,NX2-1;\ j_y = 0,1,\cdots,NY2-1;\ j_z = 0,1,\cdots,NZ2-1)$.

Here, $\Delta = 0.5$.

NX1, NY1, NZ1, NX2, NY2, NZ2, MX, MY, MZ and ISW.

(c) Main program

```
      PROGRAM OFCR3D
! *** EXAMPLE OF QFCR3D ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER I,J,K
      INTEGER NT
      INTEGER ISW,IERR,IWK(60)
      INTEGER NX1,NX2,LX1,LX2,MX
      INTEGER NY1,NY2,LY1,LY2,MY
      INTEGER NZ1,NZ2,LZ1,LZ2,MZ
      INTEGER M0
      PARAMETER (M0 = 8)
      PARAMETER (LX1 = (M0+2)/2*2)
      PARAMETER (LY1 = M0)
      PARAMETER (LZ1 = M0)
      PARAMETER (LX2 = LX1)
      PARAMETER (LY2 = LY1)
      PARAMETER (LZ2 = LZ1)
      REAL(8) R1(LX1,LY1,LZ1),R2(LX2,LY2,LZ2)
      REAL(8) WK(5*M0+LX1*M0*M0)
      REAL(8) T
      REAL(8) XF,YF,ZF,XG,YG,ZG,DT
      PARAMETER (DT = 0.5D0)
      PARAMETER (XF = 2.0D0,YF=2.0D0,ZF=2.0D0)
      PARAMETER (XG = 2.0D0,YG=2.0D0,ZG=2.0D0)
      COMMON IWK,R1,R2,WK
!
      NT=2
      ISW=1
      NX1=XF/DT
      NY1=YF/DT
      NZ1=ZF/DT
      NX2=XG/DT
      NY2=YG/DT
      NZ2=ZG/DT
      MX=M0
      MY=M0
      MZ=M0
      WRITE (6,1000) ISW,NX1,NY1,NZ1,NX2,NY2,NZ2,MX,MY,MZ
      DO 100 K=1,NZ1
      DO 101 J=1,NY1
      DO 102 I=1,NX1
        T=DBLE(I-1)*DT
        R1(I,J,K)=T
  102 CONTINUE
  101 CONTINUE
```

```
 100 CONTINUE
     DO 200 K=1,NZ2
     DO 201 J=1,NY2
     DO 202 I=1,NX2
       T=DBLE(I-1)*DT
       R2(I,J,K)=XG-T
 202 CONTINUE
 201 CONTINUE
 200 CONTINUE
     DO 300 K=1,NZ1
     WRITE (6,1100) K,(I,(R1(I,J,K),J=1,NY1),I=1,NX1)
 300 CONTINUE
     DO 400 K=1,NZ2
     WRITE (6,1150) K,(I,(R2(I,J,K),J=1,NY2),I=1,NX2)
 400 CONTINUE
     CALL QFCR3D(NX1,NY1,NZ1,NX2,NY2,NZ2,R1,LX1,LY1,LZ1,&
      R2,LX2,LY2,LZ2,MX,MY,MZ,ISW,IWK,WK,NT,IERR)
     WRITE (6,1300)
     WRITE (6,1400) IERR
     DO 500 K=1,MZ
     WRITE (6,1200) K,(I,(R2(I,J,K),J=1,MY),I=1,MX)
 500 CONTINUE
1000 FORMAT(' ',/,/,&
            ' ***  QFCR3D  ***',/,&
            2X,'**  INPUT  **',/,&
            6X,'ISW =',I3,/,&
            6X,'(NX1,NY1,NZ1) =(',I3,',',I3,',',I3,')',/,&
            6X,'(NX2,NY2,NZ2) =(',I3,',',I3,',',I3,')',/,&
            6X,'(MX ,MY ,MZ ) =(',I3,',',I3,',',I3,')')
1100 FORMAT(12X,'DATA R1(I,J,',I3,')',/,&
            10X,'I/J   1        2        3        4',/,&
            10X,'-----------------------------------',/,&
            6(8X,I3,4F9.4,/))
1150 FORMAT(12X,'DATA R2(I,J,',I3,')',/,&
            10X,'I/J   1        2        3        4',/,&
            10X,'-----------------------------------',/,&
            6(8X,I3,4F9.4,/))
1300 FORMAT(2X,'**  OUTPUT  **')
1400 FORMAT(6X,'IERR =',I5)
1200 FORMAT(17X,'CORRELATION R2(I,J,',I3,')',/,&
            10X,'I/J   1      2      3      4      5',&
                          ',      6      7      8',/,&
            10X,'-------------------------------------',&
                ',--------------------',/,&
            8(8X,I3,8F7.2,/))
     END
```

(d) Output results

```
***  QFCR3D  ***
 **  INPUT  **
     ISW =  1
     (NX1,NY1,NZ1) =(  4,  4,  4)
     (NX2,NY2,NZ2) =(  4,  4,  4)
     (MX ,MY ,MZ ) =(  8,  8,  8)
          DATA R1(I,J,  1)
         I/J   1        2        3        4
         -----------------------------------
         1   0.0000   0.0000   0.0000   0.0000
         2   0.5000   0.5000   0.5000   0.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   1.5000   1.5000   1.5000   1.5000

           DATA R1(I,J,  2)
         I/J   1        2        3        4
         -----------------------------------
         1   0.0000   0.0000   0.0000   0.0000
         2   0.5000   0.5000   0.5000   0.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   1.5000   1.5000   1.5000   1.5000

           DATA R1(I,J,  3)
         I/J   1        2        3        4
         -----------------------------------
         1   0.0000   0.0000   0.0000   0.0000
         2   0.5000   0.5000   0.5000   0.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   1.5000   1.5000   1.5000   1.5000

           DATA R1(I,J,  4)
         I/J   1        2        3        4
         -----------------------------------
         1   0.0000   0.0000   0.0000   0.0000
         2   0.5000   0.5000   0.5000   0.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   1.5000   1.5000   1.5000   1.5000

           DATA R2(I,J,  1)
         I/J   1        2        3        4
         -----------------------------------
         1   2.0000   2.0000   2.0000   2.0000
         2   1.5000   1.5000   1.5000   1.5000
         3   1.0000   1.0000   1.0000   1.0000
         4   0.5000   0.5000   0.5000   0.5000
```

```
  DATA R2(I,J,  2)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

  DATA R2(I,J,  3)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

  DATA R2(I,J,  4)
I/J   1        2        3        4
-------------------------------------
1   2.0000   2.0000   2.0000   2.0000
2   1.5000   1.5000   1.5000   1.5000
3   1.0000   1.0000   1.0000   1.0000
4   0.5000   0.5000   0.5000   0.5000

**  OUTPUT  **
    IERR =    0
         CORRELATION R2(I,J,  1)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
2   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
3   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
4   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
5   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
6   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
8   0.00   0.00  -0.00  -0.00  -0.00   0.00   0.00   0.00

         CORRELATION R2(I,J,  2)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
2   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
3   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
4   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
5   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
6   0.50   1.00   1.50   2.00   1.50   1.00   0.50  -0.00
7   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
8   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

         CORRELATION R2(I,J,  3)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
2  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
3  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
4   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
5   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
6   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
7   0.00   0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
8   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00   0.00

         CORRELATION R2(I,J,  4)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
2  17.00  34.00  51.00  68.00  51.00  34.00  17.00  -0.00
3  16.00  32.00  48.00  64.00  48.00  32.00  16.00  -0.00
4  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
5   4.00   8.00  12.00  16.00  12.00   8.00   4.00   0.00
6   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
7  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00   0.00
8  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00

         CORRELATION R2(I,J,  5)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
2  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
3  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
4   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
5   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
6   0.75   1.50   2.25   3.00   2.25   1.50   0.75   0.00
7   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
8   0.00   0.00   0.00  -0.00   0.00  -0.00   0.00   0.00

         CORRELATION R2(I,J,  6)
I/J   1      2      3      4      5      6      7      8
-------------------------------------------------------------
1   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
2   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
3   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
4   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
5   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
```

6	0.50	1.00	1.50	2.00	1.50	1.00	0.50	0.00
7	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00

CORRELATION R2(I,J, 7)

I/J	1	2	3	4	5	6	7	8
1	3.00	6.00	9.00	12.00	9.00	6.00	3.00	0.00
2	4.25	8.50	12.75	17.00	12.75	8.50	4.25	-0.00
3	4.00	8.00	12.00	16.00	12.00	8.00	4.00	-0.00
4	2.50	5.00	7.50	10.00	7.50	5.00	2.50	-0.00
5	1.00	2.00	3.00	4.00	3.00	2.00	1.00	0.00
6	0.25	0.50	0.75	1.00	0.75	0.50	0.25	0.00
7	0.00	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	-0.00	-0.00	-0.00	0.00	0.00	0.00	0.00

CORRELATION R2(I,J, 8)

I/J	1	2	3	4	5	6	7	8
1	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	0.00
2	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
3	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
4	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
5	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
6	-0.00	-0.00	0.00	0.00	0.00	0.00	-0.00	-0.00
7	-0.00	-0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	-0.00	0.00	-0.00	0.00	0.00	0.00	0.00	0.00

6.13 POWER SPECTRUM ANALYSIS

6.13.1 QFPS2D, PFPS2D
Two-Dimensional Fourier Periodograms

(1) **Function**

QFPS2D or PFPS2D obtains the (modified) Fourier periodogram of the series u_{j_x,j_y} ($j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$). The Fourier periodogram p_{k_x,k_y} is defined by the following equation.

$$p_{k_x,k_y} = \frac{\left| \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} w_{j_x}^{(x)} w_{j_y}^{(y)} u_{j_x,j_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \right|^2}{n_x n_y \beta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. $w_{j_x}^{(x)}$ and $w_{j_y}^{(y)}$ are the truncation functions (data windows). For a raw Fourier periodogram, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = 1$ ($j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$) and $\beta = n_x n_y$ are set, and for a modified periodogram β is set as follows:

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1} (w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1} (w_{j_y}^{(y)})^2) & \text{(when a power modification expression according to a data window is used)} \\ n_x n_y & \text{(Otherwise)} \end{cases}$$

The periodogram p_{k_x,k_y} corresponds to a half period (period (n_x, n_y)) and the remainder is obtained from the relationship as follows.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y} &= p_{k_x,k_y} \\ p_{n_x-k_x,k_y} &= p_{k_x,n_y-k_y} \end{aligned}$$

Also, the total power of the corresponding series is as follows.

$$\frac{\sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \{u_{j_x,j_y}\}^2}{n_x n_y}$$

(2) **Usage**

Double precision:

CALL QFPS2D (NX, NY, R, LX, LY, ISW, IWK, WK, NT, IERR)

Single precision:

CALL PFPS2D (NX, NY, R, LX, LY, ISW, IWK, WK, NT, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Length n_x in the j_x direction of series u_{j_x,j_y} (See Note (d))
2	NY	I	1	Input	Length n_y in the j_y direction of series u_{j_x,j_y} (See Note (d))
3	R	{D / R}	LX, LY	Input	Values of series u_{j_x,j_y} (See Note (a))
				Output	Values of Fourier periodogram p_{k_x,k_y} of series u_{j_x,j_y} (See Notes (b) and (c))
4	LX	I	1	Input	Adjustable dimension of array R
5	LY	I	1	Input	Second dimension of array R
6	ISW	I	1	Input	Processing switch (See Note (e)) ISW= 0: Calculate the raw Fourier periodogram ISW=±1: Calculate the periodogram using a user-defined data window ISW=±2: Calculate the periodogram using the Hanning window ISW=±3: Calculate the periodogram using the Bartlett window ISW=±4: Calculate the periodogram using the Welch window ISW=±5: Calculate the periodogram using the Parzen window To use a power modification expression according to a data window, set ISW > 0; otherwise, set ISW < 0.
7	IWK	I	40	Work	Work area
8	WK	{D / R}	See Contents	Work	Work area When ISW=±1, enter the values of the user-defined data window. (See Note (e)) **Size**: NX + 2 × NY + LX × LY
9	NT	I	1	Input	Number of tasks to be generated
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) NX > 1 and NY > 1

(c) LX $\geq$ NX $+ 1$ and LY $\geq$ NY (when NX is odd)
LX $\geq$ NX $+ 2$ and LY $\geq$ NY (when NX is even)

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3030	Restriction (c) was not satisfied.	
3040	Restriction (d) was not satisfied.	
4000	When ISW=1, the user-defined data window was $w_{j_x}^{(x)} = 0$ $(j_x = 0, \cdots, n_x - 1)$.	
4010	When ISW=1, the user-defined data window was $w_{j_y}^{(y)} = 0$ $(j_y = 0, \cdots, n_y - 1)$.	

