# ADVANCED SCIENTIFIC LIBRARY
# ASL
## User's Guide
## <Basic Functions Vol.5>

# 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 basic functions, volume 5.

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 $\chi^2$ 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)<br>REAL(8) | 32bit integer Double-precision subroutine | 32bit integer library<br>(link option: -lasl_sequential) |
| 4 | 4 | INTEGER(4)<br>REAL(4) | 32bit integer Single-precision subroutine | |
| 8 | 8 | INTEGER(8)<br>REAL(8) | 64bit integer Double-precision subroutine | 64bit integer library<br>(link option: -lasl_sequential_i64) |
| 8 | 4 | INTEGER(8)<br>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 $3 \times \mathrm{N}$ or $\mathrm{N} + \mathrm{M}$.

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: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}^{\text{Note}}$ | 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<br>IPVT(i): Number of the row exchanged with row $i$ in the $i$-th step. |
| 5 | COND | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 1 | Output | Reciprocal of the condition number |
| 6 | W1 | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 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 $\left\{ \begin{array}{c} \text{double-precision} \\ \text{single-precision} \end{array} \right\}^{\text{Note}}$ real array of size (LNA , N) , IPVT is an integer array of size N and W1 is a $\left\{ \begin{array}{c} \text{double-precision} \\ \text{single-precision} \end{array} \right\}$ 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

<table>
<tr><th>Level</th><th>IERR value</th><th>Meaning</th><th>Processing result</th></tr>
<tr><td>Normal</td><td>0</td><td>Processing is terminated normally.</td><td>Results are guaranteed.</td></tr>
<tr><td>Warning</td><td>1000∼2999</td><td>Processing is terminated under certain conditions.</td><td>Results are conditionally guaranteed.</td></tr>
<tr><td rowspan="3">Fatal</td><td>3000∼3499</td><td>Processing is aborted since an argument violated its restrictions.</td><td>Results are not guaranteed.</td></tr>
<tr><td>3500∼3999</td><td>Obtained results did not satisfy a certain condition.</td><td>Obtained results are returned (the results are not guaranteed).</td></tr>
<tr><td>4000 or more</td><td>A fatal error was detected during processing. Usually, processing is aborted.</td><td>Results are not guaranteed.</td></tr>
</table>

(6) **Notes**

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

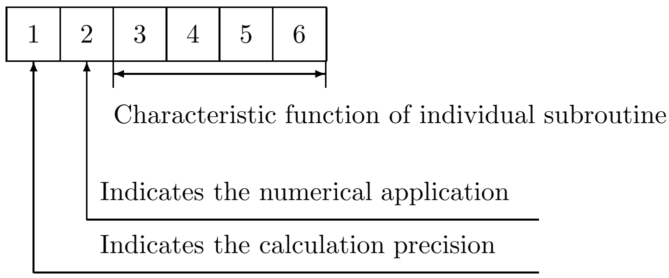
(7) **Example**

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 basic functions consists of ⟨six alphanumeric characters⟩.

Figure 1−1 Subroutine Name Components




**"1" in Figure** 1−1 **:** 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 lettererererere 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 |
| 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 |

| Letter | Application Field | Volume |
|---|---|---|
| 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 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 $\pi$ 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

# SPECIAL FUNCTIONS

## 2.1 INTRODUCTION

This chapter describes subroutines which obtain values of special functions. The following methods are available to calculate special functions.

- Taylor expansion and asymptotic expansion
- Approximation formulae
- Continued fraction
- Recursion relations

In the subroutines given here, the range of a variable is divided into intervals, and in each interval a special function is calculated with the method considered to be the best for the interval.

### 2.1.1 Notes

(1) The Bessel function of the 2nd kind $N_\nu(z)$ and the spherical Bessel function of the 2nd kind $n_\nu(z)$ are same as $Y_\nu(z)$ and $y_\nu(z)$ given here, respectively.

(2) The computation time of various Bessel functions of real number and integer orders becomes longer as the argument $z$ and the order $\nu$ increase. Therefore it is desirable to set $|\nu| < 1000.0$ and $|z| < 1000.0$

(3) To calculate values of the Bessel, modified Bessel, spherical Bessel or modified spherical Bessel functions of the 1st kind successively changing the order by one, first obtain the values for the highest two successive orders with a subroutine given here, and then calculate values for lower orders with recursion relations in the direction of decreasing order.

(4) $J_{-\nu}(x), I_{-\nu}(x), Y_{-\nu}(x), i_{-n}(x)$ and $Y_{-n}(x)$ are calculated with recursion relations. See the notes for each subroutine for the recursion relations.

(5) The Bessel and modified Bessel functions of half integer order can efficiently be calculated from the spherical Bessel function using the following relations.

$$J_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot j_n(x), \qquad Y_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot y_n(x)$$
$$I_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot i_n(x), \qquad K_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot k_n(x)$$

### 2.1.2 Algorithms Used

#### 2.1.2.1 Bessel Functions

(1) Bessel functions of the 1st kind (orders 0 and 1) $J_0(x)$ and $J_1(x)$

① $x < 0.0$:
From $J_0(x) = J_0(-x)$ and $J_1(x) = -J_1(-x)$, following methods are applied to $J_0(-x)$ and $J_1(-x)$.

② $0.0 \leq x < 4.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} J_0(x) &= \sum_{k=0}^{\infty} \frac{(-1)^k}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \\ J_1(x) &= \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1} \end{aligned}$$

③ $4.0 \leq x \leq 8.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} J_0(x) &= \sum_{k=0}^{\infty} \frac{J_0^{(k)}(6)}{k!}(x-6)^k \\ J_1(x) &= \sum_{k=0}^{\infty} \frac{J_1^{(k)}(6)}{k!}(x-6)^k \end{aligned}$$

The method of generating the best approximation is described in Section 2.1.2.22, "Coefficient Calculation Method".
($J_0^{(k)}(6)$ and $J_1^{(k)}(6)$ are $k$-th derivatives evaluated at $x = 6.0$.)

④ $x > 8.0$:
The functions are calculated from the following equations of asymptotic expansion:

$$J_0(x) \text{ or } J_1(x) = \frac{P\cos(\phi) - Q\sin(\phi)}{\sqrt{x}}.$$

Here,
for calculating $J_0(x)$

$$\begin{aligned} P &= \frac{\displaystyle\sum_{n=0}^{m} a_n^{(1)} \left(\frac{8}{x}\right)^{2n}}{\displaystyle\sum_{n=0}^{m'} b_n^{(1)} \left(\frac{8}{x}\right)^{2n}} \\ \phi &= x - \frac{\pi}{4} \end{aligned}$$

for calculating $J_1(x)$

$$\begin{aligned} Q &= \frac{\displaystyle\sum_{n=0}^{m} c_n^{(2)} \left(\frac{8}{x}\right)^{2n}}{\displaystyle\sum_{n=0}^{m'} d_n^{(2)} \left(\frac{8}{x}\right)^{2n}} \\ \phi &= x - \frac{3}{4}\pi \end{aligned}$$

See reference (2) for the coefficients $a_n^{(1)}, a_n^{(2)}, b_n^{(1)}, b_n^{(2)}, c_n^{(1)}, c_n^{(2)}, d_n^{(1)}$ and $d_n^{(2)}$.

(2) Bessel functions of the 2nd kind (orders 0 and 1) $Y_0(x)$ and $Y_1(x)$ ($x > 0.0$)

① $0.0 < x < 4.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} Y_0(x) &= \frac{2}{\pi}\left[J_0(x)\{\log\left(\frac{x}{2}\right)+\gamma\} - \sum_{k=1}^{\infty}\left\{\frac{(-1)^k}{(k!)^2}\left(\frac{x}{2}\right)^{2k}\sum_{m=1}^{k}\frac{1}{m}\right\}\right] \\ Y_1(x) &= \frac{2}{\pi}\left[J_1(x)\{\log(\frac{x}{2})+\gamma\} - \frac{1}{x} - \frac{1}{2}\sum_{k=0}^{\infty}\left\{\frac{(-1)^k}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k+1}\left(\sum_{m=1}^{k}\frac{1}{m}+\sum_{m=1}^{k+1}\frac{1}{m}\right)\right\}\right] \end{aligned}$$

The Euler's constant $\gamma$ is $0.5772\cdots$.

② $4.0 \le x \le 8.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} Y_0(x) &= \sum_{k=0}^{\infty}\frac{Y_0^{(k)}(6)}{k!}(x-6)^k \\ Y_1(x) &= \sum_{k=0}^{\infty}\frac{Y_1^{(k)}(6)}{k!}(x-6)^k \end{aligned}$$

The method of generating the best approximation is described in Section 2.1.2.22, "Coefficient Calculation Method".
The $k$-th derivative of $Y_0(6)$ and $Y_1(6)$ are obtained from recurrence relations derived from

$$\begin{aligned} Z_k'(x) &= Z_{k-1}(x) - \frac{k}{x}Z_k(x) \\ Z_1^{(k)}(x) &= -Z_0^{(k+1)}(x) \end{aligned}$$

where $Z_k(x)$ is the Bessel function to be calculated.

③ $x > 8.0$:
The functions are calculated from the following equations of asymptotic expansion:

$$Y_0(x) \text{ or } Y_1(x) = \frac{P\sin(\phi)+Q\cos(\phi)}{\sqrt{x}}.$$

Here $P$, $Q$ and $\phi$ are the same as for $J_0(x)$ and $J_1(x)$. The coefficients $a_n^{(1)}, a_n^{(2)}, b_n^{(1)}$ and $b_n^{(2)}$ are given in reference (2).

(3) Bessel function of the 1st kind (integer order) $J_n(x)$
$J_n(x)$ is expressed as in Table 2−1 using $J_{|n|}(|x|)$ depending on the sign of $x$ and $n$. $J_{|n|}(|x|)$ is calculated

Table 2−1 Expressions Equivalent to $J_n(x)$ for Different Signs of $x$ and $n$

| — | $n < 0$ | $n \ge 0$ |
|---|---|---|
| $x < 0.0$ | $J_{\|n\|}(\|x\|)$ | $(-1)^n J_{\|n\|}(\|x\|)$ |
| $x \ge 0.0$ | $(-1)^n J_{\|n\|}(\|x\|)$ | $J_{\|n\|}(\|x\|)$ |

as follows (for simple notation, $n$ and $x$ are used instead of $|n|$ and $|x|$, respectively.):

① $n = 0$ or $1$: $J_0(x)$ or $J_1(x)$ is used.

② $n \geq 2$:

(a) If

$$0.0 \leq x^2 < \left\{ \begin{array}{ll} \text{double precision} & :0.1n + 0.4 \\ \text{single precision} & :1.9n + 7.6 \end{array} \right\}$$

The function is calculated from the power series expansion of the following equation:

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{9} \frac{(-1)^k (\frac{x}{2})^{2k}}{k!(n+k)!}.$$

(b) If

$$x^2 \geq \left\{ \begin{array}{ll} \text{double precision} & :0.1n + 0.4 \\ \text{single precision} & :1.9n + 7.6 \end{array} \right\}$$

or

$$n \geq \left\{ \begin{array}{ll} \text{double precision} & :170 \\ \text{single precision} & :\ 34 \end{array} \right\}:$$

i. For $x \leq n$:

Setting $T_{n+k+1} = 0.0$, the parameters $T_i \quad (i = n, n+1, \cdots, n+k-1, n+k)$ in the continued fraction approximation are calculated using the recurrence relations below:

$$T_i = \frac{x^2}{2i - T_{i+1}} \; (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n - T_{n+1}}$$

($T_n$ has a value $\frac{J_n(x)}{J_{n-1}(x)}$.)

Here, the value $k$ is choose as in Table 2−2. It is defined that $B_n = T_n,\ B_{n-1} = 1.0$. Then

Table 2−2 Value of $k$

| single precision | double precision |
|---|---|
| $k = \lfloor \frac{(3.0+1.5\sqrt{n})x}{n} + 1.5 \rfloor$ | $k = \lfloor \frac{(3.0+2.8\sqrt{n})x}{n} + 5.2 \rfloor$ |

$B_1$ is calculated from the following recurrence relation:

$$B_{i-1} = \frac{2i}{x} B_i - B_{i+1} \; (i = n-1, n-2, \cdots, 2)$$

(Here $B_1$ has a value $\frac{J_1(x)}{J_{n-1}(x)}$)

Since the precision of computation becomes low when $J_0(x) \approx 0$ or $J_1(x) \approx 0$, $K$ is set as follows:

$$K = \lfloor 1.27324x + 3 \rfloor \pmod 4$$

If $K = 0$ or 3,

$$B = \frac{2B_1}{x} - B_2,\ J = J_0(x)$$

$K = 1$ or 2,

$$B = B_1,\ J = J_1(x)$$

Using the values of $T_n$, $B$ and $J$, $J_n(x)$ is calculated from

$$J_n(x) = \frac{T_n J}{B}$$

ii. For $x > n$:

$J_n(x)$ is calculated from the following recurrence relation starting with $J_1(x)$ and $J_0(x)$:

$$J_{k+1}(x) = \frac{2k}{x} J_k(x) - J_{k-1}(x) \; (k = 1, 2, \cdots, n-1)$$

(4) Bessel function of the 2nd kind (integer order) $Y_n(x)$ $(x > 0.0)$

① $n < 0$:
From $Y_n(x) = (-1)^n \cdot Y_{-n}(x)$, following methods are applied to $Y_{-n}(x)$.

② $n = 0$ or $1$:
$Y_0(x)$ or $Y_1(x)$ is used.

③ $n \geq 2$:
$Y_n(x)$ is calculated from the following recurrence relation starting with $Y_1(x)$ and $Y_0(x)$:

$$Y_{k+1}(x) = \frac{2k}{x} Y_k(x) - Y_{k-1}(x) \ (k = 1, 2, \cdots, n-1)$$

(5) Bessel function of the 1st kind (real number order) $J_\nu(x)$

① If $\nu$ is an integer, the subroutine for the Bessel function of the 1st kind (integer order) is used with $n = \nu$.

② If $n$ is a nonintegral real number ($x > 0.0$ and $n > 0.0$):

(a) If

$$0.0 < x^2 < \left\{ \begin{array}{ll} \text{double precision} & :0.1\nu + 0.4 \\ \text{single precision} & :1.9\nu + 7.6 \end{array} \right\}$$

and

$$\nu < \left\{ \begin{array}{ll} \text{double precision} & :170.0 \\ \text{single precision} & :\ 34.0 \end{array} \right\} :$$

The function is calculated from the power series expansion of

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{9} \frac{(-1)^k (\frac{x}{2})^{2k}}{k!\Gamma(\nu + k + 1)}.$$

(b) If

$$x^2 \geq \left\{ \begin{array}{ll} \text{double precision} & :0.1\nu + 0.4 \\ \text{single precision} & :1.9\nu + 7.6 \end{array} \right\}$$

i. For

$$x > \left\{ \begin{array}{l} \text{Double precision : } 30.0 \\ \text{Single precision } \ \text{: } 15.0 \end{array} \right\}$$

and

$$x \geq 0.55\nu^2 :$$

The function is calculated from the equation of asymptotic expansion of

$$J_\nu(x) = \sqrt{\frac{2}{\pi x}} (P \cos(\phi) - Q \sin(\phi))$$

where

$$\begin{aligned} P &= 1 + \sum_{k=1}^{m} (-1)^k \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2) \cdots (4\nu^2 - (4k-1)^2)}{(2k)!(8x)^{2k}} \\ Q &= \sum_{k=0}^{m'} (-1)^k \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2) \cdots (4\nu^2 - (4k+1)^2)}{(2k+1)!(8x)^{2k+1}} \\ \phi &= x - (\frac{\nu}{2} + \frac{1}{4})\pi \end{aligned}$$

($m$ and $m'$ are the numbers which give the last terms such that the term does not affect the result of calculation when included.)

ii. For $x$ other than the above:
The function is calculated from the (backward) recursion relation given below.

Assume that $\delta$ is the decimal part of $\nu$, $\nu$ is the integer part of $\nu$, $M$ is a sufficiently large number, and $a$ is the positive minimum number (namely the smallest positive constant in the floating point mode).

The recurrence relation to be used is

$$F_{\delta+k-1}(x) = \frac{2(\delta+k)}{x} F_{\delta+k}(x) - F_{\delta+k+1}(x) \ (k = M, M-1, \cdots, 1)$$

with $F_{\delta+M+1}(x) = 0$ and $F_{\delta+M}(x) = a$ as initial values. Then $J_n(x)$ is obtained from the following equation.

$$J_\nu(x) = \frac{F_{\delta+n}(x)(\frac{x}{2})^\delta}{\displaystyle\sum_{n=0}^{\lfloor \frac{M}{2} \rfloor} \frac{(\delta+2m)\Gamma(\delta+m)}{m!} F_{\delta+2m}(x)}$$

The value of $M$ is calculated from $x$ and $\nu$ using an approximation equation.

(6) Bessel function of the 2nd kind (real number order) $Y_\nu(x)$ $(x > 0.0)$

① If $\nu$ is an integer, the subroutine for the Bessel function of the 2nd kind (integer order) is used with $n = \nu$.

② If $\nu$ is a nonintegral real number ($\nu > 0.0$): $\nu$ is divided into the integer part $n$ and the decimal part $\delta$.

(a) If $0.0 < x \leq 4.0$:

The function is calculated with the method of Yoshida and Ninomiya where the power series expansions of $J_\nu(x)$ and $J_{-\nu}(x)$ are inserted in

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}$$

terms are grouped, and the parts which give figures at lower places are calculated with the best approximation. The procedures are shown below.

i. $0.0 < \nu \leq 0.5$:

The function is calculated from the following equation

$$Y_\nu(x) = \sum_{k=0}^{\infty} - \left(-\frac{x^2}{4}\right)^k \frac{\tilde{A}_k(\nu) + \tilde{B}_k(\nu)}{\sin(\nu\pi)}$$

Here $\tilde{A}_k(\nu)$ is calculated from the recursion relation

$$\tilde{A}_k(\nu) = \frac{\frac{1}{k!}\left\{\frac{1}{\Gamma(k-\nu)} + \frac{\cos(\nu\pi)}{\Gamma(k+\nu)}\right\} + \tilde{A}_{k-1}(\nu)}{(k+\nu)(k-\nu)}$$

with

$$\tilde{A}_0(\nu) = (\nu - \nu_0)\sum_{k=0}^{M} P_k^{(1)}\nu^k \ (\nu_0 = 0.221521\cdots)$$

as the initial value. $\tilde{B}_k(\nu)$ is calculated from

$$\tilde{B}_k(\nu) = \frac{1}{k!}\left\{\frac{\phi_1}{\Gamma(k+1-\nu)} + \frac{\phi_2\cos(\nu\pi)}{\Gamma(k+1+\nu)}\right\}$$

with

$$\phi_1 = \frac{(\frac{x}{2})^{-\nu} - 1}{\nu}, \ \phi_2 = \frac{1 - (\frac{x}{2})^\nu}{\nu}$$

for

$$\left(\frac{x}{2}\right)^\nu < 0.5 \text{ or } \left(\frac{x}{2}\right)^\nu > 2.0$$

$$\phi_1 = -f(-\nu\log(\frac{x}{2}))\log(\frac{x}{2}), \ \phi_2 = -f(\nu\log(\frac{x}{2}))\log(\frac{x}{2})$$

for

$$0.5 \leq (\frac{x}{2})^\nu \leq 2.0$$

where $f(t)$ is calculated from the best approximation equation of $\dfrac{e^t - 1}{t}$.

ii. $0.5 \leq \nu \leq 1.5$

A. $\delta \leq 0.5$:

It is set that $\delta = \delta + 1$ and $n = n - 1$, and the function is calculated in the range $0.5 < \delta 1.5$.

B. $0.5 < \delta 1.5$:

It is set that $\alpha = \delta - 1$. The function is calculated from

$$Y_\delta(x) = -\frac{\dfrac{2^{1+\alpha}}{x^{1+\alpha}\Gamma(1-\alpha)} + \displaystyle\sum_{k=0}^{\infty}\left\{\frac{x}{2}\left(-\frac{x^2}{4}\right)^k (\tilde{C}_k(\alpha) + \tilde{D}_k(\alpha))\right\}}{\sin(\alpha\pi)}$$

where $\tilde{C}_k(\alpha)$ and $\tilde{D}_k(\alpha)$ are calculated using best approximation equations as is done for $0.0 < \nu \leq 0.5$.

Furthermore the function is calculated from

$$Y_{\delta+1}(x) = \frac{-\dfrac{2^\alpha\{4(\alpha+1)+x^2\}}{x^{\alpha+2}\Gamma(1-\alpha)} + \displaystyle\sum_{k=0}^{\infty}\left\{\left(-\frac{x^2}{4}\right)^{k+1} (\tilde{E}_k(\alpha) + \tilde{F}_k(\alpha))\right\}}{\sin(\alpha\pi)}$$

where $\tilde{E}_k(\alpha)$ and $\tilde{F}_k(\alpha)$ are calculated using best approximation equations as is done for $0.0 < \nu \leq 0.5$.

iii. $\nu > 1.5$:

$Y_\nu(x)$ is calculated from the following recurrence relation using $Y_\delta(x)$ and $Y_{\delta+1}(x)$ obtained in ii.

$$Y_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}Y_{k+\delta}(x) - Y_{k+\delta-1}(x) \ (k = 1, 2, \cdots, n-1)$$

(b) If $4.0 < x \leq \left\{\begin{array}{ll} \text{Double precision} & : 30.0 \\ \text{Single precision} & : 15.0 \end{array}\right\}$

i. $0.0 < \nu < 2.0$:

If $\delta$ or $1 - \delta$ is smaller than $\sqrt{\text{unit for determining error}}/4$

The Bessel function of integer order ($n$ or $n + 1$) is taken as an approximation.

First $J_\delta(x)$ and $J_{\delta+1}(x)$ are obtained from (backward) recurrence relations. Similarly $J_t(x)$ and $J_{t+1}(x)$ are obtained by setting $t = 1 - \delta$. Then $J_{-\delta}(x)$ is calculated from

$$J_{-\delta}(x) = \frac{2(1-\delta)}{x}J_t(x) - J_{t+1}(x)$$

(See (5)$J_\nu(x)$.)

Finally $Y_\delta(x)$ and $Y_{\delta+1}(x)$ are calculated from:

$$\begin{aligned} Y_\delta(x) &= \frac{J_\delta(x)\cos(\delta\pi) - J_{-\delta}(x)}{\sin(\delta\pi)} \\ Y_{\delta+1}(x) &= \frac{J_{\delta+1}(x)\cos(\delta\pi) - \dfrac{2\delta J_{-\delta}(x)}{x} - J_t(x)}{\sin(\delta\pi)} \end{aligned}$$

ii. $\nu \geq 2.0$:

$Y_\nu(x)$ is calculated from the following recurrence relation using $Y_\delta(x)$ and $Y_{\delta+1}(x)$ obtained in i. :

$$Y_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}Y_{k+\delta}(x) - Y_{k+\delta-1}(x) \ (k = 1, 2, \cdots, n-1)$$

(c) If $\left\{\begin{array}{ll} \text{Double precision} & : 30.0 \\ \text{Single precision} & : 15.0 \end{array}\right\}$:

i. $x \leq 0.55\nu^2$:

The function is calculated from the equation of asymptotic expansion of the following relation:

$$Y_\nu(x) = \sqrt{\frac{2}{\pi x}}(P\sin(\phi) + Q\cos(\phi))$$

where $P$, $Q$ and $\phi$ are obtained as $J_\nu(x)$ in (5).

ii. $x > 0.55\nu^2$:

It is set that $m = \lfloor\sqrt{\frac{x}{0.55}}\rfloor - 1$. $Y_{m+\delta}(x)$ and $Y_{m+\delta+1}(x)$ are obtained from the equation of asymptotic expansion given above. The $Y_\nu(x)$ is calculated from the recurrence relation

$$\begin{aligned} Y_{m+\delta+k+1}(x) &= \frac{2(m+\delta+k)}{x} Y_{m+\delta+k}(x) - Y_{m+\delta+k-1}(x) \\ & \quad (k = 1, 2, \cdots, n-m-1) \end{aligned}$$

(7) Bessel function of the 1st kind with complex variable (integer order) $J_n(z)$

The function is calculated from the following equation.

$$J_n(z) = (-i)^n I_n(iz) \ (i = \sqrt{-1})$$

(8) Bessel function of the 2nd kind with complex variable (integer order) $Y_n(z)$ ($|z| > 0.0$)

① $n < 0$:

From $Y_n(z) = (-1)^n Y_{-n}(z)$, following methods are applied to $Y_{-n}(z)$.

② $n \geq 0$:

If the imaginary part of $z$ is negative, $Y_n(\bar{z}) = \overline{Y_n(z)}$ is used.

The function is calculated from

$$Y_n(z) = i^{n+1} I_n(-iz) - \frac{2}{\pi}(-i)^n K_n(-iz) \ (i = \sqrt{-1})$$

#### 2.1.2.2 Modified Bessel Functions

(1) Modified Bessel functions of the 1st kind (orders 0 and 1) $I_0(x)$ and $I_1(x)$

$x < 0.0$:

From $I_0(x) = I_0(-x)$, $I_1(x) = -I_1(-x)$, following methods are applied to $I_0(-x)$ and $I_1(-x)$.

- Single precision

① $0.0 \leq x \leq 3.75$:

The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} I_0(x) &= \sum_{k=0}^{\infty} \frac{1}{(k!)^2}\left(\frac{x}{2}\right)^{2k} \\ I_1(x) &= \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k+1} \end{aligned}$$

② $x > 3.75$:

The functions are calculated from the following approximations:

$$\begin{aligned} I_0(x) &= \frac{\sum_{n=0}^{8} a_n^{(1)} \left(\frac{3.75}{x}\right)^n}{e^{-x}\sqrt{x}} \\ I_1(x) &= \frac{\sum_{n=0}^{8} a_n^{(2)} \left(\frac{3.75}{x}\right)^n}{e^{-x}\sqrt{x}} \end{aligned}$$

The coefficients $a_n^{(1)}$ and $a_n^{(2)}$ are given in reference (1).

- Double precision

① $0.0 \leq x \leq 8.0$:
The functions are calculated from the following best approximation equations obtained from the following equations:

$$\begin{aligned} I_0(x) &= \sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \\ I_1(x) &= \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1} \end{aligned}$$

② $8.0 < x < 24.0$:
The functions are calculated from the following approximations:

$$\begin{aligned} I_0(x) &= \frac{\sum_{n=0}^{28} a_n^{(3)} x^{-n}}{e^{-x}\sqrt{x}} \\ I_1(x) &= \frac{\sum_{n=0}^{28} a_n^{(4)} x^{-n}}{e^{-x}\sqrt{x}} \end{aligned}$$

③ $x \geq 24.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} I_0(x) &= \frac{e^x}{\sqrt{2\pi x}} \sum_{k=0}^{\infty} \frac{1^2 \cdot 1^2 \cdot 3^2 \cdots (2k-1)^2}{k!(8x)^k} \\ I_1(x) &= \frac{e^x}{\sqrt{2\pi x}} \left\{ 1 + \sum_{k=1}^{\infty} \frac{(-3) \cdot 5 \cdots ((2k-1)^2 - 4)}{k!(8x)^k} \right\} \end{aligned}$$

The coefficients $a_n^{(3)}$ and $a_n^{(4)}$ are obtained with the telescoping calculation method given in reference (7). The method of generating the best approximation is described in Section 2.1.2.22.

(2) Modified Bessel functions of the 2nd kind (order 0 and 1) $K_0(x)$ and $K_1(x)$ $(x > 0.0)$

- Single precision

① $0.0 < x \leq 2.0$:
The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} K_0(x) &= -\gamma + \sum_{k=1}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \{(\sum_{m=1}^{k} \frac{1}{m}) - \gamma\} - \log\left(\frac{x}{2}\right) \{\sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k}\} \\ K_1(x) &= \frac{1 + \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+2} (2\gamma - \sum_{m=1}^{k} \frac{1}{m} - \sum_{m=1}^{k+1} \frac{1}{m})}{x} \\ &\quad + \log\left(\frac{x}{2}\right) \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1} \end{aligned}$$

(Here, $\sum_{m=1}^{0} \frac{1}{m} = 0$)

② $x > 2.0$:

The functions are calculated from the following approximations:

$$\begin{aligned} K_0(x) &= \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{6} a_n^{(1)} \left(\frac{2}{x}\right)^n \\ K_1(x) &= \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{6} a_n^{(2)} \left(\frac{2}{x}\right)^n \end{aligned}$$

The coefficient $a_n^{(1)}$ and $a_n^{(2)}$ are given in reference (1).

- Double precision

① $0.0 < x \leq 2.0$:

The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} K_0(x) &= -\gamma + \sum_{k=1}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \{(\sum_{m=1}^{k} \frac{1}{m}) - \gamma\} - \log\left(\frac{x}{2}\right) \{\sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k}\} \\ K_1(x) &= \frac{1 + \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+2} (2\gamma - \sum_{m=1}^{k} \frac{1}{m} - \sum_{m=1}^{k+1} \frac{1}{m})}{x} \\ &\quad + \log\left(\frac{x}{2}\right) \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1} \end{aligned}$$

(Here, $\sum_{m=1}^{0} \frac{1}{m} = 0$)

The Euler's constant $\gamma$ is $0.5772\cdots$.

② $2.0 < x \leq 5.0$:

The functions are calculated from the best approximation equations:

$$\begin{aligned} K_0(x) &= \sum_{k=0}^{\infty} \frac{K_0^{(k)}(3.5)}{k!} (x - 3.5)^k \\ K_1(x) &= \sum_{k=0}^{\infty} \frac{K_1^{(k)}(3.5)}{k!} (x - 3.5)^k \end{aligned}$$

($K_0^{(k)}(3.5)$ and $K_1^{(k)}(3.5)$ are the differential values of $k$-th grades for $x = 3.5$.)

The coefficients $K_0^{(k)}(3.5)$ and $K_0^{(k)}(3.5)$ are calculated from the following method.

$$t_0 = 1, t_1 = 0, t_2 = 0, u_0 = 0, u_1 = 1$$
$$v_0 = 0, v_1 = 0, w_0 = 0, w_1 = 0, w_2 = -1$$
$$i = 3, x = 3.5$$

$$\text{Repetition}\left[\begin{array}{l} t'_j = v_j - u_j \ (j = 0, \cdots, i-2) \\ t'_{i-1} = 0, t'_i = 0 \\ u'_0 = w_0 - t_0 \\ u'_j = w_j - t_j - u_{j-1} \ (j = 1, \cdots, i-1) \\ t_j = t'_j \ (j = 0, \cdots, i), u_j = u'_j \ (j = 0, \cdots, i-1) \\ K_0^{(i)}(x) = \sum_{j=0}^{i-2} t_j x^{-j} K_0(x) + \sum_{j=0}^{i-1} u_j x^{-j} K_1(x) \\ v_{j+1} = -j t_j (j = 1, i-2) \\ w_{j+1} = -j u_j (j = 1, i-1) \\ i = i+1 \end{array}\right.$$

$$K_1^{(i)}(x) = -K_0^{(i+1)}(x), K''_0(x) = K_0(x) + \frac{K_1(x)}{x}, K'_0(x) = -K_1(x)$$

③ $5.0 < x < 24.0$:

$$\begin{aligned} K_0(x) &= \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{18} a_n^{(3)} x^{-n} \\ K_1(x) &= \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{18} a_n^{(4)} x^{-n} \end{aligned}$$

The coefficients $a_n^{(3)}$ and $a_n^{(4)}$ are obtained with the telescoping calculation method given in reference (7).

④ $x \geq 24.0$:

The functions are calculated from the best approximation equations obtained from the following equations:

$$\begin{aligned} K_0(x) &= \sqrt{\frac{\pi}{2x}} e^{-x} \sum_{k=0}^{\infty} (-1)^k \frac{1^2 \cdot 1^2 \cdot 3^2 \cdots (2k-1)^2}{k!(8x)^k} \\ K_1(x) &= \sqrt{\frac{\pi}{2x}} e^{-x} \{1 + \sum_{k=1}^{\infty} \frac{3 \cdot (-5) \cdots (4 - (2k-1)^2)}{k!(8x)^k}\} \end{aligned}$$

The method of generating the best approximation is described in Section 2.1.2.22.

(3) Modified Bessel function of the 1st kind (integer order) $I_n(x)$

$I_n(x)$ is expressed as in Table 2−3 using $I_{|n|}(|x|)$ depending on the signs of $x$ and $n$. $I_{|n|}(|x|)$ is calculated

Table 2−3 Expression Equivalent to $I_n(x)$ for Different Signs of $x$ and $n$

| — | $n < 0$ | $n \geq 0$ |
|---|---|---|
| $x < 0.0$ | $(-1)^n I_{\|n\|}(\|x\|)$ | $(-1)^n I_{\|n\|}(\|x\|)$ |
| $x \geq 0.0$ | $I_{\|n\|}(\|x\|)$ | $I_{\|n\|}(\|x\|)$ |

as follows (for simple notation, $n$ and $x$ are used instead of $|n|$ and $|x|$, respectively.):

① $n = 0$ or $1$ :

$I_0(x)$ or $I_1(x)$ is used.

② $n \geq 2$ :

(a) $0.0 \le x^2 < \left\{\begin{array}{ll} \text{double precision} & :0.1n+0.4 \\ \text{single precision} & :1.9n+7.6 \end{array}\right\}$ and $n < \left\{\begin{array}{ll} \text{double precision} & :170 \\ \text{single precision} & :\ 34 \end{array}\right\}$:

The function is calculated from the power expansion of

$$I_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{9} \frac{(\frac{x}{2})^{2k}}{k!(n+k)!}$$

(b) $0.0 \le x^2 < \left\{\begin{array}{ll} \text{double precision} & :0.1n+0.4 \\ \text{single precision} & :1.9n+7.6 \end{array}\right\}$ and $n < \left\{\begin{array}{ll} \text{double precision} & :170 \\ \text{single precision} & :\ 34 \end{array}\right\}$:

i. $x \le n$:

Setting $T_{n+k+1} = 0.0$, the parameters $T_i \quad (i = n, n+1, \cdots, n+k-1, n+k)$ in the continued fraction approximation are calculated using the recurrence relations below:

$$T_i = \frac{x^2}{2i + T_{i+1}} \ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n + T_{n+1}}$$

($T_n$ has a value of $\frac{I_n(x)}{I_{n-1}(x)}$).

Here, the value $k$ is choose as in Table 2−4. Setting $B_n = T_n$ and $B_{n-1} = 1.0$, $B_i \quad (i =$

Table 2−4 Values of $k$

| Single precision | Double precision |
|---|---|
| $k = \lfloor \frac{(4.6+0.5\sqrt{n})x}{n} + 2.0 \rfloor$ | $k = \lfloor \frac{(6.0+1.2\sqrt{n})x}{n} + 6.0 \rfloor$ |

$1, 2, \cdots, n-2)$ are calculated using the following recurrence relation.

$$B_{i-1} = \frac{2i}{x} B_i + B_{i+1} \ (i = 2, \cdots, n-2, n-1)$$

($B_1$ has a value $\frac{I_1(x)}{I_{n-1}(x)}$.)

Using the obtained values of $T_n$, $B_1$ and $I_1(x)$, $I_n(x)$ is calculated from

$$I_n(x) = \frac{T_n I_1(x)}{B_1}$$

ii. $x > n$:

The function is calculated from the following recurrence relation starting with $G_{M+1}(x) = 0.0$ and $G_M(x) = a$, where $a$ is the positive minimum number (namely the smallest positive constant in the floating point mode), and $M$ is a number sufficiently larger than $n$:

$$G_{k-1}(x) = \frac{2k}{x} G_k(x) + G_{k+1}(x) \ (k = M, M-1, \cdots, 1)$$

Then

$$I_n(x) = \frac{G_n(x) e^x}{\sum_{m=0}^{M} \varepsilon_m G_m(x)} \ (\varepsilon_0 = 1, \varepsilon_m = 2 \ (m \ge 1))$$

(4) Modified Bessel function of the 2nd kind (integer order) $K_n(x)$ $(x > 0.0)$

① $n < 0$:

From $K_n(x) = K_{-n}(x)$, following methods are applied to $K_{-n}(x)$.

② $n = 0$ or 1:

$K_0(x)$ and $K_1(x)$ are used.

③ $n \ge 2$:

$K_n(x)$ is calculated from the following recurrence relation with $K_1(x)$ and $K_0(x)$ as initial values:

$$K_{k+1}(x) = \frac{2k}{x} K_k(x) + K_{k-1}(x) \ (k = 1, 2, \cdots, n-1)$$

(5) Modified Bessel function of the 1st kind (real number order) $I_\nu(x)$

① If $\nu$ is an integer, the subroutine for the modified Bessel function of the 1st kind (integer order) is used with $n = \nu$.

② If $\nu$ is a nonintegral real number($x > 0.0$ and $\nu > 0.0$):

(a) If $0.0 \le x^2 < \left\{\begin{array}{ll} \text{double precision} & 0.1\nu + 0.4 \\ \text{single precision} & 1.9\nu + 7.6 \end{array}\right\}$ and $\nu < \left\{\begin{array}{ll} \text{double precision} & 170.0 \\ \text{single precision} & 34.0 \end{array}\right\}$:

The function is calculated from the following equation of power expansion:

$$I_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{9} \frac{(\frac{x}{2})^{2k}}{k!\Gamma(\nu + k + 1)}$$

(b) If $x^2 \ge \left\{\begin{array}{ll} \text{double precision} & 0.1\nu + 0.4 \\ \text{single precision} & 1.9\nu + 7.6 \end{array}\right\}$ or $\nu \ge \left\{\begin{array}{ll} \text{double precision} & 170.0 \\ \text{single precision} & 34.0 \end{array}\right\}$:

i. If $x > \left\{\begin{array}{ll} \text{double precision} & 30.0 \\ \text{single precision} & 15.0 \end{array}\right\}$ and $x \ge 0.55\nu^2$:

The function is calculated from the following equation of asymptotic expansion:

$$I_\nu(x) = \frac{e^x}{\sqrt{2\pi x}} \sum_{k=0}^{m} (-1)^k \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2)\cdots(4\nu^2 - (2k-1)^2)}{k!(8x)^k}$$

where $m$ is the number for the last term such that the term does not affect the result of calculation up to the previous term when included.

ii. If $x$ is a real number other than above:

The function is calculated from the following (backward) recurrence relation with $G_{\delta+M+1}(x) = 0.0$ and $G_{\delta+M}(x) = a$ as initial value, where $\delta$ is the decimal part of $\nu$, $n$ is the integer part of $\nu$, $M$ a sufficiently large number and $a$ is the positive minimum number (namely the smallest positive constant in the floating point mode).

$$G_{\delta+k-1}(x) = \frac{2(\delta + k)}{x} G_{\delta+k}(x) + G_{\delta+k+1}(x) \;\; (k = M, M-1, \cdots, 1)$$

Then

$$I_\nu(x) = \frac{1}{2}(\frac{x}{2})^\delta \frac{\Gamma(2\delta + 1)}{\Gamma(\delta + 1)} e^x \frac{G_{n+\delta}(x)}{\displaystyle\sum_{k=0}^{M} \frac{(\delta + k)\Gamma(2\delta + k)}{k!} G_{\delta+k}(x)}$$

The value of $M$ is obtained from the value of $x$ and $\nu$.

(6) Modified Bessel function of the 2nd kind (real number order) $K_\nu(x)$ ($x > 0.0$)

① $\nu < 0.0$:

From $K_\nu(x) = K_{-\nu}(x)$, following methods are applied to $K_{-\nu}(x)$.

② If $\nu$ is an integer, the subroutine for the Bessel function of the 2nd kind (integer order) is used with $n = \nu$.

③ $\nu$ is a nonintegral real number ($\nu > 0.0$):

When $x$ is small, the functional is calculated with the method where the power series expansions of $I_\nu(x)$ and $I_{-\nu}(x)$ are inserted in

$$K_\nu(x) = \frac{\pi}{2} \frac{I_{-\nu}(x) - I_\nu(x)}{\sin(\nu\pi)}$$

Terms are grouped, and the parts which give figures at lower places are calculated with the best approximation.

When $x$ is large, a method which is an extension of the $\tau-$method for $K_n(x)$ extended for calculating

$K_\nu(x)$. In this way $K_\nu(x)$ is calculated for $0.0 \le \nu \le 2.5$. For $\nu > 2.5$, it is calculated from the recurrence relation

$$K_{\nu+1}(x) = \frac{2\nu}{x} K_\nu(x) + K_{\nu-1}(x)$$

The following shows the procedures of the calculation.

(a) $0.0 \le \nu \le 0.5$:

i. $x < -0.75\nu^2 + 0.0235\nu + 0.778$

The function is calculated from

$$K_\nu(x) = \frac{\sum_{k=0}^{\infty}\{\left(\frac{x}{2}\right)^{2k}(\tilde{A}_k(\nu) + \tilde{B}_k(\nu))\}}{\frac{\pi}{2}\sin(\nu\pi)}$$

Here $\tilde{A}_k(\nu)$ $(k \ge 2)$ is calculated from the recurrence relation.

$$\tilde{A}_k(\nu) = \frac{\frac{\nu}{k!}\left\{\frac{1}{\Gamma(k-\nu)} + \frac{1}{\Gamma(k+\nu)}\right\} + \tilde{A}_{k-1}(\nu)}{(k+\nu)(k-\nu)}$$

with $\tilde{A}_0(\nu)$ and $\tilde{A}_1(\nu)$ as the initial value, where $\tilde{A}_0(\nu) = \nu\sum_{k=0}^{M} p_k^{(1)}\nu^{2k}$, $\tilde{A}_1(\nu) = \nu\sum_{k=0}^{M} q_k^{(1)}\nu^{2k}$.

$\tilde{B}_k(\nu)$ is calculated from

$$\tilde{B}_k(\nu) = \frac{1}{k!}\left(\frac{\phi_1}{\Gamma(k+1-\nu)} + \frac{\phi_2}{\Gamma(k+1+\nu)}\right)$$

with

$$\phi_1 = (\frac{x}{2})^{-\nu} - 1 \qquad \phi_2 = 1 - (\frac{x}{2})^{\nu} \qquad \text{for } (\frac{x}{2})^{\nu} < 0.5 \text{ or } (\frac{x}{2})^{\nu} > 2.0$$

$$\phi_1 = f(-\nu\log(\frac{x}{2})) \quad \phi_2 = -f(\nu\log(\frac{x}{2})) \quad \text{for } 0.5 \le (\frac{x}{2})^{\nu} \le 2.0$$

where $f(t)$ is calculated from the best approximation equation of $e^t - 1$.

ii. $x \ge -0.75\nu^2 + 0.0235\nu + 0.778$:

The function is calculated from the $\tau -$ method for

$$K_\nu(x) = \sqrt{\frac{1}{x}}e^{-x}\frac{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i\{\sum_{j=0}^{i} b_{ij}(\nu^2)^j\}}{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i e_i\Psi_i}$$

Here $\Psi_i$ is obtained from

$$\Psi_0 = 1,\ \Psi i = \prod_{l=0}^{i-1}\{\nu^2 - (m-l+\frac{1}{2})^2\}\ (i \ge 1)$$

and, $e_i = \sqrt{\frac{2}{\pi}}\frac{(m-i)!}{(m+1)!}\frac{p^*_{m,m-i}}{2^i}$, and $p^*_{m,m-i}$ is the coefficients of shifted Legendre polynomial.

(b) $0.5 < \nu \le 2.5$:

$\nu$ is divided into the integer part $n$ and the decimal part $\delta$. It is set that $\alpha = \delta - 1$. When $\delta \le 0.5$, it is set that $\delta = \delta + 1$ and $n = n - 1$, $\alpha = \alpha + 1$.

i. $K_\delta$ is calculated in the following method A. or B. :

A. $x < -0.675\delta^2 + 1.973\delta - 0.12$:

$K_\delta(x)$ is calculated from

$$K_\delta(x) = -\frac{\frac{2^{1+\alpha}}{x^{1+\alpha}\Gamma(1-\alpha)} + \sum_{k=0}^{\infty}\{(\frac{x}{2})^{2k+1}(\tilde{C}_k(\alpha) + \tilde{D}_k(\alpha))\}}{\frac{\pi}{2}\sin(\alpha\pi)}$$

where $\tilde{C}_k(\alpha)$ and $\tilde{D}_k(\alpha)$ are obtained using best approximate equations as is done for $\nu \le 0.5$.

B. $x \ge -0.675\delta^2 + 1.973\delta - 0.12$:
$K_\delta(x)$ is calculated with the $\tau-$ method as is done for large $x$ with $\nu \le 0.5$.

ii. $K_{\delta+1}$ is calculated in the following method A. or B. :

A. $x < -0.277(\delta+1)^2 + 1.817(\delta+1) - 0.94$:
$K_{\delta+1}(x)$ is calculated from

$$K_{\delta+1}(x) = \frac{\dfrac{2^\alpha \alpha(4\alpha+5)}{x^\alpha \Gamma(1-\alpha)} + \displaystyle\sum_{k=0}^{\infty}\{(\frac{x}{2})^{2k+2}(\tilde{E}_k(\alpha) + \tilde{F}_k(\alpha))\}}{\frac{\pi}{2}\sin(\alpha\pi)}$$

where $\tilde{E}_k(\alpha)$ and $\tilde{F}_k(\alpha)$ are obtained using best approximation equations as is done for $v \le 0.5$.

B. $x \ge -0.277(\delta+1)^2 + 1.817(\delta+1) - 0.94$ :
$K_{\delta+1}(x)$ is calculated with the $\tau-$ method method as is done for large $x$ with $\nu \le 0.5$.

(c) $\nu > 2.5$:
$K_\nu(x)$ is calculated from the following recurrence relation using $K_\delta(x)$ and $K_{\delta+1}(x)$ obtained above:

$$K_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}K_{k+\delta}(x) + K_{k+\delta-1}(x) \ (k = 1, 2, \cdots, n-1)$$

(7) Modified Bessel function of the 1st kind with complex variable $I_n(z)$

① $n < 0$:
From $I_n(z) = I_{-n}(z)$, following methods are applied to $I_{-n}(z)$.

② $n \ge 0$:

(a) $\mathrm{Re}(z) < 0$:
From $I_n(z) = (-1)^n \cdot I_n(-z)$, following methods are applied to $I_n(-z)$.

(b) $\Re(z) > 0$ and:

i. $|z|^2 < \left\{ \begin{array}{ll} \text{Double precision} & : 0.1n + 0.4 \\ \text{Single precision} & : 1.9n + 7.6 \end{array} \right\}$

The function is calculated from the power expansion

$$I_n(z) = \left(\frac{z}{2}\right)^n \sum_{k=0}^{9} \frac{(\frac{z}{2})^{2k}}{k!(n+k)!}$$

ii. $|z|^2 \ge \left\{ \begin{array}{ll} \text{Double precision} & : 0.1n + 0.4 \\ \text{Single precision} & : 1.9n + 7.6 \end{array} \right\}$

A. If $\Re(z) > 100.0$ or $|\Im(z)| > 100.0$ and $n \le 15$:
The function is calculated from the following equation of asymptotic expansion:

$$\begin{aligned} I_n(z) &= \frac{e^z}{\sqrt{2\pi z}} \sum_{k=0}^{m} (-1)^k \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2n-1)^2)}{k!(8z)^k} \\ &+ \frac{i(-1)^n e^{-z}}{\sqrt{2\pi z}} \sum_{k=0}^{m'} \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2n-1)^2)}{k!(8z)^k} \end{aligned}$$

($m$ and $m'$ are the numbers which give the last terms such that the term does not affect the result of calculation up to the previous term when included.)

B. Otherwise:
The function is calculated from the following (backward) recurrence relation using $G_{M+1}(z) =$

0.0 $G_M(z) = a$ as initial values, where $M$ is a sufficient large number, and $a$ is the positive minimum number (namely smallest positive constant in the floating point mode).

$$G_{k-1}(z) = \frac{2k}{z}G_k(z) + G_{k+1}(z) \ (k = M, M-1, \cdots, 1)$$

Then

$$I_n(z) = \frac{G_n(z)e^z}{\sum_{m=0}^{M} \varepsilon_m G_m(z)} \ (\varepsilon_0 = 1, \varepsilon_m = 2 \ (m \geq 1))$$

The value of $M$ is obtained from $z$ and $n$ using an approximate equation.

(8) Modified Bessel function of the 2nd kind with complex variable (integer order) $K_n(z)$ ($|z| > 0.0$)

① $\Re(z) < 0$: From

$$K_n(z) = (-1)^n K_n(-z) + \pi i I_n(-z) \cdot (\text{sign of } \Im(-z))$$

following methods are applied to $K_n(-z)$.

② $\Re(z) > 0$:

(a) $n < 0$:
From $K_n(z) = K_{-n}(z)$, following methods are applied to $K_{-n}(z)$.

(b) $0 \leq n < 2$ and:

i. $|\Im(z)| < \left\{ \begin{array}{ll} \text{Double precision :} & -4.0\Re(z) + 8.0 \\ \text{Single precision} & : -2.25\Re(z) + 4.5 \end{array} \right\}$

$K_0(z)$ and $K_1(z)$ are calculated from following equations:

$$\begin{aligned} K_0(z) &= -\{\gamma + \log(\frac{z}{2})\}I_0(z) + \sum_{k=1}^{n} \frac{(\frac{z}{2})^{2k}}{(k!)^2}(\sum_{m=1}^{k} \frac{1}{m}) \\ K_1(z) &= \frac{\frac{1}{z} - I_1(z)K_0(z)}{I_0(z)} \end{aligned}$$

Where, $\gamma$ is Euler's constant.

ii. $|\Im(z)| \geq \left\{ \begin{array}{ll} \text{Double precision :} & -4.0\Re(z) + 8.0 \\ \text{Single precision} & : -2.25\Re(z) + 4.5 \end{array} \right\}$

$K_0(z)$ and $K_1(z)$ are calculated from the following equation with the $\tau -$ method:

$$K_n(z) = \sqrt{\frac{1}{z}} e^{-z} \frac{\sum_{k=0}^{m} c_k z^{k-m}}{\sum_{k=0}^{m} d_k z^{k-m}}$$

(c) $n \geq 2$: The function is calculated from the following recurrence relation using $K_0(z)$ and $K_1$ as initial values:

$$K_{k+1}(z) = \frac{2k}{z}K_k(z) + K_{k-1}(z) \ (k = 1, 2, \cdots, n-1)$$

#### 2.1.2.3 Spherical Bessel Functions

(1) Spherical Bessel function of the 1st kind (integer order) $j_n(x)$ ($x \geq 0.0$)

① $n < 0$
The function is calculated from

$$j_n(x) = (-1)^n y_{-n-1}(x).$$

② $n \geq 0$

(a) $x^2 < 0.1n + 0.47$ and $n < 35$:

The function is calculated from the power series expansion of

$$j_n(x) = x^n \sum_{k=0}^{9} \frac{(-\frac{x^2}{2})^k}{k!(2n+2k+1)!!}$$

(b) $x^2 \geq 0.1n + 0.47$ or $n \geq 35$:

i. $n = 0$ or $1$:

For $x = 0$, $j_0(0) = 1$ and $j_1(0) = 0$. Otherwise the functions are calculated from

$$j_0(x) = \frac{\sin(x)}{x},\ j_1(x) = \frac{\sin(x) - x\cos(x)}{x^2}$$

ii. $n \geq 2$:

A. $x \leq n$:

Setting $T_{n+k+1} = 0.0$, the parameters $T_i\ \ (i = n, n+1, \cdots, n+k-1, n+k)$ in the continued fraction approximation are calculated using the recurrence relations below:

$$T_i = \frac{x^2}{2i - T_{i+1} + 1}\ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n - T_{n+1} + 1}$$

($T_n$ has a value $\dfrac{j_n(x)}{j_{n-1}(x)}$.)

Here, the value $k$ is choose as in Table 2−5. Setting $B_n = T_n$ and $B_{n-1} = 1.0$, $B_i\ \ (i =$

Table 2−5 Value of $k$

| Single precision | Double precision |
|---|---|
| $k = \lfloor \frac{(3.0+1.5\sqrt{n})x}{n} + 1.5 \rfloor$ | $k = \lfloor \frac{(3.0+2.8\sqrt{n})x}{n} + 5.2 \rfloor$ |

$1, 2, \cdots, n-2)$ are calculated using the following recurrence relation.

$$B_{i-1} = \frac{2i+1}{x} B_i - B_{i+1}\ (i = 2, \cdots, n-2, n-1)$$

Since the precision of computation becomes low when $j_0(x) \simeq 0$ or $j_1(x) \simeq 0$, $K$ is set as follows:

$$K = \lfloor 1.27324x \rfloor \pmod 4$$

When $K = 1$ or $2$,

$$B = \frac{3B_1}{x} - B_2,\ j = j_0(x)$$

When $K = 0$ or $3$,

$$B = B_1,\ j = j_1(x)$$

Using the obtained values of $T_n$, $B$ and $j$, $j_n(x)$ is calculated from

$$j_n(x) = \frac{T_n j}{B}$$

B. $x > n$:

$j_n(x)$ is calculated from the following recurrence relation using $j_1(x)$ and $j_0(x)$ as initial values:

$$j_{k+1}(x) = \frac{2k+1}{x} j_k(x) - j_{k-1}(x)\ (k = 1, 2, \cdots, n-1)$$

(2) Spherical Bessel function of the 2nd kind (integer order) $y_n(x)$ $(x > 0.0)$

① $n < 0$:
The function is calculated from $y_n(x) = (-1)^{n+1} j_{-n-1}(x)$.

② $n \geq 0$:

(a) $x \leq 0.41$:
The function is calculated from the power expansion of

$$y_n(x) = -\frac{(2n-1)!!}{x^{n+1}} \left\{ 1 + \sum_{k=1}^{9} \frac{(-\frac{x^2}{2})^k}{k!(1-2n)(3-2n)\cdots(2k-1-2n)} \right\}$$

(b) $x > 0.41$:

i. $n = 0$ or $1$:
The function is calculated from

$$y_0(x) = -\frac{\cos(x)}{x}, \ y_1(x) = -\frac{\cos(x) + x\sin(x)}{x^2}$$

ii. $n \geq 2$:
$y_n(x)$ is calculated from the following recurrence relation using $y_1(x)$ and $y_0(x)$ as initial values.

$$y_{k+1}(x) = \frac{2k+1}{x} y_k(x) - y_{k-1}(x) \ (k = 1, 2, \cdots n-1)$$

(3) Modified spherical Bessel function of the 1st kind (integer order) $i_n(x)$ $(n \geq 0, x \geq 0.0)$

① $x^2 < 0.1n + 0.47$ and $n \leq 30$:
The function is calculated from the power expansion of

$$i_n(x) = x^n \sum_{k=0}^{9} \frac{(\frac{x^2}{2})^k}{k!(2n+2k+1)!!}$$

② $x^2 \geq 0.1n + 0.47$ or $n > 30$

(a) $n = 0$ or $1$:
The function is calculated from

$$i_0(x) = \frac{\sinh(x)}{x}$$

$$i_1(x) = \frac{x\cosh(x) - \sinh(x)}{x^2} = \frac{e^x(x-1) + e^{-x}(x+1)}{2x^2}$$

(b) $n \geq 2$:

i. $x \leq n$:
Setting $T_{n+k+1} = 0.0$, the parameters $T_i \ \ (i = n, n+1, \cdots, n+k-1, n+k)$ in the continued fraction approximation are calculated using the recurrence relations below:

$$T_i = \frac{x^2}{2i + T_{i+1} + 1} \ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n + T_{n+1} + 1}$$

($T_n$ has a value $\frac{i_n(x)}{i_{n-1}(x)}$.)
Here, the value $k$ is choose as in Table 2–6. Setting $B_n = T_n$ and $B_{n-1} = 1.0$, $B_k \ \ (k =$

Table 2–6 Value of $k$

| Single precision | Double precision |
|---|---|
| $k = \lfloor \frac{(4.6+0.5\sqrt{n})x}{n} + 2.0 \rfloor$ | $k = \lfloor \frac{(6.0+1.2\sqrt{n})x}{n} + 6.0 \rfloor$ |

$1, 2, \cdots, n-2)$ are calculated using the following recurrence relation.

$$B_{k-1} = \frac{2k+1}{x} B_k + B_{k+1} \ (k = 2, \cdots, n-2, n-1)$$

(Here $B_1$ has a value $\frac{i_1(x)}{i_{n-1}(x)}$.)
Using the obtained values of $T_n$, $B_1$ and $i_1(x)$, $i_n(x)$ is calculated from

$$i_n(x) = \frac{T_n i_1(x)}{B_1}$$

ii. $x < n$:
The function is calculated from the following recurrence relation using $G_{M+1}(x) = 0.0$ and $G_M(x) = a$ as initial values, where $a$ is the positive minimum number (namely the smallest positive constant in the floating point mode), and $M$ is a number sufficiently larger than $n$.

$$G_{k-1}(x) = \frac{2k+1}{x} G_k(x) + G_{k+1}(x) \ (k = M, M-1, \cdots, 1)$$

Then

$$i_n(x) = \frac{G_n(x)e^x}{2\sum_{m=0}^{M}(m+0.5)G_m(x)}$$

(4) Modified spherical Bessel function of the 2nd kind (integer order) $k_n(x)$ $(x > 0.0)$

① $n < 0$:
From $k_n(x) = k_{-n}(x)$, following methods are applied to $k_{-n}(x)$.

② $n \geq 0$:

(a) $n = 0$ or 1:
The function is calculated from

$$k_0(x) = \frac{\pi}{2}\frac{e^{-x}}{x}, \ k_1(x) = \frac{\pi}{2}\frac{e^{-x}}{x}(1 + \frac{1}{x})$$

(b) $n \geq 2$:
$k_n(x)$ is calculated from the following recurrence relation using $k_1(x)$ and $k_0(x)$ as initial values:

$$k_{i+1}(x) = \frac{2i+1}{x} k_i(x) + k_{i-1}(x) \ (i = 1, 2, \cdots, n-1)$$

#### 2.1.2.4 Functions Related To Bessel Functions

(1) Hankel functions of the 1st and 2nd kinds (integer order) $H_n^{(1)}(z), H_n^{(2)}(z)$
The function is calculated from

$$\begin{aligned} H_n^{(1)}(z) &= J_n(z) + iY_n(z) \\ H_n^{(2)}(z) &= J_n(z) - iY_n(z) \end{aligned}$$

(2) Kelvin functions $\mathrm{ber}_n(x), \mathrm{bei}_n(x)$

① $n < 0$ or $x < 0.0$
First $\mathrm{ber}_{|n|}(|x|)$ or $\mathrm{bei}_{|n|}(|x|)$ is calculated as described in the case ② $n \geq 0$ and $x \geq 0.0$ then $\mathrm{ber}_n(x)$ or $\mathrm{bei}_n(x)$ is calculated from the following expressions

(a) $n < 0$ or $x < 0$:

$$\mathrm{ber}_n(x) = (-1)^n \,\mathrm{ber}_{|n|}(|x|), \ \mathrm{bei}_n(x) = (-1)^n \,\mathrm{bei}_{|n|}(|x|)$$

(b) $n < 0$ and $x < 0$:

$$\mathrm{ber}_n(x) = \mathrm{ber}_{|n|}(|x|), \ \mathrm{bei}_n(x) = \mathrm{bei}_{|n|}(|x|)$$

② $n \geq 0$ and $x \geq 0.0$:

(a) $x \le \begin{Bmatrix} \mathrm{ber_n(x)} : 5.65 + 0.25\mathrm{n} \\ \mathrm{bei_n(x)} : 6.92 + 0.25\mathrm{n} \end{Bmatrix}$:

The function is calculated from the following approximation.

$$\begin{aligned} \mathrm{ber}_n(x) &\simeq \sum_{k=0}^{m} \frac{\cos\{\frac{1}{4}(3n+2k)\pi\}}{k!(n+k)!} \left(\frac{x}{2}\right)^{n+2k} \\ \mathrm{bei}_n(x) &\simeq \sum_{k=0}^{m} \frac{\sin\{\frac{1}{4}(3n+2k)\pi\}}{k!(n+k)!} \left(\frac{x}{2}\right)^{n+2k} \end{aligned}$$

However, for $n = 0$, both $\mathrm{ber}(x)$ and $\mathrm{bei}(x)$ are calculated from the expressions shown above by generating best approximations. The method of generating the both approximation is described in Section 2.1.2.22.