(6) **Notes**

(a) The elements of array R and the values of the series u_{j_x,j_y} are associated as follows.

$$u_{j_x,j_y} \quad \leftrightarrow \quad \mathrm{R(j_x + 1, j_y + 1)}$$

Here, $j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$, and no values need be entered in other elements. **The adjustable dimensions of array** R **should be set so that** LX**/2 and** LY **are odd numbers to avoid bank conflict of main memory. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX**=**NX**+4 is set.**

(b) The values of the Fourier periodogram p_{k_x,k_y} are associated as follows with the elements of array R.

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{R(k_x + 1, k_y + 1)} \ \ (\mathrm{k_x} = 0, \cdots, \lfloor \tfrac{\mathrm{n_x}}{2} \rfloor; \ \mathrm{k_y} = 0, \cdots, \mathrm{n_y} - 1)$$

$\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

(c) The frequencies (ξ_{k_x}, η_{k_y}) corresponding to obtained Fourier periodogram p_{k_x,k_y} $(k_x = 0, 1, \cdots, n_x - 1; \ k_y = 0, 1, \cdots, n_y - 1)$ are given as follows.

$$\xi_{k_x} = \frac{k_x}{n_x \Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y \Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y \Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

Where Δ is the sampling interval.

(d) The calculations can be performed more efficiently by setting the length NX and NY of the series u_{j_x,j_y} to a value for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, rather than setting NX $= 289 (= 17^2)$, it is usually more efficient to set NX $= 300 (= 2^2 \times 3 \times 5^2)$, NX $= 320 (= 2^6 \times 5)$, NX $= 384 (= 2^7 \times 3)$ or

the like. When the number of data cannot be increased, adjust NX by supplying the required number of zeros at the end of the data to perform the calculations.

(e) The truncation function (data window) can be changed as follows according to the value of the processing switch ISW.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{ISW= } \pm 2 \text{ (Hanning window)} \\ 1 - |2v_j - 1| & \text{ISW= } \pm 3 \text{ (Bartlett window)} \\ 1 - (2v_j - 1)^2 & \text{ISW= } \pm 4 \text{ (Welch window)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & \text{ISW= } \pm 5 \text{ (Parzen window)} \end{cases}$$

Here, $v_j = \frac{j}{n}$, and $j = j_x$ and $n = n_x$ are set for $w_{j_x}^{(x)}$ and $j = j_y$ and $n = n_y$ are set for $w_{j_y}^{(y)}$. Therefore, when the data windows shown above are used, the elements u_{0,j_y} and $u_{j_x,0}$ of the series u_{j_x,j_y} do not affect the calculation of the modified periodogram. To avoid this situation, you should specify values for NX and NY that are larger by 1 than the lengths of the series for which you actually want to calculate the periodogram and set the effective data in elements corresponding to 1 and after for j_x and j_y. The data windows are represented as follows as time (or space) domain functions that are nonzero only for $|x| \le 1$.

$$w(x) = \begin{cases} \dfrac{1 + \cos \pi x}{2} = \cos^2 \dfrac{\pi x}{2} & \text{Hanning window} \\ 1 - |x| & \text{Bartlett window} \\ 1 - x^2 & \text{Welch window} \\ \left\{ \begin{array}{ll} 1 - 6x^2 + 6|x|^3 & |x| \le \frac{1}{2} \\ 2(1 - |x|)^3 & \frac{1}{2} \le |x| \le 1 \end{array} \right\} & \text{Parzen window} \end{cases}$$

Also, to use user-defined data window values $w_{j_x}^{(x)}$ and $w_{j_y}^{(y)}$, set ISW = ±1, set the values in work array WK as follows:

$$\mathrm{WK(j_x + 1) = w_{j_x}^{(x)} \ \ (j_x = 0, \cdots, n_x - 1), WK(n_x + j_y + 1) = w_{j_y}^{(y)} \ \ (j_y = 0, \cdots, n_y - 1)}$$

and then call this subroutine.

(f) From its definition, the raw periodogram should be regarded as an approximation of a discrete Fourier transform of the autocorrelation function. Since the effective data length of the autocorrelation function of a discrete function having effective number of data n is $2n - 1$, approximating the power spectrum of a general function by a raw periodogram corresponds to truncating the function by using a square truncation function $w(k)$ for which one period is given as follows.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n - 1 \\ 0 & \text{Otherwise} \end{cases}$$

When the frequency is f for the Fourier transform of the square function, a $\frac{\sin f}{f}$ type function form is assumed having a sidelobe that is not small around the central frequency. Therefore, when a periodic function is sampled, for example, by simply truncating it using a width that is not an integer multiple of one period, since the raw periodogram will be the convolution of the Fourier transform of the periodic function for which the power spectrum is to be obtained and the $\frac{\sin f}{f}$ type function in the frequency domain, an excess frequency component called **leakage** occurs. To suppress this kind of leakage, simple truncation is not performed, and a truncation function having a small sidelobe in the frequency domain, such as the Hanning window, is used. However, in general, the more the leakage is suppressed, the

more the result of the discrete Fourier transform widens and blurs. Therefore, when estimating the power spectrum, you must select a suitable truncation function according to your objectives, that is, according to whether the spectral width or the central frequency is to be the problem, for example.

(g) To raise the resolution (sampling interval in the frequency domain) $\frac{1}{nT}$ of the discrete Fourier transform, you should increase the number of sample data n or increase the sampling interval T. However, to raise the precision of the power spectrum estimate while holding the sampling interval and resolution fixed, a technique is often used of taking m groups of samples for which the number of samples is n, obtaining the modified periodogram for each of those m groups, and then taking the average of those values. In this case, a technique is also proposed in which the m groups of sample data are taken from the series so that they overlap. For details, refer to the Reference Bibliography.

(h) When obtaining the power spectrum, the property related to the frequency transition of the Fourier transform, that is, the multiplication by $e^{2\pi\sqrt{-1}f_0 t}$ in the time (or space) domain, is associated with the shifting of the frequency by f_0 in the frequency domain, and a technique is often used of reducing the number of data points required for the calculation in which the central frequency of the power spectrum is shifted in advance, using the property that the function shape does not change. This kind of operation is known as **modulation**. However, LX=NX+1 (when NX is odd) or
LX=NX+2 (when NX is even)
and LY=NY.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the waveform defined by the following equation and estimate the power spectrum by calculating the Fourier periodogram.

$$f(x, y) = \cos 2\pi f_1 x + \cos 2\pi f_2 y$$

(b) Input data
Sampling data
$\mathrm{R}(\mathrm{j_x}+1, \mathrm{j_y}+1) = \mathrm{f}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{NX}-1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{NY}-1)$.
Here, $\Delta = 0.5$.
NX, NY and ISW.

(c) Main program

```
      PROGRAM OFPS2D
! *** EXAMPLE OF QFPS2D ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NX,NY,LX,LY,ISW,IERR,IWK(40)
      INTEGER I,J,M,ND2,IS
      INTEGER N0,ISW0
      INTEGER NT
      PARAMETER (N0 =8)
      PARAMETER (ISW0=4)
      PARAMETER (LX = N0+2, LY=N0)
      REAL(8) R(LX,LY,-1:ISW0),WK(3*N0+LX*LY)
      REAL(8) P(-1:ISW0)
      REAL(8) TX,TY,DT,DFX,DFY
      REAL(8)  PAI,F0,F1,F2
      PARAMETER(PAI=3.141592D0)
      COMMON IWK,R,R2,WK,P
!
      NT=2
      NX=N0
      NY=N0
      WRITE (6,1000) ISW0+1,NX,NY
      DT=0.5D0
      F0=1.0D0/(2.0*DT)
      F1=0.62D0*F0
      F2=0.14D0*F0
      ND2=(NX+1)/2
      DFX=1.0D0/(DT*NX)
      DFY=1.0D0/(DT*NY)
      P(-1)=0.0D0
```

```
      DO 100 J=1,NY
        TY=DBLE(J-1)*DT
      DO 101 I=1,NX
        TX=DBLE(I-1)*DT
        R(I,J,-1)=COS(2*PAI*F1*TX)+COS(2*PAI*F2*TY)
        P(-1)=P(-1)+R(I,J,-1)*R(I,J,-1)
  101 CONTINUE
  100 CONTINUE
      P(-1)=P(-1)/(DBLE(NX)*DBLE(NY))
      WRITE (6,1100) (I,(R(I,J,-1),J=1,NY),I=1,NX)
      WRITE (6,1150) P(-1)
      WRITE (6,1160) F1,F2
      IS=0
      DO 400 ISW=0,ISW0
      DO 200 J=1,NY
      DO 201 I=1,NX
        R(I,J,ISW)=R(I,J,-1)
  201 CONTINUE
  200 CONTINUE
      IF (ISW.NE.0) IS=ISW+1
      CALL QFPS2D(NX,NY,R(1,1,ISW),LX,LY,IS,IWK,WK,NT,IERR)
      P(ISW)=0.0D0
      IF (MOD(NX,2).EQ.0) THEN
        M=ND2-1
        DO 500 J=1,NY
        DO 501 I=2,M
          P(ISW)=P(ISW)+2.0D0*R(I,J,ISW)
  501   CONTINUE
  500   CONTINUE
        DO 300 J=1,NY
          P(ISW)=P(ISW)+R(1,J,ISW)+R(ND2,J,ISW)
  300   CONTINUE
      ELSE
        M=ND2
        DO 600 J=1,NY
        DO 601 I=2,M
          P(ISW)=P(ISW)+2.0D0*R(I,J,ISW)
  601   CONTINUE
  600   CONTINUE
        DO 700 J=1,NY
          P(ISW)=P(ISW)+R(1,J,ISW)
  700   CONTINUE
      ENDIF
  400 CONTINUE
      WRITE (6,1200)
      WRITE (6,1300) IERR
!
      ISW=0
      WRITE (6,1410) 'RAW',P(ISW),&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,ISW),J=1,(NY+1)/2),I=1,ND2)
!
      ISW=1
      WRITE (6,1410) 'HANNING',P(ISW),&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,ISW),J=1,(NY+1)/2),I=1,ND2)
!
      ISW=2
      WRITE (6,1410) 'BARTLETT',P(ISW),&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,ISW),J=1,(NY+1)/2),I=1,ND2)
!
      ISW=3
      WRITE (6,1410) 'WELCH',P(ISW),&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,ISW),J=1,(NY+1)/2),I=1,ND2)
!
      ISW=4
      WRITE (6,1410) 'PARZEN',P(ISW),&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,ISW),J=1,(NY+1)/2),I=1,ND2)
!
 1000 FORMAT(' ',/,/,&
             ' ***  QFPS2D  ***',/,&
            2X,'**  INPUT  **',/,&
            6X,'ISW =0, 2 TO ',I3,/,&
            6X,'NX =',I3,/,&
            6X,'NY =',I3)
 1100 FORMAT(12X,'DATA R(I,J)',/,&
            4X,'I/J   1        2        3        4',&
```

```
            ’       5       6       7       8’,/,&
            4X,’-----------------------------------’,&
            ’-----------------------------------’,/,&
            8(2X,I3,8F9.4,/))
 1150 FORMAT(6X,’TIME DOMAIN POWER =’,F10.4)
 1160 FORMAT(6X,’SIGNAL FREQUENCY =(’,F10.4,’,’,F10.4,’)’)
 1200 FORMAT(2X,’**  OUTPUT  **’)
 1300 FORMAT(6X,’IERR =’,I5)
 1410 FORMAT(6X,’(MODIFIED) PERIODOGRAM(’,A,’)’,/,&
            3X,’FREQUENCY DOMAIN POWER=’,F8.4,/,&
            2X,’X/Y-FRQ’,8F8.2,/,&
            2X,’-----------------------------------’,&
            ’-----------------------------------’)
 1420 FORMAT(8(2X,F7.2,8F8.4,/))
      END
```