(b) $x \ge \begin{Bmatrix} \text{Double precision : } 20.0 \\ \text{Single precision : } 10.0 \end{Bmatrix}$ and $x \ge 0.5n^2$:

The function is calculated from the asymptotic expansion expressions

$$\begin{aligned} \mathrm{ber}_n(x) &\simeq \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}}\{P\cos(\Psi) + Q\sin(\Psi)\} \\ \mathrm{bei}_n(x) &\simeq \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}}\{P\sin(\Psi) - Q\cos(\Psi)\} \end{aligned}$$

where:

$$\begin{aligned} P &= 1 + \sum_{k=1}^{m}(-1)^k \cos\left(\frac{k\pi}{4}\right) \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k} \\ Q &= \sum_{k=1}^{m}(-1)^k \sin\left(\frac{k\pi}{4}\right) \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k} \\ \Psi &= \frac{x}{\sqrt{2}} + (\frac{n}{2} - \frac{1}{8})\pi \end{aligned}$$

However, for $n = 0$, both $P$ and $Q$ are calculated by generating best approximation is described in Section 2.1.2.22.

($m$ is a value such that the final term will not affect values calculated before it.)

(c) Cases other than those described in (a) and (b):

The function is calculated from the Bessel function of the 1st kind with complex variable (integer order) $J_n(z)$ where:

$$\begin{aligned} \mathrm{ber}_n(x) &= \Re\left(J_n(-\frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}})\right) \\ \mathrm{bei}_n(x) &= \Im\left(J_n(-\frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}})\right) \end{aligned}$$

(3) Kelvin function $\mathrm{ker}_n(x), \mathrm{kei}_n(x)$ $(x > 0.0)$

① $n < 0$:

First, $\mathrm{ker}_{|n|}(x)$ or $\mathrm{kei}_{|n|}(x)$ is described in case ③ $n > 0$. Then $\mathrm{ker}_n(x)$ or $\mathrm{kei}_n(x)$ is calculated from the following expressions.

$$\mathrm{ker}_n(x) = (-1)^n \,\mathrm{ker}_{|n|}(x), \quad \mathrm{kei}_n(x) = (-1)^n \,\mathrm{kei}_{|n|}(x)$$

② $n = 0$ and $x \le \begin{Bmatrix} \mathrm{ker(x)} : 5.77 \\ \mathrm{kei(x)} : 6.56 \end{Bmatrix}$:

The function is calculated by generating best approximations from

$$\mathrm{ker}(x) = -\log(\frac{x}{2})\,\mathrm{ber}(x) + \frac{\pi}{4}\,\mathrm{bei}(x) + \sum_{n=0}^{\infty}\left\{\frac{(-1)^n}{((2n)!)^2}\left(\sum_{s=1}^{2n}\frac{1}{s} - \gamma\right)(\frac{x}{2})^{4n}\right\}$$

$$\mathrm{kei}(x) = -\log(\frac{x}{2})\,\mathrm{bei}(x) - \frac{\pi}{4}\,\mathrm{ber}(x) + \sum_{n=0}^{\infty}\left\{\frac{(-1)^n}{((2n+1)!)^2}\left(\sum_{s=1}^{2n}\frac{1}{s} - \gamma\right)(\frac{x}{2})^{4n+2}\right\}$$

($\gamma$: Euler's constant: $0.57721566\cdots$).

The method of generating the best approximation is described in Section 2.1.2.22.

③ $n > 0$:

(a) $x \geq \left\{\begin{array}{l} \text{Double precision : } 20.0 \\ \text{Single precision : } 10.0 \end{array}\right\}$ and $x \geq 0.5n^2$:

The function is calculated from the asymptotic expansions

$$\mathrm{ker}_n(x) \simeq \sqrt{\frac{\pi}{2x}}e^{-x\sqrt{2}}\{P\cos(\Psi) - Q\sin(\Psi)\}$$

$$\mathrm{kei}_n(x) \simeq \sqrt{\frac{\pi}{2x}}e^{-x\sqrt{2}}\{-P\sin(\Psi) - Q\cos(\Psi)\}$$

where:

$$P = 1 + \sum_{k=1}^{m}\cos\left(\frac{k\pi}{4}\right)\frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k}$$

$$Q = \sum_{k=1}^{m}\sin\left(\frac{k\pi}{4}\right)\frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k}$$

$$\Psi = \frac{x}{\sqrt{2}} + \left(\frac{n}{2} - \frac{1}{8}\right)\pi$$

However, for $n = 0$, both $P$ and $Q$ are calculated by generating best approximation is described in Section 2.1.2.22.

($m$ is a value such that the final term will not affect values calculated before it.)

(b) Case other than those described in (a):

The function is calculated from the modified Bessel function of the 2nd kind with complex variable (integer order) $K_n(z)$ where $A$ is the real part of $K_n(\frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}})$, $B$ is the imaginary part. If the remainder of $\frac{n}{4}$

i. 0:
$\mathrm{ker}_n(x) = A$, $\mathrm{kei}_n(x) = B$

ii. 1:
$\mathrm{ker}_n(x) = B$, $\mathrm{kei}_n(x) = -A$

iii. 2:
$\mathrm{ker}_n(x) = -A$, $\mathrm{kei}_n(x) = -B$

iv. 3:
$\mathrm{ker}_n(x) = -B$, $\mathrm{kei}_n(x) = A$

(4) Struve function $\mathbf{H}_0(x), \mathbf{H}_1(x), \mathbf{H}_0(x) - Y_0(x), \mathbf{H}_1(x) - Y_1(x)$

① $x < 0.0$:

From $\mathbf{H}_0(x) = -\mathbf{H}_0(-x)$ and $\mathbf{H}_1(x) = \mathbf{H}_1(-x)$, following methods applied to $\mathbf{H}_0(-x)$ or $\mathbf{H}_1(-x)$. However, the difference with the Bessel function $\mathbf{H}_0(x) - Y_0(x)$ or $\mathbf{H}_1(x) - Y_1(x)$ gives fatal error.

② $0.0 \leq x \leq 8.0$;

The function is calculated by generating best approximations from the following expressions

$$\mathbf{H}_0(x) = \frac{2}{\pi}\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k+1}}{((2k+1)!!)^2}$$

$$\mathbf{H}_1(x) = \frac{2}{\pi}\sum_{k=1}^{\infty}\frac{(-1)^k(2k+1)x^{2k}}{((2k+1)!!)^2}$$

The method of generating the best approximation is described in Section 2.1.2.22.
$\mathbf{H}_0(x) - Y_0(x)$ or $\mathbf{H}_1(x) - Y_1(x)$ is calculated by subtracting the Bessel function of the 2nd kind $Y_0(x)$ or $Y_1(x)$ from the value of $\mathbf{H}_0(x)$ or $\mathbf{H}_1(x)$ obtained in this way.

③ $x > 8.0$:
The function is calculated by using the following approximation expressions

$$\mathbf{H}_0(x) - Y_0(x) = \frac{1}{x}\sum_{k=0}^{n} a_k^{(1)}\left(\frac{1}{x}\right)^{2n} \qquad n = \left\{\begin{array}{l}\text{Double precision : 28}\\ \text{Single precision : 6}\end{array}\right\}$$

$$\mathbf{H}_1(x) - Y_1(x) = \sum_{k=0}^{n} a_k^{(2)}\left(\frac{1}{x}\right)^{2n} \qquad n = \left\{\begin{array}{l}\text{Double precision : 25}\\ \text{Single precision : 5}\end{array}\right\}$$

The coefficients $a_n^{(1)}$ and $a_n^{(2)}$ are obtained by a telescoping calculation of the approximation expressions as described in reference (7).
$\mathbf{H}_0(x)$ or $\mathbf{H}_1(x)$ is obtained by adding $Y_0(x)$ or $Y_1(x)$ to the value of the difference with the Bessel function obtained in this way.

(5) Airy functions and their derived functions $\mathrm{Ai}(x), \mathrm{Bi}(x), \mathrm{Ai}'(x), \mathrm{Bi}'(x)$
Assume $\zeta = \frac{2}{3}|x|^{\frac{3}{2}}$.

① $x < \left\{\begin{array}{l}-3.8315472 \text{ (for } \mathrm{Ai}(x), \mathrm{Bi}(x))\\ -5.2414828 \text{ (for } \mathrm{Ai}'(x), \mathrm{Bi}'(x))\end{array}\right\}$:

$$\begin{aligned}
\mathrm{Ai}(x) &= \frac{\sqrt{-x}}{3}\{\frac{4}{3\zeta}J_{\frac{2}{3}}(\zeta) - J_{\frac{5}{3}}(\zeta) + J_{\frac{1}{3}}(\zeta)\}\\
\mathrm{Bi}(x) &= \sqrt{\frac{-x}{3}}\{\frac{4}{3\zeta}J_{\frac{2}{3}}(\zeta) - J_{\frac{5}{3}}(\zeta) - J_{\frac{1}{3}}(\zeta)\}\\
\mathrm{Ai}'(x) &= \frac{x}{3}\{\frac{2}{3\zeta}J_{\frac{1}{3}}(\zeta) - J_{\frac{4}{3}}(\zeta) - J_{\frac{2}{3}}(\zeta)\}\\
\mathrm{Bi}'(x) &= \frac{-x}{\sqrt{3}}\{\frac{2}{3\zeta}J_{\frac{1}{3}}(\zeta) - J_{\frac{4}{3}}(\zeta) + J_{\frac{2}{3}}(\zeta)\}
\end{aligned}$$

② $\left\{\begin{array}{l}-3.8315472 \text{ (for } \mathrm{Ai}(x), \mathrm{Bi}(x))\\ -5.2414828 \text{ (for } \mathrm{Ai}'(x), \mathrm{Bi}'(x))\end{array}\right\} \le x < \left\{\begin{array}{l}3.8315472 \text{ (for } \mathrm{Ai}(x), \mathrm{Bi}(x))\\ 5.2414828 \text{ (for } \mathrm{Ai}'(x), \mathrm{Bi}'(x))\end{array}\right\}$:

$$\begin{aligned}
C_1 &= \frac{3^{-\frac{2}{3}}}{\Gamma(\frac{2}{3})}\\
C_2 &= \frac{3^{-\frac{1}{3}}}{\Gamma(\frac{1}{3})}\\
f(x) &= 1 + \frac{1}{3!}x^3 + \frac{1\cdot 4}{6!}x^6 + \frac{1\cdot 4\cdot 7}{9!}x^9 + \cdots\\
g(x) &= x + \frac{2}{4!}x^4 + \frac{2\cdot 5}{7!}x^7 + \frac{2\cdot 5\cdot 8}{10!}x^{10} + \cdots
\end{aligned}$$

The domain is divided into the intervals $x < 0$ and $x \ge 0$. The function is calculated by generating best approximations from the following expressions

$$\begin{aligned}
\mathrm{Ai}(x) &= C_1 f(x) - C_2 g(x)\\
\mathrm{Bi}(x) &= \sqrt{3}(C_1 f(x) + C_2 g(x))
\end{aligned}$$

$\mathrm{Ai}'(x)$ is calculated by creating an expression by differentiating the approximation of $\mathrm{Ai}(x)$.
$\mathrm{Bi}'(x)$ is calculated by creating an expression by differentiating the approximation of $\mathrm{Bi}(x)$.
The method of generating the best approximation is described in Section 2.1.2.22.

③ $x \geq \left\{ \begin{array}{l} 3.8315472 \text{ (for } \mathrm{Ai}(x), \mathrm{Bi}(x)) \\ 5.2414828 \text{ (for } \mathrm{Ai}'(x), \mathrm{Bi}'(x)) \end{array} \right\}$:

$$\begin{aligned}
\mathrm{Ai}(x) &= e^{-\zeta} x^{-\frac{1}{4}} \sum_{k=0}^{m1} a_k^{(1)} \left(\frac{1}{\zeta}\right)^k \\
\mathrm{Bi}(x) &= e^{\zeta} x^{-\frac{1}{4}} \sum_{k=0}^{m2} a_k^{(2)} \left(\frac{1}{\zeta}\right)^k + \sqrt{\frac{x}{3}} I_{\frac{5}{3}}(\zeta) \\
\mathrm{Ai}'(x) &= e^{-\zeta} x^{\frac{1}{4}} \sum_{k=0}^{m3} a_k^{(3)} \left(\frac{1}{\zeta}\right)^k \\
\mathrm{Bi}'(x) &= e^{\zeta} x^{\frac{1}{4}} \sum_{k=0}^{m4} a_k^{(4)} \left(\frac{1}{\zeta}\right)^k + \frac{x}{\sqrt{3}} I_{\frac{4}{3}}(\zeta)
\end{aligned}$$

The coefficients $a_k^{(1)}$ through $a_k^{(4)}$ are new coefficients obtained as described in reference (7).

#### 2.1.2.5 Gamma Functions

(1) Gamma function $\Gamma(x)$ with real variable and logarithmic Gamma function $\log_e(\Gamma(x))$ with real variable

① $x < 0.0$:
From

$$\begin{aligned}
\Gamma(x) &= \frac{\pi}{-x \sin(\pi x) \Gamma(-x)} \\
\log_e(\Gamma(x)) &= \log_e \pi - \log_e((-x)\sin(\pi x)) - \log_e(\Gamma(-x))
\end{aligned}$$

following methods are applied to $\Gamma(-x)$ and $\log_e(\Gamma(-x))$. where GAMMA and ALGAMA are the FORTRAN intrinsic functions for the Gamma function and logarithmic Gamma function.

② $x \geq 0.0$:
Use the FORTRAN intrinsic functions GAMMA and ALGAMA.

(2) Gamma function $\Gamma(z)$ with complex variable and logarithmic Gamma function $\log_e(\Gamma(z))$ with complex variable

① $\Im(z) = 0.0$:
The function is calculated using the logarithmic Gamma function with real variable.

② $\Re(z) = 0.0$ and $|\Im(z)| > 12.0$:
The function is calculated from the following equation of asymptotic expansion:

$$\begin{aligned}
\Re(\log_e(\Gamma(z))) &\sim \frac{1}{2}(\log_e(2\pi) - \pi\Im(z) - \log_e(\Im(z))) \\
\Im(\log_e(\Gamma(z))) &\sim \Im(z)\log_e(\Im(z)) - \Im(z) - \frac{1}{4}\pi - \sum_{n=1}^{\infty} \frac{(-1)^{n-1} B_{2n}}{(2n-1)(2n)(\Im(z))^{2n-1}}
\end{aligned}$$

where, $B_n$ is Bernoulli numbers.

③ $\Im(z) < 0.0$:
From

$$\log_e(\Gamma(z)) = \log_e \pi - \log_e(-z)\sin(\pi z) - \log_e(\Gamma(-z))$$

following methods are applied to $\log_e(\Gamma(-z))$.

④ $\Im(z) > 0.0$:
The function is calculated with the following procedure.

(a) $|\Re(z)| < 11.0$: From

$$\log_e(\Gamma(z)) = \log_e(\Gamma(n+z)) - \log_e((n-1+z)(n-2+z)\cdots(z))$$

following methods are applied to $\log_e(\Gamma(n+z))$. Where, $n = \lfloor 12.0 - \Re(z) \rfloor$

(b) $|\Re(z)| \geq 11.0$: The function is calculated from the following equation of asymptotic expansion:

$$\log_e(\Gamma(z)) \quad \sim \quad (z - \frac{1}{2})\log_e z - z + \frac{1}{2}\log_e(2\pi) + \sum_{n=1}^{\infty} \frac{B_{2n}}{(2n-1)(2n)z^{2n-1}}$$

⑤ The Gamma function with complex variable is returned as $\exp(\log_e(\Gamma(z)))$.

(3) Incomplete Gamma function of the 1st kind $\gamma(\nu, x)$, $(\nu \geq 0.0, x \geq 0.0)$

① $x = 0.0$:
$\gamma(\nu, x) = 0.0$

② $x \leq \nu - 0.5$ or $x \leq 3.5$:
For $\nu \leq \dfrac{e^{3.5}}{(\text{Maximum value})}$

$$\gamma(\nu, x) = \frac{1}{\nu}$$

For cases other than above,

$$\gamma(\nu, x) = \frac{e^{\nu \log(x) - x} P}{\nu}$$

where

$$P = 1 + \sum_{k=1}^{m} \frac{x^k}{(\nu+1)(\nu+2)\cdots(\nu+k)}$$

($m$ is a value such that the final term will not affect values calculated before it.)

③ For values of $x$ other than above:
The function is calculated from

$$\gamma(\nu, x) = \Gamma(\nu) - \Gamma(\nu, x)$$

where $\Gamma(\nu, x)$ is the incomplete Gamma function of the 2nd kind.

(4) Incomplete Gamma function of the 2nd kind $\Gamma(\nu, x)$ $(\nu \geq 0.0, x \geq 0.0)$

① $\nu \leq \dfrac{1.0}{(\text{Maximum value})}$:
$-\,\mathrm{Ei}(-x)$ is calculated by using the subroutine for the exponential integral.

② $x = 0.0$:
Let $\Gamma(\nu, x) = \Gamma(\nu)$.

③ $\nu$ is an integer $x > 0.015\nu^2$:

$$\Gamma(\nu, x) = e^{(\nu-1)\log(x) - x}\{1 + \sum_{k=1}^{m} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\}$$

($m$ is a value such that the final term will not affect values calculated before it or such that $m \leq \nu + 2$.)

④ $x \leq \nu - 0.5$:
Let $\Gamma(\nu, x) = \Gamma(\nu) - \gamma(\nu, x)$.

⑤ $x \geq \left\{ \begin{array}{ll} \text{Double precision} & : 49.0 \\ \text{Single precision} & : 23.0 \end{array} \right\}$:
Perform the same procedure as described in ③ for $\nu$ an integer and $x \geq 0.015\nu^2$.

⑥ $\nu < 1.0$:

(a) $x < 0.36\nu + 0.85$:

The function is calculated from

$$\Gamma(\nu, x) = (\nu - 1)\frac{\sum_{k=0}^{M} p_k^{(2)}\nu^k}{\sum_{k=0}^{N} q_k^{(2)}\nu^k} - \frac{e^{\nu\log(x)} - 1}{\nu} - x^{\nu}\sum_{k=1}^{M'}\frac{(-1)^k x^k}{k!(k+\nu)}$$

$$(M': \begin{Bmatrix} \text{Double precision : 21} \\ \text{Single precision : 12} \end{Bmatrix})$$

(b) $x \le 1.5\nu + 2.1$:

The function is calculated from

$$\Gamma(\nu, x) = \Gamma(1+\nu)e^{-x}\sum_{k=0}^{N'}(x^k\tilde{A}_k(\nu) + x^k\frac{\phi(\nu, x)}{\Gamma(k+1+\nu)}$$

$$(N': \begin{Bmatrix} \text{Double precision : 23} \\ \text{Single precision : 14} \end{Bmatrix})$$

where

$$\begin{aligned} \tilde{A}_0(\nu) &= (1+\nu)(\tilde{A}_1(\nu) - 1) \\ \tilde{A}_1(\nu) &\simeq \sum_{k=0}^{M} p_k^{(1)}\nu^k \end{aligned}$$

and

$$\begin{aligned} \tilde{A}_k(\nu) &= \frac{1}{k+\nu}\{\tilde{A}_{k-1}(\nu) + \frac{1}{k!}\} \\ \phi(\nu, x) &= -\frac{e^{\nu\log(x)} - 1}{\nu} \end{aligned}$$

(c) For cases other than the above:

The function is calculated from

$$\Gamma(\nu, x) = e^{\nu\log(x) - x}\cdot\frac{1|}{|x} + \underset{n=1}{\overset{\infty}{\Phi}}\frac{n-\nu|}{|1} + \frac{n|}{|x}$$

where the following values are assumed for the continued fractions:

$$\begin{Bmatrix} \text{Double precision :} & \lfloor\frac{120}{x} + 5\rfloor \\ \text{Single precision :} & \lfloor\frac{25}{x-0.25} + 2\rfloor \end{Bmatrix}$$

⑦ For cases other than those described in ① through ⑥:

(a) $x \le 5.6$:

Let $\alpha = \nu - \lfloor\nu\rfloor$ and $\mu = \alpha + 1.0$, the function is calculated from

$$\Gamma(\mu, x) = \Gamma(\mu)e^{-x}[1 + \alpha\sum_{k=0}^{M''}(x^{k+1}\tilde{C}_k(\alpha) + x^{k+1}\frac{\phi(\alpha, x)}{\Gamma(k+1+\mu)})]$$

$$(M'': \begin{Bmatrix} \text{Double precision : 20} \\ \text{Single precision : 12} \end{Bmatrix})$$

where

$$\tilde{C}_0(\alpha) = \tilde{A}_1(\alpha)$$

and

$$\begin{aligned} \tilde{C}_k(\alpha) &= \frac{1}{k+\nu}\{\tilde{C}_{k-1}(\alpha) + \frac{1}{(k+1)!}\} \\ \phi(\alpha, x) &= -\frac{e^{\alpha\log(x)} - 1}{\alpha} \end{aligned}$$

- $\nu > 2.0$:
  For $m = \nu - \mu$
  The function is calculated from
  $$\begin{aligned}\Gamma(\nu, x) &= e^{(\nu-1)\log(x)-x}\left\{1 + \sum_{k=1}^{m-1} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\right\} \\ &\quad +(\nu-1)(\nu-2)\cdots(\nu-k)\Gamma(\mu, x)\end{aligned}$$
- $\nu \leq 2.0$:
  $$\Gamma(\nu, x) = \Gamma(\mu, x)$$

(b) For cases other than (a):
The function is calculated from
$$\begin{aligned}\Gamma(\nu, x) &= e^{(\nu-1)\log(x)-x}\{1 + \sum_{k=1}^{m-1} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\} \\ &\quad +(\nu-1)(\nu-2)\cdots(\nu-m)e^{(\nu-m)\log(x)-x} \\ &\quad \cdot \left[\frac{1|}{|x} + \mathop{\Phi}_{n-1}^{\infty}\left(\frac{n-(\nu-m)|}{|1} + \frac{n|}{|x}\right)\right]\end{aligned}$$
($m$ is the integer part of $\nu$.)

#### 2.1.2.6 Functions Related To The Gamma Function

(1) Digamma function $\Psi(x)$ (if $x < 0.0$ then $x \neq$integer)

① $x < -2.0$:
The function is calculated from
$$\Psi(x) = \log(1-x) - \frac{1}{2(1-x)} + \frac{\sum_{k=0}^{m} a_k^{(2)}(1-x)^{-2k}}{\sum_{k=0}^{m} b_k^{(2)}(1-x)^{-2k}} - \pi\cot(\pi x)$$

② $-2.0 < x < 0.5$:
The function is calculated from
$$\Psi(x) = (1-c-x)\frac{\sum_{k=0}^{m} a_k^{(1)}(1-x)^{k}}{\sum_{k=0}^{m} b_k^{(1)}(1-x)^{k}} - \pi\cot(\pi x)$$

③ $0.5 \leq x \leq 3.0$:
The function is calculated from
$$\Psi(x) = (x-c)\frac{\sum_{k=0}^{m} a_k^{(1)}x^{k}}{\sum_{k=0}^{m} b_k^{(1)}x^{k}}$$

④ $x > 3.0$:

The function is calculated from

$$\Psi(x) = \log(x) - \frac{1}{2x} + \frac{\sum_{k=0}^{m} a_k^{(2)} x^{-2k}}{\sum_{k=0}^{m} b_k^{(2)} x^{-2k}}$$

The value of $c$ is assumed to be $c = 1.46163214496836234126$ and the coefficients $a_k^{(1)}$, $b_k^{(1)}$, $a_k^{(2)}$ and $b_k^{(2)}$ are described in reference (4).

(2) Beta function $B(p,q)$ $(p > 0.0, q > 0.0)$

The beta function is defined with the following formula.

$$B(p,q) = \int_0^1 x^{p-1}(1-x)^{q-1}dx = B(q,p) \ (p,q > 0)$$

The function is calculated as follows depending on the values of $p$ and $q$:

① $p < 12.0$ and $q < 12.0$:

The function is calculated using the Gamma function according to the following expression.

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$$

② $p < 12.0$ and $q \geq 12.0$:

The function is calculated using the Gamma function and the Stirling formula as follows:

$$B(p,q) = \Gamma(p)e^{(q-\frac{1}{2})\log(q)+\Phi(q)-(p+q-\frac{1}{2})\log(p+q)+p-\Phi(p+q)}$$

where $\Phi(x)$ is obtained by generating the best approximation of

$$\Phi(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}B_n}{(2n)(2n-1)x^{2n-1}}$$

($B_n$: Bernoulli numbers).

The method of generating the best approximation is described in Section 2.1.2.22.

③ $p \geq 12.0$ and $q < 12.0$:

The function is calculated from $B(p,q) = B(q,p)$ using the same formula as described for $p < 12.0$ and $q \geq 12.0$.

④ $p \geq 12.0$ and $q \geq 12.0$:

The function is calculated from the Stirling formula as follows:

$$B(p,q) = e^{(p-\frac{1}{2})\log(p)+(q-\frac{1}{2})\log(q)-(p+q-\frac{1}{2})\log(p+q)+\log(\sqrt{2\pi})+\Phi(p)+\Phi(q)-\Phi(p+q)}$$

#### 2.1.2.7 Elliptic Functions And Elliptic Integrals

(1) Complete elliptic integrals of the 1st and 2nd kinds $K(m), E(m)$ $(0.0 \leq m \leq 1.0)$

① $0.0 \leq m \leq 0.25$:

The functions are calculated by generating the best approximations of

$$\begin{aligned} K(m) &= \frac{\pi}{2}\sum_{k=0}^{\infty}\left(\frac{(2k-1)!!}{(2k)!!}\right)^2 m^k \\ E(m) &= \frac{\pi}{2}\left\{1-\sum_{k=1}^{\infty}\left(\frac{(2k-1)!!}{(2k)!!}\right)^2 \frac{m^k}{2k-1}\right\} \end{aligned}$$

The method of generating the best approximation is described in Section 2.1.2.22.

② $0.25 < m < 1.0$:

The functions are calculated from

$$K(m) = \sum_{k=0}^{l} \left\{ a_k^{(1)} (1-m)^k \right\} - \log(1-m) \sum_{k=0}^{l} \left\{ b_k^{(1)} (1-m)^k \right\}$$

$$E(m) = \sum_{k=0}^{l} \left\{ a_k^{(2)} (1-m)^k \right\} - \log(1-m) \sum_{k=0}^{l} \left\{ b_k^{(2)} (1-m)^k \right\}$$

③ $m = 1.0$:

The following values are assumed.

$$K(m) = +\infty$$

$$E(m) = 1.0$$

The coefficients $a_k^{(1)}$, $b_k^{(1)}$, $a_k^{(2)}$ and $b_k^{(2)}$ are described in reference (2).

(2) Incomplete elliptic integrals of the 1st and 2nd kinds $F(x, m), E(x, m)$ $(0.0 \le x \le 1.0, 0.0 \le m \le 1.0)$

① $x = 0.0$:

The function values are as follows:

$$F(0.0, m) = E(0.0, m) = 0.0$$

② $x = 1.0$:

The function values are calculated from the complete elliptic integrals as follows:

$$F(1.0, m) = K(m)$$

$$E(1.0, m) = E(m)$$

③ For cases other than those in ① and ②.

(a) $m = 0.0$:

The following values are returned.

$$F(x, 0.0) = \sin^{-1}(x)$$

$$E(x, 0.0) = \sin^{-1}(x)$$

(b) $0.0 < m < 1.0$:

The following values are obtained from initial values

$$a_{k+1} = \frac{a_k + b_k}{2}$$

$$b_{k+1} = \sqrt{a_k b_k}$$

$$c_{k+1} = \frac{a_k - b_k}{2} \quad (k = 0, 1, \cdots)$$

with $a_0 = 1.0$, $b_0 = \sqrt{1-m}$ and $c_0 = \sqrt{m}$ as initial values.

The convergence criterion is assumed to be

$$c_N < a_k \cdot (\text{units for determining error}).$$

The values $\phi_1, \cdots, \phi_N$ are obtained by using Newton method to solve for $\phi_{k+1} (> \phi_k)$

$$\tan(\phi_{k+1} - \phi_k) - \frac{b_k}{a_k} \tan(\phi_k) = 0 \ (k = 0, \cdots, N-1)$$

with $\phi_0 = \sin^{-1}(x)$. Then the function is calculated from

$$F(x, m) = \frac{\phi_N}{2^N a_N}$$

$$E(x, m) = \left( 1 - \frac{1}{2} \sum_{k=0}^{N} 2^k c_k^2 \right) F(x, m) + \sum_{k=1}^{N} c_k \sin(\phi_k).$$

Another method is using $F(x,m) = (1+k_1)F(x_1,m_1)$ with

$$k_1 = \frac{1-\sqrt{1-m}}{1+\sqrt{1-m}},\ m_1 = k_1^2,\ m = k^2,\ x_1 = \frac{2}{1+\sqrt{1-mx^2}}\frac{x}{1+k_1}.$$

Setting

$$K_{n+1} = \frac{K_n^2}{(1+\sqrt{1-K_n^2})^2},\ X_{n+1} = \frac{2}{1+\sqrt{1-K_n^2X_n^2}}\frac{X_n}{1+K_{n+1}},$$

we have

$$F(x,m) = (1+K_1)(1+K_2)\cdots(1+K_{n+1})F(X_{n+1},\ K_{n+1}^2),$$

and $F(X_{n+1},\ K_{n+1}^2)$ is almost equal to $\sin^{-1} X_{n+1}$ since small $n$ is enough to make $K_{n+1}$ very small.

(c) $m = 1.0$:

The following function values are assumed.

$$F(x,m) = \text{artanh(x)}$$
$$E(x,m) = x$$

(3) Incomplete Modified Elliptic Integral and Incomplete Elliptic Integral of the Weierstrass Type

As defined already, $F(r,m)$ denotes the first kind incomplete elliptic integral, $E(r,m)$ denotes the second kind incomplete elliptic integral, which are defined as

$$F(r,m) = \int_0^r \frac{dx}{\sqrt{(1-x^2)(1-mx^2)}}, E(r,m) = \int_0^r \sqrt{\frac{1-mx^2}{1-x^2}}dx.$$

(A) Incomplete Modified Elliptic Integral

For real numbers $m \geq 0, a, b$ and $p \geq 0$, incomplete modified elliptic integral

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}}$$

is obtained with the following method.

When $m = 0,\ 1$ namely incomplete modified elliptic integral can be degenerated and can be expressed with elementary functions, this is obtained by using logarithmic function or opposite tangential function.

(a) If $m$ is not near 1 and $m > 1$, then the problem is reduced to the case $m < 1$ (b) by applying integration by substitution.

(b) If $m$ is not near 1 and $m < 1$, then by setting $p = r/\sqrt{1-r^2}$, it holds that

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}} = \frac{b-ma}{1-m}F(r,1-m) + \frac{a-b}{1-m}E(r,1-m)$$

(c) If $m$ is near 1, then by setting $p = \tan\alpha, 0 \leq \alpha < \pi/2$,

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}} = \int_0^\alpha \frac{a+(b-a)\sin^2 u}{\sqrt{1-(1-m)\sin^2 u}}du$$

is applied since the value $I_n(\alpha) = \int_0^\alpha \sin^{2n} u du$ is obtained from the recurrence formula

$$(2n+2)I_{n+1}(\alpha) = (2n+1)I_n(\alpha) - \sin^{2n+1}\alpha\cos\alpha,\ n \geq 0$$

and since the integrated function is expanded by using the formula

$$\frac{1}{\sqrt{1-v}} = 1 + \sum_{n=1}^{\infty}\frac{I_n(\pi/2)}{\pi/2}v^n,\ |v| < 1.$$

(B) Incomplete Elliptic Integral of the Weierstrass Type

When incomplete elliptic integral can be degenerated and can be expressed with elementary functions, this can be obtained using logarithmic function or opposite tangential function. In the other cases, set

$$v_1 = \sqrt{\frac{(z-x)p}{1+zp}}, v_2 = \sqrt{\frac{z-x}{z}}, m = \frac{z-y}{z-x}$$

for real parameters which satisfy $z > y > x > 0$ to obtain

$$\frac{1}{2}\int_0^{1/p} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}} = \frac{1}{\sqrt{z-x}}(F(v_2, m) - F(v_1, m))$$

(4) Elliptic function of Jacobi $\mathrm{sn}(u, m), \mathrm{cn}(u, m), \mathrm{dn}(u, m)$ $(0.0 \leq m \leq 1.0)$

① $u = 0.0$:

$$\mathrm{sn}(0.0, m) = 0.0, \mathrm{cn}(0.0, m) = \mathrm{dn}(0.0, m) = 1.0$$

② $m = 1.0$:

$$\begin{aligned} \mathrm{sn}(u, 1.0) &= \tanh(u) \\ \mathrm{cn}(u, 1.0) &= \mathrm{dn}(u, 1.0) = \mathrm{sech}(u) \end{aligned}$$

③ $m = 0.0$:

$$\begin{aligned} \mathrm{sn}(u, 0.0) &= \sin(u) \\ \mathrm{cn}(u, 0.0) &= \cos(u) \\ \mathrm{dn}(u, 0.0) &= 1.0 \end{aligned}$$

④ For cases other than those in ①, ② and ③:
The following values are obtained

$$\begin{aligned} a_{k+1} &= \frac{a_k + b_k}{2} \\ b_{k+1} &= \sqrt{a_k b_k} \\ c_{k+1} &= \frac{a_k - b_k}{2} \quad (k = 0, 1, \cdots) \end{aligned}$$

with $a_0 = 1.0$ and $b_0 = \sqrt{1-m}$ as initial values.
The convergence criterion is assumed to be

$$c_N < a_{N-1} \cdot \text{(units for determining error)}.$$

The following values are obtained

$$\begin{aligned} K(m) &= \frac{\pi}{2a_N} \\ y_N &= \frac{a_N}{\sin(a_N u)} \\ y_{N-1} &= y_N + \frac{a_N c_N}{y_N} \\ &\vdots \\ y_0 &= y_1 + \frac{a_1 c_1}{y_1} \end{aligned}$$

and the functions are calculated from

$$
\begin{aligned}
\mathrm{sn}(u,m) &= \frac{1}{y_0} \\
\mathrm{cn}(u,m) &= \begin{cases} \sqrt{1-(\frac{1}{y_0})^2} & (\text{when } \mathrm{mod}(\lfloor\frac{|u|}{K(m)}\rfloor,4) = 0 \text{ or } 3) \\ -\sqrt{1-(\frac{1}{y_0})^2} & (\text{when } \mathrm{mod}(\lfloor\frac{|u|}{K(m)}\rfloor,4) = 1 \text{ or } 2) \end{cases} \\
\mathrm{dn}(u,m) &= \sqrt{1-m(\frac{1}{y_0})^2}
\end{aligned}
$$

If the value of the difference of $\lfloor\frac{u}{K(m)}\rfloor$ and $\frac{u}{K(m)}$ is less than or equal to 0.03125, then $\mathrm{sn}(u,m) \simeq 1.0$. Since many digits will be lost in the calculation of $\sqrt{1-\mathrm{sn}^2(u,m)}$, the function is calculated from the following expressions

$$
\begin{aligned}
v &= \frac{u}{2K(m)} \\
\mathrm{sn}(u,m) &= \frac{\vartheta_1(v,q)}{\sqrt[4]{m}\vartheta_4(v,q)} \\
\mathrm{cn}(u,m) &= \frac{\vartheta_2(v,q)\sqrt[4]{\frac{1-m}{m}}}{\vartheta_4(v,q)} \\
\mathrm{dn}(u,m) &= \frac{\vartheta_3(v,q)\sqrt[4]{(1-m)}}{\vartheta_4(v,q)}
\end{aligned}
$$

(For the value of $q$, see next paragraph, “Nome $q$”.)

(5) Nome $q (0.0 \le m \le 1.0)$
If $m > 0.5$, then replace $m$ with $m = 1-m$ and exchange the values of $q$ and $q'$, $K(m)$ and $K(m')$ and $E(m)$ and $E(m')$ respectively with the following procedure.

① $m = 0.0$:

$$
\begin{aligned}
q &= 0.0 \\
q' &= 1.0 \\
K(m) &= \frac{\pi}{2} \\
K(m') &= (\text{Maximum value}) \\
E(m) &= \frac{\pi}{2} \\
E(m') &= 1.0
\end{aligned}
$$

② For cases other than the one in ①:
If $\varepsilon$ are set as follows:

$$
\varepsilon = \frac{0.5m}{(2(1+\sqrt[4]{1-m}(1+\sqrt{1-m})+\sqrt{1-m})-m)}
$$

then

$$
q = c_4\varepsilon^{13} + c_3\varepsilon^9 + c_2\varepsilon^5 + c_1\varepsilon
$$

(Where, $c_1 = 1, c_2 = 2, c_3 = 15, c_4 = 150$ for double precision; $c_1 = 1, c_2 = 2, c_3 = 0, c_4 = 0$ for single precision. )

$$
q' = \exp(\frac{\pi^2}{\log(q)})
$$

If $\delta$ is set as follows:

$$\delta = (d_4 q^9 + d_3 q^4 + d_2 q + d_1)^2$$

(Where, $d_1 = 1, d_2 = 2, d_3 = 1, d_4 = 1$ for double precision; $d_1 = 1, d_2 = 2, d_3 = 1, d_4 = 0$ for single precision. ) then

$$\begin{aligned}
K(m) &= \frac{\pi}{2}\delta \\
K(m') &= (-0.5)\log(q)\delta \\
E(m) &= (\frac{\pi}{6\delta})(1 + (2-m)\delta^2 - (e_7 q^{14} + e_6 q^{12} + e_5 q^{10} + e_4 q^8 + e_3 q^6 + e_2 q^4 + e_1 q^2) \\
E(m') &= K(m')(1 - \frac{E(m)}{K(m)}) + \frac{1}{\delta}
\end{aligned}$$

(Where, $e_1 = 24, e_2 = 72, e_3 = 96, e_4 = 168, e_5 = 144, e_6 = 288, e_7 = 192$ for double precision; $e_1 = 24, e_2 = 72, e_3 = 96, e_4 = 0, e_5 = 0, e_6 = 0, e_7 = 0$ for single precision.)

(6) Elliptic theta function $\vartheta_i(v, q)$ $(0 \le i \le 4, 0.0 \le q \le 1.0)$
If $i = 0$, then set $i = 4$. If $i = 2$, then set $v = v + 0.5$. If $i = 3$, then set $v = v - 0.5$.

① $i = 1$ or 2:
Set $s = \mathrm{SIGN}(v)$ and $v = \mid v \mid$. If $v = v - \mathrm{INT}(v)$ or $\mathrm{INT}(v)$ is an odd number, then let $s = -s$.

(a) $q = 0.0$ or $v = 0.0$, then

$$\vartheta(v, q) = 0.0$$

(b) $q \le 0.125$, then by setting $sw$, $s3w$ and $s5w$ as follows:

$$\begin{aligned}
sw &= \sin(\pi \cdot v) \\
s3w &= (-4 \cdot sw^2 + 3) \cdot sw \\
s5w &= ((16 \cdot sw^2 - 20) \cdot sw^2 + 5) \cdot sw
\end{aligned}$$

the function is calculated from the following expressions.

- Single precision

$$\vartheta(v, q) = 2 \cdot s \cdot \sqrt[4]{q} \cdot ((s5w \cdot q^4 - s3w) \cdot q^2 + sw)$$

- Double precision
  By setting $s7w$ and $s9w$ as follows:

$$\begin{aligned}
s7w &= (((-64 \cdot sw^2 + 112) \cdot sw^2 - 56) \cdot sw^2 + 7) \cdot sw \\
s9w &= ((((256 \cdot (1 - sw^2) - 448) \cdot \\
&\qquad (1 - sw^2) + 240) \cdot (1 - sw^2) - 40) \cdot (1 - sw^2) + 1) \cdot sw
\end{aligned}$$

  the function is calculated from

$$\vartheta_i(v, q) = 2 \cdot s \cdot \sqrt[4]{q} \cdot ((((s9w \cdot q^8 - s7w) \cdot q^6 + s5w) \cdot q^4 - s3w) \cdot q^2 + sw)$$

(c) $q > 0.125$:
If $v > 0.5$, then $v = 1.0 - v$.
By setting $PLQ$, $wQ$ and $w$ as follows:

$$\begin{aligned}
PLQ &= \pi^2 \cdot (1/\log(q)) \\
wQ &= \exp(PLQ) \\
w &= -\exp(2 \cdot v \cdot PLQ)
\end{aligned}$$

the function is calculated from the following expressions.

- Single precision

$$\vartheta_i(v,q) = s\cdot\sqrt{\pi\cdot(-1/\log(q))}\cdot\exp((v^2-v+0.25)\cdot PLQ)\cdot(((wQ^6\cdot w+wQ^2)\cdot w +1)\cdot w+1+(wQ^4/w+1)\cdot(wQ^2/w))$$

- Double precision

$$\vartheta_i(v,q) = s\cdot\sqrt{\pi\cdot(-1/\log(q))}\cdot\exp((v^2-v+0.25)\cdot PLQ) \cdot\{(((wQ^{12}\cdot w+wQ^6)\cdot w+wQ^2)\cdot w+1)\cdot w+1 +((wQ^8/w+wQ^2)\cdot(wQ^2/w)+1)\cdot(wQ^2/w)\}$$

② $i=3$ or 4:
Let $v=\mid v\mid -\mathrm{INT}(\mid v\mid)$.

(a) If $q=0.0$, then

$$\vartheta_i(v,q)=1.0$$

(b) If $q\le 0.125$, then by setting $cw$, $c2w$ and $c3w$ as follows:

$$\begin{aligned} cw &= \cos(2\cdot\pi\cdot v)\\ c2w &= 2\cdot cw^2-1\\ c3w &= (4\cdot cw^2-3)\cdot cw \end{aligned}$$

the function is calculated from the following expressions.

- Single precision

$$\vartheta_i(v,q)=2\cdot((-c3w\cdot q^5+c2w)\cdot q^3-cw)\cdot q+1$$

- Double precision
By setting $c4w$ as follows:

$$c4w=(8\cdot cw^2-8)\cdot cw^2+1$$

the function is calculated from

$$\vartheta_i(v,q)=2\cdot(((c4w\cdot q^7-c3w)\cdot q^5+c2w)\cdot q^3-cw)\cdot q+1$$

(c) $q>0.125$:
If $v>0.5$, then let $v=1.0-v$.
By setting $PLQ$, $wQ$ and $w$ as follows:

$$\begin{aligned} PLQ &= \pi^2\cdot(1/\log(q))\\ wQ &= \exp(PLQ)\\ w &= \exp(2\cdot v\cdot PLQ) \end{aligned}$$

the function is calculated from the following expressions.

- Single precision

$$\vartheta_i(v,q) = \sqrt{\pi\cdot(-1/\log(q))}\cdot\exp((v^2-v+0.25)\cdot PLQ)\cdot(((wQ^6\cdot w +wQ^2)\cdot w+1)\cdot w+1+(wQ^4/w+1)\cdot(wQ^2/w))$$

- Double precision

$$\vartheta_i(v,q) = \sqrt{\pi\cdot(-1/\log(q))}\cdot\exp((v^2-v+0.25)\cdot PLQ)\cdot((((wQ^{12}\cdot w+wQ^6)\cdot w +wQ^2)\cdot w+1)\cdot w+1+((wQ^6/w+wQ^2)\cdot(wQ^2/w)+1)\cdot(wQ^2/w))$$

(7) Zeta function of Jacobi $Z(u)$ $(0.0\le m\le 1.0)$
$K(m)$, $K(m')$, nome $q$ and complementary nome $q'$ are calculated from $m$.
The values $v$, $s$ and $w$ are set as follows:

$$v=u/(2\cdot K(m))$$

$s = \mathrm{SIGN}(v), v = \mid v \mid - \mathrm{INT}(\mid v \mid), w = 2\pi \cdot v$

① $q \leq 0.125$:

- Single precision

$$Z(u) = \frac{\pi \cdot 2q}{K(m)} \cdot \frac{\sin(w) - 2q^3 \cdot \sin(2w) + 3q^8 \cdot \sin(3w)}{1 - 2q(\cos(w) - q^3 \cdot \cos(2w) + q^8 \cdot \cos(3w))} \cdot s$$

- Double precision

$$Z(u) = \frac{\pi \cdot 2q}{K(m)} \cdot \frac{\sin(w) - 2q^3 \cdot \sin(2w) + 3q^8 \cdot \sin(3w) - 4q^{15} \cdot \sin(4w)}{1 - 2q(\cos(w) - q^3 \cdot \cos(2w) + q^8 \cdot \cos(3w) - q^{15} \cdot \cos(4w))} \cdot s$$

② $q > 0.125$:
Assume $z = \exp(-\pi u/K(m'))$.
The function is calculated from the following expressions.

- Single precision

$$Z(u) = \frac{\pi}{2k(m')} \cdot \left( \frac{-5q'^6 z^5 - 3q'^2 z^4 - z^3 + z^2 + 3q'^2 z + 5q'^6}{q'^6 z^5 + q'^2 z^4 + z^3 + z^2 + q'^2 z + q'^6} - 2v \right) \cdot s$$

- Double precision

$$Z(u) = \frac{\pi}{2k(m')} \cdot ( \frac{(z^3 - z^4) + 3q'^2(z^2 - z^5) + 5q'^6(z - z^6) + 7q'^{12}(1 - z^7)}{(z^3 + z^4) + q'^2(z^2 + z^5) + q'^6(z + z^6) + q'^{12}(1 + z^7)} - 2v) \cdot s$$

③ $q > \left\{ \begin{array}{l} \text{Double precision} : 0.8 \\ \text{Single precision} \quad : 0.6 \end{array} \right\}$ and $v \leq 0.5$:
The function is calculated from the following expression.

$$Z(u) = \tanh(u) - 2v$$

(8) Epsilon function of Jacobi $E(u)$ $(0.0 \leq m \leq 1.0)$
The function is calculated from the following expression.

$$E(u) = Z(u) + \frac{E(m)}{K(m)} \cdot u$$

($E(m)$ and $K(m)$ are calculated by using the nome $q$.)

(9) Theta function of Jacobi $\Theta(u)$ $(0.0 \leq m \leq 1.0)$
The function is calculated from the following expression.

$$\Theta(u) = \vartheta_4(u/2 \cdot K(m), q)$$

($q$ and $K(m)$ are calculated by using the nome $q$ and $\vartheta_4$ is calculated by using the theta function.)
However, if $m = 1.0$, $\Theta(u) = 0.0$.

(10) Pi function $\Pi(u, \alpha)$ $(0.0 \leq m < 1.0)$
The function is calculated from the following expression.

$$\Pi(u, \alpha) = \frac{1}{2} \log \frac{\Theta(u - \alpha)}{\Theta(u + \alpha)} + u \cdot Z(\alpha)$$

#### 2.1.2.8 Indefinite Integrals Of Elementary Functions

(1) Exponential integrals $\overline{\mathrm{Ei}}(x), \mathrm{Ei}(-x)$ $(x > 0.0)$

If $x < 0.0$, $\mathrm{Ei}(x)$ is calculated; if $x > 0.0$, $\overline{\mathrm{Ei}}(x)$ is calculated. The calculation methods are described below.

① $x < -4.0$:

The function is calculated from

$$\mathrm{Ei}(x) = \frac{e^x}{x}\left\{1 + (-\frac{1}{x})\frac{\sum_{k=0}^{m} a_k^{(1)}(-\frac{1}{x})^k}{\sum_{k=0}^{m} b_k^{(1)}(-\frac{1}{x})^k}\right\}$$

② $-4.0 \le x < -1.0$:

The function is calculated from

$$\mathrm{Ei}(x) = -e^x \frac{\sum_{k=0}^{m} a_k^{(2)}(-\frac{1}{x})^k}{\sum_{k=0}^{m} b_k^{(2)}(-\frac{1}{x})^k}$$

③ $-1.0 \le x < 0.0$:

The function is calculated by generating the best approximation of

$$\mathrm{Ei}(x) = \log(-x) + \gamma + \sum_{n=1}^{\infty} \frac{(-x)^n}{n!n}$$

($\gamma$ is the Euler number.)

The method of generating the best approximation is described in Section 2.1.2.22.

④ $0.0 < x \le 1.0$:

The function is calculated by generating the best approximation of

$$\overline{\mathrm{Ei}}(x) = \log(x) + \gamma + \sum_{n=1}^{\infty} \frac{x^n}{n!n}$$

The method of generating the best approximation is described in Section 2.1.2.22.

⑤ $1.0 < x \le 6.0$:

The function is calculated from

$$\overline{\mathrm{Ei}}(x) = \log(\frac{x}{x_0}) + (x - x_0)\frac{\sum_{k=0}^{m} a_k^{(3)} x^k}{\sum_{k=0}^{m} b_k^{(3)} x^k}$$

($x_0 = 0.37250741078136663446$)

⑥ $6.0 < x \le 12.0$:

The function is calculated from

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left(a_0^{(4)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(4)} \big|}{\big| a_k^{(4)} + x}\right)$$

⑦ $12.0 < x \le 24.0$:

The function is calculated from

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left(a_0^{(5)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(5)} \big|}{\big| a_{k-1}^{(5)} + x}\right)$$

⑧ $x > 24.0$:
The function is calculated from

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left\{1 + \frac{1}{x}\left(a_0^{(6)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(6)}\,\big|}{\big|\, a_k^{(6)} + x}\right)\right\}$$

The coefficients $a_k^{(1)} \sim a_k^{(6)}, b_k^{(1)} \sim b_k^{(6)}$ are described in references (5) and (6). $a_k^{(3)}$ and $b_k^{(3)}$ are obtained with telescoping the coefficients of shifted Chebyshev polynomials.

(2) Logarithmic integral $\mathrm{Li}(x)$ $(x \geq 1.0)$
The function is calculated using the exponential integral from

$$\mathrm{Li}(x) = \overline{\mathrm{Ei}}(\log(x))$$

(Logarithmic integral $\mathrm{Li}(x)$ is also denoted by $\mathrm{li}(x)$.)

(3) Sine integral $\mathrm{Si}(x)$

① $x < 0.0$:
From $\mathrm{Si}(x) = -\,\mathrm{Si}(-x)$, following methods are applied to $\mathrm{Si}(-x)$.

② $x \geq 0.0$:

(a) $x \geq \left\{\begin{array}{l}\text{Double precision} : 2^{50} \cdot \pi \\ \text{Single precision} \ \ : 2^{18} \cdot \pi\end{array}\right\}$:
The function value is given as

$$Si(x) = \frac{\pi}{2}$$

(b) For cases other than those described in (a)

i. $x \leq 5.0$:
The function is calculated by generating the best approximation from

$$Si(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1) \cdot (2k+1!)}$$

The method of generating the best approximation is described in Section 2.1.2.22.

ii. $x \geq 42.0$:
The function is calculated from

$$Si(x) = \frac{\pi}{2} - f(x) \cdot \cos(x) - g(x) \cdot \sin(x)$$

where $f(x)$ and $g(x)$ are obtained by generating approximations of the following equations.

$$f(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (2k)!}{x^{2k+1}}$$

$$g(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (2k+1)!}{x^{2k+2}}$$

The method of generating the best approximation is described in Section 2.1.2.22.

iii. $5.0 < x < 42.0$:
Let $Q = Si(x_n)$ (where $x_n$ is the closest integer to $x$ and $Si(x_n)$ is the actual value of the sine integral at $x_n$) and let $z = x - x_n$.

A. If $|z| <$ (units for determining error) $\cdot\, x$, then the function value is given as

$$Si(x) = Si(x_n).$$

B. Otherwise, the following calculations are repeated until $| QQ |< \|Q \cdot \text{(units for determining error)}\|$

$$\begin{aligned} f^{(J)}(x_n) &= \{(\sin(x_n))^{(J-1)} - (J-1) \cdot f^{(J-1)}(x_n)\}/x_n \\ QQ &= f^{(J)}(x_n) \cdot Z^J/J! \\ Si(x) &= Q \end{aligned}$$

(4) Cosine integral $\mathrm{Ci}(x)$ $(x \geq 0.0)$

① $x \leq 2.0$:
The function is calculated by generating the best approximation from

$$\mathrm{Ci}(x) = \gamma + \log(x) + \sum_{k=1}^{\infty} \frac{(-1)^k \cdot x^{2k}}{2k \cdot (2k)!}$$

where $\gamma$ is the Euler number ($\gamma = 0.57721566490153286061$).
The method of generating the best approximation is described in Section 2.1.2.22.

② $x \geq 42.0$:
The function is calculated from

$$\mathrm{Ci}(x) = f(x) \cdot \sin(x) - g(x) \cdot \cos(x)$$

where $f(x)$ and $g(x)$ are the same as described in (35) for the sine integral.

③ $2.0 < x < 42.0$:
Let $Q = \mathrm{Ci}(x_n)$(where $x_n$ is the closest integer to $x$ and $\mathrm{Ci}(x_n)$ is the actual value of the cosine integral at $x_n$) and let $z = x - x_n$.

(a) $|z| <$ (units for determining error) $\cdot\, x$, then the function value is given as

$$\mathrm{Ci}(x) = \mathrm{Ci}(x_n)$$

(b) Otherwise, assuming that

$$\begin{aligned} f^{(1)}(x_n) &= (\cos(x_n) - 1)/x_n \\ Q &= Q + f^{(1)}(x_n) \cdot z \end{aligned}$$

the following calculations are repeated until $| QQ |<| Q \cdot \text{(units for determining error)} |$

$$\begin{aligned} f^{(J)}(x_n) &= \{(\cos(x_n))^{(J-1)} - (J-1) \cdot f^{(J-1)}(x_n)\}/x_n \\ QQ &= f^{(J)}(x_n) \cdot z^J/J! \\ Q &= Q + QQ \qquad (J = 2, 3, \cdots\cdots) \end{aligned}$$

and then the function value is given as

$$\mathrm{Ci}(x) = Q + \log(\frac{x}{x_n})$$

(5) Fresnel integral $S(x)$ and $C(x)$
Fresnel integrals are defined by the following equations.

$$\begin{aligned} S(x) &= \int_0^x \sin(\frac{\pi}{2} \cdot t^2)dt \\ C(x) &= \int_0^x \cos(\frac{\pi}{2} \cdot t^2)dt \end{aligned}$$

① $x < 0.0$:
From $S(x) = -S(-x)$ or $C(x) = -C(-x)$, following methods are applied to $S(-x)$ or $C(-x)$.

② $x \geq 0.0$:

(a) $x \leq 1.6$:
The functions are calculated by generating the best approximation from

$$S(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (\frac{\pi}{2})^{2k+1}}{(2k+1)! \cdot (4k+3)} \cdot x^{4k+3}$$

$$C(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (\frac{\pi}{2})^{2k}}{(2k)! \cdot (4k+1)} \cdot x^{4k+1}$$

The method of generating the best approximation is described in Section 2.1.2.22.

(b) $x \geq 5.0$:
The functions are calculated from

$$S(x) = \frac{1}{2} - \left\{ f(x) \cdot \cos\left(\frac{\pi}{2} \cdot x^2\right) + g(x) \cdot \sin\left(\frac{\pi}{2} \cdot x^2\right) \right\}$$

$$C(x) = \frac{1}{2} + \left\{ f(x) \cdot \sin\left(\frac{\pi}{2} \cdot x^2\right) - g(x) \cdot \cos\left(\frac{\pi}{2} \cdot x^2\right) \right\}$$

where $f(x)$ and $g(x)$ are obtained by generating the best approximations of the following equations.

$$f(x) = x \cdot \sum_{k=0}^{m} \frac{(-1)^k \cdot (4k-1)!!}{(\pi \cdot x^2)^{2k+1}}$$

$$g(x) = x \cdot \sum_{k=0}^{m} \frac{(-1)^k \cdot (4k+1)!!}{(\pi \cdot x^2)^{2k+2}}$$

The method of generating the best approximation is described in Section 2.1.2.22.

(c) $1.6 < x < 5.0$:
Using the conversion $Y = \frac{\pi}{2} \cdot x^2$, the functions are calculated from

$$S(x) = \frac{1}{2} - \{f(Y) \cdot \cos(Y) + g(Y) \cdot \sin(Y)\}$$

$$C(x) = \frac{1}{2} + \{f(Y) \cdot \sin(Y) - g(Y) \cdot \cos(Y)\}$$

where $f(Y)$ and $g(Y)$ are as follows: Single precision:

$$f(Y) = \frac{1}{x} \sum_{k=0}^{7} a_k^{(1)} (\frac{4}{Y})^k$$

$$g(Y) = \frac{1}{x} \sum_{k=0}^{8} b_k^{(1)} (\frac{4}{Y})^k$$

Double precision:

$$f(Y) = \frac{1}{x} \sum_{k=0}^{34} a_k^{(2)} (\frac{4}{Y})^k$$

$$g(Y) = \frac{1}{x} \sum_{k=0}^{34} b_k^{(2)} (\frac{4}{Y})^k$$

The coefficients $a_k^{(1)}$, $b_k^{(1)}$, $a_k^{(2)}$ and $b_k^{(2)}$ are obtained with telescoping.

(6) Dawson integral $F(x)$
The value of the Dawson integral is defined by the following equation.

$$F(x) = e^{-x^2} \int_0^x e^{t^2} dt$$

① $x \geq 0.0$:
From $F(x) = -F(-x)$, following methods are applied to $F(-x)$.

② $x \geq 0.0$:

(a) $x \leq 2.5$:
The function is calculated from

$$F(x) = x\frac{\sum_{k=0}^{m} a_k^{(1)} x^{2k}}{\sum_{k=0}^{m} b_k^{(1)} x^{2k}}$$

(b) $2.5 < x \leq 3.5$:
The function is calculated from

$$F(x) = \frac{1}{x} \cdot (a_0^{(2)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(2)} |}{| a_k^{(2)} + x^2})$$

(c) $3.5 < x \leq 5.0$:
The function is calculated from the same equation in case (b) except that the coefficients are $a_k^{(3)}$ and $b_k^{(3)}$.

(d) $5.0 < x \leq 1/\varepsilon$($\varepsilon$ :units for determining error)
The function is calculated from

$$F(x) = \frac{1}{2x} \cdot \{1 + x^{-2} \cdot (a_0^{(4)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(4)} |}{| a_k^{(4)} + x^2})\}$$

(e) $x > 1/\varepsilon$:
The function is calculated from $F(x) = \frac{1}{2x}$.
The coefficients $a_k^{(1\sim4)}$ and $b_k^{(1\sim4)}$ are described in reference (6).

(7) Normal distribution function and complementary normal distribution function $\Phi(x)$ and $\Psi(x)$
The function are calculated from the following equations.

$$\begin{aligned} \Phi(x) &= \frac{1}{2} \cdot \mathrm{ERF}(x/\sqrt{2}) \\ \Psi(x) &= \frac{1}{2} \cdot \mathrm{ERFC}(x/\sqrt{2}) \end{aligned}$$

where error function ERF and complementary error function ERFC will be given later.

#### 2.1.2.9 Associated Legendre Functions

(1) Associated Legendre function of the 1st kind $P_n^m(x)$

① For normalized Functions, see paragraph of Normalized Spherical Harmonics.

② $n < 0$:
From $P_n^m(x) = P_{-n-1}^m(x)$, following methods are applied to $P_{-n-1}^m(x)$.

③ $m < 0$ and $n \geq 0$:
First, $P_n^{|m|}(x)$ is obtained using the methods described in ③ or ⑤, then $P_n^m(x)$ is calculated from the following expressions.