(d) Output results

```
***  QFPS2D  ***
 **  INPUT  **
     ISW =0, 2 TO   5
     NX =  8
     NY =  8
           DATA R(I,J)
   I/J   1        2        3        4        5        6        7        8
   ------------------------------------------------------------------------
   1   2.0000   1.9048   1.6374   1.2487   0.8126   0.4122   0.1237   0.0020
   2   0.6319   0.5367   0.2693  -0.1194  -0.5555  -0.9559  -1.2444  -1.3662
   3   0.2710   0.1759  -0.0915  -0.4803  -0.9164  -1.3168  -1.6053  -1.7270
   4   1.9048   1.8097   1.5423   1.1535   0.7174   0.3170   0.0285  -0.0932
   5   1.0628   0.9676   0.7002   0.3115  -0.1246  -0.5250  -0.8135  -0.9352
   6   0.0489  -0.0462  -0.3136  -0.7024  -1.1384  -1.5388  -1.8274  -1.9491
   7   1.6374   1.5422   1.2748   0.8861   0.4500   0.0496  -0.2389  -0.3606
   8   1.4818   1.3866   1.1192   0.7304   0.2944  -0.1060  -0.3946  -0.5163

     TIME DOMAIN POWER =    1.0626
     SIGNAL FREQUENCY =(    0.6200,    0.1400)
 **  OUTPUT  **
     IERR =    0
     (MODIFIED) PERIODOGRAM(RAW)
  FREQUENCY DOMAIN POWER=  0.9717
 X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 ------------------------------------------------------------------------
    0.00  0.0158  0.0188  0.0350  0.2150  0.0218  0.2150  0.0350  0.0188
    0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

     (MODIFIED) PERIODOGRAM(HANNING)
  FREQUENCY DOMAIN POWER=  0.5980
 X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 ------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0054  0.0632  0.0105  0.0632  0.0054  0.0001
    0.25  0.0000  0.0000  0.0013  0.0095  0.0056  0.0236  0.0013  0.0000
    0.50  0.0000  0.0000  0.0000  0.0204  0.0814  0.0204  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0205  0.0820  0.0205  0.0000  0.0000

     (MODIFIED) PERIODOGRAM(BARTLETT)
  FREQUENCY DOMAIN POWER=  0.5835
 X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 ------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0820  0.0109  0.0820  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0122  0.0025  0.0178  0.0002  0.0000
    0.50  0.0000  0.0005  0.0000  0.0156  0.0855  0.0156  0.0000  0.0005
    0.75  0.0000  0.0004  0.0000  0.0095  0.0762  0.0191  0.0000  0.0004

     (MODIFIED) PERIODOGRAM(WELCH)
  FREQUENCY DOMAIN POWER=  0.7072
 X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 ------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.1263  0.0124  0.1263  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0140  0.0014  0.0127  0.0000  0.0000
    0.50  0.0002  0.0003  0.0010  0.0195  0.1065  0.0054  0.0009  0.0003
    0.75  0.0002  0.0003  0.0008  0.0064  0.0941  0.0142  0.0009  0.0003

     (MODIFIED) PERIODOGRAM(PARZEN)
  FREQUENCY DOMAIN POWER=  0.4909
 X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 ------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0093  0.0253  0.0070  0.0253  0.0093  0.0002
    0.25  0.0000  0.0001  0.0022  0.0022  0.0127  0.0279  0.0064  0.0001
    0.50  0.0000  0.0000  0.0006  0.0171  0.0558  0.0323  0.0030  0.0000
    0.75  0.0000  0.0000  0.0013  0.0206  0.0485  0.0214  0.0014  0.0000
```

6.13.2 QFPS3D, PFPS3D
Three-Dimensional Fourier Periodograms

(1) **Function**

QFPS3D or PFPS3D obtains the (modified) Fourier periodogram of the series u_{j_x,j_y,j_z} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$. The Fourier periodogram p_{k_x,k_y,k_z} is defined by the following equation.

$$p_{k_x,k_y,k_z} = \frac{\left| \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \sum_{j_z=0}^{n_z-1} w_{j_x}^{(x)} w_{j_y}^{(y)} w_{j_z}^{(z)} u_{j_x,j_y,j_z} e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z}\right)} \right|^2}{n_x n_y n_z \beta}$$
$$(k_x = 0, 1, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1; k_z = 0, 1, \cdots, n_z - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ and $w_{j_z}^{(z)}$ are the truncation functions (data windows). For a raw Fourier periodogram, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = w_{j_z}^{(z)} = 1$ $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ and $\beta = n_x n_y n_z$ are set, and for a modified periodogram β is set as follows:

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1} (w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1} (w_{j_y}^{(y)})^2)(\sum_{j_z=0}^{n_z-1} (w_{j_z}^{(z)})^2) & \text{(when a power modification expression according to a data window is used)} \\ n_x n_y n_z & \text{(Otherwise)} \end{cases}$$

The periodogram p_{k_x,k_y,k_z} corresponds to a half period (period (n_x, n_y, n_z)) and the remainder is obtained from the relationship as follows.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y,n_z-k_z} &= p_{k_x,k_y,k_z} \\ p_{n_x-k_x,k_y,k_z} &= p_{k_x,n_y-k_y,n_z-k_z} \\ p_{n_x-k_x,n_y-k_y,k_z} &= p_{k_x,k_y,n_z-k_z} \end{aligned}$$

Also, the total power of the corresponding series is as follows.

$$\frac{\sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \sum_{j_z=0}^{n_z-1} \{u_{j_x,j_y,j_z}\}^2}{n_x n_y n_z}$$

(2) **Usage**

Double precision:

```
CALL QFPS3D  (NX, NY, NZ, R, LX, LY, LZ, ISW, IWK, WK, NT, IERR)
```

Single precision:

```
CALL PFPS3D  (NX, NY, NZ, R, LX, LY, LZ, ISW, IWK, WK, NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	NX	I	1	Input	Length n_x in the j_x direction of series u_{j_x,j_y,j_z} (See Note (d))
2	NY	I	1	Input	Length n_y in the j_y direction of series u_{j_x,j_y,j_z} (See Note (d))
3	NZ	I	1	Input	Length n_z in the j_z direction of series u_{j_x,j_y,j_z} (See Note (d))
4	R	{D / R}	LX, LY, LZ	Input	Values of series u_{j_x,j_y,j_z} (See Note (a))
				Output	Values of Fourier periodogram p_{k_x,k_y,k_z} of series u_{j_x,j_y,j_z} (See Notes (b) and (c))
5	LX	I	1	Input	Adjustable dimension of array R
6	LY	I	1	Input	Second dimension of array R
7	LZ	I	1	Input	Third dimension of array R
8	ISW	I	1	Input	Processing switch (See Note (e)) ISW= 0: Calculate the raw Fourier periodogram ISW=±1: Calculate the periodogram using a user-defined data window ISW=±2: Calculate the periodogram using the Hanning window ISW=±3: Calculate the periodogram using the Bartlett window ISW=±4: Calculate the periodogram using the Welch window ISW=±5: Calculate the periodogram using the Parzen window To use a power modification expression according to a data window, set ISW > 0; otherwise, set ISW < 0.
9	IWK	I	60	Work	Work area
10	WK	{D / R}	See Contents	Work	Work area When ISW=±1, enter the values of the user-defined data window. (See Note (e)) **Size**: NX + 2 × (NY + NZ) + LX × LY × LZ
11	NT	I	1	Input	Number of tasks to be generated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ISW $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) NX > 1 and NY > 1 and NZ > 1

(c) LX $\geq$ NX $+ 1$ and LX is even and LY $\geq$ NY and LZ $\geq$ NZ (when NX is odd)
LX $\geq$ NX $+ 2$ and LX is even and LY $\geq$ NY and LZ $\geq$ NZ (when NX is even)

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3030	Restriction (c) was not satisfied.	
3040	Restriction (d) was not satisfied.	
4000	When ISW=1, the user-defined data window was $w_{j_x}^{(x)} = 0$ $(j_x = 0, \cdots, n_x - 1)$.	
4010	When ISW=1, the user-defined data window was $w_{j_y}^{(y)} = 0$ $(j_y = 0, \cdots, n_y - 1)$.	
4020	When ISW=1, the user-defined data window was $w_{j_z}^{(z)} = 0$ $(j_z = 0, \cdots, n_z - 1)$.	

(6) **Notes**

(a) The elements of array R and the values of the series u_{j_x,j_y,j_z} are associated as follows.

$$u_{j_x,j_y,j_z} \quad \leftrightarrow \quad \mathrm{R(j_x + 1, j_y + 1, j_z + 1)}$$

Here, $j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$; $j_z = 0, \cdots, n_z - 1$ and no values need be entered in other elements. **The adjustable dimensions of array** R **should be set so that** LX**/2,** LY**, and** LZ **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** R**. Usually, when** NX**, for example, is (a multiple of 4)+2,** LX=NX**+4 is set.**

(b) The values of the Fourier periodogram p_{k_x,k_y,k_z} are associated as follows with the elements of array R.

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{R(k_x + 1, k_y + 1, k_z + 1)}$$
$$(k_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

$\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

(c) The frequencies $(\xi_{k_x}, \eta_{k_y}, \zeta_{k_z})$ corresponding to obtained Fourier periodogram p_{k_x,k_y,k_z} $(k_x = 0, 1, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1\ k_z = 0, \cdots, n_z - 1)$ are given as follows.

$$\xi_{k_x} = \frac{k_x}{n_x \Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y \Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y \Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

$$\zeta_{k_z} = \begin{cases} \dfrac{k_z}{n_z \Delta} & (k_z = 0, 1, \cdots, \lfloor \dfrac{n_z}{2} \rfloor) \\ \dfrac{k_z - n_z}{n_z \Delta} & (k_z = \lfloor \dfrac{n_z}{2} \rfloor + 1, \cdots, n_z - 1) \end{cases}$$

Where Δ is the sampling interval.

(d) The calculations can be performed more efficiently by setting the length NX, NY and NZ of the series u_{j_x,j_y,j_z} to a value for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, rather than setting NX = 289(=17^2), it is usually more efficient to set NX = 300(=$2^2 \times 3 \times 5^2$), NX = 320(=$2^6 \times 5$), NX = 384(=$2^7 \times 3$) or the like. When the number of data cannot be increased, adjust NX by supplying the required number of zeros at the end of the data to perform the calculations.

(e) The truncation function (data window) can be changed as follows according to the value of the processing switch ISW.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{ISW= } \pm 2 \text{ (Hanning window)} \\ 1 - |2v_j - 1| & \text{ISW= } \pm 3 \text{ (Bartlett window)} \\ 1 - (2v_j - 1)^2 & \text{ISW= } \pm 4 \text{ (Welch window)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & \text{ISW= } \pm 5 \text{ (Parzen window)} \end{cases}$$

Here, $v_j = \frac{j}{n}$, and $j = j_x$ and $n = n_x$ are set for $w_{j_x}^{(x)}$, $j = j_y$ and $n = n_y$ are set for $w_{j_y}^{(y)}$, and $j = j_z$ and $n = n_z$ are set for $w_{j_z}^{(z)}$. Therefore, when the data windows shown above are used, the elements u_{0,j_y,j_z}, $u_{j_x,0,j_z}$ and $u_{j_x,j_y,0}$ of the series u_{j_x,j_y,j_z} do not affect the calculation of the modified periodogram. To avoid this situation, you should specify values for NX, NY and NZ that are larger by 1 than the lengths of the series for which you actually want to calculate the periodogram and set the effective data in elements corresponding to 1 and after for j_x, j_y and j_z. The data windows are represented as follows as time (or space) domain functions that are nonzero only for $|x| \le 1$.

$$w(x) = \begin{cases} \dfrac{1 + \cos \pi x}{2} = \cos^2 \dfrac{\pi x}{2} & \text{Hanning window} \\ 1 - |x| & \text{Bartlett window} \\ 1 - x^2 & \text{Welch window} \\ \left\{ \begin{array}{ll} 1 - 6x^2 + 6|x|^3 & |x| \le \frac{1}{2} \\ 2(1 - |x|)^3 & \frac{1}{2} \le |x| \le 1 \end{array} \right\} & \text{Parzen window} \end{cases}$$

Also, to use user-defined data window values $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ and $w_{j_z}^{(z)}$, set ISW = ± 1, set the values in work array WK as follows:

$$\mathrm{WK}(\mathrm{j_x}+1) = \mathrm{w}_{\mathrm{j_x}}^{(\mathrm{x})} \quad (\mathrm{j_x} = 0, \cdots, \mathrm{n_x} - 1), \mathrm{WK}(\mathrm{n_x} + \mathrm{j_y} + 1) = \mathrm{w}_{\mathrm{j_y}}^{(\mathrm{y})} \quad (\mathrm{j_y} = 0, \cdots, \mathrm{n_y} - 1),$$
$$\mathrm{WK}(\mathrm{n_x} + \mathrm{n_y} + \mathrm{j_z} + 1) = \mathrm{w}_{\mathrm{j_z}}^{(\mathrm{z})} \quad (\mathrm{j_z} = 0, \cdots, \mathrm{n_z} - 1)$$

and then call this subroutine.

(f) From its definition, the raw periodogram should be regarded as an approximation of a discrete Fourier transform of the autocorrelation function. Since the effective data length of the autocorrelation function of a discrete function having effective number of data n is $2n - 1$, approximating the power spectrum of a general function by a raw periodogram corresponds to truncating the function by using a square truncation function $w(k)$ for which one period is given as follows.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n - 1 \\ 0 & \text{Otherwise} \end{cases}$$

When the frequency is f for the Fourier transform of the square function, a $\frac{\sin f}{f}$ type function form is assumed having a sidelobe that is not small around the central frequency. Therefore, when a periodic

function is sampled, for example, by simply truncating it using a width that is not an integer multiple of one period, since the raw periodogram will be the convolution of the Fourier transform of the periodic function for which the power spectrum is to be obtained and the $\frac{\sin f}{f}$ type function in the frequency domain, an excess frequency component called **leakage** occurs. To suppress this kind of leakage, simple truncation is not performed, and a truncation function having a small sidelobe in the frequency domain, such as the Hanning window, is used. However, in general, the more the leakage is suppressed, the more the result of the discrete Fourier transform widens and blurs. Therefore, when estimating the power spectrum, you must select a suitable truncation function according to your objectives, that is, according to whether the spectral width or the central frequency is to be the problem, for example.

(g) To raise the resolution (sampling interval in the frequency domain) $\frac{1}{nT}$ of the discrete Fourier transform, you should increase the number of sample data n or increase the sampling interval T. However, to raise the precision of the power spectrum estimate while holding the sampling interval and resolution fixed, a technique is often used of taking m groups of samples for which the number of samples is n, obtaining the modified periodogram for each of those m groups, and then taking the average of those values. In this case, a technique is also proposed in which the m groups of sample data are taken from the series so that they overlap. For details, refer to the Reference Bibliography.

(h) When obtaining the power spectrum, the property related to the frequency transition of the Fourier transform, that is, the multiplication by $e^{2\pi\sqrt{-1}f_0 t}$ in the time (or space) domain, is associated with the shifting of the frequency by f_0 in the frequency domain, and a technique is often used of reducing the number of data points required for the calculation in which the central frequency of the power spectrum is shifted in advance, using the property that the function shape does not change. This kind of operation is known as **modulation**. However, LX=NX+1 (when NX is odd) or
LX=NX+2 (when NX is even)
LY=NY and LZ=NZ.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the waveform defined by the following equation and estimate the power spectrum by calculating the Fourier periodogram.

$$f(x, y, z) = \cos 2\pi f_1 x + \cos 2\pi f_2 y + \cos 2\pi f_3 z$$

(b) Input data
Sampling data
$\mathrm{R}(\mathrm{j_x}+1, \mathrm{j_y}+1, \mathrm{j_z}+1) = \mathrm{f}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ ($\mathrm{j_x} = 0, 1, \cdots, \mathrm{NX}-1$; $\mathrm{j_y} = 0, 1, \cdots, \mathrm{NY}-1$; $\mathrm{j_z} = 0, 1, \cdots, \mathrm{NZ}-1$).
Here, $\Delta = 0.5$.
NX, NY, NZ and ISW.

(c) Main program