(a) $|x| \leq 1.0$:

$$P_n^m(x) = (-1)^m \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} P_n^{|m|}(x)$$

(b) $|x| > 1.0$:

$$P_n^m(x) = \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} P_n^{|m|}(x)$$

④ $m > n \geq 0$:

$$P_n^m(x) = 0.0$$

⑤ $m = 0$:

The function $P_n^m(x)$ is equal to Legendre polynomial $P_n(x)$, its value is calculated using initial values $P_0(x) = 1$ and $P_1(x) = x$ and the recurrence relation

$$P_k(x) = \frac{2k-1}{k} \cdot x \cdot P_{k-1}(x) - \frac{k-1}{k} \cdot P_{k-2}(x)(k = 2, \cdots, n)$$

⑥ $n \geq m > 0$:

(a) $n = 1$ and $x \geq \sqrt{1/\varepsilon}$:

The function is calculated from

$$P_n^m(x) = x$$

(b) $n = m$:

The function is calculated from

$$P_n^m(x) = (2n-1)!! \cdot (\sqrt{|\ 1 - x^2\ |})^m$$

(c) $n = m + 1$:

The function is calculated from

$$P_n^m(x) = (2n-1)!! \cdot x \cdot (\sqrt{|\ 1 - x^2\ |})^m$$

(d) $n \geq m + 2$:

The function is calculated using initial values

$$F_m = (2m-1)!!, F_{m+1} = (2m+1)!! \cdot x = F_m \cdot (2m+1) \cdot x$$

and the recurrence relation

$$\begin{aligned} F_k &= \frac{2k-1}{k-m} \cdot x \cdot F_{k-1} - \frac{k+m-1}{k-m} \cdot F_{k-2} \qquad (k = m+2, \cdots n) \\ P_n^m(x) &= (\sqrt{|\ 1 - x^2\ |})^m \cdot F_n \end{aligned}$$

(2) Associated Legendre function of the second kind $Q_n^m(x)$ $(n \geq 0, |x| = 1.0)$

① $m < 0$:

First, $Q_n^{|m|}$ is obtained using described in the case ② or ③, then $Q_n^m(x)$ is calculated from following expressions.

(a) $|x| < 1.0$:

$$Q_n^m(x) = (-1)^m \frac{(n-|\ m\ |)!}{(n+|\ m\ |)!} Q_n^{|m|}(x)$$

(b) $|x| > 1.0$:

$$Q_n^m(x) = \frac{(n-|\ m\ |)!}{(n+|\ m\ |)!} Q_n^{|m|}(x)$$

② $|x| < 1.0$ and $m \geq 0$:

The values $Q_n^0(x)$ and $Q_{n-1}^0(x)$ are obtained bye using initial values

$$Q_0^0(x) = \mathrm{artanh(x)}, \mathrm{Q_1^0(x) = x \cdot Q_0^0(x) - 1}$$

and the following recurrence relation

$$Q_{k+1}^0(x) = \{(2k+1) \cdot x \cdot Q_k^0(x) - k \cdot Q_{k-1}^0(x)\}/(k+1) \qquad (k = 1, 2, \cdots, n-1)$$

(If $m = 0$, the calculation ends here.)

next, the value $Q_n^1(x)$ is obtained from

$$Q_n^1(x) = \{-n \cdot x \cdot Q_n^0(x) + n \cdot Q_{n-1}^0(x)\}/\sqrt{1 - x^2}$$

If $n = 0$, the following equation is used

$$Q_n^1(x) = -1/\sqrt{1 - x^2}$$

(If $m = 1$, the calculation ends here.)

The $Q_n^m(x)$ is obtained by calculating the following recurrence relation

$$\begin{aligned} Q_n^{k+2}(x) &= 2\cdot(k+1)\cdot(x/\sqrt{1-x^2})\cdot Q_n^{k+1}(x) - (n-k)\cdot(n+k+1)\cdot Q_n^k(x) \\ & (k = 0, 1, \cdots, m-2) \end{aligned}$$

③ $|x| > 1.0$ and $m \geq 0$:

(a) $x > \sqrt{n+2}$ and $n \geq m$

i. $n = m = 0$:
The function is calculated from

$$Q_n^m(x) = \text{artanh}(1/\text{x})$$

ii. $|x| > \sqrt{(n+0.5)/\varepsilon}$:
The function is calculated from

$$Q_n^m(x) = (-1)^m \frac{(n+m)!}{(2n+1)!!} x^{-n-1}$$

iii. Otherwise:
The function is calculated from the following series expansion

$$\begin{aligned} Q_n^m(x) &= (-1)^m (x^2-1)^{\frac{m}{2}} \frac{(n+m)!}{(2n+1)!!} x^{-n-m-1} \\ & \cdot \left(1 + \sum_{k=1}^{\infty} \frac{(n+m+1)\cdot(n+m+2)\cdots(n+m+2\cdot k)}{2k!!\cdot(2n+3)\cdot(2n+5)\cdots(2n+2k+1)} \cdot \frac{1}{x^{2k}}\right) \end{aligned}$$

Where the summation $\Sigma$ is calculated until the last term is sufficiently small compared with the first term.

(b) For values other than those in (a)
First, $Q_0^0(x)$ is calculated from the following expression

$$Q_0^0(x) = \text{artanh}(1/\text{x})$$

(If $n = m = 0$, the calculation ends here.)
Next, $Q_n^0(x)$ and $Q_{n-1}^0(x)$ are calculated using the series expansion of $f$ as follows:

$$\begin{aligned} f &= \sum_{k=n}^{\infty} \frac{1}{(k+1)\cdot P_k(x)\cdot P_{k+1}(x)} \\ Q_n^0(x) &= P_n(x)\cdot f \\ Q_{n-1}^0(x) &= P_{n-1}(x)\cdot f + \frac{1}{n\cdot p_n\cdot\ (x)} \end{aligned}$$

where, $P_k(x)$ is solved using the algorithm shown in 2.1.2.10 (1).
(If $m = 0$, the calculation ends.)
Next, $Q_n^1(x)$ is obtained from

$$Q_n^1(x) = \{n\cdot x\cdot Q_n^0(x) - n\cdot Q_{n-1}^0(x)\}/\sqrt{x^2-1}$$

However, if $n = 0$, $Q_n^1(x)$ is calculated from

$$Q_n^1(x) = -1/\sqrt{x^2-1}.$$

(If $m = 1$, the calculation ends here.)
Finally $Q_n^m(x)$ is obtained by calculating the following recurrence relation

$$\begin{aligned} Q_n^{k+2}(x) &= -2\cdot(k+1)\cdot(x/\sqrt{x^2-1})\cdot Q_n^{k+1}(x) + (n-k)\cdot(n+k+1)\cdot Q_n^k(x) \\ & (k = 0, 1, \cdots, m-2) \end{aligned}$$

#### 2.1.2.10 Orthogonal Polynomials

(1) Legendre polynomial $P_n(x)$ $(n \geq 0)$
$P_n(x)$ is obtained using initial values $P_0(x) = 1.0$ and $P_1(x) = x$ and the following recurrence relation

$$P_k(x) = \frac{2k-1}{k} \cdot x \cdot P_{k-1}(x) - \frac{k-1}{k} \cdot P_{k-2}(x) \qquad (k = 2, 3, \cdots, n)$$

(2) Laguerre polynomial $L_n(x)$ $(n \geq 0)$
$L_n(x)$ is obtained using initial values $L_0(x) = 1.0$ and $L_1(x) = 1.0 - x$ and the following recurrence relation

$$L_k(x) = \frac{(2k-x-1)}{k} \cdot L_{k-1}(x) - \frac{(k-1)}{k} \cdot L_{k-2}(x) \qquad (k = 2, 3, \cdots, n)$$

(3) Hermite polynomial $H_n(x)$ $(n \geq 0)$
$H_n(x)$ is obtained using initial values $H_0(x) = 1.0$ and $H_1(x) = 2 \cdot x$ and the following recurrence relation

$$H_k(x) = 2 \cdot x \cdot H_{k-1}(x) - 2 \cdot (k-1) \cdot H_{k-2}(x) \qquad (k = 2, 3, \cdots, n)$$

(4) Chebyshev polynomial $T_n(x)$ $(n \geq 0)$
$T_n(x)$ is obtained using initial values $T_0(x) = 1.0$ and $T_1(x) = x$ and the following recurrence relation

$$T_k(x) = 2 \cdot x \cdot T_{k-1}(x) - T_{k-2}(x)(k = 2, 3, \cdots, n)$$

(5) Chebyshev Function of the 2nd kind $U_n(x)(n \geq 0, |x| \leq 1.0)$
$U_n(x)$ is obtained using initial values $U_0(x) = 0.0$ and $U_1(x) = \sqrt{1-x^2}$ and the following expression

$$U_{n+1}(x) = 2 \cdot x \cdot U_n(x) - U_{n-1}(x)$$

(6) Generalized Laguerre polynomial $L_n^{(\alpha)}(x)$ $(n \geq 0)$
$L_n^{\alpha}(x)$ is obtained by using initial values $L_0^{(\alpha)}(x) = 1.0$ and $L_1^{(\alpha)}(x) = (1+\alpha) - x$ and the following recurrence relation

$$L_n^{(\alpha)}(x) = ((2 \cdot n + \alpha - x - 1) \cdot L_{n-1}^{(\alpha)}(x) + (1 - \alpha - n) \cdot L_{n-2}^{(\alpha)}(x))/n$$

#### 2.1.2.11 Mathieu functions of integer orders

Consider the second order simultaneous ordinary differential equation as below

$$\frac{d^2y}{dx^2} + (a - 2q\cos 2x)y = 0.$$

For a real parameter $q$, let $a = c_0, c_1, \cdots$ be the real parameters such that the solutions are $2\pi$ periodic and even functions. Here,

$$c_0 < c_1 < c_2 < \cdots.$$

Also, let $a = s_1, s_2, \cdots$ be the real parameters such that the solutions are $2\pi$ periodic and odd functions. Here,

$$s_1 < s_2 < s_3 < \cdots.$$

For $a = c_n$ with an integer $n \geq 0$, normalize the periodic even solution $c_n(x, q)$ of the differential equation so that the following condition is satisfied.

$$\int_0^{2\pi} ce_n(x, q)^2 dx = \pi$$

The $ce_n(x, q)$ is referred to as Matheiu function of integer order. Similarly, for $a = s_n$ with an integer $n \geq 1$, normalize the periodic odd solution $s_n(x, q)$ of the differential equation so that the following condition is satisfied.

$$\int_0^{2\pi} se_n(x,q)^2 dx = \pi$$

The $se_n(x, q)$ is also referred to as Matheiu function of integer order.
It is known that $ce_n(x, q)$ and $se_n(x, q)$ can be represented by the following Fourier expansions:

(1) If s=0 (for $ce_n$ with even $n$)

$$ce_n(x,q) = \sum_{r=0}^{\infty} X_r \cos(2rx).$$

(2) If s=1 (for $se_n$ with odd $n$)

$$se_n(x,q) = \sum_{r=0}^{\infty} X_r \sin((2r+1)x).$$

(3) If s=2 (for $ce_n$ with odd $n$)

$$ce_n(x,q) = \sum_{r=0}^{\infty} X_r \cos((2r+1)x).$$

(4) If s=3 (for $se_n$ with positive even $n$)

$$se_n(x,q) = \sum_{r=0}^{\infty} X_r \sin((2r+2)x).$$

These are referred to as Mathieu functions of order $n$. By substituting these formulas for $ce_n$ and $se_n$ into the differential equation, the equation reduces to an eigenvalue problem of tridiagonal real symmetric matrix $A_s(q)$, and this gives the expansions $X_r$.

$$A_s(q) = \begin{bmatrix} K_s^2 + d_s q & R_s q & & & \\ R_s q & (2+K_s)^2 & q & & 0 \\ & q & (4+K_s)^2 & q & \\ & 0 & q & (6+K_s)^2 & \ddots \\ & & & \ddots & \ddots \end{bmatrix}$$

Here,

(1) $R_0 = \sqrt{2}, R_1 = R_2 = R_3 = 1$

(2) $K_0 = 0, K_1 = K_2 = 1, K_3 = 2$

(3) $d_0 = 0, d_1 = -1, d_2 = 1, d_3 = 0,$

and $s$=0, 1, 2 or 3. The $s$ is determined by the kind of Mathieu functions (whether it is $ce_n(x, q)$ or $se_n(x, q)$), and whether order $n$ is odd or even. Eigenvalues $[R_s X_0, X_1, X_2, X_3, \cdots]^t$ of $A_s(q)$ are choose to become unit vectors. Define an integer constant $l$ with the following conditions:

(1) For $ce_n$
$l = 1 + [n/2]$

(2) For $se_n$
$l = n/2$ (If $n$ is odd.)
$l = (n+1)/2$ (If $n$ is even.)

The direction of the eigenvector which corresponds to $l$-th smallest eigenvalue is selected to satisfy $X_l > 0$. When $q \to 0$, Mathieu functions $ce_n(x,q)$ and $se_n(x,q)$ which is obtained with this procedure converge to the following functions.

(1) $ce_0(x,q) \to \frac{1}{\sqrt{2}}$

(2) $ce_n(x,q) \to \cos nx (n = 1, 2, \cdots)$

(3) $se_n(x,q) \to \sin nx (n = 1, 2, \cdots)$

#### 2.1.2.12 Langevin function

① $x < 0.0$:
From $L(x) = -L(-x)$, following methods are applied to $L(-x)$.

② $x \geq 0.0$:

(1) $x \leq 1.5$:
The function is calculated by generating the best approximation of the following equation.

$$L(x) = x \cdot \sum_{k=1}^{\infty} (-1)^{k-1} \{ \frac{2^{2k} \cdot B_k \cdot x^{2k+2}}{(2k)!} \}$$

The method of generating the best approximation is described in Section 2.1.2.22.

(2) $1.5 < x < \left\{ \begin{array}{l} \text{Double precision : } 45.0 \\ \text{Single precision \ : } 20.0 \end{array} \right\}$:
The function is calculated from

$$L(x) = \frac{2 \cdot e^{-2 \cdot x}}{1 - e^{-2 \cdot x}} - \frac{1}{x} + 1$$

(3) $x \geq \left\{ \begin{array}{l} \text{Double precision : } 45.0 \\ \text{Single precision \ : } 20.0 \end{array} \right\}$:
The function is calculated from

$$L(x) = 1 - \frac{1}{x}$$

#### 2.1.2.13 Gauss=Legendre integration formula

(1) Gauss=Legendre integration formula
Suppose an arbitrary Legendre polynomial $F(x)$ of degree $2N-1$ or less is given. The remainder of $F(x)$ divided by $P_N(x)$, the Legendre polynomial of degree $N$, can be expressed as a linear combination of $P_j(x)$ $(j = 0, 1, \cdots, N-1)$:

$$F(x) = P_N(x)Q(x) + b_0 + \sum_{j=1}^{N-1} \sqrt{2j+1} b_j P_j(x)$$

where $Q(x)$ is a polynomial of degree $N-1$ or less and $b_0, b_1, \cdots, b_{N-1}$ are constants. An arbitrary polynomial $Q(x)$ of degree $N-1$ or less can be expressed as a linear combination of $P_j(x)$ $(j = 0, 1, \cdots, N-1)$. Using the orthogonal relation

$$\int_{-1}^{+1} P_N(x)P_j(x)dx = 0 \ (j = 0, 1, \cdots, N-1),$$

it holds that

$$\int_{-1}^{+1} P_N(x)Q(x)dx = 0.$$

Also, by being aware of the fact:

$$\int_{-1}^{+1} P_j(x)dx = 0 \ (j = 1, 2, \cdots, N-1),$$

the relation below is induced from the expression formula for $F(x)$:

$$\int_{-1}^{+1} F(x)dx = 2b_0.$$

Let $\alpha_k (k = 1, 2, \cdots, N)$ be the the $N$ number of zero points of $P_N(x)$. Substituting $x = \alpha_k$ into the expression formula for $F(x)$, it holds that:

$$F(\alpha_k) = b_0 + \sum_{j=1}^{N-1} \sqrt{2j+1} b_j P_j(\alpha_k). \ (k = 1, 2, \cdots, N)$$

For $k = 1, 2, \cdots, N$, let take $d_k^2$ such that:

$$d_k^2 = \sum_{n=0}^{N-1} (2n+1)P_n^2(\alpha_k).$$

From the above expression for $F(\alpha_k)$ and

$$\sum_{k=1}^{N} d_k^{-2} P_j(\alpha_k) = 0 \ (j = 1, 2, \cdots, N-1),$$

$$\sum_{k=1}^{N} d_k^{-2} = 1,$$

it holds that:

$$b_0 = \sum_{k=1}^{N} d_k^{-2} F(\alpha_k).$$

Therefore, if a polynomial $F(x)$ is choose so that its values $F(\alpha_k)$ at $x = \alpha_k (k = 1, 2, \cdots, N)$ coincides with the values of the function $f(x)$, the integration formula of degree $2N-1$ holds:

$$\int_{-1}^{+1} f(x)dx \simeq \int_{-1}^{+1} F(x)dx = 2b_0 = 2\sum_{k=1}^{N} d_k^{-2} F(\alpha_k)$$

$$= 2\sum_{k=1}^{N} d_k^{-2} f(\alpha_k)$$

Here, the polynomial $F(x)$ of degree $2N-1$ or less needs only to satisfy the conditions at the $N$ number of zero points. This means that the choice of $F(x)$ has freedom of degree $N$ at most. If the integrand function $f(x)$ can not be approximated by some polynomial $F(x)$ (for instance when $f(x)$ is a vibrating function), it is necessary to set the degree $N$ sufficiently large when applying this integration formula.

(2) Zero points of Legendre polynomial
We set $X_n = \sqrt{2n+1}P_n(\alpha)$. If $\alpha$ satisfies $P_N(\alpha) = 0$ for $a_n = n/\sqrt{4n^2-1}$, it holds that

$$a_1X_1 = \alpha X_0, a_2X_2 + a_1X_0 = \alpha X_1, a_3X_3 + a_2X_1 = \alpha X_2, \cdots,$$

$$a_{N-1}X_{N-1} + a_{N-2}X_{N-3} = \alpha X_{N-2}, a_{N-1}X_{N-2} = \alpha X_{N-1}.$$

Therefore, since all the zero points of Legendre polynomial are single solutions, they are eigenvalues of a real symmetric tridiagonal matrix whose diagonals are zero and whose subdiagonals are $a_1, a_2, \cdots, a_{N-1}$, and they are obtained with root-free QR method.
For a Legendre polynomial of a large degree, all the zero points gather closely together in the open interval (-1,1). In this case all of the $N$ number of zero points can be calculated certainly by solving the eigenproblem of this real symmetric tridiagonal matrix using root free QR method, instead of computing them using Newton method.

#### 2.1.2.14 Zero points of Bessel Functions

(1) Positive solution for the transcendency equation $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ containing the Bessel function
Positive solution $\alpha$ for the transcendency equation

$$aJ_0(\alpha) + \alpha J_1(\alpha) = 0$$

is obtained with the following method.
By using the following method, the positive solution for the transcendency equation can be calculated up to $M = 50$ in rough. (If it is calculated with the single precision, set to M $\leq$ 24, that is N $\leq$ 192.) The parameter $a$ should be set to $a$=0 and $10^{-10} \leq |a| \leq 10^4$.
1. Calculate an approximate value with the finite element method.
The differential equation

$$u''(r) + u'(r)/r + \alpha^2 u(r) = 0$$

with a boundary condition

$$u'(1) = au(1)$$

has a non-trivial solution $u(r)(0 \leq r \leq 1)$ where $u(+0)$ is finite, if and only if $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ is satisfied. Then it holds that $u(r) = CJ_0(\alpha r)$.
Let $w_j(r)(j = 0, 1, 2, \cdots, N-1, N)$ be the basis function which satisfy $w_j(j/N) = 1, w_j(k/N) = 0 (k \neq j)$ at the nodal points $r = 0, 1/N, 2/N, \cdots, (N-1)/N, 1$. Using the $w_j$, interpolate $u(r)$ using the basis functions as below.

$$u(r) = \sum_{j=0}^{N} u(j/N)w_j(r)$$

Galerkin method can be applied to get

$$\int_0^1 w_j(r)(u''(r) + u'(r)/r + \alpha^2 u(r))rdr = 0.$$

Applying integration by parts to the factor including the second differential calculus, the above relation is reduced to the following integral relation.

$$\int_0^1 w_j'(r)u'(r)rdr - aw_j(1)u(1) = \alpha^2 \int_0^1 w_j(r)u(r)rdr$$

Substitute the interpolated basis function

$$AX = \alpha^2 BX, \ X = (u(0), u(1/N), \cdots, u(1))^t.$$

Where $A$ and $B$ are

$$A_{i,j} = \int_0^1 w_j'(r)w_i'(r)rdr - \delta_{i,N}\delta_{j,N}a,$$

$$B_{i,j} = \int_0^1 w_j(r)w_i(r)rdr.$$

Therefore, the approximate values of solutions $\alpha$ ($\alpha > 0$) for $aJ_0(\alpha)+\alpha J_1(\alpha) = 0$ are obtained by calculating $\sqrt{\beta}$ which are square roots of the positive eigenvalues $\beta$ for the generalized eigenvalue problem $AX = \beta BX$.
2. Improving the Solution by Bisection and Newton method
Set $N = LM$. Here $L$ is the approximate magnification ratio, which is preferred to be 8 in rough. The square roots of the positive eigenvalues $\beta$ for the generalized eigenvalue problem the approximate the solutions for the transcendental equation.
$E$ is defined as below:

$$E = 0(a < 0)$$

$$E = \alpha(0)^2(a \geq 0)$$

Here $\alpha(0)$ is the positive smallest zero point of $J_0(x)$.
Let $\beta_1^*, \beta_2^*, \cdots, \beta_M^*$ be the eigenvalues greater than $E$ in ascending order. We examine that the number of eigenvalues that is smaller than or equal to $E$ does not reach 2. We also examine that $\beta_j^*$ is not a multiple root.

① Bisection method
If $\alpha^* = \sqrt{\beta_j^*}$ for $j \geq 2$ satisfies

$$|aJ_0(\alpha^*) + \alpha^* J_1(\alpha^*)| < 0.1 * (|J_0(\alpha^*)| + |J_1(\alpha^*)|),$$

then step forward to the process to improve the approximate solution using Newton method.
This condition holds when $j$ is 10 or less. When j is larger than 10, however, the condition is not satisfied because of the approximation error due to Galerkin method. In the latter case, therefore, improve $\alpha^*$ by Bisection method.
In Bisection method, use the property that $|aJ_0(\alpha^*) + \alpha^* J_1(\alpha^*)|$ becomes small as the approximate solution $\alpha^*$ comes close to the exact solution. The interval $(\sqrt{\beta_j^* - \delta}, \sqrt{\beta_j^* + \delta})$ such that includes

only the j-th solution is divided into two subintervals and $\alpha^*$ is replaced with one of the edges of these subintervals. Here, $\delta$ is positive real number which has the order of the approximate error. Divisions of intervals are repeated until $\alpha^*$ satisfies the condition above, and then step forward to the process to improve the approximate solution using Newton method.

② Newton method
The improvement using Newton method will be repeated until $|aJ_0(x)+xJ_1(x)| < e$ is satisfied, where
$e$=$10^{-11}$ for double precision
or
$e$=$10^{-4}$ for single precision.

(2) Positive zero points of the Bessel function $J_{m+1}(x)$ with integer order
When $m$=$1, 2, \cdots$, the holomorphic function $F_m(x)$ can be defined as

$$J_m(x) = x^m F_m(x)$$

and this satisfies

$$F''_m(x) + (2m+1)F'_m(x)/x + F_m(x) = 0.$$

Furthermore, it holds that $x^m F'_m(x) = -J_{m+1}(x)$. Therefore, the method of algorithm (1) can be applied to

$$A_{i,j} = \int_0^1 w'_j(r)w'_i(r)r^{2m+1}dr,$$

$$B_{i,j} = \int_0^1 w_j(r)w_i(r)r^{2m+1}dr.$$

The threshold $p$ for determining whether an eigenvalue for generalized eigenvalue problem is positive or not is:
$p = 10^{-3}$ (for double precision),
$p = 10^{-1}$ (for single precision).
In Bisection method, the following error criterion is applied:

$$|J_{m+1}(\alpha^*)| \leq 0.001.$$

(3) Positive zero points of each Bessel function $J_0, J_1$ of the order 0 and 1
The zero point of $J_1(x)$ is obtained as the positive zero point of the transcendent equation in the algorithm (1) when setting $a$=0.
When $a$ is set to a sufficiently large value, the solution of the transcendent equation $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ becomes the approximate value of the zero point of $J_0(x)$. Therefore the zero point of $J_0(x)$ is obtained by applying the algorithm (1) for a sufficiently large value $a$.

#### 2.1.2.15 Positive zero points of the second kind Bessel function

① Obtain Positive zero point of the second kind Bessel function $Y_n(x)$ in window interval $[x_1,\ x_1+2\pi]$ $(x_1 > 0)$ with the following procedure.

(1) Positive zero point on each window interval
Set an initial value $x = x_0$ to the middle-point of the window interval. We let $F(x) = Y_n(x)\cos x - J_n(x)\sin x$ and $G(x) = Y_n(x)\sin x + J_n(x)\cos x$. Furthermore, let $\alpha$ be an argument of the vector $(F(x), G(x))$.

$$\cos\alpha = F(x)/S$$

$$\sin\alpha = G(x)/S$$

$$S = \sqrt{(F(x)^2 + G(x)^2)}$$

Here $\alpha$ is choose so that it lies in the window interval. The improved value $x_{i+1}$ of the approximate zero point is given as the argument of the vector $(F(x), G(x))$. Repeat this improvement.

(2) Decision of existing the positive zero point When repetition of this improvement does not converge, we concluded that positive zero point does not exist in this window interval.

② Shifting the window interval for $2\pi$ per step, repeat the above procedures until the number of already obtained positive zero points reaches the number to be obtained.

#### 2.1.2.16 Zeta function of Positive definite quadratic form $x^2 + ay^2$

We initially define the zeta function $Z(s, a)$ of the positive definite quadratic form $x^2 + ay^2$ as:

$$Z(s,a) = \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (m^2 + an^2)^{-s}\ (s > 1).$$

Subtracting the function to eliminate the unique pole of $Z(s, a)$, the analytic continuation of $Z(s, a)$ is obtained as described below:

$$\begin{aligned} & Z(s,a) - \frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)} \\ = \ & \frac{\pi^s a^{-s/2}}{\Gamma(s)} \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} \left(A_{m,n}^{-s}\int_{A_{m,n}}^{\infty} e^{-t}t^{s-1}dt + A_{m,n}^{s-1}\int_{A_{m,n}}^{\infty} e^{-t}t^{-s}dt\right) - \frac{\pi^s a^{-s/2}}{\Gamma(s+1)}. \end{aligned}$$

Each integral in the above formula is a second kind incomplete gamma integral. Therefore, we obtain

$$\begin{aligned} & Z(s,a) - \frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)} \\ = \ & \frac{\pi^s a^{-s/2}}{\Gamma(s)} \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (A_{m,n}^{-s}\Gamma(s, A_{m,n}) + A_{m,n}^{s-1}\Gamma(1-s, A_{m,n})) - \frac{\pi^s a^{-s/2}}{\Gamma(s+1)} \end{aligned}$$

with $A_{m,n} = \pi(\sqrt{a}m^2 + n^2/\sqrt{a})$.
The summation in the last expression should be taken for all those integer pairs $(m, n) \neq (0, 0)$ that satisfy $A_{m,n} < 40$.

#### 2.1.2.17 Di-log function

The value of

$$Li_2(x) = -\int_0^x \log|t-1|\frac{dt}{t}$$

is given as follows:

① For $x = 1$
The functional value $Li_2(x)$ is $\pi^2/6$.

② For $x > 1$
The functional value $Li_2(x)$ is $\pi^2/3 - \frac{1}{2}(\log x)^2 - Li_2(\frac{1}{x})$, where $Li_2(\frac{1}{x})$ is obtained as below.

③ For $0 \leq x < 1$

(1) If $x > \frac{1}{2}$, $Li_2(x)$ is calculated by $Li_2(x) = \pi^2/6 - \log(1-x)\log x - Li_2(1-x)$. Where, $Li_2(1-x)$ is obtained by applying 2.

(2) If $x \leq \frac{1}{2}$, $Li_2(x)$ is

$$Li_2(x) = \alpha - \frac{1}{4}\alpha^2 - \sum_{k=1}^{8} \frac{c_{2k}}{2k+1}\alpha^{2k+1}.$$

Where $\alpha = -\log(1-x)$ and $c_{2k}$ is Bernoulli numbers.
The functional value $Li_2(x)$ can be calculated by using this relation.

#### 2.1.2.18 Debye function

The Debye function $F_D(x)$ is defined as below:

$$F_D(x) = \frac{3}{x^3}\int_0^x \frac{e^t t^4}{(e^t-1)^2}dt.$$

This is transformed as:

$$F_D(x) = \frac{12}{x^3}\int_0^x \frac{t^3}{e^t-1}dt - \frac{3x}{e^x-1}.$$

The value of this function is calculated by applying two kind of methods as below.

① For $0 \leq x \leq \log 2$ It can be obtained with

$$F_D(x) = -\frac{3x}{e^x-1} + 12(\frac{1}{3} - \frac{x}{8} - \sum_{k=1}^{\infty} \frac{c_{2k}}{2k+3}x^{2k}) = 1 + \sum_{k=1}^{\infty} \frac{3(2k-1)}{2k+3}c_{2k}x^{2k}.$$

Here $c_{2k}$ is Bernoulli numbers. The summation of series for $k$ is performed until the added factor becomes is less than or equal to the unit for determining error (Note Appendix B).

② For $x > \log 2$
It can be obtained with

$$F_D(x) = \frac{12}{x^3}F(x) - \frac{3x}{e^x-1},$$

$$F(x) = \frac{\pi^4}{15} + S_1(y) + S_2(y)\log y + S_3(y)(\log y)^2 + y(\log y)^3,$$

$$S_j(y) = \sum_{k=1}^{\infty} b_{k,j} y^k (j = 1, 2, 3).$$

Here $y = -\log(1 - e^{-x})$. The summation of series for $k$ is performed until the added factor becomes less than or equal to the unit for determining error (Note Appendix B). The coefficients $b_{k,j}$ are calculated with the following procedure.
For $F(x) = \int_0^x t^3 dt/(e^t - 1)$, use the parameter $y(0 < y < \log 2)$

$$F(x) = \int_y^{\infty} (-\log(1 - e^{-u}))^3 du = \int_0^{\infty} - \int_0^y = I_1 - I_2$$

Here,

$$I_1 = \int_0^{\infty} = F(\infty) = \frac{\pi^4}{15},$$

$$I_2 = \int_0^y = \int_0^y (\log(\frac{u}{1 - e^{-u}}) - \log u)^3 du.$$

Furthermore, $b_{k,j}$ is calculated by using the series expansion of $\log(\frac{u}{1-e^{-u}})$

$$\log(\frac{u}{1 - e^{-u}}) = \frac{u}{2} + \sum_{k=1}^{\infty} \frac{c_{2k}}{2k} u^{2k},$$

where $c_{2k}$ is Bernoulli numbers.

#### 2.1.2.19 Normalized Spherical Harmonics

For a real number $x(-1 \le x \le 1)$, calculate the normalized spherical harmonic function (the normalized Legendre function)

$$P_n^{*m}(x) = \frac{1}{4\pi i^m} A_{n,m} \int_{-\pi}^{\pi} (x + i\sqrt{1 - x^2}\cos\psi)^n \cos(m\psi) d\psi.$$

Here, $i = \sqrt{-1}$,

$$A_{n,0} = \sqrt{\frac{2n+1}{\pi}}$$

and

$$A_{n,m} = \sqrt{\frac{2(2n+1)(n-m)!(n+m)!}{\pi(n!)^2}} \ (1 \le m \le n).$$

To obtain the integral factor, apply the following Fourier expansion (set $x = \cos\theta,\ 0 \le \theta \le \pi$):

$$(\cos\theta + i\sin\theta\cos\psi)^n = \sum_{m=0}^{n} i^m C_{n,m}(\theta)\cos(m\psi).$$

If we define $I_{n,m}$ as

$$I_{n,m} = \frac{1}{i^m \sin^m\theta} \int_{-\pi}^{\pi} (\cos\theta + i\sin\theta\cos\psi)^n \cos(m\psi) d\psi,$$

the coefficients $C_{n,m}(\theta)$ of the Fourier expansion can be obtained using the following recurrence formula:

$$I_{n,n} = \frac{\pi}{2^{n-1}},\ I_{n,n+1} = 0$$

$$I_{n,m-2} = \frac{2}{m-n-2}\{-(m-1)\cos\theta I_{n,m-1} + (m+n)\frac{\sin^2\theta}{2}I_{n,m}\}$$

$$C_{n,m}(\theta) = \frac{(2-\delta_{m,0})\sin^m\theta}{2\pi}I_{n,m}$$

Using the coefficients $C_{n,m}(\theta)$ of the Fourier expansion which can be obtained as procedure above, the normalized spherical harmonic function (the normalized Legendre function) $P_n^{*m}(\cos\theta)$ of the order $n$ can be obtained as:

$$P_n^{*m}(\cos\theta) = C_{n,m}(\theta)\frac{1}{h_{n,m}}.$$

Here, the coefficients $h_{n,m}$ are:

$$g_{n,0} = 1, g_{n,m} = g_{n,m-1}(1+\frac{m}{n})^{-1}(1-\frac{m-1}{n})\ \ (m = 1, 2, \cdots)$$

$$h_{n,m} = 2\sqrt{\frac{(2-\delta_{m,0})\pi g_{n,m}}{2n+1}}.$$

#### 2.1.2.20 Hurwitz Zeta function for a real variable

(1) Definition

For real region $s > -1$ and $s \neq 1$ and for a parameter $a > 0$, the Hurwitz zeta function $\zeta(s,a)$ is defined as follows :

$$\zeta(s,a) = \frac{a^{-s+1}}{s-1} + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx.$$

Here, $\psi(x)$ is a periodic function (of period 1) defined as $x - [x] - \frac{1}{2}$.

(a) Definition for $s > 1$

The Hurwitz zeta function is expressed as follows:

$$\begin{aligned}
\zeta(s,a) &= \int_0^\infty (x+a)^{-s}dx + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx \\
&= \sum_{n=0}^{\infty}\{\int_n^{n+1}(x+a)^{-s}dx + \int_n^{n+1} -s\psi(x)(x+a)^{-s-1}dx\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^{\infty}\{\int_n^{n+1}\frac{d}{dx}(\psi(x)(x+a)^{-s})dx\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^{\infty}\{\frac{1}{2}(n+a)^{-s} + \frac{1}{2}(n+1+a)^{-s}\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^{\infty}(n+a)^{-s}.
\end{aligned}$$

Therefore, the Hurwitz zeta function is equal to the infinite series sum $\sum_{n=0}^{\infty}(n+a)^{-s}$ for $s > 1$, which converges to a finite value.

(b) Definition for $s > -1$

An infinite integral

$$\int_0^\infty -s\psi(x)(x+a)^{-s-1}dx$$

is convergent also for $s > -1$. This is easy to see from the fact that the alternating series

$$\sum_{m=0}^{M} \int_{\frac{m}{2}}^{\frac{m+1}{2}} -s\psi(x)(x+a)^{-s-1}dx$$

converge as $M \to \infty$. Therefore, the Hurwitz zeta function can also be defined in this region.

Now it can be said that the Hurwitz zeta function

$$\zeta(s,a) = \frac{a^{-s+1}}{s-1} + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx$$

is a function defined in the region $s \neq 1$, which is an extension of the infinite series sum $\sum_{n=0}^{\infty}(n+a)^{-s}$ that is defined for $s > 1$. Also,

$$\zeta(s,a) - \frac{1}{s-1} = \int_0^1 \{\zeta(s,a) - \zeta(s,x+1)\}dx$$

has an integral expression

$$\int_a^1 x^{-s}dx + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx,$$

which can be defined for $s > -1$, including the case $s = 1$.

(2) Algorithm for Calculation

(a) Set $N$ to a fixed natural number 10; $N = 10$. The Hurwitz zeta function can be expanded as the formula below:

$$\zeta(s,a) = \sum_{n=0}^{N-1}(n+a)^{-s} + Z(s,a) + \frac{(N+a)^{-s}}{2} + R(s,a),$$

$$Z(s,a) = \frac{(N+a)^{-s+1}}{s-1},$$

and

$$R(s,a) = \int_N^\infty -s(x+a)^{-s-1}\psi(x)dx.$$

This expansion formula is obtained by applying the integral expression to the second summation term in the right hand side of the Hurwitz zeta function

$$\zeta(s,a) = \sum_{n=0}^{N-1}(n+a)^{-s} + \sum_{n=N}^{\infty}(n+a)^{-s}.$$

(b) The value of $R(s,a)$ is obtained from the following formula whose error is of the order $10^{-9}$( for single precision ) $10^{-18}$ ( for double precision ):

$$R(s,a) = \sum_{j=0}^{m-2}(-s)\cdots(-s-j)(N+a)^{-s-1-j}\psi_{2+j}(0).$$

Here, $\psi_k(x) \quad (k = 2, 3, \cdots)$ are defined as periodic functions (of period 1)

$$-\psi(x) = \psi_2'(x), \ -\psi_2(x) = \psi_3'(x), \ -\psi_3(x) = \psi_4'(x), \ \cdots$$

satisfying

$$\int_0^1 \psi_k(x)dx = 0.$$

Depending on whether double precision or single precision is assumed and depending on the value $s$, the function $R(s,a)$ is obtained as follows:

i. For single precision
If $s > 10$ then, set $R(s,a) = 0$.
If $0 \le s \le 10$, then apply expansion formula with $m = 10$.

ii. For double precision
If $s > 20$, then set $R(s,a) = 0$.
If $10 < s \le 20$, then apply expansion formula with $m = 10$.
If $5 < s \le 10$, then apply expansion formula with $m = 15$.
If $0 \le s \le 5$, then apply expansion formula with $m = 20$.

Using the Bernoulli numbers $B_l$, $\psi_k(0)$ is represented as

$$\psi_{2j+1}(0) = 0$$

and

$$\psi_{2j}(0) = (-1)^j \frac{B_j}{(2j)!}.$$

For arbitrary real numbers $x$ and $t \quad (|t| < 2\pi)$, it holds that

$$\frac{te^{x_1 t}}{e^t - 1} = 1 - \sum_{k=1}^{\infty} \psi_k(x)(-t)^k.$$

Here, $x_1$ is fractional part of $x$. The Bernoulli polynomials $\psi_k(x)$ are expressed using the Fourier expansions as

$$(-1)^{\frac{k+1}{2}} 2 \sum_{n=1}^{\infty} \frac{\sin(2\pi nx)}{(2\pi n)^k} \quad (k : \text{oddnumber})$$

and

$$(-1)^{\frac{k}{2}} 2 \sum_{n=1}^{\infty} \frac{\cos(2\pi nx)}{(2\pi n)^k} \quad (k : \text{evennumber}).$$

This implies that these absolute values are of the order $(2\pi)^{-k}$.

(3) Removal of Singular point
Since the Hurwitz zeta function has a pole at the point $s = 1$, it is not able to obtain the value of the function when $s$ is equal to 1. But, as $\zeta(s,a) - 1/(s-1)$ can be defined as a regular function, as far as $|s-1| < 10^{-4}$( for single precision ) or $|s-1| < 10^{-8}$ ( for double precision ), we replace $Z(s,a)$ in the expansion formula of the Hurwitz zeta function $\zeta(s,a)$ with:

$$-\log(N+a)(1 - \frac{1}{2}\log(N+a)(s-1)(1 - \frac{1}{3}\log(N+a)(s-1)))$$

which is an approximation formula for

$$\frac{(N+a)^{-s+1} - 1}{s-1}$$

whose error is less than $10^{-9}$( single precision ) $10^{-18}$ ( double precision ). Here

$$\lim_{s\to 1}(\zeta(s,a)-\frac{1}{s-1})=-\frac{\Gamma'(a)}{\Gamma(a)}.$$

#### 2.1.2.21 The functions related to the error function

(1) The error function for complex values $e^{-z^2}\,\mathrm{Erfc}(-iz)$

The error function for complex values $e^{-z^2}\,\mathrm{Erfc}(-iz)$ can be obtained from the value of definite integral

$$e^{-z^2}\int_0^z e^{w^2}dw.$$

(a) When the absolute value of $\Im(z)$ is small and less than $10^{-4}$, we apply the following formula to avoid the small denominator due to some special values of $\Re(z)$ ( also including the case $\Re(z)=0$ )

$$f(z)=f_0+f_1(z-z_0)+f_2(z-z_0)^2/2+f_3(z-z_0)^3/6,$$

where $f(z)=e^{-z^2}+\frac{2i}{\sqrt{\pi}}D_w(z)(D_w(z)$ is the Dawson integral) and

$$f_0=f(z_0),f_1=-2z_0f_0+\frac{2i}{\sqrt{\pi}},f_2=-2z_0f_1-2f_0,f_3=-2z_0f_2-4f_1,z_0=\Re(z).$$

(b) Assume that $z$ is not real and set $z=\beta=R+iI(R$ and $I$ are real and $I>0)$. If the imaginary part is negative, this problem is reduced to the case when the imaginary part is positive using the fact that the relation

$$F(z)+F(-z)=2e^{-z^2}$$

holds for $F(z)=e^{-z^2}\,\mathrm{Erfc}(-iz)$.

(c) Suppose that $z=\beta=R+iI(R,I$ are real and $I>0)$. Set

$$f(x)=\int_{-\infty}^{\infty}e^{-xt^2}/(t+\beta)dt=\int_0^{\infty}2\beta e^{-xt^2}/(\beta^2-t^2)dt(x\geq 0).$$

For $x>0$, since

$$(e^{\beta^2x}f(x))'=(\sqrt{\pi}\beta/\sqrt{x})e^{\beta^2x},$$

it holds that

$$e^{\beta^2}f(1)-f(0)=\sqrt{\pi}\beta\int_0^1\frac{e^{\beta^2x}}{\sqrt{x}}dx.$$

Therefore,

$$\int_0^{\beta}e^{w^2}dw=\frac{1}{2\sqrt{\pi}}(e^{\beta^2}f(1)-f(0)).$$

Furthermore, in $f(0)=\int_0^{\infty}2\beta/(\beta^2-t^2)dt$, changing the integration pass $[0,\infty)$ to $[0,i\beta\infty)$ and considering the residual that is added with this change, it follows that

$$e^{-\beta^2}\int_0^{\beta}e^{w^2}dw=\frac{1}{2\sqrt{\pi}}(\int_{-\infty}^{\infty}e^{-t^2}/(t+\beta)dt+\pi ie^{-\beta^2}).$$

Now we apply the Poisson formula. Setting

$$g(x)=h\sum_{n=-\infty}^{\infty}\frac{e^{-h^2(n+x)^2}}{h(n+x)+\beta},$$

$g(x)$ is a periodic function of the period 1 and can be expanded by the Fourier series. Evaluate each coefficient as

$$\int_0^1 g(x)e^{-2\pi inx}dx = \int_{-\infty}^{\infty} h\frac{e^{-h^2x^2}}{hx+\beta}e^{-2\pi inx}dx$$

$$= \int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta}e^{-2\pi inx/h}dx$$

$$= \int_{-\infty}^{\infty} \frac{e^{-(x+\pi in/h)^2}}{x+\beta}dxe^{-\pi^2n^2/h^2},$$

here, moving the integration pass and considering the residuals if they are added, we get

$$= e^{-\pi^2n^2/h^2}\int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta-\pi in/h}dx(\pi n/h < I)$$

$$= e^{-\pi^2n^2/h^2}(\int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta-\pi in/h}dx - 2\pi ie^{-(-\beta+\pi in/h)^2})(\pi n/h > I).$$

Therefore, taking $h$ such that $\pi/h > \sqrt{\log(10^{16})}$ and neglecting the terms of the order $10^{-16}$, it is obtained that:

$$h\sum_{n=-\infty}^{\infty}\frac{e^{-h^2n^2}}{hn+\beta} = \int_{-\infty}^{\infty}\frac{e^{-x^2}}{x+\beta}dx - 2\pi i\sum_{\pi n/h>I}\exp(2\pi in\beta/h-\beta^2).$$

The infinite series on the both sides of this equation converge in a few terms. $e^{-z^2}\,\mathrm{Erfc}(-iz)$ becomes:

$$\frac{i}{\pi}\int_{-\infty}^{\infty}\frac{e^{-x^2}}{x+\beta}dx$$

with $z = \beta$.

(2) The inverse function of the co-error function

The co-error function is defined as follows:

$$y = \mathrm{Erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-t^2}dt$$

Here, $y$ is a non-negative real number.

To obtain values of its inverse function $y = \mathrm{Erfc}^{-1}(x)$, calculate $y$ satisfying

$$f(y) = (\frac{2}{\sqrt{\pi}}\int_y^{\infty} e^{-t^2}dt - x)e^{y^2} = 0$$

using Newton method first. The derivative of $f(y)$ is given by $f'(y) = 2yf(y) - 2/\sqrt{\pi}$. Classifying by the range of $x$, the initial value $y_{init}$ for $y$ is taken as follows:

(a) When $x \geq \mathrm{Erfc}(2)$

Set $y_{init} = 0$.

(b) When $x < \mathrm{Erfc}(2)$

For a given $x$, the initial value $y_{init}$ is calculated using the formula below:

$$y_1 = 1/(\sqrt{\pi}x), y_2 = \sqrt{\log y_1}, y_3 = 1/(\sqrt{\pi}xy_2), y_{init} = \sqrt{\log y_3}$$

The initial value $y_{init}$ obtained by this procedure is an approximation of $y$ which satisfies:

$$\frac{\sqrt{\pi}}{2}x = e^{-y^2}\frac{1}{2y}(1+O(\frac{1}{y^2}))$$

(3) The error function and the co-error function

(a) For $|x| \leq 0.476563$

i. Co-error function is obtained as $\mathrm{Erfc}(x) = 1 - \mathrm{Erf}(x)$.

ii. The error function $\mathrm{Erf}(x)$ is obtained as follows:

$$\mathrm{Erf}(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \cfrac{2x}{1 - \cfrac{2x^2}{3 + \cfrac{4x^2}{5 - \cfrac{6x^2}{7 + \cfrac{8x^2}{9 - \cdots}}}}}$$

(b) For $|x| > 0.476563$

i. Co-error function $\mathrm{Erfc}(x)$ is obtained as follows:
For $0 \leq x \leq 8.0$, using the best-fit formula.
For $8.0 < x \leq 26.628736$,

$$\mathrm{Erfc}(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \cfrac{1.0}{x + \cfrac{0.5}{x + \cfrac{1.0}{x + \cfrac{1.5}{x + \cdots}}}}$$

and if $x > 26.628736$ ,then Erfc($x$)=0.
For negative value of $x$, $\mathrm{Erfc}(x) = 2 - \mathrm{Erfc}(-x)$ is applied.
Then error function is obtained as $\mathrm{Erf}(x) = 1 - \mathrm{Erfc}(x)$.

#### 2.1.2.22 Coefficient Calculation Method

The subroutines in this library adopt arithmetic techniques to save the amount of calculation without degrading the accuracy.

#### 2.1.2.23 Method of Calculating Related Special Functions

(1) Incomplete elliptic integral of the 3rd kind

$$\begin{aligned}
\Pi(\varphi; c, m) &= \int_0^{\varphi} \frac{d\theta}{(1 + c\sin^2\theta)\sqrt{1 - m\sin^2\theta}} \\
&= \int_0^{x} \frac{dt}{(1 + c\,t^2)\sqrt{(1 - t^2)(1 - mt^2)}} \qquad (x = \sin\varphi) \\
&= \int_0^{u} \frac{dw}{1 + c \cdot \mathrm{sn}^2\, w} \qquad (x = \mathrm{sn}\ u)
\end{aligned}$$

If this integral is represented using the $\Pi$ function, it appears as follows:

$$\Pi(\varphi; c, m) = u + \frac{\mathrm{sn}\,\alpha}{\mathrm{cn}\,\alpha \cdot \mathrm{dn}\,\alpha} \cdot \Pi(u, \alpha)$$

($\alpha$ is set so that $\mathrm{sn}^2\,\alpha = -\frac{c}{m}$.)
($u = F(x, m) = F(\sin\varphi, m)$)
Another method is described below.

- $-m < c < 0$:

$$\begin{aligned}
\epsilon &= \sqrt{-\frac{c}{m}} \\
\beta &= \frac{\pi}{2} \cdot F(\epsilon, m)/K(m) \\
v &= \frac{\pi}{2} \cdot F(x, m)/K(m) \qquad (x = \sin\varphi) \\
\delta_1 &= \sqrt{\frac{-c}{(1+c)(m+c)}}
\end{aligned}$$

$$\Pi(\varphi; c, m) = \delta_1 \left[ -\frac{1}{2}\log\frac{\vartheta_4(v+\beta)}{\vartheta_4(v-\beta)} + v \cdot \frac{\vartheta_1'(\beta)}{\vartheta_1(\beta)} \right]$$

Assume that the nome $q$ for modulus $m$. Then the above function can be calculated using the following expressions.

$$\begin{aligned}
\frac{1}{2}\log\frac{\vartheta_4(v+\beta)}{\vartheta_4(v-\beta)} &= 2\sum_{s=1}^{\infty} \frac{q^s}{s \cdot (1-q^{2s})} \cdot \sin(2 \cdot s \cdot v) \cdot \sin(2 \cdot s \cdot \beta) \\
\frac{\vartheta_1'(\beta)}{\vartheta_1(\beta)} &= \cot\beta + 4\sum_{s=1}^{\infty} \frac{q^{2s}}{1 - 2 \cdot q^{2s} \cdot \cos(2 \cdot \beta) + q^{4s}} \cdot \sin(2 \cdot \beta)
\end{aligned}$$

The summation calculation are continued until the last term is sufficiently small relative to the first term.

- $c < -1$:

Assume $N = \frac{m}{c}, p_1 = \sqrt{(-c-1)(1+\frac{m}{c})}$. Then the function is calculated using the following expressions.

$$\Pi(\varphi; c, m) = -\pi(\varphi; N, m) + F(x, m) + \frac{1}{2p_1}\log[\frac{\Delta(\varphi) + p_1\tan\varphi}{\Delta(\varphi) - p_1\tan\varphi}]$$

where, $\Delta(\varphi) = \sqrt{1 - m\sin^2\varphi}$.

$\Pi(\varphi; N, m)$ is obtained using the calculations described above for $-m < c < 0$.

- $-1 < c < -m$:

$$\begin{aligned}
\epsilon &= \sqrt{\frac{1+c}{1-m}} \\
\beta &= \frac{\pi}{2} \cdot F(\epsilon, 1-m)/K(m) \\
v &= \frac{\pi}{2} \cdot F(x, m)/K(m) \\
\delta_2 &= \sqrt{\frac{c}{(1+c)(m+c)}} \\
\lambda &= a \cdot \tan(\tanh\beta \cdot \tan v) + 2\sum_{s=1}^{\infty} \frac{(-1)^{s-1} \cdot q^{2s}}{s \cdot (1-q^{2s})} \cdot \sin(2 \cdot s \cdot v) \cdot \sinh(2 \cdot s \cdot \beta) \\
\mu &= \frac{\displaystyle\sum_{s=1}^{\infty} s q^{s^2} \sinh(2 \cdot s \cdot \beta)}{1 + 2\sum_{s=1}^{\infty} q^{s^2}\cosh(2 \cdot s \cdot \beta)}
\end{aligned}$$

$$\Pi(\varphi; c, m) = \delta_2 \cdot (\lambda - 4 \cdot \mu \cdot v)$$

- $c > 0.0$:

$$\begin{aligned}
N &= -\frac{m+c}{1+c} \\
p_2 &= \sqrt{\frac{c \cdot (m+c)}{1+c}}
\end{aligned}$$

$$
\begin{aligned}
\Pi(\varphi;c,m) &= \left\{\sqrt{(1+N)\cdot\left(1+\frac{m}{N}\right)}\cdot\Pi(\varphi;N,m)+\frac{m\cdot F(x,m)}{p_2}\right.\\
&\left.\quad + a\tan(\frac{1}{2}p_2\cdot\sin(\frac{2\cdot\varphi}{\Delta(\varphi)}))\right\}/\sqrt{(1+c)(1+\frac{m}{c})}
\end{aligned}
$$

Where, $\Delta(\varphi)=\sqrt{1-m\sin^2\varphi}$.

$\Pi(\varphi;N,m)$ is obtained using the calculations described above for $-m<c<0.0$.

(Refer to bibliography reference (1).)

(2) Heuman's lambda function

$$
\Lambda_0(\varphi\backslash\alpha)=\frac{2}{\pi}\{K(\alpha)E(\varphi\backslash 90^\circ-\alpha)-(K(\alpha)-E(\alpha))\cdot F(\varphi\backslash 90^\circ-\alpha)\}
$$

Let $m=\sin^2\alpha$. Then lambda function is obtained from the following equations.

$$
\begin{aligned}
K(\alpha) &= K(m)\\
E(\varphi\backslash 90^\circ-\alpha) &= E(\sin\varphi,1-m)\\
E(\alpha) &= E(m)\\
F(\varphi\backslash 90^\circ-\alpha) &= F(\sin\varphi,1-m)
\end{aligned}
$$

(Refer to bibliography reference (1).)

(3) Legendre function

- $x>1.0$:

$$
\begin{aligned}
P_{-1/2}(x) &= \frac{2}{\pi}\sqrt{\frac{2}{x+1}}K\left(\frac{x-1}{x+1}\right)\\
P_{1/2}(x) &= \frac{2}{\pi}\sqrt{x-\sqrt{x^2-1}}E\left(\frac{2\sqrt{x^2-1}}{x+\sqrt{x^2-1}}\right)\\
Q_{-1/2}(x) &= \sqrt{\frac{2}{x+1}}K\left(\frac{2}{x+1}\right)\\
Q_{1/2}(x) &= x\sqrt{\frac{2}{x+1}}K\left(\frac{2}{x+1}\right)-\sqrt{2(x+1)}E\left(\frac{2}{x+1}\right)
\end{aligned}
$$

- $x\le 1.0$:

$$
\begin{aligned}
P_{-1/2}(x) &= \frac{2}{\pi}K\left(\frac{1-x}{2}\right)\\
P_{1/2}(x) &= \frac{2}{\pi}\left\{2E\left(\frac{1-x}{2}\right)-K\left(\frac{1-x}{2}\right)\right\}\\
Q_{-1/2}(x) &= K\left(\frac{1+x}{2}\right)\\
Q_{1/2}(x) &= K\left(\frac{1+x}{2}\right)-2E\left(\frac{1+x}{2}\right)
\end{aligned}
$$

($E$ and $K$ are complete elliptic integrals)

### 2.1.3 Reference Bibliography

(1) Abramowitz, M. and Stegur, I. A. , eds. , "Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables", Dover Publications, Inc. (1965).

(2) Hart, J. F. , ed. , "Computer approximation", John Wiley and Sons (1968).

(3) Cody W. J. , Strecok A. J. and Thacher H. C. , "Chebyshev approximations for the psi function", Math. Comp. , Vol.27, No.121, pp.123–127 (1973).

(4) Cody W. J. and Thacher H. C. , "Rational Chebyshev approximations for the exponential integral $E_1(x)$", Math. Comp. , Vol.22, pp.641–649 (1968).

(5) Cody W. J. and Thacher H. C. , "Chebyshev approximations for the exponential integral $E_i(x)$", Math. Comp. , Vol.23, pp.289–303 (1969).

(6) Cody W. J. , Paciorek K. A. and Thacher H. C. Jr. , "Chebyshev approximations for Dawson's integral", Math. Comp. , Vol.24, pp.171–178 (1970).

(7) Luke Y. L. , "The special functions and their approximations, II", Academic Press, (1969).

## 2.2 BESSEL FUNCTIONS

### 2.2.1 WIBJ0X, VIBJ0X
### Bessel Function of the 1st Kind (Order 0)

(1) **Function**

For $x = X_i$, calculates values of the Bessel function of the 1st kind (order 0)

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t))dt$$

(2) **Usage**

Double precision:
  CALL WIBJ0X (NV, XI, XO, IERR)
Single precision:
  CALL VIBJ0X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $J_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $|\text{XI(i)}| \leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) Bessel function of the 1st kind $J_\nu(z)$ is the basic solution of Bessel's differential equation

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}.$$

(b) Bessel function of the 1st kind is also called cylindrical function of the 1st kind.

(7) **Example**

(a) Problem

Obtain $J_0(x)$ for $x = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIBJ0X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBJ0X', CFNC='    J0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBJ0X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBJ0X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    J0(  0.000000)=  1.000000
    J0(  0.100000)=  0.997502
    J0(  0.200000)=  0.990025
    J0(  0.300000)=  0.977626
    J0(  0.400000)=  0.960398
    J0(  0.500000)=  0.938470
    J0(  0.600000)=  0.912005
    J0(  0.700000)=  0.881201
    J0(  0.800000)=  0.846287
    J0(  0.900000)=  0.807524
```

### 2.2.2 WIBY0X, VIBY0X Bessel Function of the 2nd Kind (Order 0)

(1) **Function**

For $x = X_i$, calculates values of the Bessel function of the 2nd kind (order 0)

$$Y_0(x) = -\frac{2}{\pi}\int_0^\infty \cos(x\cosh(t))dt \quad (x > 0.0)$$

(2) **Usage**

Double precision:
 CALL WIBY0X (NV, XI, XO, IERR)
Single precision:
 CALL VIBY0X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\begin{cases} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{cases}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Input | $X_i$ |
| 3 | XO | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Output | $Y_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\geq 0.0$

(c) XI(i) $\leq M$
 where $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI(i) = 0.0 (overflow) | XO(i) = (Minimum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by XI(i). | |

(6) **Notes**

(a) Bessel function of the 2nd kind $Y_\nu(z)$ is the basic solution of Bessel's differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

and defined as

$$Y_\nu(z)=\frac{J_\nu(z)\cos\nu\pi-J_{-\nu}(z)}{\sin\nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$Y_n(z)=\lim_{\nu\to n}Y_\nu(z)$$

(b) Bessel function of the 2nd kind is also called cylindrical function of the 2nd kind.

(c) The Neumann function $N_\nu(z)$ is the same as the Bessel function of the 2nd kind $Y_\nu(z)$.

(7) **Example**

(a) Problem

Obtain $Y_0(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIBY0X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBY0X', CFNC='    Y0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBY0X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBY0X *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
    Y0(  0.100000)= -1.534239
    Y0(  0.200000)= -1.081105
    Y0(  0.300000)= -0.807274
    Y0(  0.400000)= -0.606025
    Y0(  0.500000)= -0.444519
    Y0(  0.600000)= -0.308510
```

```
Y0(  0.700000)= -0.190665
Y0(  0.800000)= -0.086802
Y0(  0.900000)=  0.005628
Y0(  1.000000)=  0.088257
```

### 2.2.3 WIBJ1X, VIBJ1X Bessel Function of the 1st Kind (Order 1)

(1) **Function**

For $x = X_i$, calculates values of the Bessel function of the 1st kind (order 1)

$$J_1(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t) - t)dt.$$

(2) **Usage**

Double precision:
  CALL WIBJ1X (NV, XI, XO, IERR)
Single precision:
  CALL VIBJ1X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $J_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $|\text{XI(i)}| \leq M$
with $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) Bessel function of the 1st kind $J_\nu(z)$ is the basic solution of Bessel's differential equation

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}.$$

(b) Bessel function of the 1st kind is also called cylindrical function of the 1st kind.

(7) **Example**

(a) Problem

Obtain $J_1(x)$ for $x = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIBJ1X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBJ1X', CFNC='    J1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBJ1X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBJ1X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    J1(  0.000000)=  0.000000
    J1(  0.100000)=  0.049938
    J1(  0.200000)=  0.099501
    J1(  0.300000)=  0.148319
    J1(  0.400000)=  0.196027
    J1(  0.500000)=  0.242268
    J1(  0.600000)=  0.286701
    J1(  0.700000)=  0.328996
    J1(  0.800000)=  0.368842
    J1(  0.900000)=  0.405950
```

## 2.2.4 WIBY1X, VIBY1X Bessel Function of the 2nd Kind (Order 1)

(1) **Function**

For $x = X_i$, calculates values of the Bessel function of the 2nd kind (order 1)

$$Y_1(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin(t) - t)dt - \frac{1}{\pi}\int_0^{\infty} e^{-x\sinh(t)}[e^t - e^{-t}]dt.$$

(2) **Usage**

Double precision:
CALL WIBY1X (NV, XI, XO, IERR)
Single precision:
CALL VIBY1X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex
R:Single precision real  C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $Y_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\geq 0.0$

(c) XI(i) $\leq M$
where $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000+$i$ | XI(i) $\leq$ 1.0/(Maximum value) (overflow) | XO(i) = (Minimum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by XI(i). | |

(6) **Notes**

(a) Bessel function of the 2nd kind $Y_\nu(z)$ is the basic solution of Bessel's differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

and defined as

$$Y_\nu(z)=\frac{J_\nu(z)\cos\nu\pi-J_{-\nu}(z)}{\sin\nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$Y_n(z)=\lim_{\nu\to n}Y_\nu(z)$$

(b) Bessel function of the 2nd kind is also called cylindrical function of the 2nd kind.

(c) The Neumann function $N_\nu(z)$ is the same as the Bessel function of the 2nd kind $Y_\nu(z)$.

(7) **Example**

(a) Problem

Obtain $Y_1(x)$ for $x=0.1,0.2,\cdots,1.0$.

(b) Main program

```
      PROGRAM EIBY1X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBY1X', CFNC='    Y1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBY1X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBY1X *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
    Y1(  0.100000)= -6.458951
    Y1(  0.200000)= -3.323825
    Y1(  0.300000)= -2.293105
    Y1(  0.400000)= -1.780872
    Y1(  0.500000)= -1.471472
    Y1(  0.600000)= -1.260391
    Y1(  0.700000)= -1.103250
    Y1(  0.800000)= -0.978144
    Y1(  0.900000)= -0.873127
    Y1(  1.000000)= -0.781213
```

### 2.2.5 DIBJNX, RIBJNX Bessel Function of the 1st Kind (Integer Order)

(1) **Function**

Calculates a value of the Bessel function of the 1st kind (integer order)

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t) - nt)dt.$$

(2) **Usage**

Double precision:
  CALL DIBJNX (N, XI, XO, IERR)
Single precision:
  CALL RIBJNX (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Output | Value of $J_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{XI}| \leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\|n\|(\log_e\|\frac{n}{x}\| - M_1) > M_2$ (See Note (c)) (XI $\neq$ 0.0 and N $\neq$ 0) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $J_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $|\mathrm{XI}| < 1000.0$.

(b) To calculate $J_n(x), J_{n+1}(x), J_{n+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation:

$$J_{n-1} = \frac{2n}{x} J_n(x) - J_{n+1}(x)$$

(c) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =${double precision: 709.7827, single precision: 88.72284}

(d) Bessel function of the 1st kind $J_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}.$$

(e) Bessel function of the 1st kind is also called cylindrical function of the 1st kind.

(7) **Example**

(a) Problem
Obtain the value of $J_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBJNX
! *** EXAMPLE OF DIBJNX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBJNX(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBJNX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF JN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
 *** DIBJNX ***

  ** INPUT **

    N =  5     XI =   1.50

  ** OUTPUT**

    IERR =     0

    VALUE OF JN(X)

      XO =   0.1799421767D-02
```

## 2.2.6 DIBYNX, RIBYNX
## Bessel Function of the 2nd Kind (Integer Order)

(1) **Function**

Calculates a value of the Bessel function of the 2nd kind (integer order)

$$Y_n(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin(t) - nt)dt - \frac{1}{\pi}\int_0^{\infty} e^{-x\sinh(t)}[e^{nt} + (-1)^n e^{-nt}]dt.$$

(2) **Usage**

Double precision:
CALL DIBYNX (N, XI, XO, IERR)
Single precision:
CALL RIBYNX (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $Y_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq 0.0$

(b) XI $\leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $\leq 2.0/$(Maximum value) or $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (See Note (c)) (XI $\neq 0.0$ and N $\neq 0$) (overflow) | If N $\geq 0$, XO = (Minimum value) is performed. If N $< 0$, XO = (Minimum value)$\times(-1)^{\text{N}}$ is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $Y_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $\mathrm{XI} < 1000.0$.

(b) To calculate $Y_n(x), Y_{n+1}(x), Y_{n+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly.
Recurrence relation:

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

(c) When IERR becomes 2000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Bessel function of the 2nd kind $Y_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} + (z^2 - \nu^2) w = 0$$

and defined as

$$Y_\nu(z) = \frac{J_\nu(z) \cos \nu\pi - J_{-\nu}(z)}{\sin \nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$Y_n(z) = \lim_{\nu \to n} Y_\nu(z)$$

(e) Bessel function of the 2nd kind is also called cylindrical function of the 2nd kind.

(f) The Neumann function $N_\nu(z)$ is the same as the Bessel function of the 2nd kind $Y_\nu(z)$.

(7) **Example**

(a) Problem
Obtain the value of $Y_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
$\mathrm{N} = 5$ and $\mathrm{XI} = 1.5$.

(c) Main program

```
      PROGRAM BIBYNX
! *** EXAMPLE OF DIBYNX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBYNX(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBYNX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF YN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
*** DIBYNX ***
 ** INPUT **
   N =  5     XI =   1.50

 ** OUTPUT**
   IERR =     0
   VALUE OF YN(X)
     X0 =  -0.3719030840D+02
```

## 2.2.7 DIBJMX, RIBJMX
## Bessel Function of the 1st Kind (Real Number Order)

(1) **Function**

Calculates a value of the Bessel function of the 1st kind (real number order)

$$J_\nu(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin(t) - \nu t)dt - \frac{\sin(\pi\nu)}{\pi}\int_0^\infty e^{-x\sinh(t)}e^{-\nu t}dt.$$

(2) **Usage**

Double precision:

CALL DIBJMX (R, XI, XO, IERR)

Single precision:

CALL RIBJMX (R, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | R | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Input | Order $\nu$ |
| 2 | XI | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Output | Value of $J_\nu(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) When R corresponds with an integer:
$|\text{R}| \le M_1$
$|\text{XI}| \le M_2$

(b) When R does not correspond with an integer:
$0 < \text{R} \le M_1$
$0 < \text{XI} \le M_2$
where, $M_1$ ={double precision: $2^{31}$, single precision: $2^{31}$},
$M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ (See Note (e)) (XI $\neq$ 0.0 and R $\neq$ 0.0) (underflow)<br>(Note: When $\nu$ corresponds with an integer, $\lvert\nu\rvert$ and $\lvert x\rvert$ are used for judging.) | XO = 0.0 is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $J_\nu(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set R < 1000.0 and XI < 1000.0.

(b) If the order is half an integer (a half of an odd integer), the spherical Bessel function should be used instead.

$$J_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} j_n(x)$$

(c) If $\nu$ is negative and is not an integer, the Bessel function of the 1st kind cannot be calculated by using this subroutine. Therefore, it should be calculated by using a recurrence relation.

(d) To calculate $J_\nu(x), J_{\nu+1}(x), J\nu + 2(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $\nu$. Therefore the recurrence relation should be used with decreasing $\nu$.
Recurrence relation:

$$J_{\nu-1}(x) = \frac{2\nu}{x} J_\nu(x) - J_{\nu+1}(x)$$

(e) When IERR becomes 1000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =${double precision: 709.7827, single precision: 88.72284}

(f) Bessel function of the 1st kind $J_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}.$$

(g) Bessel function of the 1st kind is also called cylindrical function of the 1st kind.

(7) **Example**

(a) Problem

Obtain the value of $J_\nu(x)$ at $x = 1.5$ for $\nu = 3.3$.

(b) Input data

R = 3.3 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBJMX
! *** EXAMPLE OF DIBJMX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) R
      READ (5,*) XI
      WRITE(6,1000) R,XI
      CALL DIBJMX(R,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBJMX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'R =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF JM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBJMX ***

      ** INPUT **

        R =  3.30     XI =   1.50

      ** OUTPUT**

        IERR =     0

        VALUE OF JM(X)

          XO =   0.3827927999D-01
```

## 2.2.8 DIBYMX, RIBYMX
## Bessel Function of the 2nd Kind (Real Number Order)

(1) **Function**

Calculates a value of the Bessel function of the 2nd kind (real number order)

$$Y_\nu(x) = \frac{1}{\pi}\int_0^\pi \sin(x\sin(t) - \nu t)dt - \frac{1}{\pi}\int_0^\infty e^{-x\sinh(t)}[e^{\nu t} + \cos(\pi\nu)e^{-\nu t}]dt.$$

(2) **Usage**

Double precision:
CALL DIBYMX (R, XI, XO, IERR)
Single precision:
CALL RIBYMX (R, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | R | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Input | Order $\nu$ |
| 2 | XI | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 1 | Output | Value of $Y_\nu(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) When R corresponds with an integer,
$|\text{R}| \leq M_1$

(b) When R does not corresponds with an integer,
$0 < \text{R} \leq M_1$
where, $M_1$={double precision: $2^{31}$, single precision: $2^{31}$}

(c) XI $\geq 0.0$

(d) XI $\leq M_2$
where, $M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $\leq$ 2.0/(Maximum value) or $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ (See Note (e)) (XI $\neq$ 0.0 and R $\neq$ 0) (overflow) Note: When $\nu$ corresponds with an integer, $\|\nu\|$ is used for judging. | XO = (Minimum value) is performed. Note: If R $<$ 0, XO = $(-1)^{\mathrm{R}+1}$ $\times$ (Maximum value) is performed. |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The Bessel function of the 2nd kind $N_n(x)$ is the same as $Y_n(x)$.

(b) The computation time of $Yn(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set R $<$ 1000.0 and XI $<$ 1000.0.

(c) If the order is half an integer (a half of an odd integer), the spherical Bessel function should be used instead.

$$Y_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}}y_n(x)$$

(d) If $\nu$ is negative and is not an integer, the Bessel function of the 1st kind cannot be calculated by using this subroutine. Therefore, it should be calculated by using a recurrence relation.

$$Y_{\nu-1}(x) = \frac{2\nu}{x}Y_\nu(x) - Y_{\nu+1}(x)$$

(e) To calculate $Y_\nu(x), Y_{\nu+1}(x), Y\nu + 2(x), \cdots$ at a time, it is faster to successively use the recurrence relation than to call this subroutine repeatedly.

(f) When IERR becomes 2000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(g) Bessel function of the 2nd kind $Y_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$Y_n(z) = \lim_{\nu\to n} Y_\nu(z)$$

(h) Bessel function of the 2nd kind is also called cylindrical function of the 2nd kind.

(i) The Neumann function $N_\nu(z)$ is the same as the Bessel function of the 2nd kind $Y_\nu(z)$.

(7) **Example**

(a) Problem
Obtain the value of $Y_\nu(x)$ at $x = 1.5$ for $\nu = 3.3$.

(b) Input data

R = 3.3 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBYMX
! *** EXAMPLE OF DIBYMX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) R
      READ (5,*) XI
      WRITE(6,1000) R,XI
      CALL DIBYMX(R,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBYMX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'R =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF YM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBYMX ***

      ** INPUT **

        R =  3.30     XI =   1.50

      ** OUTPUT**

        IERR =     0

        VALUE OF YM(X)

          XO =  -0.2895226697D+01
```

### 2.2.9 ZIBJNZ, CIBJNZ
### Bessel Function of the 1st Kind with Complex Variable (Integer Order)

(1) **Function**

Calculates a value of the Bessel function of the 1st kind with complex variable (integer order)

$$J_n(z) = \frac{1}{\pi}\int_0^{\pi} \cos(z\sin(t) - nt)dt.$$

(2) **Usage**

Double precision:
 CALL ZIBJNZ (N, ZI, ZO, IERR)
Single precision:
 CALL CIBJNZ (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Output | Value of $J_n(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\Im(\text{ZI})| \leq M_1$
 where, $M_1$ ={double precision: 709.7827, single precision: 88.72284}

(b) $|\text{ZI}| \leq M_2$
 where, $M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\|n\|(\log_e \frac{\|n\|}{z} - M_3) > M_4$ (See Note (c)) ($\|\text{ZI}\| \neq 0.0$ and $\text{N} \neq 0$) (underflow) | $\text{ZO} = (0.0, 0.0)$ is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $J_n(z)$ becomes longer as $|z|$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $|\mathrm{ZI}| < 1000.0$.

(b) To calculate $J_n(z), J_{n+1}(z), J_{n+2}(z), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation:

$$J_{n-1}(z) = \frac{2n}{z} J_n(z) - J_{n+1}(z)$$

(c) When IERR becomes 1000 in this subroutine, the values of $M_3$ and $M_4$ are as follows: $M_3 = 0.3068$, $M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Bessel function of the 1st kind $J_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

and defined as

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}.$$

(e) Bessel function of the 1st kind is also called cylindrical function of the 1st kind.

(7) **Example**

(a) Problem
Obtain the value of $J_n(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
$\mathrm{N} = 3$ and $\mathrm{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBJNZ
! *** EXAMPLE OF ZIBJNZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBJNZ(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBJNZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF JN(Z)',/,/,10X,'ZO = (',D18.10,'  ,',D18.10 ,'  )' )
      END
```

(d) Output results

```
     *** ZIBJNZ ***

      ** INPUT **

        N =  3     ZI = (  1.00   ,  2.00   )

      ** OUTPUT**

        IERR =     0

        VALUE OF JN(Z)

          ZO = ( -0.2810396668D+00   ,  0.1717506200D-01   )
```

## 2.2.10 ZIBYNZ, CIBYNZ
## Bessel Function of the 2nd Kind with Complex Variable (Integer Order)

(1) **Function**

Calculates a value of the Bessel function of the 2nd kind with complex variable (integer order)

$$Y_n(z) = \frac{1}{\pi}\int_0^{\pi} \sin(z\sin(t) - nt)dt - \frac{1}{\pi}\int_0^{\infty} e^{-z\sinh(t)}[e^{nt} + (-1)^n e^{-nt}]dt.$$

(2) **Usage**

Double precision:
  CALL ZIBYNZ (N, ZI, ZO, IERR)
Single precision:
  CALL CIBYNZ (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Output | Value of $Y_n(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{ZI}| > 0.0$

(b) $|\Im(\text{ZI})| \leq M_1$
where, $M_1$ ={double precision: 709.7827, single precision: 88.72284}

(c) $|\text{ZI}| \leq M_2$
where, $M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000 | $|\text{ZI}| \leq 2.0/(\text{Maximum value})$<br>or $|n|(\log_e \frac{|n|}{|z|} - M_3) > M_4$ (See Note (d))<br>($|\text{ZI}| \neq 0.0$ and $\text{N} \neq 0$) | |

(6) **Notes**

(a) The Bessel function of the 2nd kind $N_n(z)$ is the same as $Y_n(z)$.

(b) The computation time of $Y_n(z)$ becomes longer as $|z|$ and $n$ increase. Generally it is desirable to set $|\text{N}| < 1000$ and $|\text{ZI}| < 1000.0$.

(c) To calculate $Y_n(z), Y_{n+1}(z), Y_{n+2}(z), \cdots$ at a time, it is faster to successively use the recurrence relation than to call this subroutine repeatedly.
Recurrence relation:

$$Y_{n+1}(z) = \frac{2n}{z} Y_n(z) - Y_{n-1}(z)$$

(d) When IERR becomes 4000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_4 = \{$double precision: 709.7827, single precision: 88.72284$\}$

(e) Bessel function of the 2nd kind $Y_\nu(z)$ is the basic solution of Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} + (z^2 - \nu^2) w = 0$$

and defined as

$$Y_\nu(z) = \frac{J_\nu(z) \cos \nu\pi - J_{-\nu}(z)}{\sin \nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$Y_n(z) = \lim_{\nu \to n} Y_\nu(z)$$

(f) Bessel function of the 2nd kind is also called cylindrical function of the 2nd kind.

(g) The Neumann function $N_\nu(z)$ is the same as the Bessel function of the 2nd kind $Y_\nu(z)$.

(7) **Example**

(a) Problem
Obtain the value of $Y_n(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
$\text{N} = 3$ and $\text{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBYNZ
! *** EXAMPLE OF ZIBYNZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBYNZ(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBYNZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF YN(Z)',/,/,10X,'ZO = (',D18.10,'  ,',D18.10,'  )' )
      END
```

(d) Output results

```
     *** ZIBYNZ ***

      ** INPUT **

        N =  3     ZI = (  1.00   ,  2.00  )

      ** OUTPUT**

        IERR =     0

        VALUE OF YN(Z)

          ZO = (  0.2901532942D+00   , -0.2121187705D+00   )
```

## 2.3 ZERO POINTS OF THE BESSEL FUNCTIONS

### 2.3.1 DIZBS0, RIZBS0
### Positive Zero Points of the Bessel Function of the 1st Kind (Order 0)

(1) **Function**

Obtain positive zero points of the Bessel function of the first kind of the order 0.

(2) **Usage**

Double precision:

CALL DIZBS0 (N, Z, IERR)

Single precision:

CALL RIZBS0 (N, Z, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Number of zero points |
| 2 | Z | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Output | Zero points (stored in ascending order) |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $1 \leq N \leq 50$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Set N=20 to get the positive zero points of $J_0(x)$ to 20-th one.

(b) Main program

```
      PROGRAM BIZBS0
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=20)
      REAL(8)   Z(N)
      CALL DIZBS0(N,Z,IERR)
      WRITE(6,60)
      WRITE(6,80)
      WRITE(6,90) N
      WRITE(6,100)
      WRITE(6,110) 0,IERR
      DO 1000 I=1,N
      CALL DIBJ0X(Z(I),Y,IERR)
      WRITE(6,6000) I,0 ,Z(I),Y
 1000 CONTINUE
      STOP
   60 FORMAT(1X,'*** DIZBS0 ***',/,/)
   80 FORMAT(1X,' *** INPUT *** ',/,/)
   90 FORMAT(1X,' N= ',I3,/,/)
  100 FORMAT(1X,' *** OUTPUT *** ',/,/)
  110 FORMAT(1X,'ORDER = ',I2,'  IERR = ',I4,/,/)
 6000 FORMAT(1X,I2,' TH ZERO OF J',I2,'  ',F13.10, ' ERR=',E10.3)
      END
```

(c) Output results

```
*** DIZBS0 ***

 *** INPUT ***

 N=  20

 *** OUTPUT ***

ORDER =  0  IERR =    0

 1 TH ZERO OF J 0   2.4048255577 ERR= 0.000E+00
 2 TH ZERO OF J 0   5.5200781103 ERR=-0.555E-16
 3 TH ZERO OF J 0   8.6537279129 ERR=-0.855E-16
 4 TH ZERO OF J 0  11.7915344390 ERR= 0.561E-15
 5 TH ZERO OF J 0  14.9309177085 ERR= 0.314E-16
 6 TH ZERO OF J 0  18.0710639679 ERR= 0.206E-16
 7 TH ZERO OF J 0  21.2116366299 ERR= 0.235E-15
 8 TH ZERO OF J 0  24.3524715307 ERR=-0.287E-15
 9 TH ZERO OF J 0  27.4934791320 ERR=-0.163E-15
10 TH ZERO OF J 0  30.6346064684 ERR=-0.461E-16
11 TH ZERO OF J 0  33.7758202136 ERR=-0.170E-15
12 TH ZERO OF J 0  36.9170983537 ERR=-0.458E-15
13 TH ZERO OF J 0  40.0584257646 ERR=-0.374E-15
14 TH ZERO OF J 0  43.1997917132 ERR= 0.223E-15
15 TH ZERO OF J 0  46.3411883717 ERR= 0.411E-15
16 TH ZERO OF J 0  49.4826098974 ERR= 0.353E-16
17 TH ZERO OF J 0  52.6240518411 ERR= 0.292E-15
18 TH ZERO OF J 0  55.7655107550 ERR=-0.231E-15
19 TH ZERO OF J 0  58.9069839261 ERR=-0.845E-16
20 TH ZERO OF J 0  62.0484691902 ERR=-0.862E-16
```

## 2.3.2 DIZBS1, RIZBS1
## Positive Zero Points of the Bessel Function of the 1st Kind (Order 1)

(1) **Function**

Evaluate positive zero points of the Bessel function of the first kind of order 1.

(2) **Usage**

Double precision:
CALL DIZBS1 (N, Z, IERR)
Single precision:
CALL RIZBS1 (N, Z, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Number of zero points |
| 2 | Z | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Output | Positive zero points (stored in ascending order) |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $1 \leq \text{N} \leq 50$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Set N=20 to get positive zero points of $J_1(x)$ to 20-th one.

(b) Main program

```
      PROGRAM BIZBS1
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=20)
      REAL(8)   Z(N)
      CALL DIZBS1(N,Z,IERR)
      WRITE(6,60)
      WRITE(6,80)
      WRITE(6,90) N
      WRITE(6,100)
      WRITE(6,110) 1,IERR
      DO 1000 I=1,N
      CALL DIBJ1X(Z(I),Y,IERR)
      WRITE(6,6000) I,1 ,Z(I),Y
 1000 CONTINUE
      STOP
   60 FORMAT(1X,'*** DIZBS1 ***',/,/)
   80 FORMAT(1X,' *** INPUT *** ',/,/)
   90 FORMAT(1X,' N= ',I3,/,/)
  100 FORMAT(1X,' *** OUTPUT *** ',/,/)
  110 FORMAT(1X,'ORDER = ',I2,'  IERR = ',I4,/,/)
 6000 FORMAT(1X,I2,' TH ZERO OF J',I2,'  ',F13.10, ' ERR=',E10.3)
      END
```

(c) Output results

```
*** DIZBS1 ***

 *** INPUT ***

 N=  20

 *** OUTPUT ***

ORDER =  1  IERR =    0

 1 TH ZERO OF J 1   3.8317059702 ERR= 0.000E+00
 2 TH ZERO OF J 1   7.0155866698 ERR= 0.555E-16
 3 TH ZERO OF J 1  10.1734681351 ERR= 0.892E-16
 4 TH ZERO OF J 1  13.3236919363 ERR= 0.159E-15
 5 TH ZERO OF J 1  16.4706300509 ERR= 0.186E-15
 6 TH ZERO OF J 1  19.6158585105 ERR=-0.281E-15
 7 TH ZERO OF J 1  22.7600843806 ERR=-0.247E-15
 8 TH ZERO OF J 1  25.9036720876 ERR= 0.777E-16
 9 TH ZERO OF J 1  29.0468285349 ERR=-0.187E-15
10 TH ZERO OF J 1  32.1896799110 ERR= 0.356E-15
11 TH ZERO OF J 1  35.3323075501 ERR= 0.384E-15
12 TH ZERO OF J 1  38.4747662348 ERR= 0.337E-16
13 TH ZERO OF J 1  41.6170942128 ERR=-0.511E-16
14 TH ZERO OF J 1  44.7593189977 ERR=-0.161E-15
15 TH ZERO OF J 1  47.9014608872 ERR= 0.306E-15
16 TH ZERO OF J 1  51.0435351836 ERR=-0.192E-15
17 TH ZERO OF J 1  54.1855536411 ERR= 0.132E-15
18 TH ZERO OF J 1  57.3275254379 ERR=-0.490E-16
19 TH ZERO OF J 1  60.4694578453 ERR=-0.125E-15
20 TH ZERO OF J 1  63.6113566985 ERR=-0.785E-16
```

### 2.3.3 DIZBSN, RIZBSN Positive Zero Points of Bessel Function of the 1st Kind (Integer Order)

(1) **Function**

Evaluate positive zero points of Bessel function of the first kind and integer order $J_m(x)$.

(2) **Usage**

Double precision:
CALL DIZBSN (N,M,LF, Z, WORK, IERR)
Single precision:
CALL RIZBSN (N,M,LF, Z, WORK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Number of positive zero points |
| 2 | M | I | 1 | Input | $m$ |
| 3 | LF | I | 1 | Input | Approximate magnification ratio |
| 4 | Z | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | N | Output | Positive zero points (stored in ascending order) |
| 5 | WORK | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | See Contents | Work | Work area<br>size:$2 \times (LF \times N+1) \times (LF \times N+2)$ |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $1 \leq N \leq 50$ ( M=-1,0,+1 ), $1 \leq N$ ( otherwise )

(b) $LF \geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3500 | The solution could not be improved. | |
| 3600 | The solution of eigenvalue problem was not obtained. | |

(6) **Notes**

(a) N should be about 50 at most.

(b) Maximum iteration count for iterative improvement is LF×N. This value is also used as the order of the eigenvalue problem which has to be solved to obtain an initial approximation value for iterative improvement. If this value is not sufficiently large, the precision for approximation of the initial value which is used in iterative improvement may become bad, which may cause IERR=3500,3600. On the other hand, if this value is too large, processing time required to calculate the initial approximation value which is used in iterative improvement becomes large. As a criterion, N×LF may be taken to be no less than 24 if M is around 10, and may be taken to be no less than 30 if M is around 18.

(c) For M=-1,0,+1, processing time becomes rather small because this subroutine refers to an numerical table.

(7) **Example**

(a) Problem
Set N=20 and M=10 to obtain positive zero points of $J_{10}(x)$ to 20-th one.

(b) Main program

```
      PROGRAM BIZBSN
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=20)
      REAL(8)   Z(N), WORK(2*(1+8*N)*(2+8*N))
      II=10
      CALL DIZBSN(N,II,8,Z,WORK,IERR)
      WRITE(6,10)
      WRITE(6,20)
      WRITE(6,30) N,II
      WRITE(6,40)
      WRITE(6,50) II,IERR
      DO 1000 I=1,N
      CALL DIBJNX(II, Z(I),    DERR, IERR)
      WRITE(6,6000) I,II ,Z(I),DERR
 1000 CONTINUE
      STOP
   10 FORMAT(1X, ' *** DIZBSN *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,' N= ',I3,' M= ',I3,/,/)
   40 FORMAT(1X,' *** OUTPUT *** ',/,/)
   50 FORMAT(1X,'ORDER = ',I3,'  IERR = ',I4,/,/)
 6000 FORMAT(1X,I2,' TH ZERO OF J',I2,'  ',F13.10,&
                 ' ERR = ',E13.3)
      END
```

(c) Output results

```
 *** DIZBSN ***

 *** INPUT ***

 N=  20 M=  10

 *** OUTPUT ***

ORDER =  10  IERR =    0

 1 TH ZERO OF J10  14.4755006866 ERR =    -0.167E-15
 2 TH ZERO OF J10  18.4334636670 ERR =     0.194E-15
 3 TH ZERO OF J10  22.0469853647 ERR =     0.194E-15
 4 TH ZERO OF J10  25.5094505542 ERR =    -0.264E-15
 5 TH ZERO OF J10  28.8873750635 ERR =     0.139E-16
 6 TH ZERO OF J10  32.2118561997 ERR =    -0.319E-15
 7 TH ZERO OF J10  35.4999092054 ERR =     0.444E-15
 8 TH ZERO OF J10  38.7618070179 ERR =     0.194E-15
 9 TH ZERO OF J10  42.0041902367 ERR =     0.555E-16
10 TH ZERO OF J10  45.2315741035 ERR =     0.250E-15
11 TH ZERO OF J10  48.4471513873 ERR =    -0.555E-16
12 TH ZERO OF J10  51.6532516682 ERR =     0.167E-15
13 TH ZERO OF J10  54.8516190760 ERR =    -0.160E-15
14 TH ZERO OF J10  58.0435879282 ERR =    -0.694E-16
15 TH ZERO OF J10  61.2301979773 ERR =    -0.340E-15
16 TH ZERO OF J10  64.4122724129 ERR =     0.354E-15
```

```
17 TH ZERO OF J10  67.5904720737 ERR =    -0.354E-15
18 TH ZERO OF J10  70.7653339962 ERR =    -0.607E-15
19 TH ZERO OF J10  73.9372993818 ERR =     0.461E-15
20 TH ZERO OF J10  77.1067342469 ERR =    -0.163E-15
```

## 2.3.4 DIZBYN, RIZBYN<br>Positive Zero Points of the Second Kind Bessel Function

(1) **Function**

Evaluate positive zero points of Bessel function of the second kind and integer order $Y_m(x)$.

(2) **Usage**

Double precision:

CALL DIZBYN (N, M, Z, NCONV, IERR)

Single precision:

CALL RIZBYN (N, M, Z, NCONV, 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 zero points |
| 2 | M | I | 1 | Input | Degree $m$ |
| 3 | Z | {D / R} | N | Output | Positive zeros (from smallest one) |
| 4 | NCONV | I | 1 | Output | Maximum number of iteration |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq$ 1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | The iterative improvement did not converge within the maximum number of iterations. | |

(6) **Notes**

(a) The double precision version should be used to get zero points for the second kind Bessel function with the absolute value of the degree $\geq 5$.

(7) **Example**

(a) Problem

For $Y_{10}(x)$, obtain zero points for 20-th one.

(b) Input data

N=20 and M=10.

(c) Main program

```
      PROGRAM BIZBYN
! *** EXAMPLE OF DIZBYN ***
      IMPLICIT NONE
!
      INTEGER N
      INTEGER M,NCONV,IERR,I
      PARAMETER( N = 20 )
      REAL(8) Z(N),DERR
!
      M = 10
!
      WRITE(6,6000) N,M
!
      CALL DIZBYN(N,M,Z,NCONV,IERR)
!
      WRITE(6,6010) IERR, NCONV, M
      DO 100 I=1,N
         CALL DIBYNX(M, Z(I), DERR, IERR)
         IF(IERR.NE.0) WRITE(6,6020)
         WRITE(6,6030) I,M ,Z(I),DERR
  100 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  DIZBYN  ***',/,/,&
             1X,' **  INPUT  ** ',/,/,&
             1X,'     N= ',I3,'     M= ',I3,/)
 6010 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/,/,&
             1X,'     NCONV = ',I4,/,/,&
             1X,'     NO  ','Y',I2,&
                '(  ZERO  POINT  )     VALUE OF YM(X)',/)
 6020 FORMAT(1X,'     ** ERROR IN DIBYNX',/)
 6030 FORMAT(1X,5X,I2,'  Y',I2,'( ',F13.10,' )   ',D13.3)
      END
```

(d) Output results

```
***  DIZBYN  ***

 **  INPUT  **

     N=  20     M=  10


 **  OUTPUT  **

     IERR =    0

     NCONV =   29

     NO  Y10(  ZERO  POINT  )     VALUE OF YM(X)

      1  Y10( 12.1289277044 )      -0.335D-11
      2  Y10( 16.5222843948 )      -0.468D-12
      3  Y10( 20.2659845012 )       0.695D-13
      4  Y10( 23.7916697195 )       0.145D-12
      5  Y10( 27.2065688816 )      -0.144D-13
      6  Y10( 30.5550200110 )       0.247D-13
      7  Y10( 33.8596838727 )      -0.322D-14
      8  Y10( 37.1336497603 )       0.178D-13
      9  Y10( 40.3851175938 )      -0.357D-14
     10  Y10( 43.6195330856 )       0.541D-15
     11  Y10( 46.8406766306 )       0.798D-15
     12  Y10( 50.0512658519 )       0.352D-14
     13  Y10( 53.2533105567 )       0.125D-15
     14  Y10( 56.4483324889 )      -0.101D-14
     15  Y10( 59.6375070056 )       0.194D-14
     16  Y10( 62.8217575527 )      -0.196D-14
     17  Y10( 66.0018204937 )       0.330D-14
     18  Y10( 69.1782907026 )      -0.251D-13
     19  Y10( 72.3516543205 )       0.351D-14
     20  Y10( 75.5223127385 )      -0.410D-14
```

## 2.3.5 DIZBSL, RIZBSL
## Positive Zero Points of the Function $aJ_0(\alpha) + \alpha J_1(\alpha)$

(1) **Function**

Evaluate the positive solutions $\alpha$ of the transcendental equation $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$

(2) **Usage**

Double precision:
CALL DIZBSL (N,A,LF, Z, WORK, IERR)
Single precision:
CALL RIZBSL (N,A,LF, Z, WORK, 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 | N | I | 1 | Input | Number of positive solutions |
| 2 | A | {D, R} | 1 | Input | $a$ |
| 3 | LF | I | 1 | Input | Approximate magnification ratio |
| 4 | Z | {D, R} | N | Output | Positive solutions $\alpha$ (stored in ascending order) |
| 5 | WORK | {D, R} | See Contents | Work | Work area<br>size:$2 \times$ (LF$\times$N+1) $\times$ (LF$\times$N+2) |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq$ 1

(b) LF $\geq$ 1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3500 | The solution could not be improved. | |
| 3600 | The solution of eigenvalue problem was not obtained. | |

(6) **Notes**

(a) The effective range for parameter $a$ (input value A) is $10^{-10} \leq |a| \leq 10^4$ or $a = 0$.

(b) N should be about 50 at most.

(c) Maximum iteration count for iterative improvement is LF×N.

(d) As a criterion, LF may be taken to be about 8.

(7) **Example**

(a) Problem

Set $a = -\beta \dfrac{J_1(\beta)}{J_0(\beta)} (\beta = 2.304780)$, N=20 and LF=8 to obtain the positive solutions $\alpha$ of $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$.

(b) Main program

```
      PROGRAM BIZBSL
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=20)
      REAL(8)   Z(N), WORK(2*(1+8*N)*(2+8*N))
!
      A=2.304780D0
      CALL DIBJOX(A,F,IERR)
      CALL DIBJ1X(A,D,IERR)
      A=-D*A/F
      WRITE(6,10)
      WRITE(6,20)
      WRITE(6,6000) A
      CALL DIZBSL(N,A,8,Z,WORK,IERR)
      WRITE(6,30)
      WRITE(6,40) IERR
      DO 1000 J=1,N
      CALL DIBJOX(Z(J),F,IERR)
      CALL DIBJ1X(Z(J),D,IERR)
      P=A*F+D*Z(J)
      WRITE(6,6100) J,Z(J),P
 1000 CONTINUE
      STOP
   10 FORMAT(1X,' *** DIZBSL *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,'*** OUTPUT *** ',/,/)
   40 FORMAT(1X,'  IERR = ',I4,/,/)
 6000 FORMAT(1X,' INPUT A = ',F10.6,/,/)
 6100 FORMAT(1X,'ALPHA(',I2,')= ',F10.6,'   ERR= ',E11.4)
      END
```

(c) Output results

```
 *** DIZBSL ***


 *** INPUT ***


 INPUT A = -23.456847


*** OUTPUT ***


  IERR =    0


ALPHA( 1)=   2.304780   ERR=  0.0000E+00
ALPHA( 2)=   5.293488   ERR=  0.1332E-14
ALPHA( 3)=   8.306597   ERR= -0.4885E-14
ALPHA( 4)=  11.333025   ERR= -0.1243E-13
ALPHA( 5)=  14.371644   ERR=  0.6217E-14
ALPHA( 6)=  17.421959   ERR= -0.5773E-14
ALPHA( 7)=  20.483189   ERR=  0.4441E-14
ALPHA( 8)=  23.554295   ERR=  0.1332E-14
ALPHA( 9)=  26.634127   ERR=  0.0000E+00
ALPHA(10)=  29.721552   ERR=  0.5773E-14
ALPHA(11)=  32.815519   ERR= -0.3553E-14
ALPHA(12)=  35.915098   ERR=  0.1421E-13
ALPHA(13)=  39.019482   ERR= -0.1865E-13
ALPHA(14)=  42.127984   ERR=  0.1510E-13
ALPHA(15)=  45.240020   ERR= -0.4441E-15
ALPHA(16)=  48.355101   ERR= -0.6661E-14
ALPHA(17)=  51.472814   ERR=  0.1554E-13
ALPHA(18)=  54.592810   ERR=  0.1732E-13
ALPHA(19)=  57.714796   ERR=  0.2309E-13
ALPHA(20)=  60.838523   ERR=  0.1199E-13
```

## 2.4 MODIFIED BESSEL FUNCTIONS

### 2.4.1 WIBI0X, VIBI0X
### Modified Bessel Function of the 1st Kind (Order 0)

(1) **Function**

For $x = X_i$, calculates values of the Modified Bessel function of the 1st kind (order 0)

$$I_0(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}dt.$$

(2) **Usage**

Double precision:
  CALL WIBI0X (NV, XI, XO, IERR)
Single precision:
  CALL VIBI0X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $I_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 2000+$i$ | $\|$XI(i)$\|$ > M (See Note (a)) (overflow) | XO(i) = (Maximum value) is performed. |

(6) **Notes**

(a) When IERR = 2000 in this subroutine, the value of $M$ is as follows:
$M$ ={double precision: 713.067, single precision: 90.978}

(b) Modified Bessel function of the 1st kind $I_\nu(z)$ is the particular solution of modified Bessel's differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0,$$

and defined as

$$I_{\pm\nu}(z)=e^{\mp\sqrt{-1}\pi/2}J_{\pm\nu}(\sqrt{-1}z).$$

(7) **Example**

(a) Problem

Obtain $I_0(x)$ for $x=0.0,0.1,\cdots,0.9$.

(b) Main program

```
      PROGRAM EIBI0X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBI0X', CFNC='    I0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBI0X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBI0X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    I0(  0.000000)=  1.000000
    I0(  0.100000)=  1.002502
    I0(  0.200000)=  1.010025
    I0(  0.300000)=  1.022627
    I0(  0.400000)=  1.040402
    I0(  0.500000)=  1.063483
    I0(  0.600000)=  1.092045
    I0(  0.700000)=  1.126303
    I0(  0.800000)=  1.166515
    I0(  0.900000)=  1.212985
```

## 2.4.2 WIBK0X, VIBK0X Modified Bessel Function of the 2nd Kind (Order 0)

(1) **Function**

For $x = X_i$, calculates values of the modified Bessel function of the 2nd kind (order 0)

$$K_0(x) = \int_0^\infty e^{-x\cosh(t)}dt.$$

(2) **Usage**

Double precision:

CALL WIBK0X (NV, XI, XO, IERR)

Single precision:

CALL VIBK0X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $K_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq$ 1

(b) XI(i) $\geq$ 0.0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI(i) > M (See Note (a)) (underflow) | XO(i) = 0.0 is performed. |
| 2000 | XI(i) = 0.0 (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+i | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the value of $M$ is as follows:
$M$ ={double precision: 705.117, single precision: 85.114}

(b) Modified Bessel function of the 2nd kind $K_\nu(z)$ is the particular solution of modified Bessel's differential equation

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2+\nu^2)w = 0$$

and defined as

$$K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z) - I_\nu(z)}{\sin\nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$K_n(z) = \lim_{\nu\to n} K_\nu(z)$$

(7) **Example**

(a) Problem
Obtain $K_0(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIBK0X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBK0X', CFNC='    K0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBK0X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBK0X *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
    K0(  0.100000)=  2.427069
    K0(  0.200000)=  1.752704
    K0(  0.300000)=  1.372460
    K0(  0.400000)=  1.114529
    K0(  0.500000)=  0.924419
```

```
K0(  0.600000)=  0.777522
K0(  0.700000)=  0.660520
K0(  0.800000)=  0.565347
K0(  0.900000)=  0.486730
K0(  1.000000)=  0.421024
```

### 2.4.3 WIBI1X, VIBI1X Modified Bessel Function of the 1st Kind (Order 1)

(1) **Function**

For $x = X_i$, calculates values of the modified Bessel function of the 1st kind (order 1)

$$I_1(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}\cos(t)dt.$$

(2) **Usage**

Double precision:
  CALL WIBI1X (NV, XI, XO, IERR)
Single precision:
  CALL VIBI1X (NV, XI, XO, IERR)

(3) **Arguments**

| | | |
|---|---|---|
| D:Double precision real | Z:Double precision complex | I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$ |
| R:Single precision real | C:Single precision complex | |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $I_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 2000+i | $\lvert$XI(i)$\rvert$ > M (See Note (a))<br>(overflow) | In the case where XI(i) $\geq$ 0.0,<br>XO(i) = (Maximum value) is performed.<br>In the case where XI(i) < 0.0,<br>XO(i) = $-$(Maximum value)<br>is performed. |

(6) **Notes**

(a) When IERR becomes 2000 in this subroutine, the value of $M$ is as follows:
$M$ ={double precision: 713.067, single precision: 90.978}

(b) Modified Bessel function of the 1st kind $I_\nu(z)$ is the particular solution of modified Bessel's differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0,$$

and defined as

$$I_{\pm\nu}(z)=e^{\mp\sqrt{-1}\pi/2}J_{\pm\nu}(\sqrt{-1}z).$$

(7) **Example**

(a) Problem
Obtain $I_1(x)$, for $x=0.0,0.1,0.2,\cdots,0.9$.

(b) Main program

```
      PROGRAM EIBI1X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBI1X', CFNC='    I1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBI1X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBI1X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    I1(  0.000000)=  0.000000
    I1(  0.100000)=  0.050063
    I1(  0.200000)=  0.100501
    I1(  0.300000)=  0.151694
    I1(  0.400000)=  0.204027
    I1(  0.500000)=  0.257894
    I1(  0.600000)=  0.313704
    I1(  0.700000)=  0.371880
    I1(  0.800000)=  0.432865
    I1(  0.900000)=  0.497126
```

## 2.4.4 WIBK1X, VIBK1X Modified Bessel Function of the 2nd Kind (Order 1)

(1) **Function**

For $x = X_i$, calculates values of the modified Bessel function of the 2nd kind (order 1)

$$K_1(x) = \int_0^\infty e^{-x\cosh(t)}\cosh(t)dt.$$

(2) **Usage**

Double precision:
CALL WIBK1X (NV, XI, XO, IERR)
Single precision:
CALL VIBK1X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Output | $K_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\geq 0.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI(i) > M (See Note (a)) (underflow) | XO(i) = 0.0 is performed. |
| 2000 | XI(i) $\leq$ 1.0/(Maximum value) (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+i | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the value of $M$ is as follows:
$M$ ={double precision: 705.117, single precision: 85.114}

(b) Modified Bessel function of the 2nd kind $K_\nu(z)$ is the particular solution

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2+\nu^2)w = 0,$$

and defined as

$$K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z) - I_\nu(z)}{\sin\nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$K_n(z) = \lim_{\nu\to n} K_\nu(z)$$

(7) **Example**

(a) Problem
Obtain $K_1(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIBK1X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBK1X', CFNC='    K1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBK1X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBK1X *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
    K1(  0.100000)=  9.853845
    K1(  0.200000)=  4.775973
    K1(  0.300000)=  3.055992
    K1(  0.400000)=  2.184354
    K1(  0.500000)=  1.656441
    K1(  0.600000)=  1.302835
    K1(  0.700000)=  1.050284
```

```
K1(  0.800000)=  0.861782
K1(  0.900000)=  0.716534
K1(  1.000000)=  0.601907
```

## 2.4.5 DIBINX, RIBINX
## Modified Bessel Function of the 1st Kind (Integer Order)

(1) **Function**

Calculates a value of the modified Bessel function of the 1st kind (integer order)

$$I_n(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}\cos(nt)dt.$$

(2) **Usage**

Double precision:
  CALL DIBINX (N, XI, XO, IERR)
Single precision:
  CALL RIBINX (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $I_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{XI}| \leq M$
where, $M =${double precision: 713.067, single precision: 90.978}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\|n\|(\log_e\|\frac{n}{x}\| - M_1) > M_2$ (See Note (c)) (XI $\neq$ 0.0 and N $\neq$ 0) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $I_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set |N| < 1000 and |XI| < 1000.0.

(b) To calculate $I_n(x), I_{n+1}(x), I_{n+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation :

$$I_{n-1} = \frac{2n}{x} I_n(x) + I_{n+1}(x)$$

(c) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =${double precision: 709.7827, single precision: 88.72284}

(d) Modified Bessel function of the 1st kind $I_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} - (z^2 + \nu^2) w = 0,$$

and defined as

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z).$$

(7) **Example**

(a) Problem
Obtain the value of $I_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBINX
! *** EXAMPLE OF DIBINX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBINX(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBINX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF IN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBINX ***

      ** INPUT **

        N =  5     XI =   1.50


      ** OUTPUT**

        IERR =     0

        VALUE OF IN(X)

          XO =   0.2170559569D-02
```

## 2.4.6 DIBKNX, RIBKNX Modified Bessel Function of the 2nd Kind (Integer Order)

(1) **Function**

Calculates a value of the modified Bessel function of the second kind (integer order)

$$K_n(x) = \int_0^\infty e^{-x\cosh(t)} \cosh(nt)dt.$$

(2) **Usage**

Double precision:
CALL DIBKNX (N, XI, XO, IERR)
Single precision:
CALL RIBKNX (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $K_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq 0.0$

(b) XI $\leq M$
where, $M$ ={double precision: 705.117, single precision: 85.114}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| | Normal termination. | |
| 2000 | XI $\leq$ 2.0/(Maximum value) or $\vert n\vert(\log_e \frac{\vert n\vert}{x} - M_1) > M_2$ (See Note (c)) (XI $\neq 0.0$ and N $\neq 0$) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $K_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $\mathrm{XI} < 1000.0$.

(b) To calculate $K_n(x), K_{n+1}(x), K_{n+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly.
Recurrence relation:

$$K_{n+1}(x) = \frac{2n}{x} K_n(x) + K_{n-1}(x)$$

(c) When IERR becomes 2000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Modified Bessel function of the 2nd kind $K_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} - (z^2 + \nu^2) w = 0,$$

and defined as

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$K_n(z) = \lim_{\nu \to n} K_\nu(z)$$

(7) **Example**

(a) Problem
Obtain the value of $K_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBKNX
! *** EXAMPLE OF DIBKNX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBKNX(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBKNX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF KN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBKNX ***

      ** INPUT **

        N =  5     XI =   1.50

      ** OUTPUT**

        IERR =     0

        VALUE OF KN(X)

          XO =   0.4406778116D+02
```

### 2.4.7 DIBIMX, RIBIMX Modified Bessel Function of the 1st Kind (Real Number Order)

(1) **Function**

Calculates a value of the modified Bessel function of the 1st kind (real number order)

$$I_\nu(x) = \frac{1}{\pi}\int_0^\pi e^{x\cos(t)}\cos(\nu t)dt - \frac{\sin(\pi\nu)}{\pi}\int_0^\infty e^{-x\cosh(t)-\nu t}dt.$$

(2) **Usage**

Double precision:
CALL DIBIMX (R, XI, XO, IERR)
Single precision:
CALL RIBIMX (R, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | R | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Order $\nu$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $I_\nu(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) When R corresponds with an integer:
$|\text{R}| \leq M_1$
$|\text{XI}| \leq M_2$

(b) When R does not correspond with an integer:
$0 < \text{R} \leq M_1$
$0 < \text{XI} \leq M_2$
where, $M_1 =$ {double precision: $2^{31}$, single precision: $2^{31}$},
$M_2 =$ {double precision: 713.067, single precision: 90.978}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ (See Note (e)) (XI $\neq$ 0.0 and R $\neq$ 0.0) (underflow) (Note: When $\nu$ corresponds with an integer, $\lvert\nu\rvert$ and $\lvert x\rvert$ are used for judging.) | XO = 0.0 is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $I_\nu(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set R < 1000.0 and XI < 1000.0.

(b) If the order is half an integer (a half of an odd integer), the spherical Bessel function should be used instead.

$$I_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} i_n(x)$$

(c) If $\nu$ is negative and is not an integer, the Bessel function of the 1st kind cannot be calculated by using this subroutine. Therefore, it should be calculated by using a recurrence relation.

(d) To calculate $I_\nu(x), I\nu + 1(x), I_{\nu+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $\nu$. Therefore the recurrence relation should be used with decreasing $\nu$.
Recurrence relation:

$$I_{\nu-1}(x) = \frac{2\nu}{x} I_\nu(x) + I_{\nu+1}(x)$$

(e) When IERR becomes 1000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(f) Modified Bessel function of the 1st kind $I_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} - (z^2 + \nu^2) w = 0,$$

and defined as

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z).$$

(7) **Example**

(a) Problem

Obtain the value of $I_\nu(x)$ at $x = 1.5$ for $\nu = 3.3$.

(b) Input data

R = 3.3 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBIMX
! *** EXAMPLE OF DIBIMX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) R
      READ (5,*) XI
      WRITE(6,1000) R,XI
      CALL DIBIMX(R,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBIMX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'R =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF IM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
    *** DIBIMX ***

     ** INPUT **

       R =  3.30     XI =   1.50

     ** OUTPUT**

       IERR =     0

       VALUE OF IM(X)

         XO =   0.4973088526D-01
```

### 2.4.8 DIBKMX, RIBKMX
### Modified Bessel Function of the 2nd Kind (Real Number Order)

(1) **Function**

Calculates a value of the modified Bessel function of the 2nd kind (real number order)

$$K_\nu(x) = \int_0^\infty e^{-x\cosh(t)} \cosh(\nu t) dt.$$

(2) **Usage**

Double precision:
CALL DIBKMX (R, XI, XO, IERR)
Single precision:
CALL RIBKMX (R, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | R | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Order $\nu$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of $K_\nu(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{R}| \leq M_1$
where, $M_1$={double precision: $2^{31}$, single precision: $2^{31}$}

(b) XI $\geq 0.0$

(c) XI $\leq M_2$
where, $M_2$ ={double precision: 705.117, single precision: 85.114}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $\leq$ 2.0/(Maximum value) or $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ (See Note (d)) (XI $\neq$ 0.0 and R $\neq$ 0.0) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $K_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set R $<$ 1000.0 and XI $<$ 1000.0.

(b) If the order is half an integer (a half of an odd integer), the spherical Bessel function should be used instead.

$$K_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} k_n(x)$$

(c) To calculate $K_\nu(x), K_{\nu+1}(x), K_{\nu+2}(x), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly.
Recurrence relation:

$$K_{\nu+1}(x) = \frac{2\nu}{x} K_\nu(x) + K_{\nu-1}(x)$$

(d) When IERR becomes 2000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(e) Modified Bessel function of the 2nd kind $K_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} - (z^2 + \nu^2) w = 0,$$

and defined as

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$K_n(z) = \lim_{\nu \to n} K_\nu(z)$$

(7) **Example**

(a) Problem

Obtain the value of $K_\nu(x)$ at $x = 1.5$ for $\nu = 3.3$.

(b) Input data

R = 3.3 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBKMX
! *** EXAMPLE OF DIBKMX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) R
      READ (5,*) XI
      WRITE(6,1000) R,XI
      CALL DIBKMX(R,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBKMX ***',/,/,6X,'** INPUT **',&
      /,/,8X,'R =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF KM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBKMX ***

      ** INPUT **

        R =  3.30     XI =   1.50


      ** OUTPUT**

        IERR =     0

        VALUE OF KM(X)

          XO =   0.2759863620D+01
```

### 2.4.9 ZIBINZ, CIBINZ Modified Bessel Function of the 1st Kind with Complex Variable (Integer Order)

(1) **Function**

Calculates a value of the modified Bessel function of the 1st kind with complex variable (integer order)

$$I_n(z) = \frac{1}{\pi}\int_0^{\pi} e^{z\cos(t)}\cos(nt)dt.$$

(2) **Usage**

Double precision:
  CALL ZIBINZ (N, ZI, ZO, IERR)
Single precision:
  CALL CIBINZ (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Output | Value of $I_n(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{ZI}| \leq M_1$
  where, $M_1$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(b) $|\Re(\text{ZI})| \leq M_2$
  where, $M_2$ ={double precision: 709.7827, single precision: 88.72284}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{\lvert z\rvert} - M_3) > M_4$ (See Note (c)) ($\lvert\text{ZI}\rvert \neq 0.0$ and $\text{N} \neq 0$) (underflow) | ZO = $(0.0, 0.0)$ is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $I_n(z)$ becomes longer as $|z|$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $|\mathrm{ZI}| < 1000.0$.

(b) To calculate $I_n(z), I_{n+1}(z), I_{n+2}(z), \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly. The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation:

$$I_{n-1}(z) = \frac{2n}{z} I_n(z) + I_{n+1}(z)$$

(c) When IERR becomes 1000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Modified Bessel function of the 1st kind $I_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0,$$

and defined as

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z).$$

(7) **Example**

(a) Problem
Obtain the value of $I_n(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
$\mathrm{N} = 3$ and $\mathrm{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBINZ
! *** EXAMPLE OF ZIBINZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBINZ(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBINZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF IN(Z)',/,/,10X,'ZO = (',D18.10,'  ,',D18.10,'  )' )
      END
```

(d) Output results

```
     *** ZIBINZ ***

      ** INPUT **

        N =  3     ZI = (  1.00   ,  2.00  )

      ** OUTPUT**

        IERR =     0

        VALUE OF IN(Z)

          ZO = ( -0.1753534440D+00  , -0.8243079895D-01  )
```

### 2.4.10 ZIBKNZ, CIBKNZ Modified Bessel Function of the 2nd Kind with Complex Variable (Integer Order)

(1) **Function**

Calculates a value of the modified Bessel function of the second kind with complex variable (integer order)

$$K_n(z) = \int_0^{\infty} e^{-z\cosh(t)} \cosh(nt)dt.$$

(2) **Usage**

Double precision:

CALL ZIBKNZ (N, ZI, ZO, IERR)

Single precision:

CALL CIBKNZ (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{matrix}\text{Z}\\ \text{C}\end{matrix}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{matrix}\text{Z}\\ \text{C}\end{matrix}\right\}$ | 1 | Output | Value of $K_n(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{ZI}| > 0.0$

(b) $|\text{ZI}| \leq M_1$
where, $M_1$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(c) $|\Re(\text{ZI})| \leq M_2$
where, $M_2$ ={double precision: 709.7827, single precision: 88.72284}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000 | $\lvert\mathrm{ZI}\rvert \leq 2.0/$(Maximum value) or $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{\lvert z\rvert} - M_3) > M_4$ (See Note (c)) ($\lvert\mathrm{ZI}\rvert \neq 0.0$ and $\mathrm{N} \neq 0$) | |

(6) **Notes**

(a) The computation time of $K_n(z)$ becomes longer as $|z|$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $|\mathrm{ZI}| < 1000.0$.

(b) To calculate $K_n(z), K_{n+1}(z), K_{n+2}(z), \cdots$ at a time, it is faster to successively use the recurrence relation given below than to call this subroutine repeatedly.
Recurrence relation:

$$K_{n+1}(z) = \frac{2n}{z} K_n(z) + K_{n-1}(z)$$

(c) When IERR becomes 4000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Modified Bessel function of the 2nd kind $K_\nu(z)$ is the particular solution of modified Bessel's differential equation:

$$z^2 \frac{d^2 w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0,$$

and defined as

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}.$$

When $\nu$ is equal to integer $n$, the following limiting value is used for definition.

$$K_n(z) = \lim_{\nu \to n} K_\nu(z)$$

(7) **Example**

(a) Problem
Obtain the value of $K_n(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
$\mathrm{N} = 3$ and $\mathrm{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBKNZ
! *** EXAMPLE OF ZIBKNZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBKNZ(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBKNZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF KN(Z)',/,/,10X,'ZO = (',D18.10,'  ,',D18.10,'  )' )
      END
```

(d) Output results

```
*** ZIBKNZ ***
 ** INPUT **
   N =  3     ZI = (  1.00   ,  2.00  )

 ** OUTPUT**
   IERR =     0
   VALUE OF KN(Z)
     ZO = ( -0.6814364280D+00   ,  0.6251546546D+00  )
```

## 2.5 SPHERICAL BESSEL FUNCTIONS

### 2.5.1 DIBSJN, RIBSJN
### Spherical Bessel Function of the 1st Kind (Integer Order)

(1) **Function**

Calculates a value of the spherical Bessel function of the 1st kind (integer order)

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+\frac{1}{2}}(x).$$

(2) **Usage**

Double precision:
CALL DIBSJN (N, XI, XO, IERR)
Single precision:
CALL RIBSJN (N, XI, XO, 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 | N | I | 1 | Input | Order $n$ |
| 2 | XI | {D, R} | 1 | Input | Value of variable $x$ |
| 3 | XO | {D, R} | 1 | Output | Value of $j_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq 0.0$

(b) XI $\leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\|n\|(\log_e \frac{\|n\|}{x} - M_1) > M_2$ (See Note (c)) (XI $\neq$ 0.0 and N $\neq$ 0) (underflow or overflow) | If N $\geq$ 0, XO = 0.0 is performed. If N < 0, XO = $(-1)^{\mathrm{N}+1}$ × (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $j_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and XI < 1000.0.

(b) To calculate $j_n(x), j_{n+1}(x), j_{n+2}(x) \cdots$ at a time, it is faster to successively use the recurrence relation bellow than to call this subroutine repeatedly.
The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation:

$$j_{n-1}(x) = \frac{2n+1}{x} j_n(x) - j_{n+1}(x)$$

(c) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 = \{$double precision: 709.7827, single precision: 88.72284$\}$

(d) Spherical Bessel function of the 1st kind $j_n(z)$ is the particular solution of differential equation:

$$z^2 \frac{d^2w}{dz^2} + 2z\frac{dw}{dz} + \{z^2 - n(n+1)\}w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

and defined as

$$j_n(z) = \sqrt{\frac{\pi}{2z}} J_{n+\frac{1}{2}}(z).$$

(7) **Example**

(a) Problem

Obtain the value of $j_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data

N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBSJN
! *** EXAMPLE OF DIBSJN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBSJN(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBSJN ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF SPHERICAL JN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBSJN ***

      ** INPUT **

        N =  5     XI =   1.50


      ** OUTPUT**

        IERR =     0

        VALUE OF SPHERICAL JN(X)

          XO =   0.6696205963D-03
```

## 2.5.2 DIBSYN, RIBSYN<br>Spherical Bessel Function of the 2nd Kind (Integer Order)

(1) **Function**

Calculate a value of the spherical Bessel function of the 2nd kind (integer Order)

$$y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+\frac{1}{2}}(x).$$

(2) **Usage**

Double precision:

CALL DIBSYN (N, XI, XO, IERR)

Single precision:

CALL RIBSYN (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of $y_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq 0.0$

(b) XI $\leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $\leq$ 2.0/(Maximum value)<br>or $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (See Note (c))<br>(XI $\neq$ 0.0 and N $\neq$ 0)<br>(overflow or underflow) | If N $\geq$ 0, XO = (Minimum value) is performed.<br>If N $<$ 0, XO = 0.0 is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $y_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $\mathrm{XI} < 1000.0$.

(b) To calculate $y_n(x), y_{n+1}(x), y_{n+2}(x) \cdots$ at a time, it is faster to successively use the recurrence relation below than to call this subroutine repeatedly.
Recurrence relation:

$$y_{n+1}(x) = \frac{2n+1}{x} y_n(x) - y_{n-1}(x)$$

(c) When IERR becomes 2000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Spherical Bessel function of the 2nd kind $y_n(z)$ is the particular solution of differential equation:

$$z^2 \frac{d^2 w}{dz^2} + 2z \frac{dw}{dz} + \{z^2 - n(n+1)\} w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

and defined as

$$y_n(z) = \sqrt{\frac{\pi}{2z}} Y_{n+\frac{1}{2}}(z).$$

(e) The spherical Neumann function $n_n(z)$ is the same as the spherical Bessel function of the 2nd kind $y_n(z)$.

(7) **Example**

(a) Problem
Obtain the value of $y_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
$\mathrm{N} = 5$ and $\mathrm{XI} = 1.5$.

(c) Main program

```
      PROGRAM BIBSYN
! *** EXAMPLE OF DIBSYN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBSYN(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBSYN ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF SPHERICAL YN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
*** DIBSYN ***
 ** INPUT **
   N =  5     XI =   1.50

 ** OUTPUT**
   IERR =     0
   VALUE OF SPHERICAL YN(X)
     X0 =  -0.9423611009D+02
```

### 2.5.3 DIBSIN, RIBSIN Modified Spherical Bessel Function of the 1st Kind (Integer Order)

(1) **Function**

Obtain the value of the modified spherical Bessel function of the 1st kind (integer Order)

$$i_n(x) = \sqrt{\frac{\pi}{2x}} I_{n+\frac{1}{2}}(x).$$

(2) **Usage**

Double precision:

CALL DIBSIN (N, XI, XO, IERR)

Single precision:

CALL RIBSIN (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $i_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{N} \geq 0$

(b) $\text{XI} \geq 0.0$

(c) $\text{XI} \leq M$

where, $M$ ={double precision: 709.7827, single precision: 88.72284}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $n(\log_e \frac{n}{x} - M_1) > M_2$ (See Note (d)) (XI $\neq 0.0$ and N $\neq 0$) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $i_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set N $<$ 1000 and XI $<$ 1000.0.

(b) If $n$ is negative and is not an integer, the Bessel function of the 1st kind cannot be calculated by using this subroutine. Therefore, it should be calculated by using a recurrence relation.

(c) To calculate $i_n(x), i_{n+1}(x), i_{n+2}(x) \cdots$ at a time, it is faster to successively use the recurrence relation than to call this subroutine repeatedly.
The computation, however, becomes unstable if it is done with increasing $n$. Therefore the recurrence relation should be used with decreasing $n$.
Recurrence relation:

$$i_{n-1}(x) = \frac{2n+1}{x} i_n(x) + i_{n+1}(x)$$

(d) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(e) Modified spherical Bessel function of the 1st kind $i_n(z)$ is the particular solution of differential equation:

$$z^2 \frac{d^2w}{dz^2} + 2z \frac{dw}{dz} - \{z^2 + n(n+1)\}w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

and defined as

$$i_n(z) = \sqrt{\frac{\pi}{2z}} I_{n+\frac{1}{2}}(z).$$

(7) **Example**

(a) Problem
Obtain the value of $i_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBSIN
! *** EXAMPLE OF DIBSIN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBSIN(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBSIN ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF SPHERICAL IN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
    *** DIBSIN ***

     ** INPUT **

       N =  5     XI =   1.50


     ** OUTPUT**

       IERR =     0

       VALUE OF SPHERICAL IN(X)

         XO =   0.7961612655D-03
```

## 2.5.4 DIBSKN, RIBSKN
## Modified Spherical Bessel Function of the 2nd Kind (Integer Order)

(1) **Function**

Calculate a value of the modified spherical Bessel function of the 2nd kind (integer order)

$$k_n(x) = \sqrt{\frac{\pi}{2x}} K_{n+\frac{1}{2}}(x).$$

(2) **Usage**

Double precision:

CALL DIBSKN (N, XI, XO, IERR)

Single precision:

CALL RIBSKN (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of $k_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq 0.0$

(b) XI $\leq M$
where, $M =${double precision: 702.293, single precision: 83.364}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $\leq$ 2.0/(Maximum value) or $\|n\|(\log_e \frac{\|n\|}{x} - M_1) > M_2$ (See Note (c)) (XI $\neq$ 0.0 and N $\neq$ 0) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The computation time of $k_n(x)$ becomes longer as $x$ and $n$ increase. Generally it is desirable to set $|\mathrm{N}| < 1000$ and $\mathrm{XI} < 1000.0$.