```
      PROGRAM OFPS3D
! *** EXAMPLE OF QFPS3D ***
!     IMPLICIT REAL(8) (A-H,O-Z)
      IMPLICIT NONE
      INTEGER NX,NY,NZ,LX,LY,LZ,ISW,IERR,IWK(60)
      INTEGER I,J,K,M,ND2,IS
      INTEGER N0,ISW0
      INTEGER NT
      PARAMETER (N0 =8)
      PARAMETER (ISW0=4)
      PARAMETER (LX = (N0+2)/2*2,LY=N0,LZ=N0)
      REAL(8) R(LX,LY,LZ,-1:ISW0),WK(5*N0+LX*LY*LZ)
      REAL(8) P(-1:ISW0)
      REAL(8) TX,TY,TZ,DT,DFX,DFY,DFZ
```

```
      REAL(8)  PAI,F0,F1,F2,F3
      PARAMETER(PAI=3.141592D0)
      COMMON IWK,R,WK,P
!
      NT=2
      NX=N0
      NY=N0
      NZ=N0
      WRITE (6,1000) ISW0+1,NX,NY,NZ
      DT=0.5D0
      F0=1.0D0/(2.0*DT)
      F1=0.62D0*F0
      F2=0.14D0*F0
      F3=0.55D0*F0
      ND2=(NX+1)/2
      DFX=1.0D0/(DT*NX)
      DFY=1.0D0/(DT*NY)
      DFZ=1.0D0/(DT*NZ)
      P(-1)=0.0D0
      DO 100 K=1,NZ
        TZ=DBLE(K-1)*DT
      DO 101 J=1,NY
        TY=DBLE(J-1)*DT
      DO 102 I=1,NX
        TX=DBLE(I-1)*DT
        R(I,J,K,-1)=COS(2*PAI*F1*TX)+COS(2*PAI*F2*TY)+COS(2*PAI*F3*TZ)
        P(-1)=P(-1)+R(I,J,K,-1)*R(I,J,K,-1)
  102 CONTINUE
  101 CONTINUE
  100 CONTINUE
      P(-1)=P(-1)/(DBLE(NX)*DBLE(NY)*DBLE(NZ))
      DO 110 K=1,NZ
        WRITE (6,1100) K,(I,(R(I,J,K,-1),J=1,NY),I=1,NX)
  110 CONTINUE
      WRITE (6,1150) P(-1)
      WRITE (6,1160) F1,F2,F3
      IS=0
      DO 400 ISW=0,ISW0
      DO 200 K=1,NZ
      DO 201 J=1,NY
      DO 202 I=1,NX
        R(I,J,K,ISW)=R(I,J,K,-1)
  202 CONTINUE
  201 CONTINUE
  200 CONTINUE
      IF (ISW.NE.0) IS=ISW+1
      CALL QFPS3D(NX,NY,NZ,R(1,1,1,ISW),LX,LY,LZ,IS,IWK,WK,NT,IERR)
      P(ISW)=0.0D0
      IF (MOD(NX,2).EQ.0) THEN
        M=ND2-1
        DO 500 K=1,NZ
        DO 501 J=1,NY
        DO 502 I=2,M
          P(ISW)=P(ISW)+2.0D0*R(I,J,K,ISW)
  502   CONTINUE
  501   CONTINUE
  500   CONTINUE
        DO 300 K=1,NZ
        DO 301 J=1,NY
          P(ISW)=P(ISW)+R(1,J,K,ISW)+R(ND2,J,K,ISW)
  301   CONTINUE
  300   CONTINUE
      ELSE
        M=ND2
        DO 600 K=1,NZ
        DO 601 J=1,NY
        DO 602 I=2,M
          P(ISW)=P(ISW)+2.0D0*R(I,J,K,ISW)
  602   CONTINUE
  601   CONTINUE
  600   CONTINUE
        DO 700 K=1,NZ
        DO 701 J=1,NY
          P(ISW)=P(ISW)+R(1,J,K,ISW)
  701   CONTINUE
  700   CONTINUE
      ENDIF
  400 CONTINUE
      WRITE (6,1200)
      WRITE (6,1300) IERR
!
      ISW=0
      WRITE (6,1410) 'RAW',P(ISW)
      DO 800 K=(NZ+1)/2+1,NZ
      WRITE (6,1430) (K-(NZ+1))*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                     (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  800 CONTINUE
      DO 805 K=1,(NZ+1)/2
      WRITE (6,1430) (K-1)*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
```

```
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  805 CONTINUE
!
      ISW=1
      WRITE (6,1410) 'HANNING',P(ISW)
      DO 810 K=(NZ+1)/2+1,NZ
      WRITE (6,1430) (K-(NZ+1))*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  810 CONTINUE
      DO 815 K=1,(NZ+1)/2
      WRITE (6,1430) (K-1)*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  815 CONTINUE
!
      ISW=2
      WRITE (6,1410) 'BARTLETT',P(ISW)
      DO 820 K=(NZ+1)/2+1,NZ
      WRITE (6,1430) (K-(NZ+1))*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  820 CONTINUE
      DO 825 K=1,(NZ+1)/2
      WRITE (6,1430) (K-1)*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  825 CONTINUE
!
      ISW=3
      WRITE (6,1410) 'WELCH',P(ISW)
      DO 830 K=(NZ+1)/2+1,NZ
      WRITE (6,1430) (K-(NZ+1))*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  830 CONTINUE
      DO 835 K=1,(NZ+1)/2
      WRITE (6,1430) (K-1)*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  835 CONTINUE
!
      ISW=4
      WRITE (6,1410) 'PARZEN',P(ISW)
      DO 840 K=(NZ+1)/2+1,NZ
      WRITE (6,1430) (K-(NZ+1))*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  840 CONTINUE
      DO 845 K=1,(NZ+1)/2
      WRITE (6,1430) (K-1)*DFZ,&
          ((J-(NY+1))*DFY,J=(NY+1)/2+1,NY),&
          ((J-1)*DFY,J=1,(NY+1)/2)
      WRITE (6,1420)&
          ((I-1)*DFX,(R(I,J,K,ISW),J=(NY+1)/2+1,NY),&
                      (R(I,J,K,ISW),J=1,(NY+1)/2),I=1,ND2)
  845 CONTINUE
!
 1000 FORMAT(' ',/,/,&
             ' ***  QFPS3D  ***',/,&
             2X,'**  INPUT  **',/,&
             6X,'ISW =0, 2 TO ',I3,/,&
             6X,'NX =',I3,/,&
             6X,'NY =',I3,/,&
             6X,'NZ =',I3)
 1100 FORMAT(12X,'DATA R(I,J,',I3,')',/,&
             4X,'I/J   1        2        3        4',&
             '        5        6        7        8',/,&
             4X,'-------------------------------------',&
             '-------------------------------------',/,&
             8(2X,I3,8F9.4,/))
 1150 FORMAT(6X,'TIME DOMAIN POWER =',F10.4)
```

```
 1160 FORMAT(6X,'SIGNAL FREQUENCY =(',F10.4,',',F10.4,',',F10.4,')')
 1200 FORMAT(2X,'**  OUTPUT  **')
 1300 FORMAT(6X,'IERR =',I5)
 1410 FORMAT(6X,'(MODIFIED) PERIODOGRAM(',A,')',/,&
             3X,'FREQUENCY DOMAIN POWER=',F8.4)
 1430 FORMAT(17X,' Z-FRQ=',F8.2,/,&
             2X,'X/Y-FRQ',8F8.2,/,&
             2X,'-------------------------------------',&
             '-------------------------------------')
 1420 FORMAT(8(2X,F7.2,8F8.4,/))
      END
```

(d) Output results

```
***  QFPS3D  ***
 **  INPUT  **
     ISW =0, 2 TO   5
     NX =  8
     NY =  8
     NZ =  8
          DATA R(I,J,  1)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   3.0000   2.9048   2.6374   2.2487   1.8126   1.4122   1.1237   1.0020
  2   1.6319   1.5367   1.2693   0.8806   0.4445   0.0441  -0.2444  -0.3662
  3   1.2710   1.1759   0.9085   0.5197   0.0836  -0.3168  -0.6053  -0.7270
  4   2.9048   2.8097   2.5423   2.1535   1.7174   1.3170   1.0285   0.9068
  5   2.0628   1.9676   1.7002   1.3115   0.8754   0.4750   0.1865   0.0648
  6   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
  7   2.6374   2.5422   2.2748   1.8861   1.4500   1.0496   0.7611   0.6394
  8   2.4818   2.3866   2.1192   1.7304   1.2944   0.8940   0.6054   0.4837