(b) To calculate $k_n(x), k_{n+1}(x), k_{n+2}(x) \cdots$ at a time, it is faster to successively use the recurrence relation given bellow than to call this subroutine repeatedly.
Recurrence relation:

$$k_{n+1}(x) = \frac{2n+1}{x} k_n(x) + k_{n-1}(x)$$

(c) When IERR becomes 2000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(d) Modified spherical Bessel function of the 2nd kind $k_n(z)$ is the particular solution of differential equation:

$$z^2 \frac{d^2w}{dz^2} + 2z\frac{dw}{dz} - \{z^2 + n(n+1)\}w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

and defined as

$$k_n(z) = \sqrt{\frac{\pi}{2z}} K_{n+\frac{1}{2}}(z).$$

(7) **Example**

(a) Problem
Obtain the value of $k_n(x)$ at $x = 1.5$ for $n = 5$.

(b) Input data
N = 5 and XI = 1.5.

(c) Main program

```
      PROGRAM BIBSKN
! *** EXAMPLE OF DIBSKN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBSKN(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBSKN ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF SPHERICAL KN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBSKN ***

      ** INPUT **

        N =  5     XI =   1.50


      ** OUTPUT**

        IERR =     0

        VALUE OF SPHERICAL KN(X)

          XO =   0.1152469739D+03
```

# 2.6 FUNCTIONS RELATED TO BESSEL FUNCTIONS

## 2.6.1 ZIBH1N, CIBH1N
## Hankel Function of the 1st Kind

(1) **Function**

Calculates the value of the Hankel function of the 1st kind

$$H_n^{(1)}(z) = -\frac{2\sqrt{-1}}{\pi} e^{-\sqrt{-1}n\pi/2} \int_0^\infty e^{\sqrt{-1}z\cosh(t)} \cosh(nt)dt \quad (0 < \arg z < \pi).$$

(2) **Usage**

Double precision:

CALL ZIBH1N (N, ZI, ZO, IERR)

Single precision:

CALL CIBH1N (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{array}{c}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Output | Value of $H_n^{(1)}(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{ZI}| > 0.0$

(b) $|\Im(\text{ZI})| \leq M_1$
where, $M_1$ ={double precision: 709.7827, single precision: 88.72284}

(c) $|\text{ZI}| \leq M_2$
where, $M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000 | $\lvert ZI \rvert \leq 2.0/$(Maximum value) or $\lvert n \rvert(\log_e \frac{\lvert n \rvert}{\lvert z \rvert} - M_3) > M_4$ (See Note (a)) | |

(6) **Notes**

(a) When IERR becomes 4000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =${double precision: 709.7827, single precision: 88.72284}

(b) Hankel function of the 1st kind $H_\nu^{(1)}(z)$ is the particular solution of Bessel's differential equation:

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0,$$

and defined as

$$H_\nu^{(1)}(z) = -\frac{1}{\pi}\int_{L_1} e^{-\sqrt{-1}z\sin\tau + \sqrt{-1}\nu\tau} d\tau$$

where the path of integration $L_1$ is taken as $(0, -\infty) \to (0, 0) \to (-\pi, 0) \to (-\pi, \infty)$.

(c) Hankel functions of the 1st kind and of the 2nd kind are also called Bessel function of the 3rd kind or cylindrical function of the 3rd kind.

(7) **Example**

(a) Problem
Obtain the value of $H_n^{(1)}(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
N $= 3$ and ZI $= (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBH1N
! *** EXAMPLE OF ZIBH1N ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBH1N(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBH1N ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF H1N(Z)',&
                           /,/,10X,'ZO = (',D18.10,'  ,',D18.10 ,'  )' )
      END
```

(d) Output results

```
     *** ZIBH1N ***

      ** INPUT **

        N =  3     ZI = (  1.00   ,  2.00  )

      ** OUTPUT**

        IERR =     0

        VALUE OF H1N(Z)

          ZO = ( -0.6892089637D-01   ,  0.3073283562D+00   )
```

### 2.6.2 ZIBH2N, CIBH2N Hankel Function of the 2nd Kind

(1) **Function**

Calculates the value of the Hankel function of the 2nd kind

$$H_n^{(2)}(z) = \frac{2\sqrt{-1}}{\pi} e^{\sqrt{-1}n\pi/2} \int_0^\infty e^{-\sqrt{-1}z\cosh(t)} \cosh(nt)dt \quad (0 < \arg z < \pi).$$

(2) **Usage**

Double precision:
  CALL ZIBH2N (N, ZI, ZO, IERR)
Single precision:
  CALL CIBH2N (N, ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | ZI | $\left\{\begin{array}{l}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 3 | ZO | $\left\{\begin{array}{l}\text{Z}\\ \text{C}\end{array}\right\}$ | 1 | Output | Value of $H_n^{(2)}(z)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{ZI}| > 0.0$

(b) $|\Im(\text{ZI})| \leq M_1$
where, $M_1$ ={double precision: 709.7827, single precision: 88.72284}

(c) $|\text{ZI}| \leq M_2$
where, $M_2$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000 | $|\text{ZI}| \leq 2.0/(\text{Maximum value})$ or $|n|(\log_e \frac{|n|}{|z|} - M_3) > M_4$ (See Note (a)) | |

(6) **Notes**

(a) When IERR becomes 4000 in this subroutine, the values of $M_3$ and $M_4$ are as follows:
$M_3 = 0.3068$,
$M_4 =${double precision: 709.7827, single precision: 88.72284}

(b) Hankel function of the 2nd kind $H_\nu^{(2)}(z)$ is the particular solution of Bessel's differential equation:

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0,$$

and defined as

$$H_\nu^{(2)}(z) = -\frac{1}{\pi}\int_{L_2} e^{-\sqrt{-1}z\sin\tau+\sqrt{-1}\nu\tau}d\tau$$

where the path of integration $L_2$ is taken as $(\pi,\infty) \to (\pi,0) \to (0,0) \to (0,-\infty)$.

(c) Hankel functions of the 1st kind and of the 2nd kind are also called Bessel function of the 3rd kind or cylindrical function of the 3rd kind.

(7) **Example**

(a) Problem
Obtain the value of $H_n^{(2)}(z)$ at $z = 1 + 2\sqrt{-1}$ for $n = 3$.

(b) Input data
N $= 3$ and ZI $= (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIBH2N
! *** EXAMPLE OF ZIBH2N ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) N,ZI
      CALL ZIBH2N(N,ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIBH2N ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF H2N(Z)',&
                           /,/,10X,'ZO = (',D18.10,'  ,',D18.10 ,'  )' )
      END
```

(d) Output results

```
    *** ZIBH2N ***

     ** INPUT **

       N =  3     ZI = (  1.00   ,  2.00  )


     ** OUTPUT**

       IERR =     0

       VALUE OF H2N(Z)

         ZO = ( -0.4931584373D+00   , -0.2729782322D+00   )
```

## 2.6.3 DIBBER, RIBBER
## Kelvin Function $\mathrm{ber}_n(x)$

(1) **Function**

Calculates the Kelvin function

$$\mathrm{ber}_n(x) = \Re\{J_n(xe^{3\sqrt{-1}\pi/4})\}.$$

(2) **Usage**

Double precision:
CALL DIBBER (N, XI, XO, IERR)
Single precision:
CALL RIBBER (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{ber}_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{XI}| \le M$
where, $M$ ={double precision: 1003.784, single precision: 125.473}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $|\text{N}|(\log_e \frac{|\text{N}|}{|\text{XI}|} - M_1) > M_2$ (See Note (a)) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:

$$M_1 = 0.3068$$
$$M_2 = 709.7827$$

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, and $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ are the solutions of the differential equation:

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0.$$

(7) **Example**

(a) Problem

Obtain the value of $\mathrm{ber}_n(x)$ at $x = 1.0$ for $n = 3$.

(b) Input data

N = 3 and XI = 1.0.

(c) Main program

```
      PROGRAM BIBBER
! *** EXAMPLE OF DIBBER ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBBER(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBBER ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF BERNX',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBBER ***

      ** INPUT **

        N =  3     XI =   1.00


      ** OUTPUT**

        IERR =     0

        VALUE OF BERNX

          XO =   0.1378798405D-01
```

## 2.6.4 DIBBEI, RIBBEI
## Kelvin Function $\mathrm{bei}_n(x)$

(1) **Function**

Calculates the Kelvin function

$$\mathrm{bei}_n(x) = \Im\{J_n(xe^{3\sqrt{-1}\pi/4})\}.$$

(2) **Usage**

Double precision:
CALL DIBBEI (N, XI, XO, IERR)
Single precision:
CALL RIBBEI (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{bei}_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $|\text{XI}| \le M$
where, $M$ ={double precision: 1003.784, single precision: 125.473}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\lvert\text{N}\rvert(\log_e \frac{\lvert\text{N}\rvert}{\lvert AGX\rvert} - M_1) > M_2$ (See Note (a)) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2$ ={double precision: 709.7827, single precision: 88.72284}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, and $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ are the solutions of the differential equation:

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0.$$

(7) **Example**

(a) Problem

Obtain the value of $\mathrm{bei}_n(x)$ at $x = 1.0$ for $n = 3$.

(b) Input data

N = 3 and XI = 1.0.

(c) Main program

```
      PROGRAM BIBBEI
! *** EXAMPLE OF DIBBEI ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBBEI(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBBEI ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF BEINX',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBBEI ***

      ** INPUT **

        N =  3     XI =   1.00


      ** OUTPUT**

        IERR =     0

        VALUE OF BEINX

          XO =   0.1562876861D-01
```

## 2.6.5 DIBKER, RIBKER Kelvin Function $\mathrm{ker}_n(x)$

(1) **Function**

Calculates the Kelvin function

$$\mathrm{ker}_n(x) = \Re\{e^{-\sqrt{-1}n\pi/2}K_n(xe^{\sqrt{-1}\pi/4})\}.$$

(2) **Usage**

Double precision:
CALL DIBKER (N, XI, XO, IERR)
Single precision:
CALL RIBKER (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{ker}_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0.0 < \text{XI} \leq M$
where, $M = \left\{\begin{array}{c}\text{double precision : 1003.784}\\ \text{single precision : 125.473}\end{array}\right\}$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | XI $\leq$ 2.0/(Maximum value) or $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (See Note (a)) | |

(6) **Notes**

(a) When IERR becomes 4000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$$M_2 = \begin{Bmatrix} \text{double precision : } 709.7827 \\ \text{single precision : } 88.72284 \end{Bmatrix}$$

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, and $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ are solutions of the differential equation:
$$x^2 \frac{d^2 w}{dx^2} + x \frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0.$$

(7) **Example**

(a) Problem
Obtain the value of $\mathrm{ker}_n(x)$ at $x = 1.0$ for $n = 3$.

(b) Input data
N = 3 and XI = 1.0.

(c) Main program

```
      PROGRAM BIBKER
! *** EXAMPLE OF DIBKER ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBKER(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBKER ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF KERNX',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBKER ***

      ** INPUT **

        N =  3     XI =   1.00


      ** OUTPUT**

        IERR =     0

        VALUE OF KERNX

          XO =   0.4887273882D+01
```

## 2.6.6 DIBKEI, RIBKEI
## Kelvin Function $\mathrm{kei}_n(x)$

(1) **Function**

Calculates the Kelvin function

$$\mathrm{kei}_n(x) = \Im\{e^{-\sqrt{-1}n\pi/2}K_n(xe^{\sqrt{-1}\pi/4})\}.$$

(2) **Usage**

Double precision:
CALL DIBKEI (N, XI, XO, IERR)
Single precision:
CALL RIBKEI (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{kei}_n(x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0.0 < \text{XI} \le M$
where, $M$ ={double precision: 1003.784, single precision: 125.473}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | $\text{XI} \le 2.0/$(Maximum value) or $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (See Note (a)) | |

(6) **Notes**

(a) When IERR becomes 4000 in this subroutine, the values of $M_1$ and $M_2$ are as follows:
$M_1 = 0.3068$,
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, and $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ are solutions of the differential equation:

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0.$$

(7) **Example**

(a) Problem
Obtain the value of $\mathrm{kei}_n(x)$ at $x = 1.0$ for $n = 3$.

(b) Input data
N = 3 and XI = 1.0.

(c) Main program

```
      PROGRAM BIBKEI
! *** EXAMPLE OF DIBKEI ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIBKEI(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIBKEI ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF KEINX',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIBKEI ***

      ** INPUT **

        N =  3     XI =   1.00

      ** OUTPUT**

        IERR =     0

        VALUE OF KEINX

          XO =  -0.6269710887D+01
```

### 2.6.7 WIBH0X, VIBH0X Struve Function (Order 0)

(1) **Function**

For $x = X_i$, calculates the Struve function (order 0)

$$\mathbf{H}_0(x) = \frac{2}{\pi}\int_0^{\frac{\pi}{2}} \sin(x\cos(t))dt.$$

(2) **Usage**

Double precision:
  CALL WIBH0X (NV, XI, XO, IERR)
Single precision:
  CALL VIBH0X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | Value of variable $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\mathbf{H}_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) For struve Function of order $\nu\mathbf{H}_\nu(z)$, the following recurrence relation holds:

$$\mathbf{H}_{\nu-1}(z) + \mathbf{H}_{\nu+1}(z) = \frac{2\nu}{z}\mathbf{H}_\nu(z) + \frac{\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})}.$$

(b) The general solution of differential equation

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=\frac{4\left(\frac{z}{2}\right)^{\nu+1}}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}$$

is

$$w=aJ_\nu(z)+bY_\nu(z)+\mathbf{H}_\nu(z),$$

where $J_\nu(z)$ and $Y_\nu(z)$ are Bessel function of the 1st kind and of the 2nd kind respectively, and $a$ and $b$ are constants.

(7) **Example**

(a) Problem
Obtain $\mathbf{H}_0(x)$ for $x=0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIBH0X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBH0X', CFNC='    H0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBH0X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBH0X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    H0(  0.000000)=  0.000000
    H0(  0.100000)=  0.063591
    H0(  0.200000)=  0.126759
    H0(  0.300000)=  0.189083
    H0(  0.400000)=  0.250150
    H0(  0.500000)=  0.309556
    H0(  0.600000)=  0.366911
    H0(  0.700000)=  0.421842
    H0(  0.800000)=  0.473994
    H0(  0.900000)=  0.523035
```

### 2.6.8 WIBH1X, VIBH1X Struve Function (Order 1)

(1) **Function**

For $x = X_i$, calculates the Struve function (order 1)

$$\mathbf{H}_1(x) = \frac{2x}{\pi}\int_0^{\frac{\pi}{2}} \sin(x\cos(t))\sin^2(t)dt.$$

(2) **Usage**

Double precision:

CALL WIBH1X (NV, XI, XO, IERR)

Single precision:

CALL VIBH1X (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\mathbf{H}_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) For struve Function of order $\nu$ $\mathbf{H}_\nu(z)$, the following recurrence relation holds

$$\mathbf{H}_{\nu-1}(z) + \mathbf{H}_{\nu+1}(z) = \frac{2\nu}{z}\mathbf{H}_\nu(z) + \frac{\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})}.$$

(b) The general solution of differential equation

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = \frac{4\left(\frac{z}{2}\right)^{\nu+1}}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}$$

is

$$w = aJ_\nu(z) + bY_\nu(z) + \mathbf{H}_\nu(z),$$

where $J_\nu(z)$ and $Y_\nu(z)$ are Bessel functions of the 1st kind and of the 2nd kind respectively, and $a$ and $b$ are constants.

(7) **Example**

(a) Problem
Obtain $\mathbf{H}_1(x)$ for $x = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIBH1X
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBH1X', CFNC='    H1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIBH1X( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBH1X *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
    H1(  0.000000)=  0.000000
    H1(  0.100000)=  0.002121
    H1(  0.200000)=  0.008466
    H1(  0.300000)=  0.018984
    H1(  0.400000)=  0.033593
    H1(  0.500000)=  0.052174
    H1(  0.600000)=  0.074580
    H1(  0.700000)=  0.100632
    H1(  0.800000)=  0.130122
    H1(  0.900000)=  0.162817
```

## 2.6.9 WIBHY0, VIBHY0 Difference of Struve Function (Order 0) and Bessel Function of the 2nd Kind (Order 0)

(1) **Function**

For $x = X_i$, calculates the difference of the Struve function (order 0) and Bessel function of the 2nd kind (order 0)

$$\mathbf{H}_0(x) - Y_0(x) = \frac{2}{\pi}\int_0^\infty e^{-xt}(1+t^2)^{-\frac{1}{2}}dt.$$

(2) **Usage**

Double precision:
CALL WIBHY0 (NV, XI, XO, IERR)
Single precision:
CALL VIBHY0 (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Output | $\mathbf{H}_0(X_i) - Y_0(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq$ 1
(b) XI(i) $\geq$ 0.0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI(i) = 0.0 (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) The following relation holds:

$$\mathbf{H}_\nu(z) - Y_\nu(z) = \frac{2\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}\int_0^\infty e^{-zt}(1+t^2)^{\nu-\frac{1}{2}}dt \quad (|\arg z| < \frac{\pi}{2}).$$

(7) **Example**

(a) Problem
Obtain $\mathbf{H}_0(x) - Y_0(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIBHY0
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBHY0', CFNC=' H0-Y0' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBHY0( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBHY0 *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
 H0-Y0(  0.100000)=  1.597830
 H0-Y0(  0.200000)=  1.207864
 H0-Y0(  0.300000)=  0.996357
 H0-Y0(  0.400000)=  0.856174
 H0-Y0(  0.500000)=  0.754075
 H0-Y0(  0.600000)=  0.675421
 H0-Y0(  0.700000)=  0.612507
 H0-Y0(  0.800000)=  0.560797
 H0-Y0(  0.900000)=  0.517407
 H0-Y0(  1.000000)=  0.480400
```

### 2.6.10 WIBHY1, VIBHY1 Difference of Struve Function (Order 1) and Bessel Function of the 2nd Kind (Order 1)

(1) **Function**

For $x = X_i$, calculates the difference of the Struve function (order 1) and Bessel function of the 2nd kind (order 1)

$$\mathbf{H}_1(x) - Y_1(x) = \frac{2x}{\pi} \int_0^\infty e^{-xt}(1+t^2)^{\frac{1}{2}}dt.$$

(2) **Usage**

Double precision:
  CALL WIBHY1 (NV, XI, XO, IERR)
Single precision:
  CALL VIBHY1 (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\mathbf{H}_1(X_i) - Y_1(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\geq 0.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI(i) $\leq$ 1.0/(Maximum value) (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

(6) **Notes**

(a) The following relation holds:

$$\mathbf{H}_\nu(z) - Y_\nu(z) = \frac{2\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}\int_0^\infty e^{-zt}(1+t^2)^{\nu-\frac{1}{2}}dt \quad (|\arg z| < \frac{\pi}{2}).$$

(7) **Example**

(a) Problem
Obtain $\mathbf{H}_1(x) - Y_1(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIBHY1
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIBHY1', CFNC=' H1-Y1' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIBHY1( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIBHY1 *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
 H1-Y1(  0.100000)=  6.461072
 H1-Y1(  0.200000)=  3.332291
 H1-Y1(  0.300000)=  2.312089
 H1-Y1(  0.400000)=  1.814465
 H1-Y1(  0.500000)=  1.523646
 H1-Y1(  0.600000)=  1.334971
 H1-Y1(  0.700000)=  1.203882
 H1-Y1(  0.800000)=  1.108267
 H1-Y1(  0.900000)=  1.035944
 H1-Y1(  1.000000)=  0.979670
```

## 2.6.11 DIBAIX, RIBAIX
## Airy Function Ai($x$)

(1) **Function**

Calculates the Airy function

$$\mathrm{Ai}(x) = \begin{cases} \pi^{-1}\sqrt{\frac{x}{3}}K_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) & (x \geq 0.0) \\ \frac{1}{3}\sqrt{|x|}[J_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + J_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases} .$$

(2) **Usage**

Double precision:
CALL DIBAIX (XI, XO, IERR)
Single precision:
CALL RIBAIX (XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 2 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of Ai($x$) |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq -M$
where,
$M$={double precision: $(3 \times 2^{49}\pi)^{2/3}$, single precision: $(3 \times 2^{17}\pi)^{2/3}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI $> M_1$ (See Note (a)) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the value of $M_1$ is as follows:
$M_1$ ={double precision: 103.8, single precision: 25.3}

(b) Pairs of linearly independent solution of differential equation:

$$\frac{d^2w}{dz^2} - zw = 0$$

are $\{\mathrm{Ai}(z), \mathrm{Bi}(z)\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{2\sqrt{-1}\pi}{3}})\}$, and $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{-2\sqrt{-1}\pi}{3}})\}$.

(7) **Example**

(a) Problem

Obtain the value of $\mathrm{Ai}(x)$ at $x = -5.3$.

(b) Input data

$\mathrm{XI} = -5.3$.

(c) Main program

```
      PROGRAM BIBAIX
! *** EXAMPLE OF DIBAIX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER IERR
!
      READ(5,*) XI
      WRITE(6,1000)
      WRITE(6,2000) XI
      CALL DIBAIX(XI,XO,IERR)
      WRITE(6,3000)
      WRITE(6,4000) IERR
      WRITE(6,5000) XO
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DIBAIX ***',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'VALUE OF VARIABLE X = ',F4.1)
 3000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'VALUE OF AIRY FUNCTION AI(X) = ',D17.10)
      END
```

(d) Output results

```
     *** DIBAIX ***
      ** INPUT **
         VALUE OF VARIABLE X = -5.3


      ** OUTPUT **
         IERR =    0
         VALUE OF AIRY FUNCTION AI(X) =  0.1825679311D+00
```

## 2.6.12 DIBBIX, RIBBIX
## Airy Function Bi$(x)$

(1) **Function**

Calculates the Airy function

$$\mathrm{Bi}(x) = \begin{cases} \sqrt{\frac{x}{3}}[I_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + I_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x \geq 0.0) \\ \sqrt{\frac{|x|}{3}}[J_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) - J_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases} .$$

(2) **Usage**

Double precision:
CALL DIBBIX (XI, XO, IERR)
Single precision:
CALL RIBBIX (XI, XO, 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 | XI | {D, R} | 1 | Input | Value of variable $x$ |
| 2 | XO | {D, R} | 1 | Output | Value of Bi$(x)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq -M$
where, $M$={double precision: $(3 \times 2^{49}\pi)^{2/3}$, single precision: $(3 \times 2^{17}\pi)^{2/3}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $> M_1$ (See Note (a)) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 2000 in this subroutine, the value of $M_1$ is as follows:
$M_1$ ={double precision: 104.266, single precision: 20.066}

(b) Pairs of linearly independent solution of differential equation:

$$\frac{d^2w}{dz^2} - zw = 0$$

are $\{\mathrm{Ai}(z), \mathrm{Bi}(z)\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{2\sqrt{-1}\pi}{3}})\}$, and $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{-2\sqrt{-1}\pi}{3}})\}$.

(7) **Example**

(a) Problem
Obtain the value of Bi($x$) at $x = -5.3$.

(b) Input data
XI $= -5.3$.

(c) Main program

```
      PROGRAM BIBBIX
! *** EXAMPLE OF DIBBIX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER IERR
!
      READ(5,*) XI
      WRITE(6,1000)
      WRITE(6,2000) XI
      CALL DIBBIX(XI,XO,IERR)
      WRITE(6,3000)
      WRITE(6,4000) IERR
      WRITE(6,5000) XO
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DIBBIX ***',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'VALUE OF VARIABLE X = ',F4.1)
 3000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'VALUE OF AIRY FUNCTION BI(X) = ',D17.10)
      END
```

(d) Output results

```
     *** DIBBIX ***
      ** INPUT **
         VALUE OF VARIABLE X = -5.3


      ** OUTPUT **
         IERR =    0
         VALUE OF AIRY FUNCTION BI(X) = -0.3237160767D+00
```

## 2.6.13 DIBAID, RIBAID Derived Airy Function $\mathrm{Ai}'(x)$

(1) **Function**

Calculates the derived Airy function

$$\mathrm{Ai}'(x) = \begin{cases} -\frac{x}{\sqrt{3}\pi} K_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) & (x \geq 0.0) \\ \frac{x}{3}[J_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) - J_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases} .$$

(2) **Usage**

Double precision:
  CALL DIBAID (XI, XO, IERR)
Single precision:
  CALL RIBAID (XI, XO, IERR)

(3) **Arguments**

D:Double precision real   Z:Double precision complex   I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 2 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{Ai}'(x)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq -M$
where, $M$={double precision: $(3 \times 2^{49}\pi)^{2/3}$, single precision: $(3 \times 2^{17}\pi)^{2/3}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI $> M_1$ (See Note (a)) (underflow) | XO = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the value of $M_1$ is as follows:
$M_1$ ={double precision: 104.1, single precision: 25.7}

(7) **Example**

(a) Problem

Obtain the value of the derived function $\mathrm{Ai}'(x)$ of the Airy function $\mathrm{Ai}(x)$ at $x = -5.3$.

(b) Input data

$\mathrm{XI} = -5.3$.

(c) Main program

```
      PROGRAM BIBAID
! *** EXAMPLE OF DIBAID ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER IERR
!
      READ(5,*) XI
      WRITE(6,1000)
      WRITE(6,2000) XI
      CALL DIBAID(XI,XO,IERR)
      WRITE(6,3000)
      WRITE(6,4000) IERR
      WRITE(6,5000) XO
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DIBAID ***',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'VALUE OF VARIABLE X = ',F4.1)
 3000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'VALUE OF DERIVED AIRY FUNCTION AI''(X) = ',D17.10)
      END
```

(d) Output results

```
     *** DIBAID ***
      ** INPUT **
         VALUE OF VARIABLE X = -5.3


      ** OUTPUT **
         IERR =    0
         VALUE OF DERIVED AIRY FUNCTION AI'(X) =  0.7545754199D+00
```

## 2.6.14 DIBBID, RIBBID Derived Airy Function $\mathrm{Bi}'(x)$

(1) **Function**

Calculates the derived Airy function

$$\mathrm{Bi}'(x) = \begin{cases} \frac{x}{\sqrt{3}}[I_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + I_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x \geq 0.0) \\ -\frac{x}{\sqrt{3}}[J_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + J_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases}.$$

(2) **Usage**

Double precision:
CALL DIBBID (XI, XO, IERR)
Single precision:
CALL RIBBID (XI, XO, 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 | XI | {D, R} | 1 | Input | Value of variable $x$ |
| 2 | XO | {D, R} | 1 | Output | Value of $\mathrm{Bi}'(x)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq -M$
where, $M$={double precision: $(3 \times 2^{49}\pi)^{2/3}$, single precision: $(3 \times 2^{17}\pi)^{2/3}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI $> M_1$ (See Note (a)) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 2000 in this subroutine, the value of $M_1$ is as follows:
$M_1 = \{$double precision: 104.266, single precision: 20.066$\}$

(7) **Example**

(a) Problem
Obtain the value of the derived function $\mathrm{Bi}'(x)$ of the Airy function $\mathrm{Bi}(x)$ at $x = -5.3$.

(b) Input data
$\mathrm{XI} = -5.3$.

(c) Main program

```
      PROGRAM BIBBID
! *** EXAMPLE OF DIBBID ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER IERR
!
      READ(5,*) XI
      WRITE(6,1000)
      WRITE(6,2000) XI
      CALL DIBBID(XI,XO,IERR)
      WRITE(6,3000)
      WRITE(6,4000) IERR
      WRITE(6,5000) XO
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DIBBID ***',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'VALUE OF VARIABLE X = ',F4.1)
 3000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'VALUE OF DERIVED AIRY FUNCTION BI''(X) = ',D17.10)
      END
```

(d) Output results

```
    *** DIBBID ***
     ** INPUT **
        VALUE OF VARIABLE X = -5.3


     ** OUTPUT **
        IERR =    0
        VALUE OF DERIVED AIRY FUNCTION BI'(X) =  0.4055569409D+00
```

## 2.7 GAMMA FUNCTIONS

### 2.7.1 WIGAMX, VIGAMX<br>Gamma Function with Real Variable

(1) **Function**

For $x = X_i$, calculates the value of the Gamma function with real variable

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt.$$

(2) **Usage**

Double precision:

CALL WIGAMX (NV, XI, XO, IERR)

Single precision:

CALL VIGAMX (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Input | $X_i$ |
| 3 | XO | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Output | $\Gamma(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) is not any negative integer or 0.0.

(c) XI(i) $\geq -$M

where $M$ ={double precision: 170.4, single precision: 34.0}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | $\lvert XI(i) \rvert \leq 1.0/$(Maximum value) or XI(i) $> M_1$ (See Note (c)) (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by XI(i). | |

(6) **Notes**

(a) If IERR $= 3000 + i$ and restriction (b) is not satisfied, furthermore XI(i) is not a value close to a negative integer, $\Gamma(X_i)$ is a value close to 0.0.

(b) $x = -n$ $(n = 0, 1, 2, \cdots)$ are simple poles of the Gamma function $\Gamma(x)$, i.e.

$$\lim_{x \to -n} \frac{1}{\Gamma(x)} = 0 \quad (n = 0, 1, 2, \cdots).$$

Therefore, high precision result cannot be expected for a calculated value of gamma function at a point close to zero or at a negative integer.

(c) The value to return IERR=2000 of $M_1$ is as follows:
$M_1 =$\{double precision: 171.4, single precision: 35.0\}

(d) Gamma Function $\Gamma(z)$ is called Euler's integral of the 2nd kind. This function is also called factorial function because $\Gamma(z+1) = z\Gamma(z) = z!$ holds.

(7) **Example**

(a) Problem
Obtain $\Gamma(x)$ for $x = 0.1, 0.2, \cdots, 0.9, 1.0$.

(b) Main program

```
      PROGRAM EIGAMX
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIGAMX', CFNC=' GAMMA' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIGAMX( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIGAMX *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
 GAMMA(  0.100000)=  9.513508
 GAMMA(  0.200000)=  4.590844
 GAMMA(  0.300000)=  2.991569
 GAMMA(  0.400000)=  2.218160
 GAMMA(  0.500000)=  1.772454
 GAMMA(  0.600000)=  1.489192
 GAMMA(  0.700000)=  1.298055
 GAMMA(  0.800000)=  1.164230
 GAMMA(  0.900000)=  1.068629
 GAMMA(  1.000000)=  1.000000
```

### 2.7.2 WIGLGX, VIGLGX
### Logarithmic Gamma Function with Real Variable

(1) **Function**

For $x = X_i$, calculates the value of the logarithmic Gamma function with real variable $\log_e(\Gamma(x))$.

(2) **Usage**

Double precision:
CALL WIGLGX (NV, XI, XO, IERR)
Single precision:
CALL VIGLGX (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Output | $\log_e(\Gamma(X_i))$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) must not be any non-positive integer.

(c) XI(i) $> -M$
where $M$ ={double precision: $2^{50}$, single precision: $2^{18}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Γ(XI(i)) is negative. | The natural logarithm of \|Γ(XI(i))\| is performed. |
| 2000 | XI(i) > $M_1$ (See Note (b)) (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by XI(i). | |

(6) **Notes**

(a) When IERR = 1000, the Gamma function value is obtained as $-\exp(\mathrm{XO(i)})$ for a certain i, while it is obtained as $\exp(\mathrm{XO(i)})$ in other cases.

(b) When IERR becomes 2000 in this subroutine, the value of $M_1$ is as follows:
$M_1 = \{$double precision: $2.545 \times 10^{305}$, single precision: $4.08 \times 10^{36}\}$

(7) **Example**

(a) Problem
Obtain $\log_e(\Gamma(x))$ for $x = 0.1, 0.2, \cdots, 0.9, 1.0$.

(b) Main program

```
      PROGRAM EIGLGX
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIGLGX', CFNC='LGAMMA' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIGLGX( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIGLGX *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
LGAMMA(  0.100000)=  2.252713
LGAMMA(  0.200000)=  1.524064
LGAMMA(  0.300000)=  1.095798
LGAMMA(  0.400000)=  0.796678
LGAMMA(  0.500000)=  0.572365
LGAMMA(  0.600000)=  0.398234
LGAMMA(  0.700000)=  0.260867
LGAMMA(  0.800000)=  0.152060
LGAMMA(  0.900000)=  0.066376
LGAMMA(  1.000000)= -0.000000
```

### 2.7.3 DIGIG1, RIGIG1 Incomplete Gamma Function of the 1st Kind

(1) **Function**

Calculates the value of the incomplete Gamma function of the 1st kind

$$\gamma(\nu, x) = \int_0^x e^{-t} t^{\nu-1} dt.$$

(2) **Usage**

Double precision:
CALL DIGIG1 (V, XI, XO, IERR)
Single precision:
CALL RIGIG1 (V, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | V | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 1 | Input | Value of variable $\nu$ |
| 2 | XI | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 1 | Output | Value of $\gamma(\nu, x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq$ 0.0

(b) V $\geq$ 0.0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\nu \log_e(x) < -M_1$ (See Note (b)) (underflow) | XO = 0.0 is performed. |
| 2000 | V $\leq$ 1.0/(Maximum value) (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000 | For XI > 1.0, <br>• $\nu > (M_2 + x)/\log_e(x)$ or <br>• $x > x_m$ and $\nu > \nu_m$ <br>was satisfied (See Note (c)). | Processing is aborted. (See Note (a)) |

(6) **Notes**

(a) If IERR=4000, the value of $\gamma(\nu, x)$ is practically the (maximum value).

(b) When IERR becomes 1000 in this subroutine, the value of $M_1$ is as follows:
$M_1$ ={double precision: 708.396, single precision: 87.336}

(c) When IERR becomes 4000 in this subroutine, the values of $x_m$, $\nu_m$, and $M_2$ are as follows:
$x_m$ ={double precision: 171.0, single precision: 35.0},
$\nu_m$ ={double precision: 171.4, single precision: 35.0},
$M_2$ ={double precision: 709.782, single precision: 88.722}

(7) **Example**

(a) Problem
Obtain the value of $\gamma(\nu, x)$ at $x = 3.0$ for $\nu = 4.0$.

(b) Input data
V = 4.0 and XI = 3.0.

(c) Main program

```
      PROGRAM BIGIG1
! *** EXAMPLE OF DIGIG1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) V
      READ (5,*) XI
      WRITE(6,1000) V,XI
      CALL DIGIG1(V,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIGIG1 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'V =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF INCOMPLETE GAMMA FUNCTION OF THE FIRST KIND',&
                                           /,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIGIG1 ***

      ** INPUT **

        V =  4.00     XI =   3.00


      ** OUTPUT**

        IERR =     0

        VALUE OF INCOMPLETE GAMMA FUNCTION OF THE FIRST KIND

          XO =   0.2116608667D+01
```

### 2.7.4 DIGIG2, RIGIG2 Incomplete Gamma Function of the 2nd Kind

(1) **Function**

Calculates the value of the incomplete Gamma function of the 2nd kind

$$\Gamma(\nu, x) = \int_x^\infty e^{-t} t^{\nu-1} dt = e^{-x} \int_0^\infty e^{-t} (x+t)^{\nu-1} dt.$$

(2) **Usage**

Double precision:

CALL DIGIG2 (V, XI, XO, IERR)

Single precision:

CALL RIGIG2 (V, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | V | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $\nu$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $\Gamma(\nu, x)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) XI $\geq$ 0.0

(b) 0.0 $\leq$ V $\leq$ $M$
where, $M$ ={double precision: 171.4, single precision: 35.0}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | For XI > V,<br>$(\nu - 1)\log_e(x) - x < -M_1$<br>(See Note (b)) (underflow) | XO = 0.0 is performed. |
| 2000 | V = 0.0 and XI = 0.0 (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. (See Note (a)) |
| 4000 | For XI ≤ V − 0.5,<br>$\nu \log_e(x) > (M_2 + x)$ or $x > x_m$<br>was satisfied (See Note (c)). | Processing is aborted. |

(6) **Notes**

(a) If $\nu$ is a large value when IERR = 3000, the value of $\Gamma(\nu, x)$ is practically the (maximum value).

(b) When IERR becomes 1000 in this subroutine, the value of $M_1$ is as follows:
$M_1 =$ {double precision: 708.396, single precision: 87.336}

(c) When IERR becomes 4000 in this subroutine, the values of $x_m$ and $M_2$ are as follows:
$x_m =$ {double precision: 171.0, single precision: 35.0},
$M_2 =$ {double precision: 709.7827, single precision: 88.72284}

(7) **Example**

(a) Problem
Obtain the value of $\Gamma(\nu, x)$ at $x = 3.0$ for $\nu = 4.0$.

(b) Input data
V = 4.0 and XI = 3.0.

(c) Main program

```
      PROGRAM BIGIG2
! *** EXAMPLE OF DIGIG2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) V
      READ (5,*) XI
      WRITE(6,1000) V,XI
      CALL DIGIG2(V,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIGIG2 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'V =',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF INCOMPLETE GAMMA FUNCTION OF THE SECOND KIND',&
                                           /,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIGIG2 ***

      ** INPUT **

        V =   4.00     XI =   3.00

      ** OUTPUT**

        IERR =     0

        VALUE OF INCOMPLETE GAMMA FUNCTION OF THE SECOND KIND

          XO =   0.3883391333D+01
```

## 2.7.5 ZIGAMZ, CIGAMZ Gamma Function with Complex Variable

(1) **Function**

Calculates the value of the Gamma function with complex variable

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt \quad (\Re\{z\} > 0).$$

(2) **Usage**

Double precision:
CALL ZIGAMZ ( ZI, ZO, IERR)
Single precision:
CALL CIGAMZ ( ZI, ZO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | ZI | $\left\{\begin{array}{c}Z\\C\end{array}\right\}$ | 1 | Input | Value of variable $z$ |
| 2 | ZO | $\left\{\begin{array}{c}Z\\C\end{array}\right\}$ | 1 | Output | Value of $\Gamma(z)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ZI must not be a negative integer and 0.0, and $|\text{ZI}| > 1.0/(\text{Maximum value})$.

(b) $-M_1 \le \Re(\text{ZI}) \le M_2$
where, $M_1$ ={double precision: 170.4, single precision: 34.0},
$M_2$ ={double precision: 171.4, single precision: 35.0}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. (See Note (a)) |

(6) **Notes**

(a) When $\Re(\text{ZI}) < -M_1$ (aborted with IERR = 3000; see Restriction(b)), the value of $\Gamma(z)$ will be almost 0.0 except the case ZI is close to a negative integer.

(b) $z = -n \quad (n = 0, 1, 2, \cdots)$ is a singular point for gamma function $\Gamma(z)$ such that

$$\lim_{z\to -n} \frac{1}{\Gamma(z)} = 0 \ (n = 0, 1, 2, \cdots)$$

So, high precision result cannot be expected for a calculated value of gamma function at a point close to zero or at a negative integer.

(c) Gamma Function $\Gamma(z)$ is called Euler's integral of the 2nd kind. This function is also called a factorial function because $\Gamma(z+1) = z\Gamma(z) = z!$ holds.

(7) **Example**

(a) Problem
Obtain the value of $\Gamma(z)$ at $z = 1 + 2\sqrt{-1}$.

(b) Input data
$\mathrm{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIGAMZ
! *** EXAMPLE OF ZIGAMZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) ZI
      CALL ZIGAMZ(ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIGAMZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'ZI = (',F6.2,'   ,',F6.2,'   )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF GAMMA FUNCTION OF COMPLEX VARIABLE',&
                           /,/,10X,'ZO = (',D18.10,'   ,',D18.10 ,'   )' )
      END
```

(d) Output results

```
     *** ZIGAMZ ***

      ** INPUT **

        ZI = (   1.00   ,   2.00   )

      ** OUTPUT**

        IERR =     0

        VALUE OF GAMMA FUNCTION OF COMPLEX VARIABLE

          ZO = (   0.1519040027D+00   ,   0.1980488016D-01   )
```

### 2.7.6 ZIGLGZ, CIGLGZ Logarithmic Gamma Function with Complex Variable

(1) **Function**

Calculates the value of the logarithmic Gamma function with complex variable $\log_e(\Gamma(z))$.

(2) **Usage**

Double precision:
CALL ZIGLGZ ( ZI, ZO, IERR)
Single precision:
CALL CIGLGZ ( ZI, ZO, 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 | ZI | {Z, C} | 1 | Input | Value of variable $z$ |
| 2 | ZO | {Z, C} | 1 | Output | Value of $\log_e(\Gamma(z))$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ZI must not be a negative integer or 0.0.

(b) $\Re(\mathrm{ZI}) > -M_1$
where $M_1$ ={double precision: $2^{50}$, single precision: $2^{18}$}

(c) $|\Im(\mathrm{ZI})|, \Re(\mathrm{ZI}) \leq M_2$
where $M_2$ ={double precision: $2.545 \times 10^{305}$, single precision: $4.08 \times 10^{36}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | $\Gamma(z) < 0$ | $\mathrm{ZO} = \log_e(\lvert\Gamma(z)\rvert) + \pi\sqrt{-1}$ is performed. |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR = 1000, the Gamma function value is obtained as $\exp(\Re(\mathrm{ZO}))$, while it is obtained as $\exp(\mathrm{ZO})$ in general.

(b) Logarithmic gamma function with complex variable $\log_e(\Gamma(z))$ is many-valued function (with infinite number of values) and the difference between their values is integer times of $2\pi\sqrt{-1}$. This subroutine calculates the principal value determined so that $-\pi < \Im\{\log_e(\Gamma(z))\} \leq \pi$ hold.

(7) **Example**

(a) Problem
Obtain the value of $\log_e(\Gamma(z))$ at $z = 1 + 2\sqrt{-1}$.

(b) Input data
$\mathrm{ZI} = (1.0, 2.0)$.

(c) Main program

```
      PROGRAM AIGLGZ
! *** EXAMPLE OF ZIGLGZ ***
      IMPLICIT COMPLEX(8) (A-H,O-Z)
      READ (5,'(D6.1,D6.1)') ZI
      WRITE(6,1000) ZI
      CALL ZIGLGZ(ZI,ZO,IERR)
      WRITE(6,2000) IERR,ZO
 1000 FORMAT(' ',/,/,5X,'*** ZIGLGZ ***',/,/,6X,'** INPUT **',&
      /,/,8X,'ZI = (',F6.2,'  ,',F6.2,'  )' )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF LOGARITHMIC GAMMA FUNCTION OF COMPLEX VARIABLE',&
                         /,/,10X,'ZO = (',D18.10,'  ,',D18.10 ,'  )' )
      END
```

(d) Output results

```
     *** ZIGLGZ ***

      ** INPUT **

        ZI = (   1.00   ,   2.00   )

      ** OUTPUT**

        IERR =     0

        VALUE OF LOGARITHMIC GAMMA FUNCTION OF COMPLEX VARIABLE

          ZO = ( -0.1876078786D+01   ,   0.1296463163D+00   )
```

## 2.8 FUNCTIONS RELATED TO THE GAMMA FUNCTION

### 2.8.1 WIGDIG, VIGDIG
### Digamma Function

(1) **Function**

For $x = X_i$, calculates the value of the digamma function (psi function)

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)} = \frac{d}{dx}\log_e(\Gamma(x)).$$

(2) **Usage**

Double precision:

CALL WIGDIG (NV, XI, XO, IERR)

Single precision:

CALL VIGDIG (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\psi(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) is not negative integer or 0.0

(c) XI(i) $> -M$
where $M$ ={double precision: $2^{50}$, single precision: $2^{18}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by XI(i). | |

(6) **Notes**

(a) Digamma function $\psi(x)$ is a solution of the following equations

$$\psi(x+1) - \psi(x) = \frac{1}{x}, \quad \psi(1) = -\gamma, \quad \lim_{n\to\infty}\{\psi(x+n) - \psi(1+n)\} = 0$$

where, $\gamma$ is Euler's constant:

$$\gamma = \lim_{n\to\infty}\left(1 + \frac{1}{2} + \cdots + \frac{1}{n} - \log_e n\right)$$

(b) Derived functions of digamma function $\psi(x)$: $\psi'(x)$, $\psi''(x)$, and $\psi'''(x)$, are called as trigamma function, tetragamma function, pentagamma function, respectively. In general, derived functions of digamma function are called polygamma function.

(7) **Example**

(a) Problem

Obtain $\psi(x)$ for $x = 0.1, 0.2, \cdots, 0.9, 1.0$.

(b) Main program

```
      PROGRAM EIGDIG
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIGDIG', CFNC='DGAMMA' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIGDIG( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIGDIG *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
DGAMMA(  0.100000)=-10.423755
DGAMMA(  0.200000)= -5.289040
DGAMMA(  0.300000)= -3.502524
DGAMMA(  0.400000)= -2.561385
DGAMMA(  0.500000)= -1.963510
DGAMMA(  0.600000)= -1.540619
DGAMMA(  0.700000)= -1.220024
DGAMMA(  0.800000)= -0.965009
DGAMMA(  0.900000)= -0.754927
DGAMMA(  1.000000)= -0.577216
```

## 2.8.2 WIGBET, VIGBET
## Beta Function

(1) **Function**

For $p = X_i$ and $q = Y_i$, calculates the value of the beta function

$$B(p,q) = \int_0^1 t^{p-1}(1-t)^{q-1}dt.$$

(2) **Usage**

Double precision:
CALL WIGBET (NV, P, Q, XO, IERR)
Single precision:
CALL VIGBET (NV, P, Q, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | P | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | Q | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $Y_i$ |
| 4 | XO | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Output | $B(X_i, Y_i)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) 2.0/(maximum)$<$ P(i) $\leq M$

(c) 2.0/(maximum)$<$ Q(i) $\leq M$
where $M$ ={double precision: $1.2 \times 10^{305}$, single precision: $2.0 \times 10^{36}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | P(i) > 1000.0 and Q(i) > 1000.0 | The solution is obtained but the precision could not be guaranteed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) or (c) was not satisfied by P(i) and Q(i). | |

(6) **Notes**

(a) Beta Function $B(p, q)$ is called Euler's integral of the 1st kind.

(b) Beta Function $B(p, q)$ is represented using gamma function $\Gamma(z)$ as

$$B(p, q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}.$$

(7) **Example**

(a) Problem

Obtain $B(p, q)$ for $p = 0.1, 0.2, \cdots, 0.9, 1.0$ and $q = 0.5$.

(b) Main program

```
      PROGRAM EIGBET
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , YI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIGBET', CFNC='  BETA' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
        YI(I)=0.5D0
 1000 CONTINUE
!
      CALL WIGBET( NV, XI, YI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIGBET *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
```

```
IERR=     0
  BETA(  0.100000)= 11.323087
  BETA(  0.200000)=  6.268653
  BETA(  0.300000)=  4.554443
  BETA(  0.400000)=  3.679094
  BETA(  0.500000)=  3.141593
  BETA(  0.600000)=  2.774502
  BETA(  0.700000)=  2.505796
  BETA(  0.800000)=  2.299288
  BETA(  0.900000)=  2.134760
  BETA(  1.000000)=  2.000000
```

# 2.9 ELLIPTIC FUNCTIONS AND ELLIPTIC INTEGRALS

## 2.9.1 WIECI1, VIECI1
## Complete Elliptic Integral of the 1st Kind

(1) **Function**

For $m = X_i$, calculates the value of the complete elliptic integral of the 1st kind

$$K(m) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}} = \int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1-m\sin^2\theta}} d\theta.$$

(2) **Usage**

Double precision:

CALL WIECI1 (NV, RM, XO, IERR)

Single precision:

CALL VIECI1 (NV, RM, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | RM | $\begin{Bmatrix} D \\ R \end{Bmatrix}$ | NV | Input | Modulus $X_i$ |
| 3 | XO | $\begin{Bmatrix} D \\ R \end{Bmatrix}$ | NV | Output | $K(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM(i) $\leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | RM(i) = 1.0 (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by RM(i). | |

(6) **Notes**

(a) If the complete elliptic integral of the 1st kind is given as $K(k) = \int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1-k^2\sin^2\theta}}d\theta$, then input $k^2$ to RM(i).

(b) Evaluating both $E(m)$ and $K(m)$, it is more effective to use 2.9.8 $\begin{Bmatrix} \text{WIENMQ} \\ \text{VIENMQ} \end{Bmatrix}$.

(7) **Example**

(a) Problem
Obtain $K(m)$ for $m = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIECI1
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIECI1', CFNC='     K' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIECI1( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIECI1 *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     K(  0.000000)=  1.570796
     K(  0.100000)=  1.612441
     K(  0.200000)=  1.659624
     K(  0.300000)=  1.713889
     K(  0.400000)=  1.777519
     K(  0.500000)=  1.854075
     K(  0.600000)=  1.949568
     K(  0.700000)=  2.075363
     K(  0.800000)=  2.257205
     K(  0.900000)=  2.578092
```

## 2.9.2 WIECI2, VIECI2 Complete Elliptic Integral of the 2nd Kind

(1) **Function**

For $m = X_i$, calculates the value of the complete elliptic integral of the 2nd kind

$$E(m) = \int_0^1 \sqrt{(1-t^2)(1-mt^2)}dt = \int_0^{\frac{\pi}{2}} \sqrt{1-m\sin^2\theta}d\theta.$$

(2) **Usage**

Double precision:
CALL WIECI2 (NV, RM, XO, IERR)
Single precision:
CALL VIECI2 (NV, RM, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\begin{Bmatrix}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{Bmatrix}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | RM | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NV | Input | Modulus $X_i$ |
| 3 | XO | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NV | Output | $E(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM(i) $\leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by RM(i). | |

(6) **Notes**

(a) If the complete elliptic integral of the 2nd kind is given as $E(k) = \int_0^{\frac{\pi}{2}} \sqrt{1-k^2\sin^2\theta}d\theta$ then the value of $k^2$ must be input for RM(i).

(b) Evaluating both $E(m)$ and $K(m)$, it is more effective to use 2.9.8 $\begin{Bmatrix}\text{WIENMQ}\\ \text{VIENMQ}\end{Bmatrix}$.