          DATA R(I,J,  2)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   1.8436   1.7484   1.4810   1.0923   0.6562   0.2558  -0.0327  -0.1545
  2   0.4754   0.3803   0.1129  -0.2759  -0.7119  -1.1123  -1.4009  -1.5226
  3   0.1146   0.0194  -0.2480  -0.6367  -1.0728  -1.4732  -1.7617  -1.8834
  4   1.7484   1.6532   1.3858   0.9971   0.5610   0.1606  -0.1279  -0.2496
  5   0.9064   0.8112   0.5438   0.1550  -0.2810  -0.6814  -0.9699  -1.0917
  6  -0.1075  -0.2027  -0.4701  -0.8588  -1.2949  -1.6953  -1.9838  -2.1055
  7   1.4810   1.3858   1.1184   0.7297   0.2936  -0.1068  -0.3953  -0.5170
  8   1.3253   1.2301   0.9627   0.5740   0.1379  -0.2625  -0.5510  -0.6727

          DATA R(I,J,  3)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
  2  -0.3192  -0.4144  -0.6818  -1.0705  -1.5066  -1.9070  -2.1955  -2.3172
  3  -0.6800  -0.7752  -1.0426  -1.4313  -1.8674  -2.2678  -2.5563  -2.6781
  4   0.9538   0.8586   0.5912   0.2025  -0.2336  -0.6340  -0.9225  -1.0443
  5   0.1117   0.0166  -0.2508  -0.6396  -1.0756  -1.4760  -1.7646  -1.8863
  6  -0.9021  -0.9973  -1.2647  -1.6534  -2.0895  -2.4899  -2.7784  -2.9001
  7   0.6864   0.5912   0.3238  -0.0649  -0.5010  -0.9014  -1.1899  -1.3117
  8   0.5307   0.4355   0.1681  -0.2206  -0.6567  -1.0571  -1.3456  -1.4673

          DATA R(I,J,  4)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   2.4540   2.3588   2.0914   1.7027   1.2666   0.8662   0.5777   0.4560
  2   1.0859   0.9907   0.7233   0.3346  -0.1015  -0.5019  -0.7904  -0.9122
  3   0.7250   0.6298   0.3624  -0.0263  -0.4624  -0.8628  -1.1513  -1.2730
  4   2.3588   2.2636   1.9962   1.6075   1.1714   0.7710   0.4825   0.3608
  5   1.5168   1.4216   1.1542   0.7655   0.3294  -0.0710  -0.3595  -0.4812
  6   0.5029   0.4078   0.1404  -0.2484  -0.6844  -1.0849  -1.3734  -1.4951
  7   2.0914   1.9962   1.7288   1.3401   0.9040   0.5036   0.2151   0.0934
  8   1.9357   1.8406   1.5732   1.1844   0.7484   0.3480   0.0594  -0.0623

          DATA R(I,J,  5)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   2.8090   2.7138   2.4464   2.0577   1.6216   1.2212   0.9327   0.8110
  2   1.4409   1.3457   1.0783   0.6896   0.2535  -0.1469  -0.4354  -0.5571
  3   1.0800   0.9849   0.7175   0.3287  -0.1073  -0.5077  -0.7963  -0.9180
  4   2.7138   2.6187   2.3513   1.9625   1.5265   1.1261   0.8375   0.7158
  5   1.8718   1.7766   1.5092   1.1205   0.6844   0.2840  -0.0045  -0.1262
  6   0.8580   0.7628   0.4954   0.1067  -0.3294  -0.7298  -1.0183  -1.1401
  7   2.4464   2.3513   2.0839   1.6951   1.2591   0.8587   0.5701   0.4484
  8   2.2908   2.1956   1.9282   1.5395   1.1034   0.7030   0.4145   0.2927

          DATA R(I,J,  6)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   1.2929   1.1977   0.9303   0.5416   0.1055  -0.2949  -0.5834  -0.7051
  2  -0.0752  -0.1704  -0.4378  -0.8265  -1.2626  -1.6630  -1.9515  -2.0733
  3  -0.4361  -0.5312  -0.7987  -1.1874  -1.6235  -2.0239  -2.3124  -2.4341
  4   1.1977   1.1025   0.8351   0.4464   0.0103  -0.3901  -0.6786  -0.8003
  5   0.3557   0.2605  -0.0069  -0.3956  -0.8317  -1.2321  -1.5206  -1.6423
  6  -0.6582  -0.7533  -1.0207  -1.4095  -1.8455  -2.2459  -2.5345  -2.6562
  7   0.9303   0.8351   0.5677   0.1790  -0.2571  -0.6575  -0.9460  -1.0677
  8   0.7747   0.6795   0.4121   0.0233  -0.4127  -0.8131  -1.1017  -1.2234

          DATA R(I,J,  7)
  I/J   1        2        3        4        5        6        7        8
```

```
  ------------------------------------------------------------------------
  1   1.4122   1.3170   1.0496   0.6609   0.2248  -0.1756  -0.4641  -0.5858
  2   0.0441  -0.0511  -0.3185  -0.7072  -1.1433  -1.5437  -1.8322  -1.9539
  3  -0.3168  -0.4119  -0.6793  -1.0681  -1.5041  -1.9045  -2.1931  -2.3148
  4   1.3170   1.2219   0.9545   0.5657   0.1297  -0.2707  -0.5593  -0.6810
  5   0.4750   0.3798   0.1124  -0.2763  -0.7124  -1.1128  -1.4013  -1.5230
  6  -0.5388  -0.6340  -0.9014  -1.2902  -1.7262  -2.1266  -2.4152  -2.5369
  7   1.0496   0.9545   0.6871   0.2983  -0.1377  -0.5381  -0.8267  -0.9484
  8   0.8940   0.7988   0.5314   0.1427  -0.2934  -0.6938  -0.9823  -1.1041

         DATA R(I,J,  8)
  I/J   1        2        3        4        5        6        7        8
  ------------------------------------------------------------------------
  1   2.8910   2.7958   2.5284   2.1397   1.7036   1.3032   1.0147   0.8930
  2   1.5229   1.4277   1.1603   0.7716   0.3355  -0.0649  -0.3534  -0.4751
  3   1.1620   1.0669   0.7995   0.4107  -0.0253  -0.4257  -0.7143  -0.8360
  4   2.7958   2.7007   2.4333   2.0445   1.6085   1.2080   0.9195   0.7978
  5   1.9538   1.8586   1.5912   1.2025   0.7664   0.3660   0.0775  -0.0442
  6   0.9399   0.8448   0.5774   0.1886  -0.2474  -0.6478  -0.9364  -1.0581
  7   2.5284   2.4333   2.1659   1.7771   1.3410   0.9406   0.6521   0.5304
  8   2.3728   2.2776   2.0102   1.6215   1.1854   0.7850   0.4965   0.3747

    TIME DOMAIN POWER =    1.6439
    SIGNAL FREQUENCY =(    0.6200,    0.1400,    0.5500)
**  OUTPUT  **
    IERR =    0
    (MODIFIED) PERIODOGRAM(RAW)
 FREQUENCY DOMAIN POWER=  1.5531
                Z-FRQ=   -1.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0007  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=    0.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0158  0.0188  0.0350  0.2150  0.0583  0.2150  0.0350  0.0188
   0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

                Z-FRQ=    0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=    0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=    0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

    (MODIFIED) PERIODOGRAM(HANNING)
 FREQUENCY DOMAIN POWER=  1.0699
                Z-FRQ=   -1.00
```

```
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0001  0.0004  0.0001  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0009  0.0164  0.0045  0.0053  0.0009  0.0000
   0.25  0.0000  0.0000  0.0002  0.0027  0.0004  0.0023  0.0002  0.0000
   0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000

                Z-FRQ=    0.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0001  0.0036  0.0427  0.0087  0.0427  0.0036  0.0001
   0.25  0.0000  0.0000  0.0009  0.0064  0.0041  0.0158  0.0009  0.0000
   0.50  0.0000  0.0000  0.0000  0.0136  0.0543  0.0136  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0137  0.0547  0.0137  0.0000  0.0000

                Z-FRQ=    0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0009  0.0053  0.0045  0.0164  0.0009  0.0000
   0.25  0.0000  0.0000  0.0002  0.0006  0.0031  0.0058  0.0002  0.0000
   0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000

                Z-FRQ=    0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=    0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

    (MODIFIED) PERIODOGRAM(BARTLETT)
 FREQUENCY DOMAIN POWER=  1.0593
                Z-FRQ=   -1.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0002  0.0000  0.0079  0.0346  0.0050  0.0000  0.0002
   0.25  0.0000  0.0000  0.0000  0.0014  0.0062  0.0009  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0003  0.0001  0.0000  0.0000

                Z-FRQ=   -0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
   0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

                Z-FRQ=   -0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0001  0.0141  0.0037  0.0074  0.0001  0.0000
   0.25  0.0000  0.0000  0.0000  0.0022  0.0005  0.0017  0.0000  0.0000
   0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
```

```
   0.75  0.0000  0.0001  0.0000  0.0012  0.0096  0.0024  0.0000  0.0001

                Z-FRQ=    0.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0007  0.0594  0.0070  0.0594  0.0007  0.0000
   0.25  0.0000  0.0000  0.0001  0.0089  0.0017  0.0129  0.0001  0.0000
   0.50  0.0000  0.0003  0.0000  0.0113  0.0622  0.0113  0.0000  0.0003
   0.75  0.0000  0.0003  0.0000  0.0069  0.0554  0.0139  0.0000  0.0003

                Z-FRQ=    0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0001  0.0074  0.0037  0.0141  0.0001  0.0000
   0.25  0.0000  0.0000  0.0000  0.0011  0.0011  0.0030  0.0000  0.0000
   0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
   0.75  0.0000  0.0001  0.0000  0.0014  0.0108  0.0027  0.0000  0.0001

                Z-FRQ=    0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
   0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
   0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

                Z-FRQ=    0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0002  0.0000  0.0050  0.0346  0.0079  0.0000  0.0002
   0.25  0.0000  0.0000  0.0000  0.0009  0.0065  0.0015  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0001  0.0008  0.0002  0.0000  0.0000

    (MODIFIED) PERIODOGRAM(WELCH)
 FREQUENCY DOMAIN POWER=  1.2154
                Z-FRQ=   -1.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0005  0.0030  0.0005  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

                Z-FRQ=   -0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0001  0.0001  0.0004  0.0051  0.0357  0.0028  0.0003  0.0001
   0.25  0.0000  0.0000  0.0000  0.0005  0.0038  0.0003  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

                Z-FRQ=   -0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0003  0.0004  0.0011  0.0103  0.1247  0.0186  0.0011  0.0004
   0.25  0.0000  0.0000  0.0001  0.0011  0.0131  0.0020  0.0001  0.0000
   0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0001  0.0017  0.0002  0.0000  0.0000

                Z-FRQ=   -0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0001  0.0122  0.0012  0.0090  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0013  0.0002  0.0009  0.0000  0.0000
   0.50  0.0000  0.0000  0.0001  0.0017  0.0095  0.0005  0.0001  0.0000
   0.75  0.0000  0.0000  0.0001  0.0005  0.0078  0.0012  0.0001  0.0000

                Z-FRQ=    0.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.1034  0.0186  0.1034  0.0000  0.0000
   0.25  0.0000  0.0000  0.0000  0.0115  0.0021  0.0104  0.0000  0.0000
   0.50  0.0002  0.0003  0.0008  0.0158  0.0862  0.0044  0.0007  0.0003
   0.75  0.0002  0.0002  0.0007  0.0052  0.0760  0.0115  0.0007  0.0002

                Z-FRQ=    0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0000  0.0090  0.0012  0.0122  0.0001  0.0000
   0.25  0.0000  0.0000  0.0000  0.0010  0.0001  0.0012  0.0000  0.0000
   0.50  0.0000  0.0000  0.0001  0.0016  0.0086  0.0004  0.0001  0.0000
   0.75  0.0000  0.0000  0.0001  0.0006  0.0083  0.0012  0.0001  0.0000

                Z-FRQ=    0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0003  0.0004  0.0011  0.0186  0.1247  0.0103  0.0011  0.0004
   0.25  0.0000  0.0000  0.0001  0.0020  0.0133  0.0011  0.0001  0.0000
   0.50  0.0000  0.0000  0.0000  0.0004  0.0031  0.0003  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0004  0.0001  0.0000  0.0000