(7) **Example**

(a) Problem

Obtain $E(m)$ for $m = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIECI2
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIECI2', CFNC='     E' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIECI2( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIECI2 *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     E(  0.000000)=  1.570796
     E(  0.100000)=  1.530758
     E(  0.200000)=  1.489035
     E(  0.300000)=  1.445363
     E(  0.400000)=  1.399392
     E(  0.500000)=  1.350644
     E(  0.600000)=  1.298428
     E(  0.700000)=  1.241671
     E(  0.800000)=  1.178490
     E(  0.900000)=  1.104775
```

## 2.9.3 DIEII1, RIEII1 Incomplete Elliptic Integral of the 1st Kind

### (1) Function

Calculates the value of the incomplete elliptic integral of the 1st kind

$$F(x,m) = \int_0^x \frac{1}{\sqrt{(1-t^2)(1-mt^2)}}dt = \int_0^\psi \frac{1}{\sqrt{1-m\sin^2\theta}}d\theta \quad \text{(here, } x = \sin\psi\text{)}.$$

### (2) Usage

Double precision:

CALL DIEII1 (XI, RM, XO, IERR)

Single precision:

CALL RIEII1 (XI, RM, XO, IERR)

### (3) Arguments

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 2 | RM | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Modulus $m$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $F(x,m)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

### (4) Restrictions

(a) $0.0 \leq \text{XI} \leq 1.0$

(b) $0.0 \leq \text{RM} \leq 1.0$

### (5) Error indicator

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI = 1.0 and RM = 1.0 (overflow) | XO = (Maximum value) is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000 | The Gaussian arithmetic-geometric mean method or Newton method did not converge. | |

(6) **Notes**

(a) If the incomplete elliptic integral of the 1st kind is given as $F(x,k) = \displaystyle\int_0^{\psi} \frac{1}{\sqrt{1-k^2\sin^2\theta}} d\theta$, then the value of $k^2$ must be input for RM.

(b) The incomplete elliptic integral of the 1st kind is also represented as

$$F(\varphi\backslash\alpha) = F(\varphi|m) = \int_0^{\varphi} \frac{1}{\sqrt{1-\sin^2\alpha\sin^2\theta}} d\theta \quad \text{(here, } m = \sin^2\alpha\text{)}.$$

(c) If $m < 0.0$, use 2.9.5 $\left\{\begin{matrix}\text{DIEII3}\\ \text{RIEII3}\end{matrix}\right\}$.

(7) **Example**

(a) Problem
Obtain the value of $F(x,m)$ at $x = 0.3$ and $m = 0.5$.

(b) Input data
XI $= 0.3$ and RM $= 0.5$.

(c) Main program

```
      PROGRAM BIEII1
! *** EXAMPLE OF DIEII1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) XI
      READ (5,*) RM
      WRITE(6,1000) XI,RM
      CALL DIEII1(XI,RM,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIEII1 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'XI =',F6.2,5X,'RM = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF F(X,M)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIEII1 ***

      ** INPUT **

        XI =  0.30     RM =   0.50


      ** OUTPUT**

        IERR =     0

        VALUE OF F(X,M)

          XO =   0.3070549305D+00
```

## 2.9.4 DIEII2, RIEII2
## Incomplete Elliptic Integral of the 2nd Kind

(1) **Function**

Calculates the value of the incomplete elliptic integral of the 2nd kind

$$E(x, m) = \int_0^x \sqrt{\frac{1 - mt^2}{1 - t^2}} dt = \int_0^\psi \sqrt{1 - m \sin^2 \theta} d\theta \quad \text{(here, } x = \sin \psi\text{)}.$$

(2) **Usage**

Double precision:

CALL DIEII2 (XI, RM, XO, IERR)

Single precision:

CALL RIEII2 (XI, RM, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Value of variable $x$ |
| 2 | RM | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Modulus $m$ |
| 3 | XO | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Output | Value of $E(x, m)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0.0 \leq \text{XI} \leq 1.0$

(b) $0.0 \leq \text{RM} \leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000 | The Gaussian arithmetic-geometric mean method or Newton method did not converge. | |

(6) **Notes**

(a) If the incomplete elliptic integral of the 2nd kind is given as $E(x,k) = \int_0^{\psi} \sqrt{1-k^2\sin^2\theta}d\theta$, then the value of $k^2$ must be input for RM.

(b) The incomplete elliptic integral of the 2nd kind is also represented as

$$E(\varphi\backslash\alpha) = E(u|m) = \int_0^{\varphi} \sqrt{1-\sin^2\alpha\sin^2\theta}d\theta \quad \text{(here, } m=\sin^2\alpha, \quad \sin\varphi = \mathrm{sn}\,u\text{)}.$$

Calculating the value of $E(u|m)$ for parameter $u$, use 2.9.11 $\begin{Bmatrix}\text{WIEJEP}\\ \text{VIEJEP}\end{Bmatrix}$.

(c) If $m < 0.0$, use 2.9.5 $\begin{Bmatrix}\text{DIEII3}\\ \text{RIEII3}\end{Bmatrix}$.

(7) **Example**

(a) Problem
Obtain the value of $E(x,m)$ at $x = 0.3$ and $m = 0.5$.

(b) Input data
XI $= 0.3$ and RM $= 0.5$.

(c) Main program

```
      PROGRAM BIEII2
! *** EXAMPLE OF DIEII2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) XI
      READ (5,*) RM
      WRITE(6,1000) XI,RM
      CALL DIEII2(XI,RM,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIEII2 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'XI =',F6.2,5X,'RM = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF E(X,M)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIEII2 ***

      ** INPUT **

        XI =  0.30     RM =   0.50

      ** OUTPUT**

        IERR =     0

        VALUE OF E(X,M)

          XO =   0.3023628305D+00
```

## 2.9.5 DIEII3, RIEII3 Incomplete Modified Elliptic Integral

(1) **Function**

For real numbers $m \geq 0, a, b$ and $x \geq 0$, obtain the value of incomplete modified elliptic integral

$$f(x, m, a, b) \equiv \int_0^x \frac{a + bt^2}{\sqrt{(1+t^2)^3\ (1+mt^2)}} dt.$$

(2) **Usage**

Double precision:

CALL DIEII3 (X, DM, A, B, Y, IERR)

Single precision:

CALL RIEII3 (X, DM, A, B, Y, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | X | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Value of variable $x$ |
| 2 | DM | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Modulus $m$ |
| 3 | A | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Coefficient $a$ of $a + bt^2$ |
| 4 | B | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Coefficient $b$ of $a + bt^2$ |
| 5 | Y | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Output | value of $f(x, m, a, b)$ |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) X $\geq 0$

(b) DM $\geq 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) This function is valid for $m \geq 1$ and the factor $1 + mt^2$ is difference from the incomplete elliptic integrals:

$$F(r, m) = \int_0^r \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}}, E(r, m) = \int_0^r \sqrt{\frac{1-mt^2}{1-t^2}} dt.$$

(b) The first incomplete elliptic integral $\int_0^{\psi} \sqrt{(1 - a \sin^2\theta)^{-1}} d\theta$ is $f(\tan\psi, 1-a, 1, 1)$, here $0 \leq a < 1$ but this can be extended to $a < 1$.

(c) The second incomplete elliptic integral $\int_0^{\psi} \sqrt{1 - a \sin^2\theta} d\theta$ is $f(\tan\psi, 1-a, 1, 1-a)$, here $0 \leq a < 1$ but this can be extended to $a < 1$ .

(7) **Example**

(a) Problem
For $x = 1.0, m = 3.0, a = 4.0$ and $b = 2.0$, obtain incomplete modified elliptic integral.

(b) Input data
X=1.0, DM=3.0, A=4.0 and B=2.0.

(c) Main program

```
      PROGRAM BIEII3
! *** EXAMPLE OF DIEII3 ***
      IMPLICIT NONE
!
      INTEGER IERR
      REAL(8) X,DM,A,B,Y
      DATA X/1.D0/,DM/3.D0/,A/4.D0/,B/2.D0/
!
      WRITE(6,6000) X, DM, A, B
!
      CALL DIEII3(X, DM, A, B, Y, IERR)
!
      WRITE(6,6010) IERR
      WRITE(6,6020) Y
      STOP
!
 6000 FORMAT(/,&
             1X,'***  DIEII3  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     X= ',F10.7, ' DM= ',F10.7, ' A= ',F10.7,&
                ' B= ',F10.7,/)
 6010 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6020 FORMAT(1X,'     Y= ',F10.7)
      END
```

(d) Output results

```
***  DIEII3  ***

 **  INPUT  **

     X=  1.0000000 DM=  3.0000000 A=  4.0000000 B=  2.0000000


 **  OUTPUT  **

     IERR =    0

     Y=  2.5140399
```

## 2.9.6 DIEII4, RIEII4 Incomplete Elliptic Integral of The Weierstrass Type

(1) **Function**

For positives $x, y, z$ and real $p \geq 0$, obtain incomplete elliptic integral of the Weierstrass type

$$f(x, y, z, p) \equiv \frac{1}{2} \int_0^{1/p} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}.$$

(2) **Usage**

Double precision:
CALL DIEII4 (X, Y, Z, P, DI, IERR)
Single precision:
CALL RIEII4 (X, Y, Z, P, DI, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | X | $\left\{\begin{matrix} D \\ R \end{matrix}\right\}$ | 1 | Input | Variable $x$ |
| 2 | Y | $\left\{\begin{matrix} D \\ R \end{matrix}\right\}$ | 1 | Input | Variable $y$ |
| 3 | Z | $\left\{\begin{matrix} D \\ R \end{matrix}\right\}$ | 1 | Input | Variable $z$ |
| 4 | P | $\left\{\begin{matrix} D \\ R \end{matrix}\right\}$ | 1 | Input | Inverse number of top $p$ (See Note (a)) |
| 5 | DI | $\left\{\begin{matrix} D \\ R \end{matrix}\right\}$ | 1 | Output | Incomplete elliptic integral $f(x, y, z, p)$ (See Note (b)) |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) X, Y, Z > 0.0

(b) P $\geq$ 0.0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) If $p$ is equal to zero, then the complete integral

$$\frac{1}{2}\int_0^{\infty}\frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$$

is output to DI.

(b) Note that the the integral defining $f(x,y,z,p)$ is taken from zero to $1/p$ (See Note (a)).

(c) $f(x,y,z,p)$ is symmetric for $x,\ y,\ z.$

(d) If $h \geq 0$ ,

$$f(x+h,y+h,z+h,p)=\frac{1}{2}\int_h^{1/p+h}\frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}.$$

(7) **Example**

(a) Problem

For $x = 1.0, y = 3.0, z = 4.0 and p = 2.0$, obtain incomplete elliptic integral of the Weierstrass type.

(b) Input data

X=1.0, DM=3.0, A=4.0 and B=2.0.

(c) Main program

```
      PROGRAM BIEII4
! *** EXAMPLE OF DIEII4 ***
      IMPLICIT NONE
!
      INTEGER IERR
      REAL(8) X,Y,Z,P,DI
      DATA X/1.D0/,Y/3.D0/,Z/4.D0/,P/2.D0/
!
      WRITE(6,6000) X, Y, Z, P
!
      CALL DIEII4(X, Y, Z, P, DI, IERR)
!
      WRITE(6,6010) IERR
      WRITE(6,6020) DI
!
      STOP
 6000 FORMAT(/,&
             1X,'***  DIEII4  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     X= ',F10.7, '  Y= ',F10.7, ' Z= ',F10.7,&
                ' P= ',F10.7,/)
 6010 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6020 FORMAT(1X,'     DI= ',F10.7)
      END
```

(d) Output results

```
***  DIEII4  ***

 **  INPUT  **

     X=  1.0000000  Y=  3.0000000 Z=  4.0000000 P=  2.0000000


 **  OUTPUT  **

     IERR =    0

     DI=  0.0607049
```

## 2.9.7 WIEJAC, VIEJAC
## Elliptic Functions of Jacobi

(1) **Function**

For $u = X_i$, calculates the values of the elliptic functions of Jacobi $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$, $\mathrm{dn}(u, m)$.
Here these are defined as for $u$, setting $u = F(x, m)$, which is incomplete elliptic integral of the 1st kind with modulus $m$,
$\mathrm{sn}(u, m) = \sin\psi = x$, $\mathrm{cn}(u, m) = \cos\psi$, $\mathrm{dn}(u, m) = \sqrt{1 - m\sin^2\psi}$.

(2) **Usage**

Double precision:
CALL WIEJAC (NV, UI, RM, SN, CN, DN, IERR)
Single precision:
CALL VIEJAC (NV, UI, RM, SN, CN, DN, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | UI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | argument $X_i$ |
| 3 | RM | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Modulus $m$ |
| 4 | SN | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $\mathrm{sn}(X_i, m)$ |
| 5 | CN | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $\mathrm{cn}(X_i, m)$ |
| 6 | DN | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $\mathrm{dn}(X_i, m)$ |
| 7 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{NV} \geq 1$

(b) $0.0 \leq \mathrm{RM} \leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3001 | Restriction (b) was not satisfied. | |

(6) **Notes**

(a) Denoting $u$ by the incomplete elliptic integral of 1st kind given as

$$u = \int_0^{\psi} \frac{d\theta}{\sqrt{1 - m\sin^2\theta}},$$

which implies $\mathrm{sn}(u, m) = \sin\psi$, this subroutine evaluates $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$ and $\mathrm{dn}(u, m)$. On the other hand, denoting $u$ by

$$u = \int_0^{\psi} \frac{d\theta}{\sqrt{1 - k^2\sin^2\theta}},$$

which implies $\mathrm{sn}(u, k) = \sin\psi$, and evaluating $\mathrm{sn}(u, k)$, $\mathrm{cn}(u, k)$ and $\mathrm{dn}(u, k)$, the value $k^2$ should be input to RM. Generally, $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$ and $\mathrm{dn}(u, m)$ are denoted by $\mathrm{sn}(u|m)$, $\mathrm{cn}(u|m)$ and $\mathrm{dn}(u|m)$, respectively.

(7) **Example**

(a) Problem
Suppose that $m$=0.5. Obtain $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$ and $\mathrm{dn}(u, m)$ for $u = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIEJAC
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV,3)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIEJAC', CFNC='S C DN' ,RM=0.5D0)
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIEJAC( NV, XI, RM, XO(1,1),XO(1,2),XO(1,3), IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I,1),XO(I,2),XO(I,3)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',3F10.6 )
      END
```

(c) Output results

```
*** WIEJAC *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
S C DN(  0.000000)= -0.000000  1.000000  1.000000
S C DN(  0.100000)=  0.099751  0.995012  0.997509
S C DN(  0.200000)=  0.198022  0.980198  0.990148
S C DN(  0.300000)=  0.293413  0.955986  0.978241
S C DN(  0.400000)=  0.384672  0.923053  0.962296
S C DN(  0.500000)=  0.470750  0.882266  0.942972
S C DN(  0.600000)=  0.550831  0.834617  0.921028
S C DN(  0.700000)=  0.624340  0.781153  0.897273
S C DN(  0.800000)=  0.690935  0.722917  0.872528
S C DN(  0.900000)=  0.750478  0.660895  0.847580
```

## 2.9.8 WIENMQ, VIENMQ<br>Nome $q$ and Complete Elliptic Integrals

(1) **Function**

For $m = X_i$, calculates the values of the nome $q = e^{-\pi K(m')/K(m)}$, and complementary nome $q' = e^{-\pi K(m)/K(m')}$ (here, $m' = 1 - m$) and of the complete elliptic integrals of the 1st kinds $K(m)$, $K(m')$ and 2nd kinds $E(m)$, $E(m')$.

(2) **Usage**

Double precision:

CALL WIENMQ (NV, RM, Q, QD, K, KD, E, ED, IERR)

Single precision:

CALL VIENMQ (NV, RM, Q, QD, K, KD, E, ED, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | RM | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | Modulus $m = X_i$ |
| 3 | Q | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Nome $q$ |
| 4 | QD | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Complementary nome $q'$ |
| 5 | K | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Value of the complete elliptic integral of the 1st kind $K(m)$ |
| 6 | KD | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Value of the complete elliptic integral of the 1st kind $K(m')$ |
| 7 | E | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Value of the complete elliptic integral of the 2nd kind $E(m)$ |
| 8 | ED | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | Value of the complete elliptic integral of the 2nd kind $E(m')$ |
| 9 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM(i) $\leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | RM(i) = 0.0 or 1.0 (overflow) | If RM(i) = 0.0, KD(i) = (Maximum value) is performed. If RM(i) = 1.0, K(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by RM(i). | |

(6) **Notes**

(a) If it is sufficient to obtain only value $K(m)$ of the complete elliptic integral of the 1st kind or value $E(m)$ of the complete elliptic integral of the 2nd kind, it is more efficient to use 2.9.1 $\left\{\begin{matrix}\text{WIECI1}\\ \text{VIECI1}\end{matrix}\right\}$ and 2.9.2 $\left\{\begin{matrix}\text{WIECI2}\\ \text{VIECI2}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain the value of the nome $q$, complementary nome $q'$, complete elliptic integrals $K(m)$ and $E(m)$ and $K(m')$ and $E(m')$ for $m' = 1 - m$ at modulus $m = 0.1, 0.2, \cdots, 0.9$

(b) Main program

```
      PROGRAM EIENMQ
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=9)
      REAL(8)  XI(NV) , XO(NV,6)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIENMQ', CFNC=' PARAM' )
!
      DNV=NV+1
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIENMQ( NV, XI, XO(1,1),XO(1,2),XO(1,3),&
                           XO(1,4),XO(1,5),XO(1,6), IERR)
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I,1),XO(I,2),XO(I,3),&
                                   XO(I,4),XO(I,5),XO(I,6)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',6F10.6 )
      END
```

(c) Output results

```
*** WIENMQ *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
*** OUTPUT *
IERR=    0
 PARAM(  0.100000)=  0.006585  0.140173  1.612441  2.578092  1.530758  1.104775
 PARAM(  0.200000)=  0.013943  0.099274  1.659624  2.257205  1.489035  1.178490
 PARAM(  0.300000)=  0.022277  0.074690  1.713889  2.075363  1.445363  1.241671
 PARAM(  0.400000)=  0.031883  0.057020  1.777519  1.949568  1.399392  1.298428
 PARAM(  0.500000)=  0.043214  0.043214  1.854075  1.854075  1.350644  1.350644
 PARAM(  0.600000)=  0.057020  0.031883  1.949568  1.777519  1.298428  1.399392
 PARAM(  0.700000)=  0.074690  0.022277  2.075363  1.713889  1.241671  1.445363
 PARAM(  0.800000)=  0.099274  0.013943  2.257205  1.659624  1.178490  1.489035
 PARAM(  0.900000)=  0.140173  0.006585  2.578092  1.612441  1.104775  1.530758
```

## 2.9.9 WIETHE, VIETHE Elliptic Theta Function

(1) **Function**

For $v = X_j$, calculates the values of the elliptic theta functions of Jacobi $\vartheta_i(v, q)$.

$$\vartheta_0(v,q) = \vartheta_4(v,q) = 1 + 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos(2n\pi v)$$

$$\vartheta_1(v,q) = 2q^{1/4}\sum_{n=0}^{\infty}(-1)^n q^{n(n+1)}\sin((2n+1)\pi v)$$

$$\vartheta_2(v,q) = 2q^{1/4}\sum_{n=0}^{\infty} q^{n(n+1)}\cos((2n+1)\pi v)$$

$$\vartheta_3(v,q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}\cos(2n\pi v)$$

(2) **Usage**

Double precision:
  CALL WIETHE (NV, I, V, Q, XO, IERR)
Single precision:
  CALL VIETHE (NV, I, V, Q, XO, IERR)

(3) **Arguments**

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | I | I | 1 | Input | Order $i$ |
| 3 | V | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_j$ |
| 4 | Q | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Nome $q$ |
| 5 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $\vartheta_i(X_j, q)$ |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0 \leq$ I $\leq 4$

(c) $0 \leq$ Q $< 1.0$

(d) $|$V(j)$| < M$
where $M$ ={double precision: $2^{31}$, single precision: $2^{18}$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000+$j$ | Restriction (d) was not satisfied by V(j). | |

(6) **Notes**

(a) $\vartheta_0(v,q) = \vartheta_4(v,q)$.

(b) To obtain the value $F(x,m)$ of the incomplete elliptic integral of the 1st kind corresponding to $(x,m)$, calculate $u = F(x,m)$ from 2.9.3 $\begin{Bmatrix} \text{DIEII1} \\ \text{RIEII1} \end{Bmatrix}$, calculate nome $q$ and the value $K(m)$ of the complete elliptic integrals of the 1st kinds from 2.9.8 $\begin{Bmatrix} \text{WIENMQ} \\ \text{VIENMQ} \end{Bmatrix}$, and then apply this subroutine for $v = \frac{u}{2K(m)}$.

(c) $\vartheta_4'(v,q)$ can also be obtained from the following expression: $\vartheta_4'(v,q) = 2Z(u)K(m)\vartheta_4(v,q)$.

(7) **Example**

(a) Problem
Obtain $\vartheta_i(v,q)$ for $i = 3$, $v = 0.0, 0.1, 0.2, \cdots, 0.9$ and $q = 0.5$.

(b) Main program

```
      PROGRAM EIETHE
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIETHE', CFNC='  JTH3' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      II=3
      Q = 0.5D0
      CALL WIETHE( NV, II, XI, Q,  XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
```

```
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIETHE *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
  JTH3(  0.000000)=  2.128937
  JTH3(  0.100000)=  1.846412
  JTH3(  0.200000)=  1.204739
  JTH3(  0.300000)=  0.593025
  JTH3(  0.400000)=  0.230793
  JTH3(  0.500000)=  0.121124
  JTH3(  0.600000)=  0.230793
  JTH3(  0.700000)=  0.593025
  JTH3(  0.800000)=  1.204739
  JTH3(  0.900000)=  1.846412
```

### 2.9.10 WIEJZT, VIEJZT
### Zeta Function of Jacobi

(1) **Function**

For $u = X_i$, calculates the value of the zeta function of Jacobi

$$Z(u) = \frac{\Theta'(u)}{\Theta(u)} \quad (\text{where } \Theta(u) = \vartheta_4(v, q) = \vartheta_4(u/2K(m), q))$$

(2) **Usage**

Double precision:
CALL WIEJZT (NV, UI, RM, XO, IERR)
Single precision:
CALL VIEJZT (NV, UI, RM, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | UI | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | NV | Input | $X_i$ (value of the incomplete elliptic integral of the 1st kind $F(x, m)$) |
| 3 | RM | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Input | Modulus $m$ |
| 4 | XO | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | NV | Output | $Z(X_i)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM $\leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3001 | Restriction (b) was not satisfied. | |

(6) **Notes**

(a) The value of $u$ can be obtained from $v$ and $K(m)$ by using the expression $u = 2K(m)v$.

(7) **Example**

(a) Problem

Obtain $Z(u)$ for $u = 0.0, 0.1, 0.2, \cdots, 0.9$ with a modulus $m = 0.5$.

(b) Main program

```
      PROGRAM EIEJZT
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIEJZT', CFNC='     Z' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      RM = 0.5D0
      CALL WIEJZT( NV, XI, RM,  XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIEJZT *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     Z(  0.000000)= -0.000000
     Z(  0.100000)=  0.026987
     Z(  0.200000)=  0.052988
     Z(  0.300000)=  0.077076
     Z(  0.400000)=  0.098434
     Z(  0.500000)=  0.116389
     Z(  0.600000)=  0.130442
     Z(  0.700000)=  0.140276
     Z(  0.800000)=  0.145748
     Z(  0.900000)=  0.146872
```

### 2.9.11 WIEJEP, VIEJEP Epsilon Function of Jacobi

(1) **Function**

For $u = X_i$, calculates the value of the epsilon function of Jacobi

$$E(u|m) = \int_0^x \sqrt{\frac{1-mt^2}{1-t^2}}dt = \int_0^u \mathrm{dn}^2(t)dt \quad \left(\text{here, } u = \int_0^x \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}}\right).$$

(2) **Usage**

Double precision:
CALL WIEJEP (NV, UI, RM, XO, IERR)
Single precision:
CALL VIEJEP (NV, UI, RM, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | UI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i$ (value of the incomplete elliptic integral of the 1st kind $F(x,m)$) |
| 3 | RM | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Modulus $m$ |
| 4 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $E(X_i\|m)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM $\leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3001 | Restriction (b) was not satisfied. | |

(6) **Notes**

(a) Epsilon function of Jacobi $E(u|m)$ is the same as the incomplete elliptic integral of the 2nd kind $E(\varphi\backslash\alpha)$ and the following relation holds:

$$E(\varphi\backslash\alpha) = E(u|m) = \int_0^{\varphi} \sqrt{1-\sin^2\alpha\sin^2\theta}d\theta \quad \text{(where } m=\sin^2\alpha, \quad \sin\varphi = \mathrm{sn}\,u)$$

(See 2.9.2 $\left\{\begin{matrix}\text{WIECI2}\\ \text{VIECI2}\end{matrix}\right\}$).

(7) **Example**

(a) Problem

Obtain $E(u|m)$ for $u = 0.0, 0.1, 0.2, \cdots, 0.9$, with modulus $m = 0.5$.

(b) Main program

```
      PROGRAM EIEJEP
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIEJEP', CFNC='     E' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      RM = 0.5D0
      CALL WIEJEP( NV, XI, RM,  XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIEJEP *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     E(  0.000000)=  0.000000
     E(  0.100000)=  0.099834
     E(  0.200000)=  0.198682
     E(  0.300000)=  0.295618
     E(  0.400000)=  0.389823
     E(  0.500000)=  0.480625
     E(  0.600000)=  0.567526
     E(  0.700000)=  0.650208
     E(  0.800000)=  0.728526
     E(  0.900000)=  0.802498
```

## 2.9.12 WIEJTE, VIEJTE
## Theta Function of Jacobi

(1) **Function**

For $u = X_i$, calculates the value of the theta function of Jacobi

$$\Theta(u) = \vartheta_4(v, q) = \vartheta_4(u/2K(m), q).$$

(2) **Usage**

Double precision:
CALL WIEJTE (NV, UI, RM, XO, IERR)
Single precision:
CALL VIEJTE (NV, UI, RM, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | UI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i$ (value of the incomplete elliptic integral of the 1st kind $F(x, m)$) |
| 3 | RM | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Modulus $m$ |
| 4 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $\Theta(X_i)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM $\leq 1.0$

(c) $\frac{|\text{UI(i)}|}{2K(m)} < M$
where, $M$ ={double precision: $2^{31}$, single precision: $2^{18}$}, and $K(m)$ denotes the complete elliptic integral of the 1st kind with $m$=RM.

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3001 | Restriction (b) was not satisfied. | |
| 4000+i | Restriction (c) was not satisfied by UI(i). | |

(6) **Notes**

(a) $\Theta'(u)$ can be obtained from this subroutine with $Z(u)$ obtained from 2.9.10 $\left\{\begin{matrix}\text{WIEJZT}\\ \text{VIEJZT}\end{matrix}\right\}$ using the relation

$$\Theta'(u) = Z(u)\Theta(u).$$

(7) **Example**

(a) Problem
Obtain $\Theta(u)$ for $u = 0.0, 0.1, 0.2, \cdots, 0.9$, with modulus $m = 0.5$.

(b) Main program

```
      PROGRAM EIEJTE
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIEJTE', CFNC='JTHETA' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      RM = 0.5D0
      CALL WIEJTE( NV, XI, RM,  XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIEJTE *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
JTHETA(  0.000000)=  0.913579
JTHETA(  0.100000)=  0.914816
JTHETA(  0.200000)=  0.918493
JTHETA(  0.300000)=  0.924504
JTHETA(  0.400000)=  0.932677
JTHETA(  0.500000)=  0.942777
JTHETA(  0.600000)=  0.954517
JTHETA(  0.700000)=  0.967560
JTHETA(  0.800000)=  0.981533
JTHETA(  0.900000)=  0.996035
```

### 2.9.13 WIEPAI, VIEPAI Pi Function

(1) **Function**

For $u = X_i$, calculates the value of the pi function

$$\Pi(u,\alpha) = m \operatorname{sn}\alpha \operatorname{cn}\alpha \operatorname{dn}\alpha \int_0^u \frac{\operatorname{sn}^2 t dt}{1 - m\operatorname{sn}^2\alpha \operatorname{sn}^2 t}.$$

(2) **Usage**

Double precision:

CALL WIEPAI (NV, UI, ALF, RM, XO, IERR)

Single precision:

CALL VIEPAI (NV, UI, ALF, RM, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | UI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ (value of the incomplete elliptic integral of the 1st kind $F(x,m)$) |
| 3 | ALF | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | $\alpha$ |
| 4 | RM | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Modulus $m$ |
| 5 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\Pi(X_i,\alpha)$ |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $0.0 \leq$ RM $< 1.0$

(c) $|\frac{|\text{UI(i)}|-\text{ALF}}{2K(m)}| < M$
where, $M$ ={double precision: $2^{31}$, single precision: $2^{18}$}, and $K(m)$ denotes the complete elliptic integral of the 1st kind with $m$=RM.

(d) $|\frac{\text{ALF}}{2K(m)}| < M$
where, $M$ and $K(m)$ are the same as Restriction (c).

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3001 | Restriction (b) was not satisfied. | |
| 4000 | Restriction (d) was not satisfied. | |
| 4000+i | Restriction (c) was not satisfied by UI(i). | |

(6) **Notes**

(a) Pi Function $\Pi(u,\alpha)$ is represented as:

$$\Pi(u,\alpha) = u\frac{\Theta'(\alpha)}{\Theta(\alpha)} + \frac{1}{2}\log_e\frac{\Theta(u-\alpha)}{\Theta(u+\alpha)}.$$

(b) The incomplete elliptic integral of the 3rd kind $F(z)$ is represented as

$$F(z) = \int_0^z \frac{dz}{(1-a^2z^2)\sqrt{(1-z^2)(1-k^2z^2)}} = \frac{\operatorname{sn}\alpha}{\operatorname{cn}\alpha\operatorname{dn}\alpha}\Pi(u,\alpha) + u$$

with $z = \operatorname{sn} u$, $a^2 = k^2\operatorname{sn}^2\alpha$.

(7) **Example**

(a) Problem

Obtain $\Pi(u,\alpha)$ for $u = 0.0, 0.1, 0.2, \cdots, 0.9$, $\alpha = 1.7$ and $m = 0.5$.

(b) Main program

```
      PROGRAM EIEPAI
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIEPAI', CFNC='JACPAI' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      RM = 0.5D0
      ALF= 1.7D0
      CALL WIEPAI( NV, XI, ALF, RM,  XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIEPAI *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
```

```
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=     0
JACPAI(  0.000000)=  0.000000
JACPAI(  0.100000)=  0.000013
JACPAI(  0.200000)=  0.000103
JACPAI(  0.300000)=  0.000346
JACPAI(  0.400000)=  0.000821
JACPAI(  0.500000)=  0.001601
JACPAI(  0.600000)=  0.002762
JACPAI(  0.700000)=  0.004373
JACPAI(  0.800000)=  0.006501
JACPAI(  0.900000)=  0.009204
```

## 2.10 INDEFINITE INTEGRALS OF ELEMENTARY FUNCTIONS

### 2.10.1 WIIEXP, VIIEXP Exponential Integral

(1) **Function**

For $x = X_i$, calculates the value of the exponential integral

$$\overline{\mathrm{Ei}}(x) = P\int_{-\infty}^{x} \frac{e^t}{t} dt \quad (x > 0.0)$$

$$\mathrm{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt = \int_{-\infty}^{x} \frac{e^t}{t} dt \quad (x < 0.0)$$

where $P$ denotes Cauchy's principal value.

(2) **Usage**

Double precision:
CALL WIIEXP (NV, XI, XO, IERR)
Single precision:
CALL VIIEXP (NV, XI, XO, 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 | NV | I | 1 | Input | number of input data |
| 2 | XI | {D / R} | NV | Input | $X_i$ |
| 3 | XO | {D / R} | NV | Output | $X_i > 0.0 : \overline{\mathrm{Ei}}(X_i)$<br>$X_i < 0.0 : \mathrm{Ei}(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI(i) $< -\mathrm{M}_1$ (See Note (b)) (underflow) | XO(i) = 0.0 is performed. |
| 2000 | XI(i) = 0.0 (overflow) | XO(i) = (Minimum value) is performed. |
| 2100 | XI(i) $> \mathrm{M}_2$ (See Note (c)) (overflow) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

## (6) Notes

(a) In the case where $x = \text{XI(i)} > 0.0$, $\overline{\text{Ei}}(x) = P\int_{-\infty}^{x} \frac{e^t}{t} dt$ is evaluated, and in another case $x = \text{XI(i)} < 0.0$ $\text{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt = \int_{-\infty}^{x} \frac{e^t}{t} dt$ is also evaluated.

(b) When IERR becomes 1000 in this subroutine, the value of $M_1$ is as follows:
$M_1 = \{$double precision: 702.0, single precision: 83.0$\}$

(c) When IERR becomes 2100 in this subroutine, the value of $M_2$ is as follows:
$M_2 = \{$double precision: 709.782, single precision: 88.722$\}$

(d) The value of the exponential integral may defined as follows:

$$E_1(z) = \int_{z}^{\infty} \frac{e^t}{t} dt \quad (|\arg z| < \pi).$$

In this case, $E_1(x) = -\,\text{Ei}(-x) \quad (x > 0)$ holds.

## (7) Example

(a) Problem
Obtain $\overline{\text{Ei}}(x)$ for $x = 0.1, 0.2, \cdots, 1.0$.

(b) Main program

```
      PROGRAM EIIEXP
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIIEXP', CFNC='    Ei' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=I/DNV
 1000 CONTINUE
!
      CALL WIIEXP( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIIEXP *
*** INPUT *
XI( 1)=  0.100000
XI( 2)=  0.200000
XI( 3)=  0.300000
XI( 4)=  0.400000
XI( 5)=  0.500000
XI( 6)=  0.600000
XI( 7)=  0.700000
XI( 8)=  0.800000
XI( 9)=  0.900000
XI(10)=  1.000000
*** OUTPUT *
IERR=    0
    Ei(  0.100000)= -1.622813
    Ei(  0.200000)= -0.821761
    Ei(  0.300000)= -0.302669
    Ei(  0.400000)=  0.104765
    Ei(  0.500000)=  0.454220
    Ei(  0.600000)=  0.769881
    Ei(  0.700000)=  1.064907
    Ei(  0.800000)=  1.347397
    Ei(  0.900000)=  1.622812
    Ei(  1.000000)=  1.895118
```

### 2.10.2 WIILOG, VIILOG Logarithmic Integral

(1) **Function**

For $x = X_i$, calculates the value of the logarithmic integral

$$\mathrm{Li}(x) = \int_0^x \frac{1}{\log_e(t)} dt.$$

(2) **Usage**

Double precision:
CALL WIILOG (NV, XI, XO, IERR)
Single precision:
CALL VIILOG (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Input | $X_i$ |
| 3 | XO | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NV | Output | $\mathrm{Li}(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) XI(i) $\geq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI(i) = 1.0 (overflow) | XO(i) = (Minimum value) is performed. |
| 2100 | log(XI(i)) > $\mathrm{M}_2$ (See Note (a).) | XO(i) = (Maximum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | |

## (6) **Notes**

(a) When IERR becomes 2100 in this subroutine, the value of $M_2$ is as follows:
$M_2 =${double precision: 709.782, single precision: 88.722}

## (7) **Example**

(a) Problem
Obtain $\mathrm{Li}(x)$ for $x = 1.1, 1.2, \cdots, 2.0$.

(b) Main program

```
      PROGRAM EIILOG
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIILOG', CFNC='    Li' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=1.D0+I/DNV
 1000 CONTINUE
!
      CALL WIILOG( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIILOG *
*** INPUT *
XI( 1)=  1.100000
XI( 2)=  1.200000
XI( 3)=  1.300000
XI( 4)=  1.400000
XI( 5)=  1.500000
XI( 6)=  1.600000
XI( 7)=  1.700000
XI( 8)=  1.800000
XI( 9)=  1.900000
XI(10)=  2.000000
*** OUTPUT *
IERR=    0
    Li(  1.100000)= -1.675773
    Li(  1.200000)= -0.933787
    Li(  1.300000)= -0.480178
    Li(  1.400000)= -0.144991
    Li(  1.500000)=  0.125065
    Li(  1.600000)=  0.353748
    Li(  1.700000)=  0.553744
    Li(  1.800000)=  0.732637
    Li(  1.900000)=  0.895327
    Li(  2.000000)=  1.045164
```

### 2.10.3 DIISIN, RIISIN
### Sine Integral

(1) **Function**

Calculates the value of the sine integral

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t} dt.$$

(2) **Usage**

Double precision:

CALL DIISIN (XI, XO, IERR)

Single precision:

CALL RIISIN (XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\left\{\begin{array}{c}\mathrm{D}\\ \mathrm{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 2 | XO | $\left\{\begin{array}{c}\mathrm{D}\\ \mathrm{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{Si}(x)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

None

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |

(6) **Notes**

None

(7) **Example**

(a) Problem

Obtain the value of $\mathrm{Si}(x)$ at $x = 1.0$.

(b) Input data

XI = 1.0.

(c) Main program

```
      PROGRAM BIISIN
! *** EXAMPLE OF DIISIN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) XI
      WRITE(6,1000) XI
      CALL DIISIN(XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIISIN ***',/,/,6X,'** INPUT **',&
      /,/,8X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF SI(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIISIN ***

      ** INPUT **

        XI =   1.00


      ** OUTPUT**

        IERR =     0

        VALUE OF SI(X)

          XO =   0.9460830704D+00
```

### 2.10.4 DIICOS, RIICOS Cosine Integral

(1) **Function**

Calculates the value of the cosine integral

$$\mathrm{Ci}(x) = -\int_x^\infty \frac{\cos t}{t} dt.$$

(2) **Usage**

Double precision:
CALL DIICOS (XI, XO, IERR)
Single precision:
CALL RIICOS (XI, XO, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 2 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Value of $\mathrm{Ci}(x)$ |
| 3 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0.0 \leq \mathrm{XI} \leq M$
where, $M$={double precision: $2^{50}\pi$, single precision: $2^{18}\pi$}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | XI = 0.0 (overflow) | XO = (Minimum value) is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted.<br>(See Note (a)) |

(6) **Notes**

(a) For IERR=3000, if XI is sufficiently large, then the value of Ci($x$) will be a value extremely close to 0.0.

(7) **Example**

(a) Problem
Obtain the value of Ci($x$) at $x = 1.0$.

(b) Input data
XI = 1.0.

(c) Main program

```
      PROGRAM BIICOS
! *** EXAMPLE OF DIICOS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) XI
      WRITE(6,1000) XI
      CALL DIICOS(XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DIICOS ***',/,/,6X,'** INPUT **',&
      /,/,8X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF CI(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIICOS ***

      ** INPUT **

        XI =   1.00


      ** OUTPUT**

        IERR =     0

        VALUE OF CI(X)

          XO =   0.3374039229D+00
```

## 2.10.5 WIIFSI, VIIFSI Fresnel Sine Integral

(1) **Function**

For $x = X_i$, calculates the value of the Fresnel sine integral

$$S(x) = \int_0^x \sin(\frac{\pi}{2}t^2)dt.$$

(2) **Usage**

Double precision:
CALL WIIFSI (NV, XI, XO, IERR)
Single precision:
CALL VIIFSI (NV, XI, XO, 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 | NV | I | 1 | Input | number of input data |
| 2 | XI | {D R} | NV | Input | $X_i$ |
| 3 | XO | {D R} | NV | Output | $S(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $|\text{XI(i)}| \leq M$
where $M$={double precision: 47453132.0, single precision: 724.07734}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | Processing is aborted. (See Note (b)) |

(6) **Notes**

(a) If the Fresnel sine integral is given as $S(x) = \dfrac{1}{\sqrt{2\pi}}\displaystyle\int_0^x \frac{\sin t}{\sqrt{t}}dt$, then XI(i) must be $\sqrt{\dfrac{2x}{\pi}}$.

(b) For IERR $= 3000 + i$, if $|x|$ is sufficiently large ($x = \text{XI(i)}$), $S(x)$ will be a value extremely close to 0.5 (XI(i) $> 0.0$), and to $-0.5$(XI(i) $< 0.0$).

## (7) **Example**

(a) Problem

Obtain $S(x)$ for $x = 0.0, 0.1, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIIFSI
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIIFSI', CFNC='     S' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIIFSI( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIIFSI *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     S(  0.000000)=  0.000000
     S(  0.100000)=  0.000524
     S(  0.200000)=  0.004188
     S(  0.300000)=  0.014117
     S(  0.400000)=  0.033359
     S(  0.500000)=  0.064732
     S(  0.600000)=  0.110540
     S(  0.700000)=  0.172136
     S(  0.800000)=  0.249341
     S(  0.900000)=  0.339776
```

## 2.10.6 WIIFCO, VIIFCO Fresnel Cosine Integral

(1) **Function**

For $x = X_i$, calculates the value of the Fresnel cosine integral

$$C(x) = \int_0^x \cos(\frac{\pi}{2}t^2)dt.$$

(2) **Usage**

Double precision:
  CALL WIIFCO (NV, XI, XO, IERR)
Single precision:
  CALL VIIFCO (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | NV | Output | $C(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(b) $|\text{XI(i)}| \leq M$
where $M$={double precision: 47453132.0, single precision: 724.07734}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3000+$i$ | Restriction (b) was not satisfied by XI(i). | Processing is aborted. (See Note (b)) |

(6) **Notes**

(a) If the Fresnel cosine integral is given as $C(x) = \dfrac{1}{\sqrt{2\pi}} \displaystyle\int_0^x \frac{\cos t}{\sqrt{t}}dt$, then the value of XI(i) must be $\sqrt{\dfrac{2x}{\pi}}$.

(b) For IERR $= 3000 + i$, if $|x|$ is sufficiently large ($x =$ XI(i)), $C(x)$ will be a value extremely close to 0.5 (XI(i) $> 0.0$) and to $-0.5$(XI(i) $< 0.0$).

(7) **Example**

(a) Problem

Obtain $C(x)$ for $x = 0.0, 0.1, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIIFCO
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIIFCO', CFNC='     C' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIIFCO( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIIFCO *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     C(  0.000000)=  0.000000
     C(  0.100000)=  0.099998
     C(  0.200000)=  0.199921
     C(  0.300000)=  0.299401
     C(  0.400000)=  0.397481
     C(  0.500000)=  0.492344
     C(  0.600000)=  0.581095
     C(  0.700000)=  0.659652
     C(  0.800000)=  0.722844
     C(  0.900000)=  0.764823
```

## 2.10.7 WIIDAW, VIIDAW
## Dawson Integral

(1) **Function**

For $x = X_i$, calculates the value of the Dawson integral

$$e^{-x^2}\int_0^x e^{t^2}dt.$$

(2) **Usage**

Double precision:
CALL WIIDAW (NV, XI, XO, IERR)
Single precision:
CALL VIIDAW (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $e^{-X_i^2}\int_0^{X_i} e^{t^2}dt$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Obtain $e^{-x^2}\int_0^x e^{t^2}dt$ for $x = 0.0, 0.1, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIIDAW
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIIDAW', CFNC='DAWSON' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIIDAW( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIIDAW *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
DAWSON(  0.000000)=  0.000000
DAWSON(  0.100000)=  0.099336
DAWSON(  0.200000)=  0.194751
DAWSON(  0.300000)=  0.282632
DAWSON(  0.400000)=  0.359943
DAWSON(  0.500000)=  0.424436
DAWSON(  0.600000)=  0.474763
DAWSON(  0.700000)=  0.510504
DAWSON(  0.800000)=  0.532102
DAWSON(  0.900000)=  0.540724
```

## 2.10.8 WIICND, VIICND
## Normal Distribution Function

(1) **Function**

For $x = X_i$, calculates the value of the normal distribution function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt.$$

(2) **Usage**

Double precision:

CALL WIICND (NV, XI, XO, IERR)

Single precision:

CALL VIICND (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\Phi(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Obtain $\Phi(x)$ for $x = 0.0, 0.1, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIICND
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIICND', CFNC='   PHI' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIICND( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIICND *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
   PHI(  0.000000)=  0.000000
   PHI(  0.100000)=  0.039828
   PHI(  0.200000)=  0.079260
   PHI(  0.300000)=  0.117911
   PHI(  0.400000)=  0.155422
   PHI(  0.500000)=  0.191462
   PHI(  0.600000)=  0.225747
   PHI(  0.700000)=  0.258036
   PHI(  0.800000)=  0.288145
   PHI(  0.900000)=  0.315940
```

### 2.10.9 WIICNC, VIICNC Complementary Normal Distribution Function

(1) **Function**

For $x = X_i$, calculates the value of the normal distribution function

$$\Psi(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt.$$

(2) **Usage**

Double precision:
CALL WIICNC (NV, XI, XO, IERR)
Single precision:
CALL VIICNC (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\begin{Bmatrix} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{Bmatrix}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\begin{Bmatrix} D \\ R \end{Bmatrix}$ | NV | Input | $X_i$ |
| 3 | XO | $\begin{Bmatrix} D \\ R \end{Bmatrix}$ | NV | Output | $\Psi(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | XI(i) $>$ M (See Note (a)) (underflow) | XO(i) = 0.0 is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) When IERR becomes 1000 in this subroutine, the value of $M$ is as follows:
$M = \{$double precision: 38.485, single precision: 13.0$\}$

(7) **Example**

(a) Problem
Obtain $\Phi(x)$ for $x = 0.0, 0.1, 0.2, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIICNC
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIICNC', CFNC='   PSI' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIICNC( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIICNC *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
   PSI(  0.000000)=  0.500000
   PSI(  0.100000)=  0.460172
   PSI(  0.200000)=  0.420740
   PSI(  0.300000)=  0.382089
   PSI(  0.400000)=  0.344578
   PSI(  0.500000)=  0.308538
   PSI(  0.600000)=  0.274253
   PSI(  0.700000)=  0.241964
   PSI(  0.800000)=  0.211855
   PSI(  0.900000)=  0.184060
```

## 2.11 THE FUNCTIONS RELATED TO THE ERROR FUNCTIONS

### 2.11.1 WIERRF, VIERRF
### Error Function

(1) **Function**

For $x = X_i$, evaluate error function $\mathrm{Erf}(x)$.

(2) **Usage**

Double precision:

CALL WIERRF (NV, XV, YV, IERR)

Single precision:

CALL VIERRF (NV, XV, YV, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$

R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number of inputs |
| 2 | XV | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $x_i$ |
| 3 | YV | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $\mathrm{Erf}(x_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV>0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Evaluate values of $\mathrm{Erf}(x_i)$ with $i = 1, 2, \cdots, 10$ where each $x_i$ is given as the functional value is $0.1^i$.

(b) Main program

```
      PROGRAM EIERRF
      IMPLICIT REAL(8)(A-H,O-Z)
      INTEGER     NV
      PARAMETER(  NV = 10 )
      REAL(8)   XV(NV), YV(NV)
      ITER=0
      DO 1000 I=1,NV
         CALL DIIERF(1.D-1**I, XV(I), ITER, IERR)
         IF(IERR.GT.0) STOP
 1000 CONTINUE
      WRITE(6,10)
      WRITE(6,20)
      CALL WIERRF(NV, XV, YV, IERR)
      DO 2000 I=1,NV
      WRITE(6,6000) I, XV(I)
 2000 CONTINUE
      WRITE(6,30)
      WRITE(6,40) IERR
      DO 3000 I=1,NV
      WRITE(6,6500) I, 1.D0 - YV(I)
 3000 CONTINUE
      STOP
   10 FORMAT(1X,' *** WIERRF *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,/,/,' *** OUTPUT *** ',/,/)
   40 FORMAT(1X,'IERR = ',I4,/,/)
 6000 FORMAT(1X,I2,' TH INPUT  VALUES = ',E15.7)
 6500 FORMAT(1X,I2,' TH OUTPUT  VALUES = ',E15.7)
      END
```

(c) Output results

```
 *** WIERRF ***

 *** INPUT ***

 1 TH INPUT  VALUES =   0.1163087E+01
 2 TH INPUT  VALUES =   0.1821386E+01
 3 TH INPUT  VALUES =   0.2326754E+01
 4 TH INPUT  VALUES =   0.2751064E+01
 5 TH INPUT  VALUES =   0.3123413E+01
 6 TH INPUT  VALUES =   0.3458911E+01
 7 TH INPUT  VALUES =   0.3766563E+01
 8 TH INPUT  VALUES =   0.4052237E+01
 9 TH INPUT  VALUES =   0.4320005E+01
10 TH INPUT  VALUES =   0.4572825E+01

*** OUTPUT ***

IERR =    0

 1 TH OUTPUT  VALUES =   0.1000000E+00
 2 TH OUTPUT  VALUES =   0.1000000E-01
 3 TH OUTPUT  VALUES =   0.1000000E-02
 4 TH OUTPUT  VALUES =   0.1000000E-03
 5 TH OUTPUT  VALUES =   0.1000000E-04
 6 TH OUTPUT  VALUES =   0.1000000E-05
 7 TH OUTPUT  VALUES =   0.1000000E-06
 8 TH OUTPUT  VALUES =   0.1000000E-07
 9 TH OUTPUT  VALUES =   0.1000000E-08
10 TH OUTPUT  VALUES =   0.1000000E-09
```

### 2.11.2 WIERFC, VIERFC Co-Error Function

(1) **Function**

For $x = X_i$, evaluate co-error function Erfc($x$).

(2) **Usage**

Double precision:
CALL WIERFC (NV, XV, YV, IERR)
Single precision:
CALL VIERFC (NV, XV, YV, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number of inputs |
| 2 | XV | $\left\{\begin{array}{l}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $x_i$ |
| 3 | YV | $\left\{\begin{array}{l}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | Erfc($x_i$) |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV>0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Evaluate values of Erfc($x_i$) for $x_i$ with $i = 1, 2, \cdots, 10$ given as the functional value equals to $0.1^i$.

(b) Main program

```
      PROGRAM EIERFC
      IMPLICIT REAL(8)(A-H,O-Z)
      INTEGER     NV
      PARAMETER(  NV = 10 )
      REAL(8)   XV(NV), YV(NV)
      DO 1000 I=1,NV
         CALL DIIERF(1.D-1**I, XV(I), 0, IERR)
         IF(IERR.GT.0) STOP
 1000 CONTINUE
      WRITE(6,10)
      WRITE(6,20)
      CALL WIERFC(NV, XV, YV, IERR)
      DO 2000 I=1,NV
      WRITE(6,6000) I, XV(I)
 2000 CONTINUE
      WRITE(6,30)
      WRITE(6,40) IERR
      DO 3000 I=1,NV
      WRITE(6,6500) I, YV(I)
 3000 CONTINUE
      STOP
   10 FORMAT(1X,' *** WIERFC *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,/,/,' *** OUTPUT *** ',/,/)
   40 FORMAT(1X,'IERR = ',I4,/,/)
 6000 FORMAT(1X,I2,' TH INPUT  VALUES = ',E15.7)
 6500 FORMAT(1X,I2,' TH OUTPUT  VALUES = ',E15.7)
      END
```

(c) Output results

```
 *** WIERFC ***

 *** INPUT ***

 1 TH INPUT  VALUES =   0.1163087E+01
 2 TH INPUT  VALUES =   0.1821386E+01
 3 TH INPUT  VALUES =   0.2326754E+01
 4 TH INPUT  VALUES =   0.2751064E+01
 5 TH INPUT  VALUES =   0.3123413E+01
 6 TH INPUT  VALUES =   0.3458911E+01
 7 TH INPUT  VALUES =   0.3766563E+01
 8 TH INPUT  VALUES =   0.4052237E+01
 9 TH INPUT  VALUES =   0.4320005E+01
10 TH INPUT  VALUES =   0.4572825E+01

*** OUTPUT ***

IERR =    0

 1 TH OUTPUT  VALUES =   0.1000000E+00
 2 TH OUTPUT  VALUES =   0.1000000E-01
 3 TH OUTPUT  VALUES =   0.1000000E-02
 4 TH OUTPUT  VALUES =   0.1000000E-03
 5 TH OUTPUT  VALUES =   0.1000000E-04
 6 TH OUTPUT  VALUES =   0.1000000E-05
 7 TH OUTPUT  VALUES =   0.1000000E-06
 8 TH OUTPUT  VALUES =   0.1000000E-07
 9 TH OUTPUT  VALUES =   0.1000000E-08
10 TH OUTPUT  VALUES =   0.1000000E-09
```

## 2.11.3 DIIERF, RIIERF Inverse of Co-Error Function

(1) **Function**

Evaluate $\mathrm{Erfc}^{-1}$ i.e, for $0 < x \leq 1$, evaluate $y$ satisfying $\mathrm{Erfc}(y) = x$.

(2) **Usage**

Double precision:
CALL DIIERF (X, Y, ITER, IERR)
Single precision:
CALL RIIERF (X, Y, ITER, 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 | X | {D, R} | 1 | Input | $x$ |
| 2 | Y | {D, R} | 1 | Output | $y$ |
| 3 | ITER | I | 1 | Input | Maximum number of iterations (Normal:10) (See Note (d)) |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) 0 <X $\leq$ 1

(b) ITER $\geq$ 1 ( Excluding the case where 0 or an negative value is input to be reset to the default value )

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | X = 1 | Y = 0 is performed. |
| 3000 | X > 1 | Processing is aborted. |
| 3500 | X $\leq$ 0 | |
| 4000 | The solution did not converge within the maximum number of iterations | |

(6) **Notes**

(a) For $2 > x > 1$, $y$ can be obtained by $y = -\,\mathrm{Erfc}^{-1}(2-x)$ according to the following relation:

$$\frac{2}{\sqrt{\pi}} \int_{-y}^{\infty} e^{-t^2} dt = 2 - x.$$

(b) $\mathrm{Erfc}^{-1}(0) = \infty$, $\mathrm{Erfc}^{-1}(2) = -\infty$.

(c) Inverse of $\mathrm{Erf}(x)$ is $\mathrm{Erfc}^{-1}(1-x)$.

(d) The maximum count of iteration is set to 10 when ITER is less than 1.

(7) **Example**

(a) Problem

Evaluate $y$ satisfying ERF($y$)=0.9999 and check this result.

(b) Main program

```
      PROGRAM BIIERF
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(ONE=1.D0)
      X=ONE-0.9999D0
      WRITE(6,90)
      WRITE(6,91)
      WRITE(6,5000) X
      CALL DIIERF(X, Y, 0, IERR)
      WRITE(6,92)
      WRITE(6,93) IERR
      CALL WIERRF(1, Y, Z, IERR)
      IF(IERR.NE.0) STOP
      WRITE(6,6000) ONE-X, Y, Z
      STOP
   90 FORMAT(1X,' *** DIIERF *** ',/,/)
   91 FORMAT(1X,' *** INPUT *** ',/,/)
   92 FORMAT(1X,' *** OUTPUT *** ',/,/)
   93 FORMAT(1X,'  IERR= ',I4,/,/)
 5000 FORMAT(1X,' X = ',E15.7,/,/)
 6000 FORMAT(1X,' 1-X=',E15.7,' OUTPUT Y=',&
                          E15.7,&
                ' TEST VALUE =',E15.7)
      END
```

(c) Output results

```
*** DIIERF ***

*** INPUT ***

X =   0.1000000E-03

*** OUTPUT ***

 IERR=    0

1-X=  0.9999000E+00 OUTPUT Y=  0.2751064E+01 TEST VALUE =  0.9999000E+00
```

## 2.11.4 JIIERF, IIIERF
## Error Function for Complex Arguments

(1) **Function**

For $z{=}Z_i$, evaluate error function with complex argument $e^{-z^2}\,\mathrm{Erfc}(-iz)$.

(2) **Usage**

Double precision:
  CALL JIIERF (NV, Z, W, IERR)
Single precision:
  CALL IIIERF (NV, Z, W, 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 | NV | I | 1 | Input | Number of input values |
| 2 | Z | {Z / C} | NV | Input | $z$ |
| 3 | W | {Z / C} | NV | Output | $e^{-z^2}\,\mathrm{Erfc}(-iz)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $>$ 0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred. (See Note (b)) | |

### (6) Notes

(a) If the imaginary part of $z$ is close to 0.0, the precision becomes rather low (within an error of order $10^{-10}$ in the double precision case, though).

(b) If $\Im(z)$ is negative and $-\Re(z^2)$ is almost log(Maximum positive), overflow occurs.

### (7) Example

(a) Problem
Evaluate values in the integration pass $|z - 3i| = 1$.

(b) Main program

```
      PROGRAM KIIERF
      IMPLICIT COMPLEX(8)(A-H,O-Z)
      PARAMETER( N=100 , Z3=(0.D0,3.D0) )
      PARAMETER( Z1=(0.D0,1.D0) )
      REAL(8) R1, RI1, R3, RI3
      CHARACTER*10 HEAD1,HEAD2
      COMPLEX(8)   ZV(N+1), WV(N+1)
      HEAD1='CAUCHY :  '
      HEAD2='TRUE   :  '
      PAI=ACOS(-1.D0)
      S=(0.D0,0.D0)
      WRITE(6,10)
      WRITE(6,20)
      WRITE(6,30) N+1
      DO 1000 I=1,100
         ZV(I)=Z3+EXP(2*PAI*Z1*I/100.D0)
 1000 CONTINUE
      ZV(N+1)=Z3
      WRITE(6,40)
      CALL JIIERF(N+1, ZV, WV, IERR)
      DO 2000 I=1,100
         S=S+WV(I)/100.D0
 2000 CONTINUE
      W3=WV(N+1)
      R1=S
      RI1=S/Z1
      R3=W3
      RI3=W3/Z1
      WRITE(6,6000) HEAD1,R1,RI1
      WRITE(6,6000) HEAD2,R3,RI3
      STOP
   10 FORMAT(1X,' *** JIIERF *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,'NV = ',I3,/,/)
   40 FORMAT(1X, ' *** OUTPUT *** ',/,/)
 6000 FORMAT(1X,A10,5X,E15.7,5X,E15.7,/,/)
      END
```

(c) Output results

```
 *** JIIERF ***

 *** INPUT ***

NV = 101

 *** OUTPUT ***

CAUCHY :        0.1790012E+00      -0.7589415E-18

TRUE   :        0.1790012E+00       0.6251103E-19
```

## 2.12 ASSOCIATED LEGENDRE FUNCTIONS

### 2.12.1 DILEG1, RILEG1
### Associated Legendre Function of the 1st Kind

(1) **Function**

Calculates the value of the associated Legendre function of the 1st kind

$$P_n^m(x) = (|1-x^2|)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}.$$

(2) **Usage**

Double precision:

CALL DILEG1 (N, M, XI, XO, IERR)

Single precision:

CALL RILEG1 (N, M, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$

R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | M | I | 1 | Input | Multiple $m$ |
| 3 | XI | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Input | Value of variable $x$ |
| 4 | XO | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Output | Value of $P_n^m(x)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) When M $<0$, then $|\mathrm{M}| \leq$ N (if N $<0$, then $|\mathrm{M}| \leq -\mathrm{N}-1$)

(b) When $|\mathrm{M}| \leq$ N (if N $<0$, then $rm|\mathrm{M}| \leq -\mathrm{N}-1$), then $|\mathrm{M}| < M_1$
where, $M_1 =${double precision: 150, single precision: 27}

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | $\lvert\mathrm{XI}\rvert >$ (Maximum value)$^{1/\mathrm{N}}/2$ and N $\geq 2$ (overflow) | If XI $\geq 0.0$,<br>XO = (Maximum value) is performed.<br>If XI $< 0.0$,<br>XO = (Maximum value) $\times (-1)^{\mathrm{N}}$<br>is performed. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

(6) **Notes**

(a) If $|x| > 1.0$, then the Hobson associated Legendre function

$$P_n^m(x) = (x^2 - 1.0)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

is calculated, and if $|x| \leq 1.0$, then the Ferrers associated Legendre function

$$P_n^m(x) = (1.0 - x^2)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

is calculated.

(b) This subroutine uses double length arithmetic internally to guarantee precision.

(c) Note that the associated Legendre function of the 1st kind may be defined as

$$P_n^m(x) = (-1)^m (1.0 - x^2)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

when $|x| \leq 1.0$ or it may be defined as the associated function multiplied by $(n - m)!$.

(d) To obtain these values for many XI and for a large order $n$, it is better to use 2.15.3 $\left\{\begin{array}{c}\text{WINPLG}\\ \text{VINPLG}\end{array}\right\}$.

The relationship between the associated Legendre function of the 1st kind $P_n^m(x)$ and the normalized spherical harmonic function $P_n^{*m}(x)$ can be expressed as below.

$$P_n^{*0}(x) = \sqrt{\frac{2n+1}{4\pi}} P_n^0(x) \ (-1 \leq x \leq 1)$$

$$P_n^{*m}(x) = \sqrt{\frac{2n+1}{4\pi}} \sqrt{2\frac{(n-m)!}{(n+m)!}} P_n^m(x) \ (-1 \leq x \leq 1; m = 1, 2, \cdots, n)$$

Note that the absolute value of normalized Legendre function of the 1st kind increases steeply with the factorial order when $m$ is large.

(7) **Example**

(a) Problem

Obtain the value of $P_n^m(x)$ at $n = 4$, $m = 2$ and $x = 0.8$.

(b) Input data

N = 4, M = 2 and XI = 0.8.

(c) Main program

```
      PROGRAM BILEG1
! *** EXAMPLE OF DILEG1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) M
      READ (5,*) XI
      WRITE(6,1000) N,M,XI
      CALL DILEG1(N,M,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DILEG1 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N = ',I3,5X,'M =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF PNM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
    *** DILEG1 ***

     ** INPUT **

       N =   4     M =  2     XI =   0.80


     ** OUTPUT**
```

```
IERR =     0
VALUE OF PNM(X)
  XO =   0.9396000000D+01
```

## 2.12.2 DILEG2, RILEG2
## Associated Legendre Function of the 2nd Kind

(1) **Function**

Calculates the value of the associated Legendre function of the 2nd kind

$$Q_n^m(x) = (|1-x^2|)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}.$$

(2) **Usage**

Double precision:
CALL DILEG2 (N, M, XI, XO, IERR)
Single precision:
CALL RILEG2 (N, M, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Order $n$ |
| 2 | M | I | 1 | Input | Multiple $m$ |
| 3 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 4 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of $Q_n^m(x)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{N} \geq 0$

(b) When $\text{M} < 0$, then $|\text{M}| \leq \text{N}$

(c) $|\text{XI}| \neq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |
| 4100 | Series expansion calculations did not converge. | |

(6) **Notes**

(a) If $|x| > 1.0$, then the Hobson associated Legendre function

$$Q_n^m(x) = (x^2 - 1.0)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}$$

is calculated, and if $|x| \leq 1.0$, then the Ferrers associated Legendre function

$$Q_n^m(x) = (1.0 - x^2)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}$$

is calculated.

(b) This subroutine uses double length arithmetic internally to guarantee precision.

(c) Note that the associated Legendre function of the 2nd kind may be defined as the definition given above for $Q_n^m(x)$ multiplied by $(-1)^m$ when $|x| > 1.0$ or $|x| < 1.0$ or it may be defined as the associated function multiplied by $(n - m)!$.

(7) **Example**

(a) Problem
Obtain the value of $Q_n^m(x)$ at $n = 4, m = 2$ and $x = 1.8$.

(b) Input data
N = 4, M = 2 and XI = 1.8.

(c) Main program

```
      PROGRAM BILEG2
! *** EXAMPLE OF DILEG2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      READ (5,*) N
      READ (5,*) M
      READ (5,*) XI
      WRITE(6,1000) N,M,XI
      CALL DILEG2(N,M,XI,XO,IERR)
      WRITE(6,2000) IERR,XO
 1000 FORMAT(' ',/,/,5X,'*** DILEG2 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N = ',I3,5X,'M =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF QNM(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
    *** DILEG2 ***

     ** INPUT **

       N =   4     M =  2     XI =   1.80

     ** OUTPUT**

       IERR =     0

       VALUE OF QNM(X)

         XO =   0.7031257577D-01
```

## 2.13 ORTHOGONAL POLYNOMIALS

### 2.13.1 DIOPLE, RIOPLE
### Legendre Polynomial

(1) **Function**

Calculates the value of the Legendre polynomial

$$P_i(x) = \frac{1}{2^i i!}\frac{d^i}{dx^i}(x^2-1)^i \quad (i = 0, 1, \cdots, n).$$

(2) **Usage**

Double precision:

CALL DIOPLE (N, XI, XO, IERR)

Single precision:

CALL RIOPLE (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Highest order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 0 : N | Output | Value of $P_i(x)$ $(i = 0, 1, \cdots, n)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq$ 0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

(6) **Notes**

(a) This subroutine uses double length arithmetic internally to guarantee precision.

(7) **Example**

(a) Problem

Obtain the value of $P_n(x)$ at $n = 3$ and $x = 0.8$.

(b) Input data

N = 3 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPLE
! *** EXAMPLE OF DIOPLE ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIOPLE(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPLE ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF PN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIOPLE ***

      ** INPUT **

        N =  3     XI =   0.80


      ** OUTPUT**

        IERR =     0

        VALUE OF PN(X)

          XO =   0.8000000000D-01
```

## 2.13.2 DIZGLW, RIZGLW Gauss=Legendre Formula

(1) **Function**

Evaluate the integration points and weights of (high degree) Gauss=Legendre formula.

(2) **Usage**

Double precision:
CALL DIZGLW (N, Z, W, WORK, IERR)
Single precision:
CALL RIZGLW (N, Z, W, WORK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Degree $n$ |
| 2 | Z | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Output | Zero points of $P_n(x)$ (stored in ascending order) |
| 3 | W | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Output | Weights |
| 4 | WORK | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Work | Work area |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq$ 1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | N was equal to 1. | Processing continues. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | The solution did not converge. | |
| 5000 | Overflow occurred. | |

## (6) Notes

(a) Zero points of $P_n(x)$ are set as output.

(b) The original form of Gauss=Legendre formula is

$$\int_{-1}^{1} f(x)dx = \sum_{j=1}^{n} w_j f(z_j)$$

where $z_j$ and $w_j$ are zero points and weights of $P_n(x)$ respectively. This form holds straightly only when the interval for integration is $[-1, 1]$. If the interval for integration is other than $[-1, 1]$, it can be reduced to the above case by applying integration by substitution.

(c) It is recommended to avoid applying Gauss=Legendre formula to a vibrating function. To illustrate this, we consider the integration below:

$$\int_{-1}^{1} 5e^{-25x^2} \cos(10Ax)dx.$$

This integrand decreases steeply at the both end points. By mapping the interval so that the both end points are mapped to the both infinite points, the value of the integration are calculated as $\sqrt{\pi}e^{-A^2}$. This mapping of the interval bears an error of order $10^{-12}$. If Gauss=Legendre formula is applied with $N = 24$, the error has the order of $0.3*10^{-7}$ when $A$ =0.3. When $A$ =2.4, the calculated value becomes -0.0004898, but the true value is approximately 0.005585. This indicates that even a vibration term such as $\cos(24x)$ can become a cause of an neglectable error in integration.

(d) This subroutine can also be applied to Gauss=Legendre formulas of high degrees.

## (7) Example

(a) Problem

Set N=24 to evaluate $\displaystyle\int_{-1}^{1} \frac{1}{1+x+x^2}dx = \frac{\pi}{\sqrt{3}}$.

(b) Main program

```
      PROGRAM BIZGLW
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=24)
      REAL(8)&
         Z(N), W(2*N)
!
      WRITE(6,10)
      WRITE(6,20)
      WRITE(6,30)N
      CALL DIZGLW(N, Z, W, W(N+1), IERR)
      WRITE(6,40)
      WRITE(6,50)  IERR
      WRITE(6,60)  N
      WRITE(6,4000)
      DO  500 I=1,N
      WRITE(6,5000) I, Z(I), W(I)
  500 CONTINUE
      D=0
      DO 1000 I=1,N
         X=Z(I)
         F= 1.D0/(1+X+X*X)
         D=D+W(I)*F
 1000 CONTINUE
      S=2.D0*ASIN(1.D0)/SQRT(3.D0)
      WRITE(6,70) N
      WRITE(6,6000) D
      WRITE(6,7000) S , D-S
      STOP
   10 FORMAT(1X,' *** DIZGLW *** ',/,/)
   20 FORMAT(1X,' *** INPUT *** ',/,/)
   30 FORMAT(1X,' N = ',I3,/,/)
   40 FORMAT(1X, ' *** OUTPUT *** ',/,/)
   50 FORMAT(1X,'IERR= ',I4,/,/)
   60 FORMAT(1X,'TABLE FOR ',I3,' POINTS FORMULA',/,/)
   70 FORMAT(1X,/,/,' GAUSS-LEGENDRE ',I3,' POINTS FORMULA',/,/)
 4000 FORMAT(1X,' NO ',10X,'ZERO POINT',15X,'  WEIGHT  ',/,/)
 5000 FORMAT(1X, I4, 5X, E20.10,5X,E20.10)
```

```
 6000 FORMAT(1X,'COMPUTED VALUE= ',E20.10,/,/)
 7000 FORMAT(1X,'    TRUE VALUE= ',E20.10,' ERR = ',E20.10)
      END
```

(c) Output results

```
 *** DIZGLW ***

 *** INPUT ***

 N =  24

 *** OUTPUT ***

IERR=    0

TABLE FOR  24 POINTS FORMULA

 NO           ZERO POINT                 WEIGHT

   1        -0.9951872200E+00         0.1234122980E-01
   2        -0.9747285560E+00         0.2853138863E-01
   3        -0.9382745520E+00         0.4427743882E-01
   4        -0.8864155270E+00         0.5929858492E-01
   5        -0.8200019860E+00         0.7334648141E-01
   6        -0.7401241916E+00         0.8619016153E-01
   7        -0.6480936519E+00         0.9761865210E-01
   8        -0.5454214714E+00         0.1074442701E+00
   9        -0.4337935076E+00         0.1155056681E+00
  10        -0.3150426797E+00         0.1216704729E+00
  11        -0.1911188675E+00         0.1258374563E+00
  12        -0.6405689286E-01         0.1279381953E+00
  13         0.6405689286E-01         0.1279381953E+00
  14         0.1911188675E+00         0.1258374563E+00
  15         0.3150426797E+00         0.1216704729E+00
  16         0.4337935076E+00         0.1155056681E+00
  17         0.5454214714E+00         0.1074442701E+00
  18         0.6480936519E+00         0.9761865210E-01
  19         0.7401241916E+00         0.8619016153E-01
  20         0.8200019860E+00         0.7334648141E-01
  21         0.8864155270E+00         0.5929858492E-01
  22         0.9382745520E+00         0.4427743882E-01
  23         0.9747285560E+00         0.2853138863E-01
  24         0.9951872200E+00         0.1234122980E-01

GAUSS-LEGENDRE  24 POINTS FORMULA

COMPUTED VALUE=     0.1813799364E+01

    TRUE VALUE=     0.1813799364E+01 ERR =    -0.3774758284E-14
```

### 2.13.3 DIOPLA, RIOPLA Laguerre Polynomial

(1) **Function**

Calculates the value of the Laguerre polynomial

$$L_i(x) = \frac{e^x}{i!}\frac{d^i}{dx^i}(e^{-x}x^i) \quad (i = 0, 1, \cdots, n).$$

(2) **Usage**

Double precision:

CALL DIOPLA (N, XI, XO, IERR)

Single precision:

CALL RIOPLA (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Highest order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 0 : N | Output | Value of $L_i(x)$ $(i = 0, 1, \cdots, n)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{N} \geq 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

(6) **Notes**

(a) This subroutine uses double length arithmetic internally to guarantee precision.

(7) **Example**

(a) Problem

Obtain the value of $L_n(x)$ at $n = 3$ and $x = 0.8$.

(b) Input data

N = 3 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPLA
! *** EXAMPLE OF DIOPLA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIOPLA(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPLA ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF LN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIOPLA ***

      ** INPUT **

        N =  3     XI =   0.80


      ** OUTPUT**

        IERR =     0

        VALUE OF LN(X)

          XO =  -0.5253333333D+00
```

### 2.13.4 DIOPHE, RIOPHE Hermite Polynomial

(1) **Function**

Calculates the value of the Hermite polynomial

$$H_i(x) = (-1)^i e^{x^2} \frac{d^i}{dx^i}(e^{-x^2}) \quad (i = 0, 1, \cdots, n).$$

(2) **Usage**

Double precision:
CALL DIOPHE (N, XI, XO, IERR)
Single precision:
CALL RIOPHE (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Highest order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 0 : N | Output | Value of $H_i(x)$ $(i = 0, 1, \cdots, n)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{N} \geq 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

(6) **Notes**

(a) This subroutine uses double length arithmetic internally to guarantee precision.

(7) **Example**

(a) Problem

Obtain the value of $H_n(x)$ at $n = 3$ and $x = 0.8$.

(b) Input data

N = 3 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPHE
! *** EXAMPLE OF DIOPHE ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIOPHE(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPHE ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF HN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
*** DIOPHE ***

 ** INPUT **

   N =  3     XI =   0.80

 ** OUTPUT**

   IERR =     0

   VALUE OF HN(X)

     XO =  -0.5504000000D+01
```

## 2.13.5 DIOPCH, RIOPCH
## Chebyshev Polynomial

(1) **Function**

Calculates the value of the Chebyshev polynomial

$$T_i(x) = \frac{(-1)^i}{(2i-1)!!}\sqrt{1-x^2}\frac{d^i}{dx^i}(1-x^2)^{i-1/2} \quad (i = 0, 1, \cdots, n).$$

(2) **Usage**

Double precision:

CALL DIOPCH (N, XI, XO, IERR)

Single precision:

CALL RIOPCH (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Highest order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 0 : N | Output | Value of $T_i(x)$ $(i = 0, 1, \cdots, n)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{N} \geq 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

(6) **Notes**

(a) This subroutine uses double length arithmetic internally to guarantee precision.

(7) **Example**

(a) Problem

Obtain the value of $T_n(x)$ at $n = 3$ and $x = 0.8$.

(b) Input data

N = 3 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPCH
! *** EXAMPLE OF DIOPCH ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIOPCH(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPCH ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF TN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIOPCH ***

      ** INPUT **

        N =  3     XI =   0.80

      ** OUTPUT**

        IERR =     0

        VALUE OF TN(X)

          XO =  -0.3520000000D+00
```

### 2.13.6 DIOPC2, RIOPC2 Chebyshev Function of the 2nd Kind

(1) **Function**

Calculates the value of the Chebyshev function of the 2nd kind

$$U_i(x) = \frac{\sqrt{1-x^2}}{i}\frac{dT_i(x)}{dx} \quad (i = 0, 1, \cdots, n).$$

(2) **Usage**

Double precision:

CALL DIOPC2 (N, XI, XO, IERR)

Single precision:

CALL RIOPC2 (N, XI, XO, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Highest order $n$ |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Value of variable $x$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 0 : N | Output | Value of $U_i(x)$ $(i = 0, 1, \cdots, n)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq 0$, $|\text{XI}| \leq 1.0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

None

(7) **Example**

(a) Problem

Obtain the value of $U_3(0.8)$.

(b) Input data

N = 3 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPC2
! *** EXAMPLE OF DIOPC2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) XI
      WRITE(6,1000) N,XI
      CALL DIOPC2(N,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPC2 ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF UN(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIOPC2 ***

      ** INPUT **

        N =  3     XI =   0.80

      ** OUTPUT**

        IERR =     0

        VALUE OF UN(X)

          XO =   0.9360000000D+00
```

## 2.13.7 DIOPGL, RIOPGL Generalized Laguerre Polynomial

**(1) Function**

Calculates the value of the generalized Laguerre polynomial

$$L_i^{(\alpha)}(x) = \frac{e^x x^{-\alpha}}{i!} \frac{d^i}{dx^i}(e^{-x} x^{i+\alpha}) \quad (i = 0, 1, \cdots, n)$$

**(2) Usage**

Double precision:

CALL DIOPGL (N, ALF, XI, XO, IERR)

Single precision:

CALL RIOPGL (N, ALF, XI, XO, 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 | Highest order $n$ |
| 2 | ALF | {D / R} | 1 | Input | Value of variable $\alpha$ |
| 3 | XI | {D / R} | 1 | Input | Value of variable $x$ |
| 4 | XO | {D / R} | 0 : N | Output | Value of $L_i^{(\alpha)}(x)$ $(i = 0, 1, \cdots, n)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

**(4) Restrictions**

(a) $\mathrm{N} \geq 0$

**(5) Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | Overflow occurred during the calculation. | |

**(6) Notes**

None

(7) **Example**

(a) Problem

Obtain the value of $L_n^{(\alpha)}(x)$ for $n = 3$, $\alpha = 0.5$ and $x = 0.8$.

(b) Input data

N = 3, ALF = 0.5 and XI = 0.8.

(c) Main program

```
      PROGRAM BIOPGL
! *** EXAMPLE OF DIOPGL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION  XO(0:3)
      READ (5,*) N
      READ (5,*) ALF
      READ (5,*) XI
      WRITE(6,1000) N,ALF,XI
      CALL DIOPGL(N,ALF,XI,XO,IERR)
      WRITE(6,2000) IERR,XO(3)
 1000 FORMAT(' ',/,/,5X,'*** DIOPGL ***',/,/,6X,'** INPUT **',&
      /,/,8X,'N =',I3,5X,'ALF = ',F6.2,5X,'XI = ',F6.2  )
 2000 FORMAT(' ',/,/,6X,'** OUTPUT**',/,/,8X,'IERR = ',I5,&
      /,/,8X,'VALUE OF LNA(X)',/,/,10X,'XO = ',D18.10)
      END
```

(d) Output results

```
     *** DIOPGL ***

      ** INPUT **

        N =  3     ALF =   0.50     XI =   0.80

      ** OUTPUT**

        IERR =     0

        VALUE OF LNA(X)

          XO =  -0.2778333333D+00
```

## 2.14 MATHIEU FUNCTIONS

### 2.14.1 DIMTCE, RIMTCE
### Mathieu Functions of Integer Orders $ce_n(x, q)$

(1) **Function**

Evaluate the value of $ce_n(x, q)$.

(2) **Usage**

Double precision:

CALL DIMTCE (NORD, N, Q, X, CE, ISW, WORK, IERR)

Single precision:

CALL RIMTCE (NORD, N, Q, X, CE, ISW, WORK, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/Output | Contents |
|---|---|---|---|---|---|
| 1 | NORD | I | 1 | Input | term number for expansion−1 (See Note (a)) |
| 2 | N | I | 1 | Input | order $n$ |
| 3 | Q | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | parameter $q$ |
| 4 | X | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | variable $x$ |
| 5 | CE | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | the function $ce_n(x, q)$ |
| 6 | ISW | I | 1 | Input | switch<br>ISW =0 : after initial setting, evaluate CE<br>ISW =1 : only evaluate CE |
| | | | | Output | if initial setting was done<br>return 1 |
| 7 | WORK | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | See Contents | Input/Output | coefficient table (See Note (b))<br>**Size**: $2 \times \text{NORD}^2 + 6 \times \text{NORD} + 108$ |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $2 \leq \text{NORD} \leq 50$

(b) $0 \leq \text{N} \leq 2 \times \text{NORD} + 1$

(c) ISW=0 or ISW=1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | It became impossible to calculate in sufficient accuracy. | Processing is aborted. |
| 3000 | Restriction (a) or (b) was not satisfied. | |

(6) **Notes**

(a) If NORD is small, the precision for CE is not sufficient. Therefore the NORD is prefer to the large value. The value of NORD for the parameter Q and N must be set to the following range:

$$\min(50.0, 0.5 \times \mathrm{N} + 10 + |\mathrm{Q}|) \leq \mathrm{NORD} \leq 50.$$

(b) To obtain multiple values of Mathieu functions $ce_n(x, q)$ where the parameter q is fixed, call this subroutine once with ISW=0 and then call this subroutine again after changing only the contents of X and N. Here, the value of NORD must be set to the following range:

$$\min(50.0, 0.5 \times \mathrm{N}_{\max} + 10 + |\mathrm{Q}|) \leq \mathrm{NORD} \leq 50$$

where $\mathrm{N}_{\max}$ is the largest order. This enables you to eliminate unnecessary calculations by performing the initial setting only once.

(c) When processing ends with IERR=2000, the accuracy of a calculation result cannot be guaranteed.

(d) Mathieu functions $ce_n(x, q)$ can be represented by Fourier series expansion (with cosine terms only). In this subroutine, they are calculated by series sum up to the (NORD+1)-th term. Therefore, the calculation time and accuracy depend on the value of NORD.

(e) As |Q| or N is larger, the calculation time $ce_n(x, q)$ trend to increase. It is prefer to set $\mathrm{N} \leq 90$ and $|\mathrm{Q}| \leq 70.0$.

(7) **Example**

(a) Problem
Obtain the values of $ce_7(x, 5.0)$ on the approximation condition that the number of the expression terms is 21 for $x$=1.0, 2.0,$\cdots$,10.0.

(b) Main program

```
      PROGRAM BIMTCE
      IMPLICIT REAL(8)(A-H, O-Z)
      PARAMETER( NORD = 20 )
      PARAMETER( NSIZE= 2*NORD*NORD+6*NORD+108 )
      REAL(8) WORK(NSIZE)
!
      N7= 7
      Q = 5.0D0
      X = 1.D0
      ISW=0
      ISWO=ISW
      CALL DIMTCE(NORD, N7, Q, X, CE, ISW, WORK, IERR)
      WRITE(6,4000)
      WRITE(6,4500)
      WRITE(6,5000) N7, Q , NORD
      WRITE(6,5300)
      WRITE(6,5500)
      WRITE(6,6000) X, CE, IERR, ISWO
      DO 2000 I= 2, 10
         X= I
         ISWO=ISW
         CALL DIMTCE(NORD, N7, Q, X, CE, ISW, WORK, IERR)
         WRITE(6,6000) X, CE, IERR, ISWO
 2000 CONTINUE
      STOP
 4000 FORMAT(1X ,'*** DIMTCE*')
```

```
 4500 FORMAT(1X ,'*** INPUT *')
 5000 FORMAT(1X ,'MATHIEU FUNCTION :N=',I4,'  Q=',F15.6,' NORD=',I4)
 5300 FORMAT(1X,'*** OUTPUT *' )
 5500 FORMAT(1X,7X,'X',7X,5X,6X,'CEN',5X,6X,'  CODE',5X,'   ISW')
 6000 FORMAT(1X,F15.6,5X,F15.6,5X,I6,5X,I6)
      END
```

(c) Output results

```
*** DIMTCE*
*** INPUT *
MATHIEU FUNCTION :N=   7  Q=       5.000000 NORD=  20
*** OUTPUT *
       X                CEN              CODE         ISW
       1.000000          0.902463           0           0
       2.000000         -0.128610           0           1
       3.000000         -0.666293           0           1
       4.000000         -0.796419           0           1
       5.000000         -0.769659           0           1
       6.000000         -0.217824           0           1
       7.000000         -0.057805           0           1
       8.000000          0.858668           0           1
       9.000000          0.931020           0           1
      10.000000          0.869127           0           1
```

## 2.14.2 DIMTSE, RIMTSE
## Mathieu Functions of Integer Orders $se_n(x, q)$

(1) **Function**

Evaluate the value of $se_n(x, q)$.

(2) **Usage**

Double precision:

CALL DIMTSE (NORD, N, Q, X, SE, ISW, WORK, IERR)

Single precision:

CALL RIMTSE (NORD, N, Q, X, SE, ISW, WORK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NORD | I | 1 | Input | term number for expansion$-1$ (See Note (a)) |
| 2 | N | I | 1 | Input | order $n$ |
| 3 | Q | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Input | parameter $q$ |
| 4 | X | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Input | variable $x$ |
| 5 | SE | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | 1 | Output | the function $se_n(x, q)$ |
| 6 | ISW | I | 1 | Input | switch<br>ISW =0 : after initial setting, evaluate SE<br>ISW =1 : only evaluate SE |
| | | | | Output | if initial setting was done<br>return 1 |
| 7 | WORK | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | See Contents | Input/ Output | coefficient table (See Note (b))<br>**Size**: $2 \times \text{NORD}^2 + 6 \times \text{NORD} + 108$ |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $2 \leq \text{NORD} \leq 50$

(b) $1 \leq \text{N} \leq 2 \times \text{NORD} + 2$

(c) ISW=0 or ISW=1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000 | It became impossible to calculate in sufficient accuracy. | Processing is aborted. |
| 3000 | Restriction (a) or (b) was not satisfied. | |

(6) **Notes**

(a) If NORD is small, the precision for SE is not sufficient. Therefore the NORD is prefer to the large value. The value of NORD for the parameter Q and N must be set to the following range:

$$\min(50.0, 0.5 \times \mathrm{N} + 10 + |\mathrm{Q}|) \leq \mathrm{NORD} \leq 50.$$

(b) To obtain multiple values of Mathieu functions $se_n(x,q)$ where the parameter q is fixed, call this subroutine once with ISW=0 and then call this subroutine again after changing only the contents of X and N. Here, the value of NORD must be set to the following range:

$$\min(50.0, 0.5 \times \mathrm{N}_{\max} + 10 + |\mathrm{Q}|) \leq \mathrm{NORD} \leq 50$$

where $\mathrm{N}_{\max}$ is the largest order. This enables you to eliminate unnecessary calculations by performing the initial setting only once.

(c) When processing ends with IERR=2000, the accuracy of a calculation result cannot be guaranteed.

(d) Mathieu functions $se_n(x,q)$ can be represented by Fourier series expansion (with sine terms only). In this subroutine, they are calculated by series sum up to the (NORD+1)-th term. Therefore, the calculation time and accuracy depend on the value of NORD.

(e) As |Q| or N is larger, the calculation time $se_n(x,q)$ trend to increase. It is prefer to set N $\leq$ 90 and $|\mathrm{Q}| \leq 70.0$.

(7) **Example**

(a) Problem
Obtain the values of $se_7(x, 5.0)$ on the approximation condition that the number of the expression terms is 21 for $x$=1.0, 2.0,$\cdots$,10.0.

(b) Main program

```
      PROGRAM BIMTSE
      IMPLICIT REAL(8)(A-H, O-Z)
      PARAMETER( NORD = 20 )
      PARAMETER( NSIZE= 2*NORD*NORD+6*NORD+108 )
      REAL(8) WORK(NSIZE)
!
      N7= 7
      Q = 5.0D0
      X = 1.D0
      ISW=0
      ISW0=ISW
      CALL DIMTSE(NORD, N7, Q, X, SE, ISW, WORK, IERR)
      WRITE(6,4000)
      WRITE(6,4500)
      WRITE(6,5000) N7, Q , NORD
      WRITE(6,5300)
      WRITE(6,5500)
      WRITE(6,6000) X, SE, IERR, ISW0
      DO 2000 I= 2, 10
         X= I
         ISW0=ISW
         CALL DIMTSE(NORD, N7, Q, X, SE, ISW, WORK, IERR)
         WRITE(6,6000) X, SE, IERR, ISW0
 2000 CONTINUE
      STOP
 4000 FORMAT(1X ,'*** DIMTSE*')
```

```
 4500 FORMAT(1X ,'*** INPUT *')
 5000 FORMAT(1X ,'MATHIEU FUNCTION :N=',I4,'  Q=',F15.6,' NORD=',I4)
 5300 FORMAT(1X,'*** OUTPUT *' )
 5500 FORMAT(1X,7X,'X',7X,5X,6X,'SEN',5X,6X,'  CODE',5X,'   ISW')
 6000 FORMAT(1X,F15.6,5X,F15.6,5X,I6,5X,I6)
      END
```

(c) Output results

```
*** DIMTSE*
*** INPUT *
MATHIEU FUNCTION :N=   7  Q=       5.000000 NORD=  20
*** OUTPUT *
       X                SEN              CODE        ISW
       1.000000          0.370085          0          0
       2.000000          0.956161          0          1
       3.000000          0.815456          0          1
       4.000000          0.585207          0          1
       5.000000         -0.568524          0          1
       6.000000         -1.022084          0          1
       7.000000         -1.001135          0          1
       8.000000         -0.411058          0          1
       9.000000          0.447748          0          1
      10.000000          0.529804          0          1
```

## 2.15 OTHER SPECIAL FUNCTIONS

### 2.15.1 WIXSPS, VIXSPS Di-Log Function

(1) **Function**

For real numbers $X_i$ ($\geq 0$ , $i = 1, \cdots, N$), obtain values of di-log function defined as

$$Li_2(X_i) = -\int_0^{X_i} \log|t-1| \frac{dt}{t}.$$

(2) **Usage**

Double precision:

CALL WIXSPS (NV,XV,YV, IERR)

Single precision:

CALL VIXSPS (NV,XV,YV, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number $N$ of $X_i$ |
| 2 | XV | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ ($i = 1, \cdots$,NV) |
| 3 | YV | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $Li_2(X_i)$ ($i = 1, \cdots$,NV) |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq$ 1

(b) XV(i) $\geq$ 0 (i=1,$\cdots$,NV)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) $Li_2$(XV(i)) (i=1,$\cdots$,NV) are stored in YV(i).

(b) $Li_2(x)$ increases monotonically in $0 \leq x < 2$, reaches the peak of $\pi^2/4$ at the point $x = 2$. In $x > 2$, $Li_2(x)$ decreases as the asymptotic formula

$$Li_2(x) = -\frac{1}{2}(\log x)^2 + \frac{\pi^2}{3} + O(x^{-1}).$$

(c) The zero point of $Li_2(x)$ is the vicinity of $x = 12.59517037$.

(7) **Example**

(a) Problem

Obtain the values of $Li_2(x_i)$ for $x_i = 0.2 * i$ $(i = 1, 2, \cdots, 10)$.

(b) Input data

NV=10 and array XV.

(c) Main program

```
      PROGRAM EIXSPS
! *** EXAMPLE OF WIXSPS ***
      IMPLICIT NONE
!
      INTEGER NV
      PARAMETER( NV = 10 )
      INTEGER IERR,I
      REAL(8) XV(NV),YV(NV)
      REAL(8) FIVE
      PARAMETER( FIVE = 5.D0 )
!
      DO 100 I=1,NV
         XV(I)=DBLE(I)/FIVE
  100 CONTINUE
!
      WRITE(6,6000) NV
      DO 110 I=1,NV
        WRITE(6,6010) I,XV(I)
  110 CONTINUE
!
      CALL WIXSPS( NV, XV, YV, IERR )
!
      WRITE(6,6020) IERR
      DO 120 I=1,NV
         WRITE(6,6030) I, YV(I)
  120 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  WIXSPS  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     NV = ',I4,/)
 6010 FORMAT(1X,'     XV(',I2,')=',F10.6)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6030 FORMAT(1X,'     LI2( XV(',I2,') )=',F10.6)
      END
```

(d) Output results

```
***  WIXSPS  ***

 **  INPUT  **

     NV =   10

     XV( 1)=  0.200000
     XV( 2)=  0.400000
     XV( 3)=  0.600000
     XV( 4)=  0.800000
     XV( 5)=  1.000000
     XV( 6)=  1.200000
     XV( 7)=  1.400000
     XV( 8)=  1.600000
     XV( 9)=  1.800000
     XV(10)=  2.000000

 **  OUTPUT  **

     IERR =    0

     LI2( XV( 1) )=  0.211004
     LI2( XV( 2) )=  0.449283
     LI2( XV( 3) )=  0.727586
     LI2( XV( 4) )=  1.074795
     LI2( XV( 5) )=  1.644934
     LI2( XV( 6) )=  2.129169
     LI2( XV( 7) )=  2.319073
     LI2( XV( 8) )=  2.413131
     LI2( XV( 9) )=  2.455876
     LI2( XV(10) )=  2.467401
```

## 2.15.2 WIDBEY, VIDBEY Debye Function

(1) **Function**

For real values $X_i$ ($\geq 0, i = 1, \cdots, N$), obtain each value of the Debye function

$$F_D(X_i) = \frac{3}{{X_i}^3} \int_0^{X_i} \frac{e^t t^4}{(e^t - 1)^2} dt.$$

(2) **Usage**

Double precision:

CALL WIDBEY (NV,XV,YV, IERR)

Single precision:

CALL VIDBEY (NV,XV,YV, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$

R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number $N$ of $X_i$ |
| 2 | XV | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i(i = 1, \cdots,$NV) |
| 3 | YV | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Output | $F_D(X_i)(i = 1, \cdots,$NV) |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq$ 1

(b) XV(i) $\geq$ 0 (i=1,$\cdots$,NV)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) $F_D$(XV(i)) (i=1,$\cdots$,NV) are stored in YV(i).

(b) Debye function $F_D(y)$ decreases monotonically with y.

(c) $F_D(0)$ means $\lim_{y \to +0} F_D(y)$.

(7) **Example**

(a) Problem

Obtain the value of $F_D(x_i)$ for $x_i = 0.2 * i$ $(i = 1, 2, \cdots, 10)$.

(b) Input data

NV=10 and array XV.

(c) Main program

```
      PROGRAM EIDBEY
! *** EXAMPLE OF WIDBEY ***
      IMPLICIT NONE
!
      INTEGER NV
      PARAMETER( NV = 10 )
      INTEGER IERR,I
      REAL(8) XV(NV),YV(NV)
      REAL(8) FIVE
      PARAMETER( FIVE = 5.D0 )
!
      DO 100 I=1,NV
      XV(I)=DBLE(I)/FIVE
  100 CONTINUE
!
      WRITE(6,6000) NV
      DO 110 I=1,NV
        WRITE(6,6010) I,XV(I)
  110 CONTINUE
!
      CALL WIDBEY( NV, XV, YV, IERR )
!
      WRITE(6,6020) IERR
      DO 120 I=1,NV
         WRITE(6,6030) I, YV(I)
  120 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  WIDBEY  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     NV = ',I4,/)
 6010 FORMAT(1X,'     XV(',I2,')=',F10.6)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6030 FORMAT(1X,'     FD( XV(',I2,') )=',F10.6)
      END
```

(d) Output results

```
***  WIDBEY  ***

 **  INPUT  **

     NV =   10

     XV( 1)=  0.200000
     XV( 2)=  0.400000
     XV( 3)=  0.600000
     XV( 4)=  0.800000
     XV( 5)=  1.000000
     XV( 6)=  1.200000
     XV( 7)=  1.400000
     XV( 8)=  1.600000
     XV( 9)=  1.800000
     XV(10)=  2.000000

 **  OUTPUT  **

     IERR =    0

     FD( XV( 1) )=  0.998003
     FD( XV( 2) )=  0.992045
     FD( XV( 3) )=  0.982229
     FD( XV( 4) )=  0.968717
     FD( XV( 5) )=  0.951732
     FD( XV( 6) )=  0.931545
     FD( XV( 7) )=  0.908467
     FD( XV( 8) )=  0.882842
     FD( XV( 9) )=  0.855031
     FD( XV(10) )=  0.825408
```

### 2.15.3 WINPLG, VINPLG
### Spherical Harmonic Function

(1) **Function**

For given real numbers $X_i(|X_i| \leq 1; i = 1, \cdots, N)$, this program obtains spherical harmonic functional systems ( order $m = 0, \cdots, n$ ) of the degree $n$ which are normalized and defined as

$$P_n^{*m}(X_i) = \frac{1}{4\pi\sqrt{-1}^m} A_{n,m} \int_{-\pi}^{\pi} (X_i + \sqrt{-1}\sqrt{1 - X_i^2}\cos\phi)^n \cos(m\phi) d\phi,$$

where normalize coefficients $A_{n,m}$ are

$$A_{n,0} = \sqrt{\frac{2n+1}{\pi}}\ ;\ A_{n,m} = \sqrt{\frac{2(2n+1)(n-m)!(n+m)!}{\pi(n!)^2}}\ (m = 1, \cdots, n).$$

(2) **Usage**

Double precision:
  CALL WINPLG (NV,XV,N,PLG,NVL,WORK, IERR)
Single precision:
  CALL VINPLG (NV,XV,N,PLG,NVL,WORK, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number $N$ of $X_i$ |
| 2 | XV | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NV | Input | $X_i(i = 1, \cdots,\text{NV})$ |
| 3 | N | I | 1 | Input | Degree $n$ |
| 4 | PLG | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | See Contents | Output | Spherical harmonic $P_n^{*m}(X_i)$<br>$(i = 1, \cdots,\text{NV};\ m = 0, \cdots,\text{N})$<br>(See Note (a))<br>**Size**: $\text{NVL}, \text{N} + 1$ |
| 5 | NVL | I | 1 | Input | Adjustable dimension of array PLG |
| 6 | WORK | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | See Contents | work | Work area<br>**Size**: $3 \times \text{NV} + \text{N} + 1$ |
| 7 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $1 \leq \text{NV} \leq \text{NVL}$

(b) $\text{N} \geq 1$

(c) $|\text{XV(i)}| \leq 1$ (i=1,$\cdots$,NV)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (c) was not satisfied. | |

(6) **Notes**

(a) $P_n^{*\mathrm{m}}(\mathrm{XV(i)})$ are stored in $\mathrm{PLG(i, m+1)}$ for i=1,$\cdots$,NV; m=0,$\cdots$,N.

(b) When $m = 0, \cdots, n$ for a fixed $n$, it is better to use this subroutine than to use 2.12.1 $\begin{Bmatrix} \text{DILEG1} \\ \text{RILEG1} \end{Bmatrix}$.

(c) If two integers $n_1$ and $n_2$ satisfy $n_1, n_2 \geq m$ for a non-negative integer $m$, the following integration relation is satisfied.

$$\int_{-1}^{1} P_{n_1}^{*\ m}(x) P_{n_2}^{*\ m}(x) dx = \frac{\delta_{n1,n2}}{(1+\delta_{0,m})\pi}$$

If the normalized spherical harmonic functions $P_n^{*m}(\cos\theta)\cos(m\phi)$ $(m = 0, \cdots, n)$ and $P_n^{*m}(\cos\theta)\sin(m\phi)$ $(m = 1, \cdots, n)$ are obtained for $n = 1, 2, \cdots$, then these functions consist the orthonormal basis for surface integration on a unit spherical surface $\int_{-\pi\leq\phi\leq\pi;0\leq\theta\leq\pi} \sin\theta d\theta d\phi$ .

(d) The following relation holds:

$$\frac{2n+1}{4\pi} P_n(xy + \sqrt{(1-x^2)(1-y^2)}\cos\phi) = \sum_{m=0}^{n} P_n^{*m}(x) P_n^{*m}(y) \cos(m\phi) \ (-1 \leq x \leq 1, -1 \leq y \leq 1).$$

(7) **Example**

(a) Problem

Obtain spherical harmonic functional values $P_n^{*m}(x_i)$ $(m = 0, \cdots, n)$ for $x_1 = 0.57735026919$ and $x_2 = -0.57735026919$ of the degree $n = 10$.

(b) Input data

NV=2, NVL= 2, array XV and N=10.

(c) Main program

```
      PROGRAM EINPLG
! *** EXAMPLE OF WINPLG ***
      IMPLICIT NONE
!
      INTEGER NV,N,NVL
      PARAMETER( NV = 2, N = 10, NVL = 2 )
      INTEGER IERR,I
      REAL(8) XV(NV),PLG(NVL,N+1),WORK(3*NV+N+1)
!
      XV(1)=0.57735026919D0
      XV(2)=-XV(1)
!
      WRITE(6,6000) NV,N,NVL
      WRITE(6,6010) XV(1), XV(2)
!
      CALL WINPLG( NV, XV, N, PLG, NVL, WORK, IERR )
!
      WRITE(6,6020) IERR
      DO 100 I=0,N
         WRITE(6,6030) N, I, PLG(1,I+1), PLG(2,I+1)
  100 CONTINUE
!
      STOP
 6000 FORMAT(/,&
             1X,'***  WINPLG  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     NV = ',I4,'    N = ',I4,'   NVL = ',I4,/)
 6010 FORMAT(1X,'     XV=  ',F10.6 , '   ',F10.6,/)
 6020 FORMAT(/,&
```

```
              1X,' **  OUTPUT  **',/,/,&
              1X,'     IERR =',I5,/,/,&
              1X,'     TABLE OF SPHERICAL HARMONICS FOR       XV(1)',&
                 '        XV(2)',/)
 6030 FORMAT(1X,7X,I2,' TH DEGREE HARMONIC ORDER ',&
                 I2,' = ',F10.6,3X,F10.6)
      END
```

(d) Output results

```
 ***  WINPLG  ***

  **  INPUT  **

      NV =    2    N =   10   NVL =    2

      XV=    0.577350    -0.577350


  **  OUTPUT  **

      IERR =    0

      TABLE OF SPHERICAL HARMONICS FOR       XV(1)        XV(2)

        10 TH DEGREE HARMONIC ORDER  0 =  -0.345789    -0.345789
        10 TH DEGREE HARMONIC ORDER  1 =   0.076718    -0.076718
        10 TH DEGREE HARMONIC ORDER  2 =   0.503967     0.503967
        10 TH DEGREE HARMONIC ORDER  3 =   0.061596    -0.061596
        10 TH DEGREE HARMONIC ORDER  4 =  -0.492768    -0.492768
        10 TH DEGREE HARMONIC ORDER  5 =  -0.358105     0.358105
        10 TH DEGREE HARMONIC ORDER  6 =   0.239552     0.239552
        10 TH DEGREE HARMONIC ORDER  7 =   0.634916    -0.634916
        10 TH DEGREE HARMONIC ORDER  8 =   0.586511     0.586511
        10 TH DEGREE HARMONIC ORDER  9 =   0.319568    -0.319568
        10 TH DEGREE HARMONIC ORDER 10 =   0.101056     0.101056
```

### 2.15.4 WIXSLA, VIXSLA Langevin Function

(1) **Function**

For $x = X_i$, calculates the value of the Langevin function

$$L(x) = \coth(x) - \frac{1}{x}$$

(2) **Usage**

Double precision:
CALL WIXSLA (NV, XI, XO, IERR)
Single precision:
CALL VIXSLA (NV, XI, XO, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | number of input data |
| 2 | XI | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Input | $X_i$ |
| 3 | XO | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NV | Output | $L(X_i)$ |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NV $\geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) If $L(x)$ is calculated directly according to its definition, precision will be bad at $x \simeq 0$. Therefore, this subroutine should be used.

(7) **Example**

(a) Problem

Obtain $L(x)$ for $x = 0.0, 0.1, \cdots, 0.9$.

(b) Main program

```
      PROGRAM EIXSLA
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER (NV=10)
      REAL(8)  XI(NV) , XO(NV)
      CHARACTER*6 CNAME , CFNC
      PARAMETER( CNAME='WIXSLA', CFNC='     L' )
!
      DNV=NV
      DO 1000 I=1,NV
        XI(I)=(I-1)/DNV
 1000 CONTINUE
!
      CALL WIXSLA( NV, XI, XO, IERR )
!
      WRITE(6,6000) CNAME
      WRITE(6,6100)
      DO 2000 I=1,NV
        WRITE(6,6200) I,XI(I)
 2000 CONTINUE
!
      WRITE(6,6300)
      WRITE(6,6400) IERR
      DO 3000 I=1,NV
         WRITE(6,6500) CFNC,XI(I), XO(I)
 3000 CONTINUE
      STOP
!
 6000 FORMAT(1X,'*** ',A6,' *')
 6100 FORMAT(1X,'*** INPUT *' )
 6200 FORMAT(1X,'XI(',I2,')=',F10.6 )
 6300 FORMAT(1X,'*** OUTPUT *' )
 6400 FORMAT(1X,'IERR=',I5 )
 6500 FORMAT(1X,A6,'(',F10.6,')=',F10.6 )
      END
```

(c) Output results

```
*** WIXSLA *
*** INPUT *
XI( 1)=  0.000000
XI( 2)=  0.100000
XI( 3)=  0.200000
XI( 4)=  0.300000
XI( 5)=  0.400000
XI( 6)=  0.500000
XI( 7)=  0.600000
XI( 8)=  0.700000
XI( 9)=  0.800000
XI(10)=  0.900000
*** OUTPUT *
IERR=    0
     L(  0.000000)=  0.000000
     L(  0.100000)=  0.033311
     L(  0.200000)=  0.066490
     L(  0.300000)=  0.099405
     L(  0.400000)=  0.131932
     L(  0.500000)=  0.163953
     L(  0.600000)=  0.195359
     L(  0.700000)=  0.226050
     L(  0.800000)=  0.255941
     L(  0.900000)=  0.284956
```

## 2.15.5 WIXZTA, VIXZTA Hurwitz Zeta Function

(1) **Function**

For $a > 0ands = X_i \geq 0$, obtain the value of Hurwitz zeta function subtracted by $(s-1)^{-1}$

$$\zeta(s,a) - (s-1)^{-1} = \frac{1}{\Gamma(s)}(\int_1^\infty \frac{e^{-at}}{1-e^{-t}} t^{s-1} dt + \int_0^1 (\frac{e^{-at}}{1-e^{-t}} - \frac{1}{t}) t^{s-1} dt + (s-1)^{-1}) - (s-1)^{-1}$$

where the right-hand side is an example of the analytical continuation of

$$\sum_{n=0}^\infty (n+a)^{-s} - (s-1)^{-1}$$

($\Re(s) > 1$) to the region $\Re(s) > 0$.

(2) **Usage**

Double precision:
  CALL WIXZTA (NV, X, A, Y, IERR)
Single precision:
  CALL VIXZTA (NV, X, A, Y, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | NV | I | 1 | Input | Number of inputs |
| 2 | X | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | NV | Input | Value of variable $s$ |
| 3 | A | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | 1 | Input | Parameter $a$ |
| 4 | Y | $\left\{ \begin{array}{c} D \\ R \end{array} \right\}$ | NV | Output | Value of Hurwitz zeta function $\zeta(s,a) - 1/(s-1)$ |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) A $>$ 0.0

(b) X(i) $\geq$ 0.0

(c) NV $>$ 0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a), (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) $\zeta(s,a)$ can be continued analytically to a rational function of the complex argument $s$, and has only pole $s = 1$. This subroutine does not calculates values of this function itself, but it calculates values of this function subtracted by the function $1/(s-1)$. Therefore, when X(i) = 1, $-$Y(i) equals di-gamma-function $\psi(a)$.

(b) Poly-gamma-function can be given as the values Y(i) for X(i) $= 2, 3, \cdots$ multiplied by a certain constant.

(7) **Example**

(a) Problem
Evaluate $\zeta(x,a) - (x-1)^{-1}$ for $x$=0.5i(i=1,$\cdots$,10) and $a$=1.

(b) Main program

```
      PROGRAM EIXZTA
      REAL(8)  X(10) , Y(10) , A
      INTEGER NV,IERR,I
!
      NV=10
      DO 1000 I=1, 10
         X(I) = 0.5D0*I
 1000 CONTINUE
      A=1.D0
      CALL WIXZTA(NV, X, A, Y, IERR)
      WRITE(6,5900)
      WRITE(6,5950)
      WRITE(6,6000) X(1), A
      DO 2000 I=2,NV
      WRITE(6,6050) X(I)
 2000 CONTINUE
      WRITE(6,6060)
      WRITE(6,6100) IERR
      DO 3000 I=1,NV
        IF(X(I).EQ.1.D0) THEN
           WRITE(6,6200) X(I), X(I), Y(I)
        ELSE
           WRITE(6,6300) X(I), X(I), Y(I), X(I), Y(I)+1/(X(I)-1)
        ENDIF
 3000 CONTINUE
      STOP
!
 5900 FORMAT(1X,'*** WIXZTA ***',/,/)
 5950 FORMAT(1X,'*** INPUT ***',/,/)
 6000 FORMAT(1X,'               X= ',F10.7, ' A= ',F10.7)
 6050 FORMAT(1X,'               X= ',F10.7)
 6060 FORMAT(1X,/,/,1X,'*** OUTPUT ***',/,/)
 6100 FORMAT(1X,'OUTPUT VALUES : IERR= ',I5,/,/)
 6200 FORMAT(1X,'ZETA(',F10.7,',1) -1/(',F10.7,' -1)= ',F10.7,&
                '  EULER CONSTANT '         )
 6300 FORMAT(1X,'ZETA(',F10.7,',1) -1/(',F10.7,' -1)= ',F10.7,&
                '  ZETA(',F10.7,',1)= ',F10.7)
      END
```

(c) Output results

```
*** WIXZTA ***

*** INPUT ***

               X=  0.5000000 A=  1.0000000
               X=  1.0000000
               X=  1.5000000
               X=  2.0000000
               X=  2.5000000
               X=  3.0000000
               X=  3.5000000
               X=  4.0000000
```

```
              X=  4.5000000
              X=  5.0000000


   *** OUTPUT ***

   OUTPUT VALUES : IERR=     0

   ZETA( 0.5000000,1) -1/( 0.5000000 -1)=  0.5396455  ZETA( 0.5000000,1)= -1.4603545
   ZETA( 1.0000000,1) -1/( 1.0000000 -1)=  0.5772157  EULER CONSTANT
   ZETA( 1.5000000,1) -1/( 1.5000000 -1)=  0.6123753  ZETA( 1.5000000,1)=  2.6123753
   ZETA( 2.0000000,1) -1/( 2.0000000 -1)=  0.6449341  ZETA( 2.0000000,1)=  1.6449341
   ZETA( 2.5000000,1) -1/( 2.5000000 -1)=  0.6748206  ZETA( 2.5000000,1)=  1.3414873
   ZETA( 3.0000000,1) -1/( 3.0000000 -1)=  0.7020569  ZETA( 3.0000000,1)=  1.2020569
   ZETA( 3.5000000,1) -1/( 3.5000000 -1)=  0.7267339  ZETA( 3.5000000,1)=  1.1267339
   ZETA( 4.0000000,1) -1/( 4.0000000 -1)=  0.7489899  ZETA( 4.0000000,1)=  1.0823232
   ZETA( 4.5000000,1) -1/( 4.5000000 -1)=  0.7689932  ZETA( 4.5000000,1)=  1.0547075
   ZETA( 5.0000000,1) -1/( 5.0000000 -1)=  0.7869278  ZETA( 5.0000000,1)=  1.0369278
```

### 2.15.6 DIXEPS, RIXEPS
### Zeta Function of the Positive Definite Quadratic Form $x^2 + ay^2$

(1) **Function**

For $s > -1$, obtain an analytical continuation of the zeta function subtracted by the function to cancel its pole for a positive quadratic form $x^2 + ay^2$

$$f(s;a) \equiv \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (m^2 + an^2)^{-s} - \frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)}(s > 1).$$

(2) **Usage**

Double precision:
CALL DIXEPS (S, A, Y, IERR)
Single precision:
CALL RIXEPS (S, A, Y, IERR)

(3) **Arguments**

| | | |
|---|---|---|
| D:Double precision real | Z:Double precision complex | I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$ |
| R:Single precision real | C:Single precision complex | |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | S | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Value of $s$ |
| 2 | A | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Coefficient $a$ of the form |
| 3 | Y | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Value of zeta function $f(s;a)$(See Note (a)) |
| 4 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{S} > -1$

(b) $\text{A} > 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The zeta function for positive quadratic form $x^2 + ay^2$ is a rational function which has only single pole at the point $s = 1$. This subroutine does not calculates values of this function itself, but it calculates values of this function subtracted by the function $\frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)}$.

(b) All integer pairs $(m, n)$ without $m = n = 0$.

(7) **Example**

(a) Problem
Obtain value of zeta-function of the form $x^2 + ay^2$ for $s = 3.0, a = 1.0$ and check this value by another method using Hurwitz zeta function.

(b) Input data
S=3.0 and A=1.0.

(c) Main program

```
      PROGRAM BIXEPS
! *** EXAMPLE OF DIXEPS ***
      IMPLICIT NONE
!
      REAL(8) S,Y,A,Z,PAI3,Z1,Z2,Z3,F1,F2
      PARAMETER( PAI3 = 31.006276680299820175476315067101D0/4.D0 )
      INTEGER IERR
!
      S  = 3.D0
      A  = 1.D0
      F1 = 0.25D0
      F2 = 0.75D0
!
      WRITE(6,6000) S, A
!
      CALL DIXEPS(S, A, Y, IERR)
!
      WRITE(6,6010) IERR
      WRITE(6,6020) Y
      Z = Y + PAI3
      CALL DIXZTA(S, A, Z1, IERR)
      IF(IERR.NE.0) WRITE(6,6030)
      CALL DIXZTA(S, F1, Z2, IERR)
      IF(IERR.NE.0) WRITE(6,6030)
      CALL DIXZTA(S, F2, Z3, IERR)
      IF(IERR.NE.0) WRITE(6,6030)
      Z2 = (Z2 - Z3)*F1**S
      Z1 = Z1*Z2/F1
      WRITE(6,6040) Z, Z1
!
      STOP
 6000 FORMAT(/,&
              1X,'***  DIXEPS  ***',/,/,&
              1X,' **  INPUT  **',/,/,&
              1X,'     S= ',F10.7, ' A= ',F10.7,/)
 6010 FORMAT(/,&
              1X,' **  OUTPUT  **',/,/,&
              1X,'     IERR =',I5,/)
 6020 FORMAT(1X,'     Y       = ',F10.7,/)
 6030 FORMAT(1X,'     ** ERROR IN DIXZTA **',/)
 6040 FORMAT(1X,'     ZETA FUNCTION FOR M*M+N*N  =  Y + POLER  = ',&
                F10.7,/,&
              1X,'     ANOTHER OBTAINING  4*ZETA(S)*L(S,KAI4)   = ',&
                F10.7,/)
      END
```

(d) Output results

```
*** DIXEPS ***

 ** INPUT **

    S=  3.0000000 A=  1.0000000

 ** OUTPUT **

    IERR =    0

    Y       = -3.0926556

    ZETA FUNCTION FOR M*M+N*N  =  Y + POLER  =  4.6589136
    ANOTHER OBTAINING  4*ZETA(S)*L(S,KAI4)   =  4.6589136
```

# Chapter 3

# SORTING AND RANKING

## 3.1 INTRODUCTION

This chapter describes the subroutines for sorting, ranking, and merging data.
This library provides subroutines having the following functions.

(1) Sorting a list of data

(2) Sorting a list of pairwise data

(3) Ranking of a list of data

(4) Top-N extraction

(5) Merging two sorted lists of data

(6) Merging two sorted lists of pairwise data

### 3.1.1 Algorithms Used

#### 3.1.1.1 Sorting

The algorithms for sorting in ascending order are 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 portion 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 ascending 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 sort 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 values is one:
  Do nothing.
- If the number of data values is two :
  if they are in ascending order, exchange their 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.

#### 3.1.1.2 Ranking of a list of data

Given $n$ data values, this function returns the ascending rank number corresponding to each data value and the number of data values having the same rank.

#### 3.1.1.3 Top-N extraction

Given $n$ data values $a_i(i = 1, 2, \cdots, n)$, this function obtains the first $m$ data values $a_j(j = j_1, j_2, \cdots, j_m)$ $(m < n)$ of the data sequence consisting of the original data values rearranged in descending or ascending order.

#### 3.1.1.4 Merging two sorted lists of data

This function merges two data sequences $a_i \quad (i = 1, 2, \cdots, n)$ and $b_j \quad (j = 1, 2, \cdots, m)$, which had each been sorted into ascending order, to obtain the data sequence $c_k \quad (k = 1, 2, \cdots, \ell)$, where, $c_k$ satisfies the following relationship.

$$c_1 \leq c_2 \leq \cdots \leq c_\ell$$

#### 3.1.1.5 Merging two sorted list of pairwise data

This function merges the set of data $(a_i, b_i) \quad (i = 1, 2, \cdots, n)$, which had been sorted into ascending order of $a_i$, and the set of data $(c_j, d_j) \quad (j = 1, 2, \cdots, m)$, which had been sorted into ascending order of $c_j$, to obtain the set of data $(e_k, f_k) \quad (k = 1, 2, \cdots, \ell)$, where, $e_k$ satisfies the following relationship.

$$e_1 \leq e_2 \leq \cdots \leq e_\ell$$

If a second order sort was specified, the function determines $k = 1, 2, \cdots, \ell$ so that $f_k \leq f_{k+1}$ for any $k$ for which $e_k = e_{k+1}$ is satisfied.

### 3.1.2 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.

# 3.2 SORTING

## 3.2.1 DSSTA1, RSSTA1 Sorting a List of Data

(1) **Function**

Given $n$ data values $a_{i_k}(k = 1, 2, \cdots, n)$, the DSSTA1 or RSSTA1 subroutine obtain 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 the following relationship.

For ascending order : $a_{j_1} \le a_{j_2} \le \cdots \le a_{j_n}$

For descending order : $a_{j_1} \ge a_{j_2} \ge \cdots \ge a_{j_n}$

(2) **Usage**

Double precision:
CALL DSSTA1 (A,N,ISW,WK,IWK, IERR)
Single precision:
CALL RSSTA1 (A,N,ISW,WK,IWK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 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 | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | See Contents | Work | Work area<br>**Size:**<br>N (When ISW= 4, $-4$ )<br>1 (Otherwise) |
| 5 | IWK | I | See Contents | Work | Work area<br>**Size:**<br>2×N (When ISW= 3, $-3$ )<br>1 (Otherwise) |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N≥1

(b) ISW=1,2,3,4,$-1$,$-2$,$-3$,$-4$

(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. | |

(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) | $-1$ | 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 $0(n^{1.5})$ . 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 BSSTA1
! *** EXAMPLE OF DSSTA1 ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,ISW,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
!
!     WRITE INPUT DATA
      WRITE(6,6000) ISW,N
      DO 110 I=1,N
         WRITE(6,6010) I,A(I)
  110 CONTINUE
!
!     SORT
      CALL DSSTA1(A,N,ISW,WK,IWK,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,'***  DSSTA1  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     ISW =',I6,/,&
             1X,'     N   =',I6,/)
 6010 FORMAT(1X,'     A(',I2,')=',F5.1)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
      END
```

(d) Output results

```
***  DSSTA1  ***

 **  INPUT  **

     ISW =     1
     N   =     6

     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
```

### 3.2.2 DSSTA2, RSSTA2 Sorting a List of Pairwise Data

(1) **Function**

Given two sets of n data values $a_{i_k}(k = 1, 2, \cdots, n), b_{i_k}(k = 1, 2, \cdots, n)$, this subroutine 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 DSSTA2 (A,N,B,ISW1,ISW2,WK,IWK, IERR)

Single precision:

CALL RSSTA2 (A,N,B,ISW1,ISW2,WK,IWK, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | N | Input | Data to be sorted $a_i$ |
| | | | | Output | Sorted data $a_j$ |
| 2 | N | I | 1 | Input | Size of array A |
| 3 | B | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | 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<br>ISW2=0 : Do not perform second order sort<br>ISW2=1 : Perform second order sort |
| 6 | WK | $\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$ | See Contents | Work | Work area<br>**Size:**<br>2×N (When ISW1= $4, -4$)<br>1 (Otherwise) |
| 7 | IWK | I | See Contents | Work | Work area<br>**Size:**<br>2×N (When ISW1= $3, -3$)<br>1 (Otherwise) |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N≥1

(b) ISW1=1,2,3,4,−1,−2,−3,−4

(c) ISW2=0 or 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 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 BSSTA2
! *** EXAMPLE OF DSSTA2 ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,ISW1,ISW2,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
!
!     WRITE INPUT DATA
      WRITE(6,6000) ISW1,ISW2,N
      DO 110 I=1,N
         WRITE(6,6010) I,A(I),I,B(I)
  110 CONTINUE
!
!     SORT
      CALL DSSTA2(A,N,B,ISW1,ISW2,WK,IWK,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,'***  DSSTA2  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     ISW1 =',I6,/,&
             1X,'     ISW2 =',I6,/,&
             1X,'     N    =',I6,/)
 6010 FORMAT(1X,'     A(',I2,')=',F5.1,7X,'B(',I2,')=',F5.1)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR = ',I5,/)
      END
```

(d) Output results

```
***  DSSTA2  ***

 **  INPUT  **

     ISW1 =     1
     ISW2 =     1
     N    =     6

     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
```

## 3.3 RANKING

### 3.3.1 DSSTRA, RSSTRA
### Ranking of a List of Data

(1) **Function**

Given $n$ data values, the DSSTRA or RSSTRA returns the ascending rank number corresponding to each data value and the number of data values having the same rank (See Note (a)). The precise specifications are as follows. Given $n$ data values $a_i (i = 1, 2, \cdots, n)$, if the data sequence consisting of the original data values rearranged in ascending order are given by the following $a_j$ $(j = j_1, j_2, \cdots, j_{m_1+\cdots+m_k})$:

$$\begin{aligned} a_{j_1} = \quad & a_{j_2} \cdots = a_{j_{m_1}} \leq \\ & a_{j_{m_1+1}} = a_{j_{m_1+2}} \cdots = a_{j_{m_1+m_2}} \leq \\ & \cdots \leq \\ & a_{j_{m_1+\cdots+m_{k-1}+1}} = a_{j_{m_1+\cdots+m_{k-1}+2}} \cdots = a_{j_{m_1+\cdots+m_k}} \end{aligned}$$

where $(m_1 + \cdots + m_k = n)$, the subroutine obtains the ranking data $r_j (j = 1, 2, \cdots, n)$ defined by $r_{j_{m_1+\cdots+m_{l-1}+1}} = r_{j_{m_1+\cdots+m_{l-1}+2}} = \cdots = r_{j_{m_1+\cdots+m_l}} = l$. Here, $m_l$ is the number of identical rankings for the $l$-th smallest data value. To obtain the number of identical rankings, the data is stored in $c_j (j = 1, 2, \cdots, n)$ so that $c_{j_{m_1+\cdots+m_{l-1}+1}} = c_{j_{m_1+\cdots+m_{l-1}+2}} = \cdots = c_{j_{m_1+\cdots+m_l}} = m_l$ is satisfied.

(2) **Usage**

Double precision:

CALL DSSTRA (A, N, IR, IC, ISW, IW, IERR)

Single precision:

CALL RSSTRA (A, N, IR, IC, ISW, IW, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N | Input | Data that is to be ranked $a_i$ |
| 2 | N | I | 1 | Input | Size of array A |
| 3 | IR | I | N | Output | Assigned rankings $r_j$ associated with A |
| 4 | IC | I | N | Output | When ISW=0,<br>number of identically ranked data $c_j$<br>When ISW=1<br>this is not used (See Note (b)) |
| 5 | ISW | I | 1 | Input | Identically ranked data count output switch<br>ISW=0: Output the number of identically ranked data in IC.<br>ISW=1: Do not output the number of identically ranked data. |
| 6 | IW | I | N | Work | Work area |
| 7 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N≥2

(b) ISW=0 or ISW=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. | |

(6) **Notes**

(a) A(i) is the IR(i)-th smallest data value among all of the data, and when ISW=0, the number of IR(i)-th smallest data values is IC(i).

(b) Since IC is not used when ISW=1, a dummy array can be set for the argument.

(7) **Example**

(a) ProblemRank the following data.
A(1) = 1.2
A(2) = 3.2
A(3) = 4.2
A(4) = 5.2

A(5) = 7.2
A(6) = 1.2
A(7) = 9.2
A(8) = 1.2
A(9) = 1.2
A(10) = 7.2
A(11) = 6.2
A(12) = 8.2
A(13) = 7.2
A(14) = 5.2
A(15) = 0.2
A(16) = 2.2

(b) Input data

Array A, N=16 and ISW=0.

(c) Main program

```
      PROGRAM BSSTRA
! *** EXAMPLE OF DSSTRA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(N=16,ISW=0)
      DIMENSION A(N),IR(N),IC(N),IW(N)
!
      READ(5,*) (A(I),I=1,N)
      WRITE(6,6000) N,ISW
      CALL DSSTRA(A,N,IR,IC,ISW,IW,IERR)
      WRITE(6,6010) IERR
      WRITE(6,6020)
      DO 100 I=1,N
       WRITE(6,6030) I,A(I),IR(I),IC(I)
  100 CONTINUE
      STOP
!
 6000 FORMAT('  *** DSSTRA ***',/,/,'   ** INPUT **',/,/,7X,'N   = ',I4,/,7X,&
             'ISW = ',I4,/)
 6010 FORMAT('   ** OUTPUT **',/,/,7X,'IERR = ',I4,/)
 6020 FORMAT(14X,'A',7X,'IR',6X,'IC')
 6030 FORMAT(6X,I3,2X,F5.1,3X,I5,3X,I5)
!
      END
```

(d) Output results

```
 *** DSSTRA ***

  ** INPUT **

      N   =   16
      ISW =    0

  ** OUTPUT **

      IERR =    0

             A       IR      IC
       1    1.2       2       4
       2    3.2       7       1
       3    4.2       8       1
       4    5.2       9       2
       5    7.2      12       3
       6    1.2       2       4
       7    9.2      16       1
       8    1.2       2       4
       9    1.2       2       4
      10    7.2      12       3
      11    6.2      11       1
      12    8.2      15       1
      13    7.2      12       3
      14    5.2       9       2
      15    0.2       1       1
      16    2.2       6       1
```

## 3.3.2 DSSTPT, RSSTPT
## Top-N Extraction

(1) **Function**

Given $n$ data values $a_i(i = 1, 2, \cdots, n)$, the DSSTPT or RSSTPT obtains the first $m$ data values $a_j(j = j_1, j_2, \cdots, j_m)$ $(m < n)$ of the data sequence consisting of the original data values rearranged in descending or ascending order.

(2) **Usage**

Double precision:
CALL DSSTPT (A, N, M, P, ISW, IERR)
Single precision:
CALL RSSTPT (A, N, M, P, ISW, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Input | Input data $a_i$ |
| | | | | Output | Data sorted in descending or ascending order $a_j$ |
| 2 | N | I | 1 | Input | Size of array A |
| 3 | M | I | 1 | Input | Number of data to be sorted in descending or ascending order (See Note (a)) |
| | | | | Output | Number of data actually sorted in ascending or ascending order |
| 4 | P | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Input | Parameter for obtaining the initial value of the threshold value (See Note (b)) |
| | | | | Output | Number of times threshold value is updated (See Note (b)) |
| 5 | ISW | I | 1 | Input | ISW=0: Sort in ascending order.<br>ISW=1: Sort in descending order. (See Note (a)) |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ISW$\in$ {0, 1}

(b) M$\leq$0, N$\leq$0, N<M

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The desired result is to return the elements of array A sorted in descending (ISW=1) or ascending (ISW=0) order in the first M elements of array A. For M, enter the number of elements you want to be sorted, and the number of elements that were actually sorted is output in M. Here, (number of elements you want to be sorted) $\leq$ (number of elements that were actually sorted).

(b) The processing of this subroutine proceeds by sequentially dividing the input data based on a certain threshold value into a set of data greater than the threshold value and a set less than the threshold value. When the size of the set obtained in this way approaches the number of elements you want to be sorted, that set is rearranged. The initial threshold value is calculated as follows based on the data assigned for parameter P.

$$\text{Initial value of threshold value } = \text{MAX} \times \text{P} + \text{MIN} \times (1.0 - \text{P})$$

Here, MAX represents the maximum value of the data contained in array A and MIN represents the minimum value of the data contained in array A. Therefore, the initial value of the threshold value is defined as the point where the MAX and MIN are internally divided into $(1.0 - \text{P}) : \text{P}$. When the characteristics of the data to be sorted are known, the processing speed can be increased by assigning suitable data for P, that is, the initial value of the threshold value. For example, if $n$ uniformly random numbers from the interval (0, 1) are given and you want to get the smallest $m$ data values among them, the optimum threshold value estimate is $\frac{m}{n}$. Therefore, you should specify P as follows.

$$\text{P} = \frac{m}{n}$$

The number of times the threshold value actually was updated is output in P as a $\left\{\begin{array}{c}\text{double-precision}\\\text{single-precision}\end{array}\right\}$ real number. If this value is small, it indicates that the initial value of the threshold value was suitable.