                Z-FRQ=    0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
```

```
   0.00  0.0001  0.0001  0.0003  0.0028  0.0357  0.0051  0.0004  0.0001
   0.25  0.0000  0.0000  0.0000  0.0003  0.0037  0.0005  0.0000  0.0000
   0.50  0.0000  0.0000  0.0000  0.0000  0.0003  0.0000  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0005  0.0001  0.0000  0.0000

    (MODIFIED) PERIODOGRAM(PARZEN)
 FREQUENCY DOMAIN POWER=  0.9132
                Z-FRQ=   -1.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0001  0.0023  0.0053  0.0023  0.0001  0.0000
   0.25  0.0000  0.0000  0.0001  0.0010  0.0023  0.0010  0.0001  0.0000
   0.50  0.0000  0.0000  0.0000  0.0001  0.0001  0.0001  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0006  0.0092  0.0209  0.0089  0.0006  0.0000
   0.25  0.0000  0.0000  0.0003  0.0039  0.0090  0.0038  0.0002  0.0000
   0.50  0.0000  0.0000  0.0000  0.0002  0.0005  0.0002  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

                Z-FRQ=   -0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0014  0.0162  0.0313  0.0112  0.0005  0.0000
   0.25  0.0000  0.0000  0.0005  0.0062  0.0120  0.0044  0.0002  0.0000
   0.50  0.0000  0.0000  0.0000  0.0003  0.0007  0.0004  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0003  0.0007  0.0003  0.0000  0.0000

                Z-FRQ=   -0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0001  0.0030  0.0125  0.0054  0.0016  0.0012  0.0000
   0.25  0.0000  0.0000  0.0008  0.0019  0.0010  0.0030  0.0010  0.0000
   0.50  0.0000  0.0000  0.0001  0.0032  0.0106  0.0063  0.0006  0.0000
   0.75  0.0000  0.0000  0.0003  0.0046  0.0108  0.0048  0.0003  0.0000

                Z-FRQ=    0.00
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0001  0.0047  0.0117  0.0007  0.0117  0.0047  0.0001
   0.25  0.0000  0.0000  0.0011  0.0006  0.0055  0.0140  0.0033  0.0001
   0.50  0.0000  0.0000  0.0003  0.0089  0.0291  0.0168  0.0016  0.0000
   0.75  0.0000  0.0000  0.0007  0.0107  0.0253  0.0111  0.0007  0.0000

                Z-FRQ=    0.25
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0012  0.0016  0.0054  0.0125  0.0030  0.0001
   0.25  0.0000  0.0000  0.0002  0.0005  0.0086  0.0110  0.0019  0.0000
   0.50  0.0000  0.0000  0.0002  0.0048  0.0151  0.0085  0.0008  0.0000
   0.75  0.0000  0.0000  0.0003  0.0047  0.0110  0.0049  0.0003  0.0000

                Z-FRQ=    0.50
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0005  0.0112  0.0313  0.0162  0.0014  0.0000
   0.25  0.0000  0.0000  0.0003  0.0055  0.0155  0.0081  0.0007  0.0000
   0.50  0.0000  0.0000  0.0001  0.0010  0.0028  0.0014  0.0001  0.0000
   0.75  0.0000  0.0000  0.0000  0.0003  0.0008  0.0003  0.0000  0.0000

                Z-FRQ=    0.75
X/Y-FRQ   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
   0.00  0.0000  0.0000  0.0006  0.0089  0.0209  0.0092  0.0006  0.0000
   0.25  0.0000  0.0000  0.0002  0.0039  0.0091  0.0040  0.0003  0.0000
   0.50  0.0000  0.0000  0.0000  0.0003  0.0006  0.0003  0.0000  0.0000
   0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

Chapter 7

SORTING

7.1 INTRODUCTION

This chapter describes the subroutines for sorting data.
Subroutine described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.
This library provides subroutines having the following functions.

(1) Sorting a list of data

(2) Sorting a list of pairwise data

7.1.1 Notes

Since the parallel processing overhead significantly affects the computation cost if the number of data to be sorted is small, performance may be worse than when a non-parallel subroutine is used.

7.1.2 Algorithms Used

The algorithms for sorting in ascending order are shown below. The algorithms for sorting in descending order, which differ only in terms of the relative magnitudes, are similar.

(1) Shell sort
(1) Set the spacing h.
(2) Take all subsequences of spacing h from the data sequence.
(3) Compare adjacent pairs within each subsequence to arrange them in ascending order. If they are in the reverse order, exchange their positions and confirm again the relative order with the preceding data. If they are again in the reverse order, exchange the positions and work back toward the beginning.
(4) Decrease the spacing h and repeat steps (2) and (3). When the processing for h =1 ends, sorting is completed.

(2) Heap sort
(1) Organize the assigned data into a heap tree (well-ordered binary tree in which parents have value greater than or equal to those of children).
(2) Exchange the root and the data at the very end of the tree.
(3) Let the position excluding the very last data be A.
(4) Consider A to be a new tree, and organize this into a heap tree again.
(5) Repeat steps (2) to (4). When the remaining data is only the root, sorting is completed.

(3) Quick sort
(1) Count the number of data values within the sort interval.
(2) Do the following depending on the number of data values.

- If the number of data values is less than or equal to one:
  Do nothing.
- If the number of data values is 2:
  If they are in the reverse order, exchange their positions.
- If the number of data values is greater than or equal to three:
  ① Select one pivot value from within the interval.
  ② Divide the data within the interval into two intervals consisting of values greater than the pivot value and values less than the pivot value.

(3) Repeat steps (1) and (2). When the number of data values in all data intervals is less than or equal to two, sorting is completed.

(4) Merge sort
(1) Count the number of data values within the sort interval.
(2) Do the following depending on the number of data values.

- If the number of data value is one:
  Do nothing.
- If the number of data values is two:
  If they are in the reverse order, exchange the positions.
- If the number of data values is greater than or equal to three:
  ① Divide the data within the interval in half into the top half and bottom half.

② Recursively merge sort the top half. Recursively merge sort the bottom half.
③ Merge the sorted top half and bottom half.

7.1.3 Reference Bibliography

(1) Niklaus Wirth, "ALGORITHMS + DATA STRUCTURES = PROGRAMS", Prentice–Hall Inc. (1976).

(2) Hiroto Namihira,"Sorting and Searching", CQ Publishing Co. Ltd.

(3) Yoshiyuki Kondo,"Algorithms and Data Structures", Softbank Publishing Inc.

7.2 SORTING

7.2.1 QSSTA1, PSSTA1
Sorting a List of Data

(1) **Function**

Given n data values $a_{i_k}(k = 1, 2, \cdots, n)$, the QSSTA1 or PSSTA1 obtains the data sequence $a_{j_k}(k = 1, 2, \cdots, n)$ consisting of the original data values a_i rearranged in ascending or descending order. Here,a_j satisfies ths following relationship.

For ascending order : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

For descending order : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

(2) **Usage**

Double precision:

```
CALL QSSTA1  (A,N,ISW,WK,IWK,NT, IERR)
```

Single precision:

```
CALL PSSTA1  (A,N,ISW,WK,IWK,NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{ D / R }	N	Input	Data to be sorted a_i
				Output	Sorted data a_j
2	N	I	1	Input	Size of array A
3	ISW	I	1	Input	Sort method selection switch (See Note (a))
4	WK	{ D / R }	N	Work	Work area
5	IWK	I	2×N	Work	Work area
6	NT	I	1	Input	Number of tasks to be generated
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1

(b) ISW $\in \{\pm 1, \pm 2, \pm 3, \pm 4\}$

(c) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The sort methods to be selected by ISW are as follows.

ISW	Sort Method	ISW	Sort Method
1	Shell sort (ascending order)	−4	Shell sort (descending order)
2	Heap sort (ascending order)	−2	Heap sort (descending order)
3	Quick sort (ascending order)	−3	Quick sort (descending order)
4	Merge sort (ascending order)	−4	Merge sort (descending order)

The user should select an appropriate sort method according to the properties of the input data. The features of each sort method are shown below.

· Shell sort

The average of the amount of calculation is on the order of $0(n^{1.5})$. A fast, stable sort is performed for any kind of data. Sorting is faster if part of the data sequence has been sorted.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

No work area is necessary.

· Heap sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

No work area is necessary.

· Quick sort

Although the average of the amount of calculation is on the order of $O(n \log n)$, this is an extremely inefficient sort for data having certain types of regularities such as data that has been partially sorted to begin with. This is the fastest sort method for random data.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

· Merge sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

The ordinal relationships before sorting among data having the same value is retained after sorting.

(7) **Example**

(a) Problem

Sort the following data in ascending order by using a shell sort.

A(1) = 5.0

A(2) = 4.0

A(3) = 9.0
A(4) = 6.0
A(5) = 2.0
A(6) = 5.0

(b) Input data

Array A, N=6 and ISW=1.

(c) Main program

```
      PROGRAM OSSTA1
! *** EXAMPLE OF PSSTA1 ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,ISW,NT,IWK(2*NA),IERR,I
      REAL(8) A(NA),WK(NA)
!
!     DATA SET
      DATA (A(I),I=1,6) /5.0D0,4.0D0,9.0D0,6.0D0,2.0D0,5.0D0/
      N   = 6
      ISW = 1
      NT  = 4
!
!     WRITE INPUT DATA
      WRITE(6,6000) ISW,N,NT
      DO 110 I=1,N
         WRITE(6,6010) I,A(I)
  110 CONTINUE
!
!     SORT
      CALL QSSTA1(A,N,ISW,WK,IWK,NT,IERR)
!
!     WRITE OUTPUT DATA
      WRITE(6,6020) IERR
      IF( IERR .LT. 3000 ) THEN
         DO 120 I=1,N
            WRITE(6,6010) I,A(I)
  120    CONTINUE
      ENDIF
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QSSTA1  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     ISW =',I6,/,&
             1X,'     N   =',I6,/,&
             1X,'     NT  =',I6,/)
 6010 FORMAT(1X,'     A(',I2,')=',F5.1)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
      END
```

(d) Output results

```
***  QSSTA1  ***

 **  INPUT  **

     ISW =     1
     N   =     6
     NT  =     4

     A( 1)=  5.0
     A( 2)=  4.0
     A( 3)=  9.0
     A( 4)=  6.0
     A( 5)=  2.0
     A( 6)=  5.0

 **  OUTPUT  **

     IERR =    0

     A( 1)=  2.0
     A( 2)=  4.0
     A( 3)=  5.0
     A( 4)=  5.0
     A( 5)=  6.0
     A( 6)=  9.0
```

7.2.2 QSSTA2, PSSTA2
Sorting a List of Pairwise Data

(1) **Function**

Given two sets of n data values $a_{i_k}(k = 1, 2, \cdots, n), b_{i_k}(k = 1, 2, \cdots, n)$, the QSSTA2 or PSSTA2 obtains the data sequence $a_{j_k}(k = 1, 2, \cdots, n)$ consisting of the original a_i data values rearranged in ascending or ascending order and the data sequence $b_{j_k}(k = 1, 2, \cdots, n)$ corresponding to it.
Here, a_j satisfies the following relationship.

For ascending order : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

For descending order : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

If a second order sort is specified, the subroutine determines $j = j_1, j_2, \cdots, j_n$ so that the following relationship is satisfied for any k for which $a_{j_k} = a_{j_{k+1}}$ is satisfied.

For ascending order : $b_{j_k} \leq b_{j_{k+1}}$

For descending order : $b_{j_k} \geq b_{j_{k+1}}$

(2) **Usage**

Double precision:

```
CALL QSSTA2  (A,N,B,ISW1,ISW2,WK,IWK,NT, IERR)
```

Single precision:

```
CALL PSSTA2  (A,N,B,ISW1,ISW2,WK,IWK,NT, IERR)
```

(3) **Arguments**

D:Double precision real Z:Double precision complex I: { INTEGER(4) as for 32bit Integer / INTEGER(8) as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument	Type	Size	Input/ Output	Contents
1	A	{D / R}	N	Input	Data to be sorted a_i
				Output	Sorted data a_j
2	N	I	1	Input	Size of array A
3	B	{D / R}	N	Input	Data b_i corresponding to a_i
				Output	Data b_j corresponding to sorted a_j
4	ISW1	I	1	Input	Sort method selection switch (See Note (a))
5	ISW2	I	1	Input	Second order sort switch ISW2=0 : Do not perform second order sort ISW2=1 : Perform second order sort
6	WK	{D / R}	2×N	Work	Work area
7	IWK	I	2×N	Work	Work area
8	NT	I	1	Input	Number of tasks to be generated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1

(b) ISW1 $\in \{\pm 1, \pm 2, \pm 3, \pm 4\}$

(c) ISW2 $\in \{0, 1\}$

(d) NT ≥ 1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	

(6) **Notes**

(a) The sort methods to be selected by ISW1 are as follows.

ISW1	Sort Method	ISW1	Sort Method
1	Shell sort (ascending order)	−1	Shell sort (descending order)
2	Heap sort (ascending order)	−2	Heap sort (descending order)
3	Quick sort (ascending order)	−3	Quick sort (descending order)
4	Merge sort (ascending order)	−4	Merge sort (descending order)

The user should select an appropriate sort method according to the properties of the input data. The

features of each sort method are shown below.

· Shell sort

The average of the amount of calculation is on the order of $O(n^{1.5})$. A fast, stable sort is performed for any kind of data. Sorting is faster if part of the data sequence has been sorted.

It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.

No work area is necessary.

· Heap sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.

No work area is necessary.

· Quick sort

Although the average of the amount of calculation is on the order of $O(n \log n)$, this is an extremely inefficient sort for data having certain types of regularities such as data that has been partially sorted to begin with. This is the fastest sort method for random data.

It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.

· Merge sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

The ordinal relationships before sorting among data having the same value is retained after sorting.

(7) **Example**

(a) Problem

Sort the following data for A in ascending order by using a shell sort and rearrange the corresponding data for B according to the sorted data for A. Also perform a second order sort.

A(1) = 5.0, B(1) = 3.0
A(2) = 4.0, B(2) = 4.0
A(3) = 9.0, B(3) = 2.0
A(4) = 6.0, B(4) = 3.0
A(5) = 2.0, B(5) = 8.0
A(6) = 5.0, B(6) = 1.0

(b) Input data

Arrays A and B, N=6, ISW1=1 and ISW2=1.

(c) Main program

```
      PROGRAM OSSTA2
! *** EXAMPLE OF PSSTA2 ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,ISW1,ISW2,NT,IWK(2*NA),IERR,I
      REAL(8) A(NA),B(NA),WK(2*NA)
!
!     DATA SET
      DATA (A(I),I=1,6) /5.0D0,4.0D0,9.0D0,6.0D0,2.0D0,5.0D0/
      DATA (B(I),I=1,6) /3.0D0,4.0D0,2.0D0,3.0D0,8.0D0,1.0D0/
      N    = 6
      ISW1 = 1
```

```
      ISW2 = 1
      NT   = 4
!
!     WRITE INPUT DATA
      WRITE(6,6000) ISW1,ISW2,N,NT
      DO 110 I=1,N
         WRITE(6,6010) I,A(I),I,B(I)
  110 CONTINUE
!
!     SORT
      CALL QSSTA2(A,N,B,ISW1,ISW2,WK,IWK,NT,IERR)
!
!     WRITE OUTPUT DATA
      WRITE(6,6020) IERR
      IF( IERR .LT. 3000 ) THEN
         DO 120 I=1,N
            WRITE(6,6010) I,A(I),I,B(I)
  120    CONTINUE
      ENDIF
!
      STOP
 6000 FORMAT(/,&
             1X,'***  QSSTA2  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     ISW1 =',I6,/,&
             1X,'     ISW2 =',I6,/,&
             1X,'     N    =',I6,/,&
             1X,'     NT   =',I6,/)
 6010 FORMAT(1X,'     A(',I2,')=',F5.1,7X,'B(',I2,')=',F5.1)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
      END
```

(d) Output results

```
***  QSSTA2  ***

 **  INPUT  **

     ISW1 =     1
     ISW2 =     1
     N    =     6
     NT   =     4

     A( 1)=  5.0       B( 1)=  3.0
     A( 2)=  4.0       B( 2)=  4.0
     A( 3)=  9.0       B( 3)=  2.0
     A( 4)=  6.0       B( 4)=  3.0
     A( 5)=  2.0       B( 5)=  8.0
     A( 6)=  5.0       B( 6)=  1.0

 **  OUTPUT  **

     IERR =    0

     A( 1)=  2.0       B( 1)=  8.0
     A( 2)=  4.0       B( 2)=  4.0
     A( 3)=  5.0       B( 3)=  1.0
     A( 4)=  5.0       B( 4)=  3.0
     A( 5)=  6.0       B( 5)=  3.0
     A( 6)=  9.0       B( 6)=  2.0
```

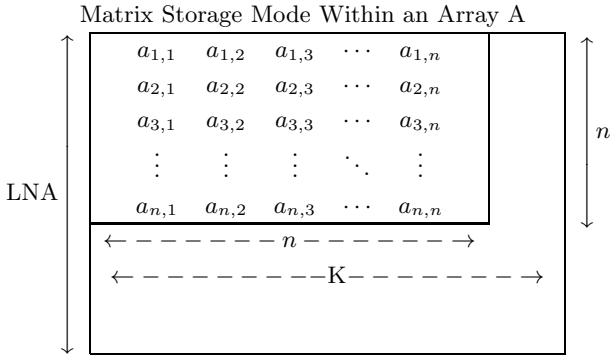
Appendix A

METHODS OF HANDLING ARRAY DATA

A.1 Methods of handling array data corresponding to matrix

Since the ASL subroutine library uses array data corresponding to matrix, this section describes various methods of handling arrays.

To call a subroutine that uses array data, you must declare that array in advance in the calling program. If the declared array is A(LNA, K), then $n \times n$ matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$ is stored in array A as shown in the figure below.



Remarks

a. LNA $\geq n$ and K $\geq n$ must hold.

b. Matrix element $a_{i,j}$ corresponds to the array element A(i, j).

Figure A−1 Matrix Storage Mode Within an Array A()

LNA is called an adjustable dimension. If a two-dimensional array is used as an argument, the adjustable must be passed to the subroutine as an argument in addition to the array name and order of the array. The matrix elements $a_{i,j}$ $(i = 1, 2, \cdots$,LNA; $j = 1, 2, \cdots$,K) must correspond to the array element A(i, j) (i $= 1, 2, \cdots,$ LNA; j $= 1, 2, \cdots,$ K) , as follows on the main memory.

$a_{1,1}$	$a_{2,1}$	$\cdots$	$a_{\mathrm{LNA},1}$	$a_{1,2}$	$a_{2,2}$	$\cdots$
$\updownarrow$	$\updownarrow$	$\cdots$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\cdots$
A$(1,1)$	A$(2,1)$	$\cdots$	A(LNA, 1)	A$(1,2)$	A$(2,2)$	$\cdots$

Example DAM1AD (Real matrix addition)

Add 3×2 matrices A and B placing the sum in matrix C. If you declare arrays of size (5, 4), the declaration and CALL statements are as follows.

```
      REAL(8) A(5, 4), B(5, 4), C(5, 4)
      INTEGER IERR
C
      CALL DAM1AD(A, 5, 3, 2, B, 5, C, 5, IERR)
```

Data is stored in A as follows. Data are stored in B and C in the same way.

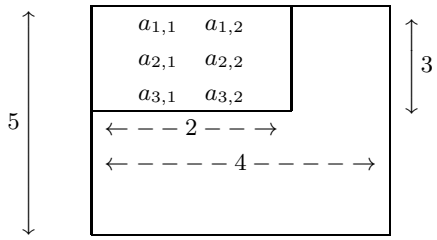


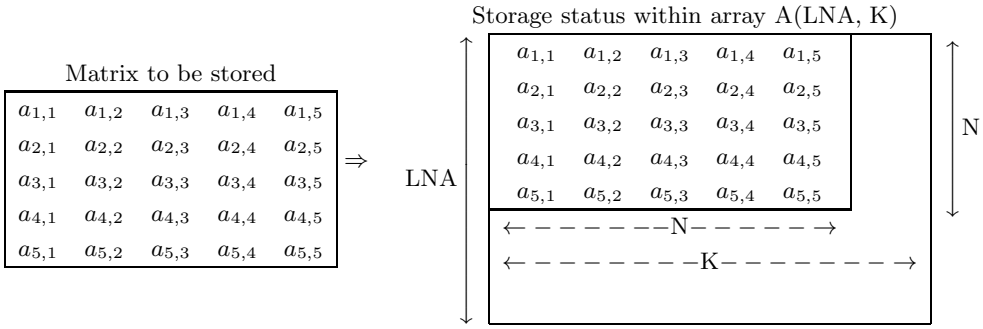
Figure A−2 Matrix Storage Mode Within an Array A

If you will be manipulating several arrays having different orders as data, you can prepare one array having LNA equal to the largest order and use that array successively for each array. However, you must always assign the LNA value as an adjustable dimension.

A.2 Data storage modes

Matrix data storage modes differ according to the matrix type. Storage modes for each type of matrix are shown below.

A.2.1 Real matrix (two-dimensional array type)



Remarks

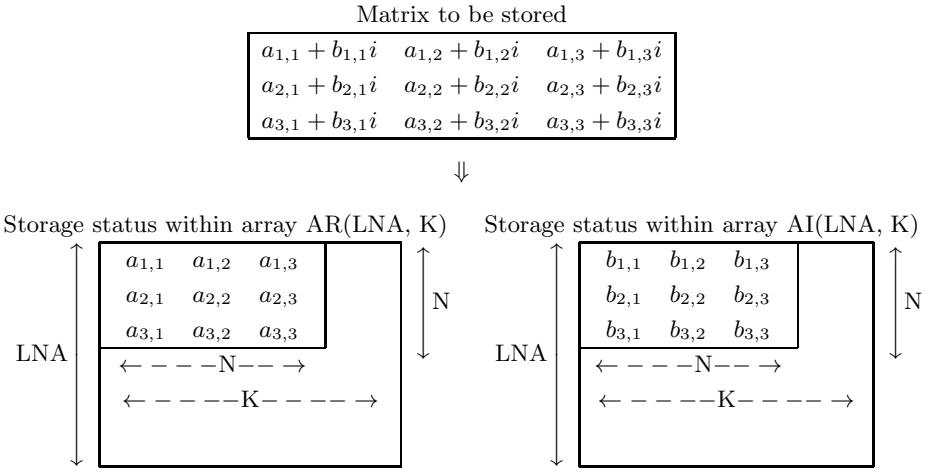
a. LNA ≥ N and K ≥ N must hold.

Figure A−3 Real Matrix (Two-Dimensional Array Type) Storage Mode

A.2.2 Complex matrix

(1) **Two-dimensional array type, real argument type**

Real and imaginary parts are stored in separate arrays.

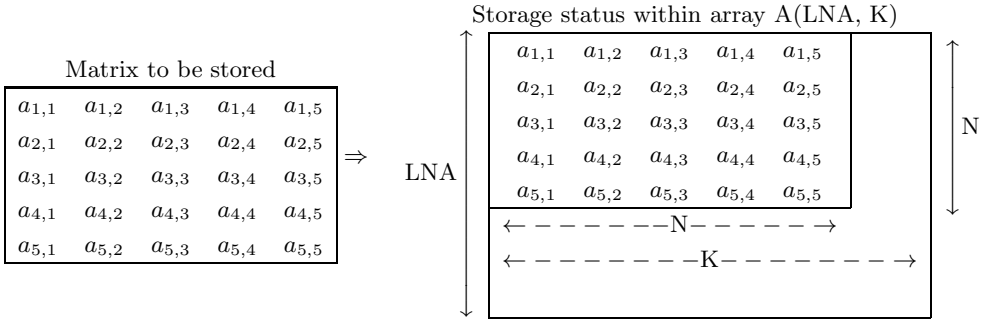


Remarks

a. LNA ≥ N and K ≥ N must hold.

Figure A−4 Complex Matrix (Two-dimensional Array Type) (Real Argument Type) Storage Mode

(2) **Two-dimensional array type, complex argument type**



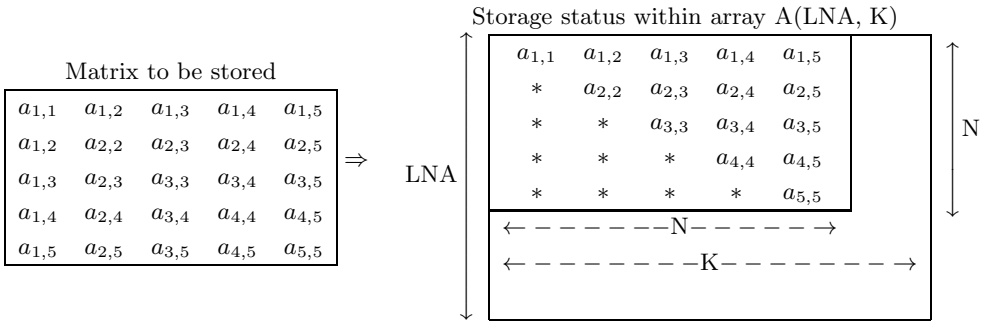
Remarks

a. LNA ≥ N and K ≥ N must hold.

Figure A−5 Complex Matrix (Two-dimensional Array Type)(Complex Argument Type) Storage Mode

A.2.3 Real symmetric matrix and positive symmetric matrix

(1) **Two-dimensional array type, upper triangular type**



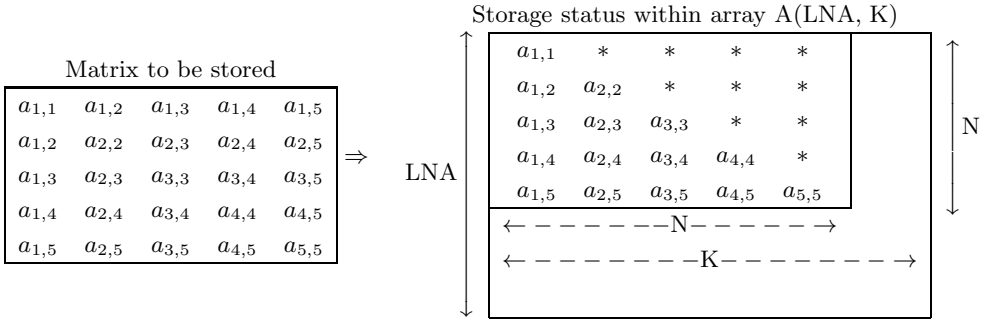
Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. LNA ≥ N and K ≥ N must hold.

Figure A−6 Real Symmetric Matrix (Two-dimensional Array Type) (Upper Triangular Type) Storage mode

(2) **Two-dimensional array type, lower triangular type**



Remarks

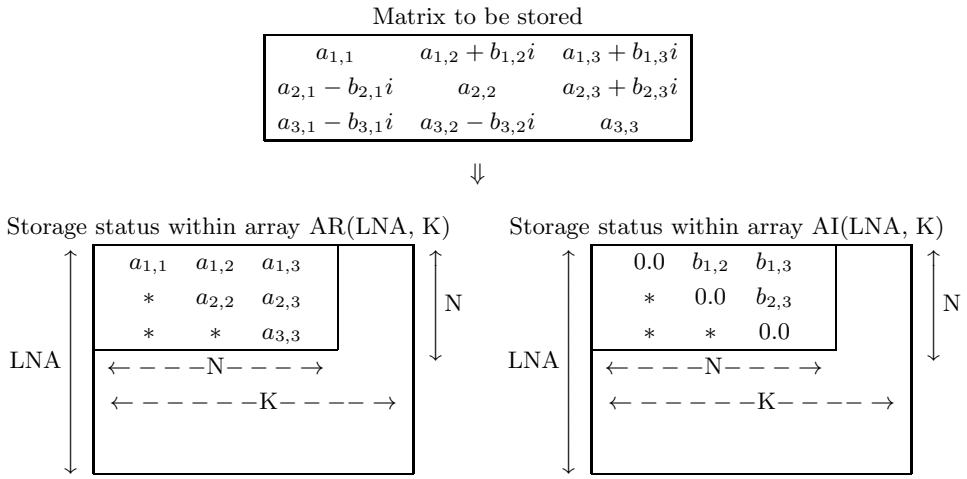
a. The asterisk (∗) indicates an arbitrary value.

b. LNA ≥ N and K ≥ N must hold.

Figure A−7 Real Symmetric Matrix (Two-dimensional Array Type, Lower Triangular Type) Storage mode

A.2.4 Hermitian matrix

(1) **Two-dimensional array type, real argument type, upper triangular type**
Upper triangular portions of the real and imaginary parts are stored in separate arrays.



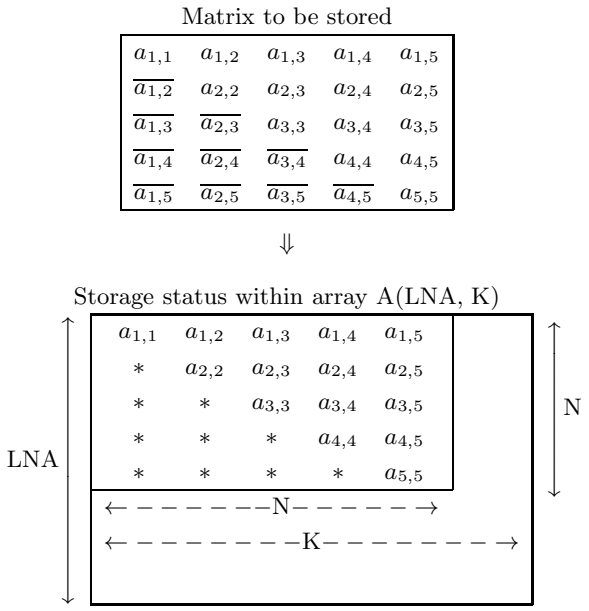
Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. LNA ≥ N and K ≥ N must hold.

Figure A−8 Hermitian Matrix (Two-dimensional Array Type) (Real Argument Type) (Upper Triangular Type) Storage Mode

(2) **Two-dimensional array type, complex argument type, upper triangular type**



Remarks

a. The $\overline{x}$ indicates the complex conjugate of x.

b. The asterisk $*$ indicates an arbitrary value.

c. LNA $\geq$ N and K $\geq$ N must hold.

Figure A−9 Hermitian Matrix (Two-dimensional Array Type) (Complex Argument Type) (Upper Triangular Type) Storage Mode

A.2.5 Random sparse matrix (For symmetric matrix only)

(1) **Sparse format (Symmetric case)**

Matrix A to be stored

$a_{1,1}$	0.0	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	0.0
0.0	$a_{2,2}$	0.0	$a_{2,4}$	0.0	0.0
$a_{1,3}$	0.0	$a_{3,3}$	0.0	$a_{3,5}$	0.0
$a_{1,4}$	$a_{2,4}$	0.0	$a_{4,4}$	0.0	$a_{4,6}$
$a_{1,5}$	0.0	$a_{3,5}$	0.0	$a_{5,5}$	0.0
0.0	0.0	0.0	$a_{4,6}$	0.0	$a_{6,6}$

⇓

Storage status of arrays AVAL(NA),JCN(NA) and IA(N)

	AVAL	JCN			IA
NA	$a_{1,1}$	1	M	N	1
	$a_{1,3}$	3			5
	$a_{1,4}$	4			7
	$a_{1,5}$	5			9
	$a_{2,2}$	2			11
	$a_{2,4}$	4			12
	$a_{3,3}$	3			
	$a_{3,5}$	5			
	$a_{4,4}$	4			
	$a_{4,6}$	6			
	$a_{5,5}$	5			
	$a_{6,6}$	6			

Remarks

a. M is the number of nonzero elements in the upper triangular part of the original matrix A including the diagonal.

b. Array AVAL contains the nonzero upper triangular elements of the original matrix A, stored sequentially beginning with the first row.

c. Array JCN contains the column numbers in the original matrix A of the elements stored in array AVAL.

d. Array IA contains values equal to the positions in array AVAL of the diagonal elements.

e. $N \leq M < NA$ must hold.

Figure A−10 Storage of Random Symmetric Sparse Matrix (Sparse format)

A.2.6 Random sparse matrix

(1) **Sparse format**

Matrix A to be stored

$a_{1,1}$	0.0	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	0.0
$a_{2,1}$	$a_{2,2}$	0.0	$a_{2,4}$	0.0	0.0
0.0	$a_{3,2}$	$a_{3,3}$	0.0	$a_{3,5}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	0.0	$a_{4,6}$
$a_{5,1}$	0.0	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	0.0
$a_{6,1}$	0.0	$a_{6,3}$	0.0	$a_{6,5}$	$a_{6,6}$

⇓

Storage status of arrays AVAL(NA), JCN(NA) and IA(N)

NA

AVAL	JCN
$a_{1,1}$	1
$a_{1,3}$	3
$a_{1,4}$	4
$a_{1,5}$	5
$a_{2,1}$	1
$a_{2,2}$	2
$a_{2,4}$	4
⋮	⋮
$a_{6,1}$	1
$a_{6,3}$	3
$a_{6,5}$	5
$a_{6,6}$	6

N

IA
1
5
8
11
15
19

Remarks

a. NA is the number of nonzero elements of the original matrix A.

b. Array AVAL contains the nonzero elements of the original matrix A, stored sequentially beginning with the first row.

c. Array JCN contains the column indices in the original matrix A of the elements stored in array AVAL.

d. Array IA contains values equal to the positions in array AVAL of the first nonzero element in each row.

e. N < NA must hold.

Figure A−11 Storage of Real Asymmetric Random Sparse Matrix (Sparse Format)

(2) **ELLPACK format**

Matrix A to be stored

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

$\Downarrow$

Storage status of arrays A(LNA, M), JA(LNA, M)

LNA

A				
$a_{1,1}$	$a_{1,2}$	$a_{1,6}$	$a_{1,4}$	$*$
$a_{2,2}$	$a_{2,1}$	$a_{2,3}$	$a_{2,4}$	$a_{2,7}$
$a_{3,3}$	$a_{3,2}$	$a_{3,6}$	$*$	$*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$a_{6,6}$	$a_{6,1}$	$a_{6,3}$	$a_{6,4}$	$a_{6,7}$
$a_{7,7}$	$a_{7,2}$	$*$	$*$	$*$
$<---$		M		$--->$

JA				
1	2	6	4	0
2	1	3	4	7
3	2	6	0	$*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
6	1	3	4	7
7	2	0	$*$	$*$
$<---$		M		$--->$

N

Remarks

a. N is order of Matrix A.

b. LNA $\geq$ N must hold.

c. M is the column number of Array A, which contains the nonzero elements of Matrix A.

d. Array A should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though M-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array JA should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array A.
For those rows in which M $-$ 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in JA in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

Figure A−12 Storage format for Asymmetric Random Sparse Matrix (ELLPACK Format)

Appendix B

MACHINE CONSTANTS USED IN ASL

B.1 Units for Determining Error

The table below shows values in ASL as units for determining error in floating point calculations. The units shown in the table are numeric values determined by the internal representation of floating point data. ASL uses these units for determining convergence and zeros.

Table B−1 Units for Determining Error

Single-precision	Double-precision
$2^{-23}(\simeq 1.19 \times 10^{-7})$	$2^{-52}(\simeq 2.22 \times 10^{-16})$

Remark: The unit for determining error ε, which is also called the machine ε, is usually defined as the smallest positive constant for which the calculation result of $1+\varepsilon$ differs from 1 in the corresponding floating point mode. Therefore, seeing the unit for determining error enables you to know the maximum number of significant digits of an operation (on the mantissa) in that floating point mode.

B.2 Maximum and Minimum Values of Floating Point Data

The table below shows maximum and minimum values of floating point data defined within ASL. Note that the maximum and minimum values shown below may differ from the maximum and minimum values that are actually used by the hardware for each floating point mode.

Table B−2 Maximum and Minimum Values of Floating Point Data

	Single-precision	Double-precision
Maximum value	$2^{127}(2-2^{-23})$ $(\simeq 3.40 \times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80 \times 10^{308})$
Positive minimum value	2^{-126} $(\simeq 1.17 \times 10^{-38})$	2^{-1022} $(\simeq 2.23 \times 10^{-308})$
Negative maximum value	-2^{-126} $(\simeq -1.17 \times 10^{-38})$	-2^{-1022} $(\simeq -2.23 \times 10^{-308})$
Minimum value	$-2^{127}(2-2^{-23})$ $(\simeq -3.40 \times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80 \times 10^{308})$

Index

(*) DMP Functions: Distributed Memory Parallel Functions
(*) SMP Functions: Shared Memory Parallel Functions