(7) **Example**

(a) Problem

Obtain the five smallest data values when the following data has been sorted in ascending order.

A(1) = 5.0
A(2) = 39.0
A(3) = 15.0
A(4) = 8.0
A(5) = 23.0
A(6) = 45.0
A(7) = 61.0
A(8) = 25.0
A(9) = 33.0
A(10) = 45.0
A(11) = 39.0

A(12) = 10.0
A(13) = 21.0
A(14) = 5.0
A(15) = 23.0
A(16) = 38.0
A(17) = 41.0
A(18) = 55.0
A(19) = 61.0
A(20) = 39.0

(b) Input data

Array A, N=20, M=5, P=0.3 and ISW=0.

(c) Main program

```
      PROGRAM BSSTPT
! *** EXAMPLE OF DSSTPT ***
      IMPLICIT REAL(8) (A-H,O-Z)
      REAL(8)    A(100)
      INTEGER  I,N,IERR,M
      INTEGER ISW
      ISW=0
!
      READ(*,5000)N,M,P,ISW
      WRITE(6,6000) N,M,ISW,P
      READ(*,5010)(A(I),I=1,20)
      DO 100 I=1,N
      WRITE(6,6010)I,A(I)
  100 CONTINUE
!
      CALL DSSTPT(A,N,M,P,ISW,IERR)
!
      WRITE(6,6020)
      WRITE(6,6030) IERR
      WRITE(6,6040) M
      DO 110 I=1,M
      WRITE(6,6010)I,A(I)
  110 CONTINUE
!
 5000 FORMAT(I2,I2,F4.1,I2)
 5010 FORMAT(20F5.1)
 6000 FORMAT(' ',/,5X,'*** DSSTPT ***',/,&
             ' ',/,6X,'** INPUT **',/,/,&
            9X,'N=',I2,'  M=',I2,'  ISW=',I2,'  P=',F5.1,/)
 6010 FORMAT(9X,'A(',I2,')',F5.1)
 6020 FORMAT(' ',/,6X,'** OUTPUT **',/)
 6030 FORMAT(9X,'IERR = ',I4)
 6040 FORMAT(9X,'M    = ',I4,/)
      END
```

(d) Output results

```
    *** DSSTPT ***

     ** INPUT **

        N=20  M= 5  ISW= 0  P=  0.3

        A( 1)  5.0
        A( 2) 39.0
        A( 3) 15.0
        A( 4)  8.0
        A( 5) 23.0
        A( 6) 45.0
        A( 7) 61.0
        A( 8) 25.0
        A( 9) 33.0
        A(10) 45.0
        A(11) 39.0
        A(12) 10.0
        A(13) 21.0
        A(14)  5.0
        A(15) 23.0
        A(16) 38.0
        A(17) 41.0
        A(18) 55.0
        A(19) 61.0
        A(20) 39.0

     ** OUTPUT **

        IERR =    0
        M    =    6
```

```
A( 1)  5.0
A( 2)  5.0
A( 3)  8.0
A( 4) 10.0
A( 5) 15.0
A( 6) 21.0
```

## 3.4 MERGING

### 3.4.1 DSMGON, RSMGON
### Merging Two Sorted Lists of Data

(1) **Function**

The DSMGON or RSMGON merges two data sequences $a_i$ $(i = 1, 2, \cdots, n)$ and $b_j$ $(j = 1, 2, \cdots, m)$, which had each been sorted into ascending order, to obtain the data sequence $c_k$ $(k = 1, 2, \cdots, \ell)$, where, $c_k$ satisfies the following relationship.

$$c_1 \leq c_2 \leq \cdots \leq c_\ell$$

(2) **Usage**

Double precision:
CALL DSMGON (A, NN, B, NM, C, NL, IERR)
Single precision:
CALL RSMGON (A, NN, B, NM, C, NL, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NN | Input | Data to be merged $a_i$. |
| 2 | NN | I | 1 | Input | Size of array A. |
| 3 | B | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NM | Input | Data to be merged $b_j$ |
| 4 | NM | I | 1 | Input | Size of array B. |
| 5 | C | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NL | Output | Merged data $c_k$ |
| 6 | NL | I | 1 | Input | Size of array C. |
| 7 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NN $\geq 1$

(b) NM $\geq 1$

(c) $1 \leq$ NL $\leq$ NN + NM

(d) A(1) $\leq$ A(2) $\leq \cdots \leq$ A(NN)

(e) B(1) $\leq$ B(2) $\leq \cdots \leq$ B(NM)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Restriction (c) was not satisfied. | NL = NN + NM is set and processing is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (b) was not satisfied. | |
| 3200 | Restriction (d) was not satisfied. | |
| 3300 | Restriction (e) was not satisfied. | |

(6) **Notes**

(a) When NL < NN + NM, only the NL smallest values of the merged result are output.

(7) **Example**

(a) Problem
Divide the sequence $a_i$ $(i = 1, 2, \cdots, n)$, which contains $n$ data stored in array X, into partial sequence of the length $n_s$, and iterate merging of the partial sequences to sort the whole of the sequence $a_i$. And, when files on an external memory device, instead of the arrays in the program below, are used, this problem becomes an external sort problem of $n$ data $a_i$.

(b) Input data
Array X and length $n$.

(c) Main program

```
      PROGRAM BSMGON
! *** EXAMPLE OF DSMGON ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,IERR
      INTEGER ILOOP,ISIZE,ISIZE2,IA,IB,IC,ICREST
      INTEGER ISTA,ISTB,ISTC,ISIZEA,ISIZEB,ISIZEC
      INTEGER I,J,K
      REAL(8) X(NA),A(NA),B(NA),C(NA)
!
!     DATA SET
      N = 17
      DO 100 I=1,N
         X(I) = DBLE( INT(SIN(DBLE(I))*100) )
  100 CONTINUE
!
!     WRITE INPUT DATA
      WRITE(6,6000) N
      DO 110 I=1,N
         WRITE(6,6010) I,X(I)
  110 CONTINUE
!
!     EXTERNAL SORT
      DO 120 I=1,N
         C(I) = X(I)
  120 CONTINUE
      IF( N .EQ. 1 )THEN
         GOTO 130
      ENDIF
!
      ILOOP = 0
      DO 140 I=1,N
         ILOOP = ILOOP + 1
         IF( 2**ILOOP .GE. N ) THEN
            GOTO 150
         ENDIF
  140 CONTINUE
  150 CONTINUE
      DO 160 I=1,ILOOP
         ISIZE  = 2 ** (I-1)
         ISIZE2 = 2 ** I
!
         IA = 0
```

```
         IB = 0
         IC = 0
         DO 170 J=1,N/ISIZE2
            DO 180 K=1,ISIZE
               IA = IA + 1
               A(IA) = C(IC+K)
  180       CONTINUE
            IC = IC + ISIZE
            DO 190 K=1,ISIZE
               IB = IB + 1
               B(IB) = C(IC+K)
  190       CONTINUE
            IC = IC + ISIZE
  170    CONTINUE
         ICREST = N - IC
         IF( (0 .LT. ICREST ) .AND. (ICREST .LE. ISIZE) ) THEN
            DO 200 K=1,ICREST
               IA = IA + 1
               A(IA) = C(IC+K)
  200       CONTINUE
         ENDIF
         IF( (ISIZE .LT. ICREST ) .AND. (ICREST .LT. (ISIZE*2)) ) THEN
            DO 210 K=1,ISIZE
               IA = IA + 1
               A(IA) = C(IC+K)
  210       CONTINUE
            IC = IC + ISIZE
            DO 220 K=1,(ICREST-ISIZE)
               IB = IB + 1
               B(IB) = C(IC+K)
  220       CONTINUE
         ENDIF
!
         DO 230 J=1,N/ISIZE2
            ISTA = (J-1) * ISIZE + 1
            ISTB = (J-1) * ISIZE + 1
            ISTC = (J-1) * ISIZE2 + 1
            ISIZEA = ISIZE
            ISIZEB = ISIZE
            ISIZEC = ISIZEA + ISIZEB
            CALL DSMGON&
            (A(ISTA),ISIZEA,B(ISTB),ISIZEB,C(ISTC),ISIZEC,IERR)
  230    CONTINUE
         IF( (0 .LT. ICREST ) .AND. (ICREST .LE. ISIZE) ) THEN
            ISTA = N/ISIZE2 * ISIZE
            ISTC = N/ISIZE2 * ISIZE2
            ISIZEA = ICREST
            DO 240 K=1,ISIZEA
               C(ISTC+K) = A(ISTA+K)
  240       CONTINUE
         ENDIF
         IF( (ISIZE .LT. ICREST ) .AND. (ICREST .LT. (ISIZE*2)) ) THEN
            ISTA = N/ISIZE2 * ISIZE + 1
            ISTB = N/ISIZE2 * ISIZE + 1
            ISTC = N/ISIZE2 * ISIZE2 + 1
            ISIZEA = ISIZE
            ISIZEB = ICREST - ISIZE
            ISIZEC = ISIZEA + ISIZEB
            CALL DSMGON&
            (A(ISTA),ISIZEA,B(ISTB),ISIZEB,C(ISTC),ISIZEC,IERR)
         ENDIF
  160 CONTINUE
  130 CONTINUE
!
!     WRITE OUTPUT DATA
      WRITE(6,6020) IERR
      IF( IERR .LT. 3000 ) THEN
         DO 250 I=1,N
            WRITE(6,6030) I,C(I)
  250    CONTINUE
      ENDIF
!
      STOP
 6000 FORMAT(/,&
             1X,'***  DSMGON  ***',/,/,&
             1X,' **  INPUT  **',/,/,&
             1X,'     N =',I6,/)
 6010 FORMAT(1X,'     X(',I2,')=',F5.1)
 6020 FORMAT(/,&
             1X,' **  OUTPUT  **',/,/,&
             1X,'     IERR =',I5,/)
 6030 FORMAT(1X,'     C(',I2,')=',F5.1)
      END
```

(d) Output results

```
***  DSMGON  ***

 **  INPUT  **

     N =    17

     X( 1)= 84.0
     X( 2)= 90.0
```

```
     X( 3)= 14.0
     X( 4)=-75.0
     X( 5)=-95.0
     X( 6)=-27.0
     X( 7)= 65.0
     X( 8)= 98.0
     X( 9)= 41.0
     X(10)=-54.0
     X(11)=-99.0
     X(12)=-53.0
     X(13)= 42.0
     X(14)= 99.0
     X(15)= 65.0
     X(16)=-28.0
     X(17)=-96.0

**   OUTPUT  **

     IERR =    0

     C( 1)=-99.0
     C( 2)=-96.0
     C( 3)=-95.0
     C( 4)=-75.0
     C( 5)=-54.0
     C( 6)=-53.0
     C( 7)=-28.0
     C( 8)=-27.0
     C( 9)= 14.0
     C(10)= 41.0
     C(11)= 42.0
     C(12)= 65.0
     C(13)= 65.0
     C(14)= 84.0
     C(15)= 90.0
     C(16)= 98.0
     C(17)= 99.0
```

### 3.4.2 DSMGPA, RSMGPA Merging Two Sorted Lists of Pairwise Data

(1) **Function**

The DSMGPA or RSMGPA merges the set of data $(a_i, b_i)$ $(i = 1, 2, \cdots, n)$, which had been sorted into ascending order of $a_i$, and the set of data $(c_j, d_j)$ $(j = 1, 2, \cdots, m)$, which had been sorted into ascending order of $c_j$, to obtain the set of data $(e_k, f_k)$ $(k = 1, 2, \cdots, \ell)$, where, $e_k$ satisfies the following relationship.

$$e_1 \leq e_2 \leq \cdots \leq e_\ell$$

If a second order sort was specified, the subroutine determines $k = 1, 2, \cdots, \ell$ so that

$$f_k \leq f_{k+1}$$

for any $k$ for which $e_k = e_{k+1}$ is satisfied.

(2) **Usage**

Double precision:
CALL DSMGPA (A, NN, B, C, NM, D, E, NL, F, ISW, IERR)
Single precision:
CALL RSMGPA (A, NN, B, C, NM, D, E, NL, F, ISW, IERR)

### (3) Arguments

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NN | Input | Data to be merged $a_i$. |
| 2 | NN | I | 1 | Input | Size of array A. |
| 3 | B | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NN | Input | Data $b_i$ corresponding to $a_i$. |
| 4 | C | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NM | Input | Data to be merged $c_j$ |
| 5 | NM | I | 1 | Input | Size of array C. |
| 6 | D | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NM | Input | Data $d_j$ corresponding to $c_j$. |
| 7 | E | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NL | Output | Merged data $e_k$. |
| 8 | NL | I | 1 | Input | Size of array E. |
| 9 | F | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | NL | Output | Data $f_k$ corresponding to $e_k$. |
| 10 | ISW | I | 1 | Input | Second order sort switch<br>ISW=0: Do not perform second order sort<br>ISW=1: Perform second order sort |
| 11 | IERR | I | 1 | Output | Error indicator |

### (4) Restrictions

(a) NN $\geq 1$

(b) NM $\geq 1$

(c) $1 \leq$ NL $\leq$ NN + NM

(d) $\mathrm{A}(1) \leq \mathrm{A}(2) \leq \cdots \leq \mathrm{A(NN)}$

(e) $\mathrm{C}(1) \leq \mathrm{C}(2) \leq \cdots \leq \mathrm{C(NM)}$

(f) ISW=0 or ISW=1

(g) When ISW=1 is specified, B($i$)≤B($i$+1) must be satisfied for any $i$ for which A($i$)=A($i$+1) is satisfied.

(h) When ISW=1 is specified, D($i$)≤D($i$+1) must be satisfied for any $i$ for which C($i$)=C($i$+1) is satisfied.

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Restriction (c) was not satisfied. | NL = NN + NM is set and processing is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (b) was not satisfied. | |
| 3200 | Restriction (d) was not satisfied. | |
| 3300 | Restriction (e) was not satisfied. | |
| 3400 | Restriction (f) was not satisfied. | |
| 3500 | Restriction (g) was not satisfied. | |
| 3600 | Restriction (h) was not satisfied. | |

(6) **Notes**

(a) When NL < NN + NM, only the NL smallest values of the merged result are output.

(7) **Example**

(a) Problem
Divide the sequence $(a_i, b_i)$ $(i = 1, 2, \cdots, n)$, which contains a couple of $n$ data stored in arrays X and Y, into partial sequences of the length $n_s$, and iterate merging of the partial sequences to sort the whole of the sequence $(a_i, b_i)$. And, when files on an external memory device, instead of the arrays in the program below, are used, this problem becomes an external sort problem of a couple of $n$ data $a_i$ and $b_i$.

(b) Input data
Array X, length $n$ and ISW=1.

(c) Main program

```
      PROGRAM BSMGPA
! *** EXAMPLE OF DSMGPA ***
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER( NA = 100 )
      INTEGER N,ISW,IERR
      INTEGER ILOOP,ISIZE,ISIZE2,IA,IB,IC,ICREST
      INTEGER ISTA,ISTB,ISTC,ISIZEA,ISIZEB,ISIZEC
      INTEGER I,J,K
      REAL(8) PI
      PARAMETER( PI = 3.1415926535897932384D0 )
      REAL(8) X(NA),A(NA), B(NA), C(NA)
      REAL(8) Y(NA),A2(NA),B2(NA),C2(NA)
!
!     DATA SET
      N = 17
      DO 100 I=1,N
         X(I) = DBLE( INT(SIN(DBLE(I))*100) )
         Y(I) = DBLE( INT(SIN(DBLE(I)+PI*0.5D0)*100) )
  100 CONTINUE
      ISW = 1
!
!     WRITE INPUT DATA
      WRITE(6,6000) N,ISW
      DO 110 I=1,N
         WRITE(6,6010) I,X(I),I,Y(I)
  110 CONTINUE
!
!     EXTERNAL SORT
      DO 120 I=1,N
         C(I)  = X(I)
         C2(I) = Y(I)
  120 CONTINUE
      IF( N .EQ. 1 )THEN
         GOTO 130
      ENDIF
```

```
!
      ILOOP = 0
      DO 140 I=1,N
         ILOOP = ILOOP + 1
         IF( 2**ILOOP .GE. N ) THEN
            GOTO 150
         ENDIF
  140 CONTINUE
  150 CONTINUE
      DO 160 I=1,ILOOP
         ISIZE  = 2 ** (I-1)
         ISIZE2 = 2 ** I
!
         IA = 0
         IB = 0
         IC = 0
         DO 170 J=1,N/ISIZE2
            DO 180 K=1,ISIZE
               IA = IA + 1
               A(IA)  = C(IC+K)
               A2(IA) = C2(IC+K)
  180       CONTINUE
            IC = IC + ISIZE
            DO 190 K=1,ISIZE
               IB = IB + 1
               B(IB)  = C(IC+K)
               B2(IB) = C2(IC+K)
  190       CONTINUE
            IC = IC + ISIZE
  170    CONTINUE
         ICREST = N - IC
         IF( (0 .LT. ICREST ) .AND. (ICREST .LE. ISIZE) ) THEN
            DO 200 K=1,ICREST
               IA = IA + 1
               A(IA)  = C(IC+K)
               A2(IA) = C2(IC+K)
  200       CONTINUE
         ENDIF
         IF( (ISIZE .LT. ICREST ) .AND. (ICREST .LT. (ISIZE*2)) ) THEN
            DO 210 K=1,ISIZE
               IA = IA + 1
               A(IA)  = C(IC+K)
               A2(IA) = C2(IC+K)
  210       CONTINUE
            IC = IC + ISIZE
            DO 220 K=1,(ICREST-ISIZE)
               IB = IB + 1
               B(IB)  = C(IC+K)
               B2(IB) = C2(IC+K)
  220       CONTINUE
         ENDIF
!
         DO 230 J=1,N/ISIZE2
            ISTA = (J-1) * ISIZE + 1
            ISTB = (J-1) * ISIZE + 1
            ISTC = (J-1) * ISIZE2 + 1
            ISIZEA = ISIZE
            ISIZEB = ISIZE
            ISIZEC = ISIZEA + ISIZEB
            CALL DSMGPA&
            (A(ISTA),ISIZEA,A2(ISTA),B(ISTB),ISIZEB,B2(ISTB),&
             C(ISTC),ISIZEC,C2(ISTC),ISW,IERR)
  230    CONTINUE
         IF( (0 .LT. ICREST ) .AND. (ICREST .LE. ISIZE) ) THEN
            ISTA = N/ISIZE2 * ISIZE
            ISTC = N/ISIZE2 * ISIZE2
            ISIZEA = ICREST
            DO 240 K=1,ISIZEA
               C(ISTC+K)  = A(ISTA+K)
               C2(ISTC+K) = A2(ISTA+K)
  240       CONTINUE
         ENDIF
         IF( (ISIZE .LT. ICREST ) .AND. (ICREST .LT. (ISIZE*2)) ) THEN
            ISTA = N/ISIZE2 * ISIZE + 1
            ISTB = N/ISIZE2 * ISIZE + 1
            ISTC = N/ISIZE2 * ISIZE2 + 1
            ISIZEA = ISIZE
            ISIZEB = ICREST - ISIZE
            ISIZEC = ISIZEA + ISIZEB
            CALL DSMGPA&
            (A(ISTA),ISIZEA,A2(ISTA),B(ISTB),ISIZEB,B2(ISTB),&
             C(ISTC),ISIZEC,C2(ISTC),ISW,IERR)
         ENDIF
  160 CONTINUE
  130 CONTINUE
!
!     WRITE OUTPUT DATA
      WRITE(6,6020) IERR
      IF( IERR .LT. 3000 ) THEN
         DO 250 I=1,N
            WRITE(6,6030) I,C(I),I,C2(I)
  250    CONTINUE
      ENDIF
!
      STOP
```

```
6000 FORMAT(/,&
            1X,'***  DSMGPA  ***',/,/,&
            1X,' **  INPUT  **',/,/,&
            1X,'     N   =',I6,/,&
            1X,'     ISW =',I6,/)
6010 FORMAT(1X,'     X(',I2,')=',F5.1,7X,' Y(',I2,')=',F5.1)
6020 FORMAT(/,&
            1X,' **  OUTPUT  **',/,/,&
            1X,'     IERR =',I5,/)
6030 FORMAT(1X,'     C(',I2,')=',F5.1,7X,'C2(',I2,')=',F5.1)
     END
```

(d) Output results

```
***  DSMGPA  ***

 **  INPUT  **

    N   =    17
    ISW =     1

    X( 1)= 84.0        Y( 1)= 54.0
    X( 2)= 90.0        Y( 2)=-41.0
    X( 3)= 14.0        Y( 3)=-98.0
    X( 4)=-75.0        Y( 4)=-65.0
    X( 5)=-95.0        Y( 5)= 28.0
    X( 6)=-27.0        Y( 6)= 96.0
    X( 7)= 65.0        Y( 7)= 75.0
    X( 8)= 98.0        Y( 8)=-14.0
    X( 9)= 41.0        Y( 9)=-91.0
    X(10)=-54.0        Y(10)=-83.0
    X(11)=-99.0        Y(11)=  0.0
    X(12)=-53.0        Y(12)= 84.0
    X(13)= 42.0        Y(13)= 90.0
    X(14)= 99.0        Y(14)= 13.0
    X(15)= 65.0        Y(15)=-75.0
    X(16)=-28.0        Y(16)=-95.0
    X(17)=-96.0        Y(17)=-27.0

 **  OUTPUT  **

    IERR =    0

    C( 1)=-99.0       C2( 1)=  0.0
    C( 2)=-96.0       C2( 2)=-27.0
    C( 3)=-95.0       C2( 3)= 28.0
    C( 4)=-75.0       C2( 4)=-65.0
    C( 5)=-54.0       C2( 5)=-83.0
    C( 6)=-53.0       C2( 6)= 84.0
    C( 7)=-28.0       C2( 7)=-95.0
    C( 8)=-27.0       C2( 8)= 96.0
    C( 9)= 14.0       C2( 9)=-98.0
    C(10)= 41.0       C2(10)=-91.0
    C(11)= 42.0       C2(11)= 90.0
    C(12)= 65.0       C2(12)=-75.0
    C(13)= 65.0       C2(13)= 75.0
    C(14)= 84.0       C2(14)= 54.0
    C(15)= 90.0       C2(15)=-41.0
    C(16)= 98.0       C2(16)=-14.0
    C(17)= 99.0       C2(17)= 13.0
```

# Chapter 4

# ROOTS OF EQUATIONS

## 4.1 INTRODUCTION

This chapter describes subroutines that obtain the roots of an algebraic equation, a nonlinear equation or a set of simultaneous nonlinear equations.
This library provides the following two types of subroutines for obtaining the roots of the algebraic equations.

(1) Subroutine that takes real-type input of real coefficients and provides real-type output of complex roots

(2) Subroutine that takes complex-type input of complex coefficients and provides complex-type output of complex roots

In addition, this library provides the following three types of subroutines for obtaining the roots of nonlinear equations.

(1) Subroutine that obtains one root when given an initial value

(2) Subroutine that obtains one root when given an interval

(3) Subroutine that obtains all roots within an interval

Among these subroutines, the ones that obtain a single root of a real function by assigning an initial value have been designed with particularly careful consideration so that a root is obtained even if the initial value is far from the root or if the function oscillates in the interval between the root and initial value.
When obtaining the roots of a set of simultaneous nonlinear equations, a Jacobian matrix calculation subroutine may or may not be given. Subroutines for both of these cases have global convergence, and careful consideration has been given so that these problems can be solved even without scaling each of the simultaneous equations.

### 4.1.1 Notes

(1) Although for real coefficient algebraic equations, roots near zero tend to be obtained first, this is not the case for complex coefficient algebraic equations.

(2) If there are multiple roots and the multiplicity of the roots is $n$, then for an algebraic equation, the precision of the solution at that root is on the order of $\sqrt[n]{\text{Unit for determining error}}$, and for a nonlinear equation for which the solution is obtained by assigning an initial value, the precision is on the order of $n \times$ (required precision).

(3) Except when obtaining a single root within an interval, convergence is determined for a nonlinear equation as follows. If $e_r$ is the required precision, $\varepsilon$ is the unit for determining error, and $\Delta x$ is the amount $x$ is updated, then convergence is considered to have occurred when the following relationships hold:

$$(\mid \Delta x \mid < e_r \times \max(1, \mid x \mid) \quad \text{and} \quad \mid f(x) \mid < e_r + 64 \times \varepsilon \times \mid x \mid) \quad \text{or} \quad f(x) = 0$$

That is, convergence is considered to have occurred when the function value is zero or when both the function value and the movement of the solution are close to zero. Therefore, even if the problem is ill conditioned, convergence is correctly determined, and if there are multiple roots, precision can be maintained although the amount of calculations increases.
However, this decision criterion tends to be more severe than a method that decides based on only the movement of the solution, based on only the function value or based on either the movement of the solution or the function value.

(4) Convergence is considered to have occurred for a set of simultaneous nonlinear equations when the following relationships hold for all $i$ $(i = 1, \cdots,$ number of order):

$$(\mid \Delta x_i \mid < e_r \times \max(1, \mid x_i \mid) \quad \text{and} \quad \|f(x)\|_\infty < e_r + 64 \times \varepsilon \times \mid x_i \mid) \quad \text{or} \quad f_i(x) = 0$$

(5) Note the following within a program that uses any of these ASL subroutines to obtain roots of a nonlinear equation or a set of simultaneous nonlinear equations.

(a) Names of equation-defining functions or subroutines that are to be used as arguments must be declared by using an EXTERNAL statement.
Example:

Nonlinear equation (Let $\boxed{\text{F}}$ and $\boxed{\text{DF}}$ have the same names in the main program and the function subprograms)

- Main program

$$\wr$$

$$\text{EXTERNAL } \boxed{\text{F}}, \boxed{\text{DF}}$$

$$\wr$$

$$\text{CALL } \begin{Bmatrix} \text{DLNRDS} \\ \text{RLNRDS} \end{Bmatrix} (\boxed{\text{F}}, \boxed{\text{DF}}, \cdots)$$

$$\wr$$

- Function subprograms

$$\begin{array}{l} \text{FUNCTION } \boxed{\text{F}} \text{ (X)} \\ \quad \wr \\ \boxed{\text{F}} = f(\text{X}) \\ \quad \wr \\ \text{FUNCTION } \boxed{\text{DF}} \text{ (X)} \\ \quad \wr \\ \boxed{\text{DF}} = f'(\text{X}) \\ \quad \wr \end{array}$$

Example:

Set of simultaneous nonlinear equations (Let $\boxed{\text{SUB}}$ and $\boxed{\text{SUBD}}$ have the same names in the main program and the subroutine subprograms)

- Main program

$$\begin{array}{l} \quad \wr \\ \text{EXTERNAL } \boxed{\text{SUB}}, \boxed{\text{SUBD}} \\ \quad \wr \\ \text{CALL} \begin{Bmatrix} \text{DLSRDS} \\ \text{RLSRDS} \end{Bmatrix} (\boxed{\text{SUB}}, \boxed{\text{SUBD}}, \cdots) \\ \quad \wr \end{array}$$

- Subroutine subprograms

$$\begin{array}{l} \text{SUBROUTINE } \boxed{\text{SUB}} \text{ (X, N, F)} \\ \text{DIMENSION X(N), F(N)} \\ \quad \wr \\ \text{F}(1) = f_1(\text{X}(1), \cdots, \text{X(N)}) \\ \quad \wr \\ \text{F(N)} = f_\text{N}(\text{X}(1), \cdots, \text{X(N)}) \\ \quad \wr \\ \text{SUBROUTINE } \boxed{\text{SUBD}} \text{ (X, N, A)} \\ \text{DIMENSION X(N), A(N, N)} \\ \quad \wr \\ \text{A}(1,1) = \partial f_1/\partial x_1 \\ \quad \wr \\ \text{A(N, N)} = \partial f_\text{N}/\partial x_\text{N} \\ \quad \wr \end{array}$$

(6) If several errors occur at the same time, only the value of the most severe error will be output for the error indicator, and information about the other errors may be lost.

### 4.1.2 Algorithms Used

#### 4.1.2.1 Roots of a real coefficient algebraic equation

##### 4.1.2.1.1 When the degree $n = 2$

The roots are obtained as follows by using the formula for the roots of a quadratic equation.
For the quadratic equation:

$$x^2 + a_1 x + a_0 = 0$$

if we let:

$$\begin{aligned} r &= -\frac{a_1}{2} \\ D &= r^2 - a_0 \end{aligned}$$

then the roots are obtained as shown below for the following three cases.

(1) $|D| \le \varepsilon$
The roots are:

$$x = r, r$$

(2) If $D < 0$
The roots are:

$$x = (r, \pm\sqrt{-D})$$

(3) If $D > 0$
The roots are:

$$x = \alpha, \frac{a_0}{\alpha}$$

where,

$$\alpha = \begin{cases} r + \sqrt{D} & (r \ge 0) \\ r - \sqrt{D} & (r < 0) \end{cases}$$

##### 4.1.2.1.2 When the degree $n = 3$

The roots are obtained as follows by using the Cardano method.
For the cubic equation:

$$x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

consider the following cases.

(1) If $|a_0| \le \varepsilon \quad (x(x^2 + a_2 x + a_1) = 0)$
By using the method described in (a) to solve the quadratic equation:

$$x^2 + a_2 x + a_1 = 0$$

to obtain the roots $\alpha$ and $\beta$, the roots of the cubic equation are:

$$x = 0, \alpha, \beta$$

(2) If $|a_2| \le \varepsilon \quad (x^3 + a_1 x + a_0 = 0)$
Consider the following two cases.

(a) $|a_1| \le \varepsilon \quad (x^3 + a_0 = 0)$
The roots are:

$$x = r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2}\right)$$

where,

$$r = \begin{cases} -\sqrt[3]{a_0} & (a_0 \ge 0) \\ \sqrt[3]{-a_0} & (a_0 < 0) \end{cases}$$

(b) Otherwise
Assume $b_1$ and $b_0$ are as follows:

$$\begin{aligned} b_1 &= \frac{a_1}{3} \\ b_0 &= -\frac{a_0}{2} \end{aligned}$$

and use the method shown in subsection iii. below to solve the equation:

$$x^3 + 3b_1 x - 2b_0 = 0$$

(3) Otherwise
The cubic equation:

$$x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

can be transformed by using the variable transformation:

$$\begin{aligned} x &= y - YMX \\ YMX &= \frac{a_2}{3} \end{aligned}$$

to:

$$y^3 + 3b_1 y - 2b_0 = 0$$

where, $b_1$ and $b_0$ are given by:

$$\begin{aligned} b_1 &= \frac{3a_1 - a_2^2}{9} \\ b_0 &= \frac{(9a_1 - 2a_2^2)a_2 - 27a_0}{54} \end{aligned}$$

Consider the following three cases.

(a) If $|b_0| \le \varepsilon \quad (y(y^2 + 3b_1) = 0)$

i. If $b_1 < 0$
$y$ and $x$ are as follows:

$$\begin{aligned} y &= 0, \pm\sqrt{-3b_1} \\ x &= -YMX, \pm\sqrt{-3b_1} - YMX \end{aligned}$$

Cancellation of significant digits that may occur in the calculation is prevented by using relationships between roots and coefficients.

ii. If $b_1 \geq 0$

$y$ and $x$ are as follows:

$$\begin{aligned} y &= 0, (0, \pm\sqrt{-3b_1}) \\ x &= -YMX, (-YMX, \pm\sqrt{-3b_1}) \end{aligned}$$

(b) If $|b_1| \leq \varepsilon \quad (y^3 - 2b_0 = 0)$

$y$ and $x$ are as follows:

$$\begin{aligned} y &= r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2}\right) \\ x &= r - YMX, \left(-\frac{r}{2} - YMX, \pm\frac{\sqrt{3}r}{2}\right) \end{aligned}$$

where,

$$r = \begin{cases} \sqrt[3]{2b_0} & (b_0 \geq 0) \\ -\sqrt[3]{-2b_0} & (b_0 < 0) \end{cases}$$

Cancellation of significant digits that may occur in the calculation is prevented by using relationships between roots and coefficients.

(c) Otherwise $\quad (y^3 + 3b_1 y - 2b_0 = 0)$

If we let:

$$y = s + t$$

we obtain the following equation:

$$s^3 + t^3 - 2b_0 + 3(st + b_1)(s + t) = 0$$

If we determine $s$ and $t$ so that the following relationships hold:

$$\begin{cases} st &= -b_1 \\ s^3 + t^3 &= 2b_0 \end{cases}$$

then we can use this to obtain $y$. $s^3$ and $t^3$ are the two roots of:

$$z^2 - 2b_0 z - b_1^3 = 0$$

If we let:

$$\begin{aligned} D &= b_0^2 + b_1^3 \quad = \quad D_s^2 D_d \\ D_s &= \begin{cases} b_0 & (|b_0| \geq |b_1|) \\ b_1 & (|b_0| < |b_1|) \end{cases} \\ D_d &= \begin{cases} 1 + b_1(\frac{b_1}{b_0})^2 & (|b_0| \geq |b_1|) \\ (\frac{b_0}{b_1})^2 + b_1 & (|b_0| < |b_1|) \end{cases} \end{aligned}$$

we can obtain $y$ and $x$ by considering the following three cases.

i. If $|D_d| \leq \varepsilon^2 \quad ((z - b_0)^2 = 0)$

$s$ and $t$ are as follows:

$$\begin{aligned} s^3 &= t^3 = b_0 \\ s, t &= r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2}\right) \end{aligned}$$

where,

$$r = \begin{cases} \sqrt[3]{b_0} & (b_0 \geq 0) \\ -\sqrt[3]{-b_0} & (b_0 < 0) \end{cases}$$

Since $s$ and $t$ satisfy $st = -b_1$, $y$ and $x$ are as follows:

$$y = 2r, -r, -r$$

$$x = 2r - YMX, -r - YMX, -r - YMX$$

Cancellation of significant digits that may occur in the calculation is prevented by using relationships between roots and coefficients.

ii. If $D < 0$

$s^3$ and $t^3$ are as follows:

$$s^3 = (b_0, \sqrt{-D})$$

$$t^3 = (b_0, -\sqrt{-D})$$

Now, from the following:

$$|s^3|^2 = |t^3|^2 = -b_1^3$$

$s$ and $t$ are as follows:

$$s = \frac{r}{2}e^{\sqrt{-1}\frac{\theta}{3}}, \frac{r}{2}e^{\sqrt{-1}(\pi+\frac{\theta-\pi}{3})}, \frac{r}{2}e^{\sqrt{-1}(\frac{\theta+\pi}{3}-\pi)}$$

$$t = \frac{r}{2}e^{-\sqrt{-1}\frac{\theta}{3}}, \frac{r}{2}e^{-\sqrt{-1}(\pi+\frac{\theta-\pi}{3})}, \frac{r}{2}e^{-\sqrt{-1}(\frac{\theta+\pi}{3}-\pi)}$$

Therefore, $y$ and $x$ are as follows:

$$y = r\cos\frac{\theta}{3}, -r\cos\frac{\pi-\theta}{3}, -r\cos\frac{\pi+\theta}{3}$$

$$x = r\cos\frac{\theta}{3} - YMX, -r\cos\frac{\pi-\theta}{3} - YMX, -r\cos\frac{\pi+\theta}{3} - YMX$$

where,

$$r = \begin{cases} -2\sqrt{b_1} & (b_1 \geq 0) \\ 2\sqrt{-b_1} & (b_1 < 0) \end{cases}$$

$$\theta = \begin{cases} \tan^{-1} D_\theta & (b_0 \geq 0) \\ \pi - \tan^{-1} D_\theta & (b_0 < 0) \end{cases}$$

$$D_\theta = \frac{|D|}{|b_0|} = \begin{cases} \sqrt{|D_d|} & (|b_0| \geq |b_1|) \\ \dfrac{|D_s|\sqrt{|D_d|}}{|b_0|} & (|b_0| < |b_1|) \end{cases}$$

Cancellation of significant digits that may occur in the calculation is prevented by using relationships between roots and coefficients.

iii. If $D > 0$

$s^3$ and $t^3$ are as follows:

$$s^3 = \alpha$$

$$t^3 = \beta$$

$$\alpha = \begin{cases} b_0 + \sqrt{D} = |D_s|r & (b_0 \geq 0) \\ b_0 - \sqrt{D} = -|D_s|r & (b_0 < 0) \end{cases}$$

$$\beta = -\frac{b_1^3}{\alpha}$$

where,

$$r = \frac{|b_0| + \sqrt{D}}{|D_s|} = \begin{cases} 1 + \sqrt{|D_d|} & (|b_0| \geq |b_1|) \\ \dfrac{|b_0|}{|D_s|} + \sqrt{|D_d|} & (|b_0| < |b_1|) \end{cases}$$

Therefore, $s$ and $t$ are as follows:

$$
\begin{aligned}
s &= \alpha', \left(-\frac{\alpha'}{2}, \pm\frac{\sqrt{3}\alpha'}{2}\right) \\
t &= \beta', \left(-\frac{\beta'}{2}, \pm\frac{\sqrt{3}\beta'}{2}\right)
\end{aligned}
$$

where,

$$
\begin{aligned}
\alpha' &= \begin{cases} \sqrt[3]{|\alpha|} & (b_0 \geq 0) \\ -\sqrt[3]{|\alpha|} & (b_0 < 0) \end{cases} \\
\beta' &= \begin{cases} \sqrt[3]{|\beta|} & (b_0 b_1 < 0) \\ -\sqrt[3]{|\beta|} & (b_0 b_1 \geq 0) \end{cases}
\end{aligned}
$$

Since $s$ and $t$ satisfy $st = -b_1$, $y$ and $x$ are as follows:

$$
\begin{aligned}
y &= u, \left(-\frac{u}{2}, \pm\frac{\sqrt{3}v}{2}\right) \\
x &= u - YMX, \left(-\frac{u}{2} - YMX, \pm\frac{\sqrt{3}v}{2}\right)
\end{aligned}
$$

where,

$$u = \alpha' + \beta', v = \alpha' - \beta'$$

Cancellation of significant digits that may occur in the calculation is prevented by using relationships between roots and coefficients.

#### 4.1.2.1.3 When the degree $n = 4$

The roots are obtained as follows by using the Ferrari method.
For the quartic (fourth degree) equation:

$$x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

consider the following cases.

(1) If $|a_0| \leq \varepsilon \quad (x(x^3 + a_3x^2 + a_2x + a_1) = 0)$
By using the method described in (b) to solve the cubic equation:

$$x^3 + a_3x^2 + a_2x + a_1 = 0$$

to obtain the roots $\alpha, \beta$ and $\gamma$, the roots of the quartic equation are:

$$x = 0, \alpha, \beta, \gamma$$

(2) If $|a_3| \leq \varepsilon \quad (x^4 + a_2x^2 + a_1x + a_0 = 0)$
Consider the following two cases.

(a) If $|a_1| \leq \varepsilon \quad (x^4 + a_2x^2 + a_0 = 0)$
If we let:

$$
\begin{aligned}
r &= -\frac{a_2}{2} \\
D &= r^2 - a_0
\end{aligned}
$$

we can obtain $x$ by considering the following three cases.

i. If $|D| \le \varepsilon \quad ((x^2 - r)^2 = 0)$
$x$ is as follows:

$$x = \begin{cases} \sqrt{r}, \sqrt{r}, -\sqrt{r}, -\sqrt{r} & (r \ge 0) \\ (0, \sqrt{-r}), (0, \sqrt{-r}), (0, -\sqrt{-r}), (0, -\sqrt{-r}) & (r < 0) \end{cases}$$

ii. If $D < 0$
$x^2$ is as follows:

$$x^2 = (r, \pm\sqrt{-D})$$

Therefore, $x$ is as follows:

$$x = (\alpha, \pm\beta), (-\alpha, \pm\beta)$$

where,

$$\begin{cases} \alpha = \sqrt{\dfrac{r + NRD}{2}}, & \beta = \dfrac{\sqrt{-D}}{2\alpha} \quad (r > 0) \\ \beta = \sqrt{\dfrac{-r + NRD}{2}}, & \alpha = \dfrac{\sqrt{-D}}{2\beta} \quad (r \le 0) \end{cases}$$

$$NRD = \sqrt{r^2 - D} = \begin{cases} |r|\sqrt{1 + \dfrac{-D}{r^2}} & (|r| \ge \sqrt{-D}) \\ \sqrt{-D}\sqrt{\dfrac{r^2}{-D} + 1} & (|r| < \sqrt{-D}) \end{cases}$$

iii. If $D > 0$
$x^2$ is as follows:

$$x^2 = \alpha, \beta$$

where,

$$\begin{aligned} \alpha &= \begin{cases} r + \sqrt{D} & (r \ge 0) \\ r - \sqrt{D} & (r < 0) \end{cases} \\ \beta &= \frac{a_0}{\alpha} \end{aligned}$$

Therefore, $x$ is as follows:

$$x = \begin{cases} \pm\sqrt{\alpha}, \pm\sqrt{\beta} & (r, \beta \ge 0) \\ \pm\sqrt{\alpha}, (0, \pm\sqrt{-\beta}) & (r \ge 0, \beta \le 0) \\ (0 \pm \sqrt{-\alpha}), \pm\sqrt{\beta} & (r < 0, \beta \ge 0) \\ (0 \pm \sqrt{-\alpha}), (0 \pm \sqrt{-\beta}) & (r, \beta < 0) \end{cases}$$

(b) Otherwise
Use the method shown in subsection iii. below to solve the equation:

$$x^4 + a_2x^2 + a_1x + a_0 = 0$$

(3) Otherwise
The quartic equation:

$$x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

can be transformed by using the variable transformation:

$$\begin{aligned} x &= y - YMX \\ YMX &= \frac{a_3}{4} \end{aligned}$$

to:

$$y^4 + b_2y^2 + b_1y + b_0 = 0$$

where, $b_2, b_1$ and $b_0$ are given by:

$$\begin{aligned}
b_2 &= \frac{8a_2 - 3a_3^2}{8} \\
b_1 &= \frac{8a_1 - a_3(4a_2 - a_3^2)}{8} \\
b_0 &= \frac{256a_0 - a_3(64a_1 - a_3(16a_2 - 3a_3^2))}{256}
\end{aligned}$$

Consider the following four cases.

(a) $b_1^2 \leq \varepsilon^2 \max(|4b_0b_2|, |b_2^3|) \qquad (y^4 + b_2y_2 + b_0 = 0)$
If we let:

$$\begin{aligned}
r &= -\frac{b_2}{2} \\
D &= r^2 - b_0
\end{aligned}$$

we can obtain $y$ and $x$ by considering the following three cases.

i. If $|D| \leq \varepsilon$
$y$ and $x$ are as follows:

$$\begin{aligned}
(y^2 - r)^2 &= 0 \\
y &= \begin{cases} \pm\sqrt{r}, \pm\sqrt{r} & (r \geq 0) \\ (0, \pm\sqrt{-r}), (0, \pm\sqrt{-r}) & (r < 0) \end{cases} \\
x &= \begin{cases} \pm\sqrt{r} - YMX, \pm\sqrt{r} - YMX & (r \geq 0) \\ (-YMX, \pm\sqrt{-r}), (-YMX, \pm\sqrt{-r}) & (r < 0) \end{cases}
\end{aligned}$$

ii. If $D < 0$
$y$ and $x$ are as follows:

$$\begin{aligned}
y^2 &= (r, \pm\sqrt{-D}) \\
y &= (\alpha, \pm\beta), (-\alpha, \pm\beta) \\
x &= (\alpha - YMX, \pm\beta), (-\alpha - YMX, \pm\beta)
\end{aligned}$$

where,

$$\begin{cases} \alpha = \sqrt{\dfrac{r + NRD}{2}}, \quad \beta = \dfrac{\sqrt{-D}}{2\alpha} & (r > 0) \\ \beta = \sqrt{\dfrac{-r + NRD}{2}}, \quad \alpha = \dfrac{\sqrt{-D}}{2\beta} & (r \leq 0) \end{cases}$$

$$NRD = \sqrt{r^2 - D} = \begin{cases} |\, r \,| \sqrt{1 + \dfrac{-D}{r^2}} & (|r| > \sqrt{-D}) \\ \sqrt{-D}\sqrt{\dfrac{r^2}{-D} + 1} & (|r| < \sqrt{-D}) \end{cases}$$

iii. If $D > 0$
$y$ and $x$ are as follows:

$$\begin{aligned}
y^2 &= \alpha, \beta \\
\alpha &= \begin{cases} r + \sqrt{D} & (r \geq 0) \\ r - \sqrt{D} & (r < 0) \end{cases} \\
\beta &= \frac{b_0}{\alpha}
\end{aligned}$$

$$y = \begin{cases} \pm\sqrt{\alpha}, \pm\sqrt{\beta} & (r, \beta \geq 0) \\ (0, \pm\sqrt{-\alpha}), \pm\sqrt{\beta} & (r < 0, \beta \geq 0) \\ \pm\sqrt{\alpha}, (0, \pm\sqrt{-\beta}) & (r \geq 0, \beta < 0) \\ (0, \pm\sqrt{-\alpha}), (0, \pm\sqrt{\beta}) & (r, \beta < 0) \end{cases}$$

$$x = \begin{cases} \pm\sqrt{\alpha} - YMX, \pm\sqrt{\beta} - YMX & (r, \beta \geq 0) \\ (-YMX, \pm\sqrt{-\alpha}), \pm\sqrt{\beta} - YMX & (r < 0, \beta \geq 0) \\ \pm\sqrt{\alpha} - YMX, (-YMX, \pm\sqrt{-\beta}) & (r \geq 0, \beta < 0) \\ (-YMX, \pm\sqrt{-\alpha}), (-YMX, \pm\sqrt{-\beta}) & (r, \beta < 0) \end{cases}$$

(b) If $|b_0| \leq \varepsilon \quad (y(y^3 + b_2 y + b_1) = 0)$

By using the method described in (b) to solve the cubic equation:

$$y^3 + b_2 y + b_1 = 0$$

to obtain the roots $\alpha, \beta$ and $\gamma$, the roots of the quartic equation are:

$$x = -YMX, \alpha - YMX, \beta - YMX, \gamma - YMX$$

(c) If $b_1^2 > 10^{-4}|b_2(b_2^2 - 4b_0)|$

If the quartic equation:

$$y^4 + b_2 y^2 + b_1 y + b_0 = 0$$

is transformed by adding $py^2 + \dfrac{p^2}{4}$ to both sides, it appears as follows:

$$\left(y^2 + \frac{p}{2}\right)^2 = (p - b_2)y^2 - b_1 y + \frac{p^2}{4} - b_0$$

To express the right-hand side in the form of the square of a linear expression, the following equation should be satisfied:

$$b_1^2 - (p - b_2)(p^2 - 4b_0) = 0$$

That is,

$$p^3 - b_2 p^2 - 4b_0 p + (4b_2 b_0 - b_1^2) = 0$$

If we solve this equation by the method described in (b) and take the real root obtained for this equation for $p$, then since the following relationship holds:

$$\left(y^2 + \frac{p}{2}\right)^2 = (p - b_2)\left\{y - \frac{b_1}{2(p - b_2)}\right\}^2$$

by solving the quadratic equation:

$$y^2 \pm \sqrt{p - b_2}\, y \mp \frac{b_1}{2\sqrt{p - b_2}} + \frac{p}{2} = 0 \quad \text{(Compound same order)}$$

the solutions of the quartic equation are obtained as follows:

$$\begin{aligned} y &= \alpha \pm \sqrt{\beta - \gamma}, -\alpha \pm \sqrt{\beta + \gamma} \\ \alpha &= \frac{\sqrt{p - b_2}}{2} \\ \beta &= -\frac{p + b_2}{4} \\ \gamma &= \frac{4b_1}{\alpha} \end{aligned}$$

However, if $p \simeq b_2$, the solution obtained by using this method is not very precise. The condition $p \simeq b_2$ occurs when:

$$|p - b_2| = \left| \frac{b_1^2}{p^2 - 4b_0} \right| \simeq \left| \frac{b_1^2}{b_2^2 - 4b_0} \right| < \delta |b_2|$$

that is, when:

$$b_1^2 < \delta |b_2(b_2^2 - 4b_0)|$$

where, $\delta$ is a sufficiently small positive number.
If the following condition holds:

$$b_1^2 \leq \varepsilon^2 \max(|b_2^3|, |4b_0 b_2|)$$

the solution of the quartic equation is obtained by ignoring $b_1$ as described earlier and solving the following equation:

$$y^4 + b_2 y^2 + b_0 = 0$$

On the other hand, if the following condition holds:

$$10^{-4} |b_2(b_2^2 - 4b_0)| \geq b_1^2 > \varepsilon^2 \max(|b_2^3|, |4b_0 b_2|)$$

the solution of the quartic equation is obtained by using the method described in subsection D. below.

(d) Otherwise
It is known that the roots of the quartic equation:

$$y^4 + b_2 y^2 + b_1 y + b_0 = 0$$

can be expressed by using the square roots of the three roots $z_1, z_2$ and $z_3$ of the cubic equation:

$$z^3 + \frac{b_2}{2} z^2 + \frac{b_2^2 - 4b_0}{16} z - \frac{b_1^2}{64} = 0$$

(where, $z_1, z_2$ and $z_3$ are generally complex numbers) as follows:

$$\begin{aligned} y &= \sqrt{z_1} + \sqrt{z_2} + \sqrt{z_3}, \\ &\quad \sqrt{z_1} - (\sqrt{z_2} + \sqrt{z_3}), \\ &\quad -\sqrt{z_1} + \sqrt{z_2} - \sqrt{z_3}, \\ &\quad -\sqrt{z_1} - (\sqrt{z_2} - \sqrt{z_3}) \end{aligned}$$

However, $\sqrt{z}$ represents one of the two square roots ($\pm\sqrt{z}$) that $z$ has. Although it is not known in advance which of the square roots should be selected, since the four root combinations are either $y$, which was shown above or $-y$, which is obtained by reversing all of the signs of the root, the root can be determined from the relationship between the roots and coefficients according to the sign of $b_1$. The cubic equation is solved by using the method described in (b).

#### 4.1.2.1.4 When the degree $n > 4$

This problem is solved by using the Hirano method, which is described below.
Assume that the given equation is as follows:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

The Hirano method obtains all roots of the equation by alternately searching for a root according to iterative improvement of an approximate root and then reducing the equation. The origin is assumed as the starting value for the approximate root, and the expansion coefficients when the polynomial $P_n(x)$ is expanded around

this approximate root are used to sequentially approach the true root. The root $x = (x_R, x_I)$ obtained in this manner is used to reduce the equation, and all roots of the equation are obtained by sequentially repeating similar operations for the reduced equation.
The root $x = (x_R, x_I)$ is determined as follows.
The convergence decision value $\zeta$ is determined as follows.
Let the calculation of the coefficients $c_k$ for $(k = n, n-1, \cdots, 0)$ have an error of $\varepsilon \mid c_k \mid$ and let the calculation of $\zeta$ have an error of up to $\varepsilon \mid x \mid$ (where, $x = z + \zeta$ is the value of the true root). Since $P_n(z+\zeta)$ can be calculated from:

(1) Let the initial value of the approximate value $z = (z_R, z_I)$ of root $x$ be $z = 0$.

(2) Until the relationship $\mid P_n(z + \zeta_m) \mid \leq \delta$ holds, replace $z$ by $z + \zeta_m$ and repeat the following calculations.

   (a) Expand the polynomial $P_n(x)$ centered on $z$ as follows:

$$P_n(\zeta + z) = c_n\zeta^n + c_{n-1}\zeta^{n-1} + \cdots + c_1\zeta + c_0 \qquad (x = \zeta + z)$$

   using synthetic division to calculate the coefficients $c_k$ for $(k = n, n-1, \cdots, 0)$.

   (b) Let $\mu = 1$.

   (c) Calculate $\zeta_k(\mu)$ for $(k = 1, \cdots, n)$ as follows:

$$\zeta_k(\mu) = \left(-\mu\frac{c_0}{c_k}\right)^{1/k} \quad \text{for} \quad (k = 1, \cdots, n)$$

   and let $\zeta_m$ be the one that has the smallest absolute value. Check whether the relationship $\mid P_n(z + \zeta_m) \mid \leq (1 - \frac{\mu}{4}) \mid P_n(z) \mid$ holds. If not, replace $\mu$ by $\frac{\mu}{2}$ and repeat this calculation sequentially until the relationship holds.

(3) Assume $z + \zeta_m$ is the root of the equation $P_n(x) = 0$.

$$\begin{aligned} b_n &= c_n \\ b_k &= b_{k+1}\zeta + c_k \quad \text{for} \quad (k = n-1, n-2, \cdots, 0) \\ P_n(z + \zeta) &= b_0 \end{aligned}$$

the error $\delta$ for the calculation of $P_n(z + \zeta)$ can be calculated from:

$$\begin{aligned} d_n &= \varepsilon \mid c_n \mid \\ d_k &= d_{k+1}(\mid \zeta \mid + \varepsilon \mid x \mid) + \varepsilon(\mid x \mid\mid b_{k+1} \mid + \mid c_k \mid) \\ &\quad \text{for} \quad (k = n-1, n-2, \cdots, 0) \\ \delta &= d_0 \end{aligned}$$

Now, in the calculation, let $\zeta = \zeta_m$, and $x$ is approximated by $z + \zeta_m$.
Although the Hirano method proceeds by sequentially obtaining roots while reducing the equation, an error analysis is performed as follows during each reduction process to determine whether reduction succeeds.

   (a) Reduction according to a single real root
   If the equation:

$$P_n(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 = 0$$

   is reduced as follows using the real root $z$ :

$$P_n(x) = (x - z)(b_nx^{n-1} + b_{n-1}x^{n-2} + \cdots + b_1) + b_0 = 0$$

the coefficients are calculated as follows:

$$\begin{aligned} b_n &= a_n \\ b_k &= a_k + b_{k+1}z \quad \text{for} \quad (k = n-1, n-2, \cdots, 0) \end{aligned}$$

Although $b_0$ originally is 0.0, if an error of up to $\sqrt{\varepsilon} \mid z \mid$ is permitted for $z$ and an error of up to $\varepsilon \mid a_k \mid$ is permitted for $a_k$, the maximum error $\delta_k$ permitted for $b_k$ is expressed as follows:

$$\begin{aligned} \delta_n &= \varepsilon \mid a_n \mid \\ \delta_k &= \varepsilon \mid a_k \mid + (\delta_{k+1} + \sqrt{\varepsilon} \mid b_{k+1} \mid) \mid z \mid \quad \text{for} \quad (k = n-1, n-2, \cdots, 0) \end{aligned}$$

Therefore, when $\mid b_0 \mid < \delta_0$, reduction is assumed to have succeeded.

(b) Reduction according to complex conjugate roots
If the equation:

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$$

is reduced as follows using the complex conjugate roots $z = (z_R, z_I)$ and $\bar{z} = (z_R, -z_I)$ :

$$P_n(x) = (x-z)(x-\bar{z})(b_n x^{n-2} + b_{n-1}x^{n-3} + \cdots + b_2) + b_1 x + b_0 = 0$$

the coefficients are calculated as follows:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + 2z_R b_n \\ b_k &= a_k + 2z_R b_{k+1} - ({z_R}^2 + {z_I}^2) b_{k+2} \quad \text{for} \quad (k = n-1, n-3, \cdots, 0) \end{aligned}$$

Although $b_0$ and $b_1$ originally are 0.0, if an error of up to $\sqrt{\varepsilon} \mid z_R \mid$ is permitted for $z_R$, an error of up to $\sqrt{\varepsilon} \mid z_I \mid$ is permitted for $z_I$, and an error of up to $\varepsilon \mid a_k \mid$ is permitted for $a_k$, the maximum error $\delta_k$ permitted for $b_k$ is expressed as follows:

$$\begin{aligned} \delta_n &= \varepsilon \mid a_n \mid \\ \delta_{n-1} &= \varepsilon \mid a_{n-1} \mid + \delta_n \mid 2z_R \mid + \sqrt{\varepsilon} \mid 2z_R b_n \mid \\ \delta_k &= \varepsilon \mid a_k \mid + \delta_{k+1} \mid 2z_R \mid + \sqrt{\varepsilon} \mid 2z_R b_{k+1} \mid + \delta_{k+2}({z_R}^2 + {z_I}^2) \\ & \quad +2\sqrt{\varepsilon} \end{aligned}$$

$$(\mid z_R \mid + \mid z_I \mid) \mid b_{k+2} \mid \quad \text{for} \quad (k = n-2, n-3, \cdots, 0)$$

Therefore, when $\mid b_0 \mid < \delta_0$ and $\mid b_1 \mid < \delta_1$, reduction is assumed to have succeeded.

#### 4.1.2.2 The roots of complex coefficient algebraic equations

This problem is solved by using the Durand-Kerner cubic method.
Assume that the given equation is as follows:

$$P_n(Z) = a_0 Z^n + a_1 Z^{n-1} + \cdots + a_{n-1}Z + a_n = 0 \quad \text{for} \quad (a_0 \neq 0)$$

The iteration formula for obtaining all roots simultaneously is as follows:

$$\begin{cases} {z_i}^{(\nu+1)} = {z_i}^{(\nu)} \Phi_i \left( {z_1}^{(\nu)}, \cdots, {z_n}^{(\nu)} \right) \\ \Phi_i(z_1^{(\nu)}, \cdots, z_n^{(\nu)}) = \dfrac{P_n({z_i}^{(\nu)}) / P'_n({z_i}^{(\nu)})}{1 - \dfrac{P_n({z_i}^{(\nu)})}{P'_n({z_i}^{(\nu)})} \displaystyle\sum_{\substack{j=1 \\ j \neq i}}^{n} \dfrac{1}{{z_i}^{(\nu)} - {z_j}^{(\nu)}}} \quad \text{for} \quad (i = 1, \cdots, n) \end{cases}$$

where $P_n'(z_i{}^{(\nu)})$ is obtained according to the following formula:

$$\left\{\begin{array}{lcl} b_0{}^{(1)} & = & a_0 \\ b_0{}^{(2)} & = & a_0 \\ b_k{}^{(1)} & = & z_i{}^{(\nu)} b_{k-1}^{(1)} + a_k \\ b_k{}^{(2)} & = & z_i^{(\nu)} b_{k-1}^{(2)} + b_k^{(1)} \quad \text{for} \quad (k = 1, \cdots, n-1) \end{array}\right.$$

$$P_n'(z_i{}^{(\nu)}) = b_{n-1}{}^{(2)}$$

Convergence is assumed to have occurred if the value of $P_n(z)$ is within the error included when it is calculated by Horner's method. That is, if $\varepsilon$ is the unit for determining error, then for:

$$\left\{\begin{array}{lcll} b_0 & = & a_0 & : \text{Where the } b_0 \text{ error is} \quad \delta_0 = 0.0 \\ b_k' & = & z_i{}^{(\nu)} b_{k-1} & : \text{Where the } b_k' \text{ error is} \quad \delta_k{}^* = \delta_{k-1} \mid z_i{}^{(\nu)} \mid + \varepsilon \mid b_k' \mid \\ b_k & = & b_k' + a_k & : \text{Where the } b_k \text{ error is} \quad \delta_k = \delta_k{}^* + \varepsilon \max(\mid a_k \mid, \mid b_k' \mid, \mid b_k \mid) \\ P_n(z) & = & b_n & \end{array}\right.$$

therefore the specified value $\delta_n$ of the maximum error of $P_n(z_i{}^{(\nu)})$ is given by:

$$\left\{\begin{array}{lcl} \delta_0 & = & 0.0 \\ \delta_k & = & \delta_{k-1} \mid z_i{}^{(\nu)} \mid + \varepsilon\{\mid b_k' \mid + \max(\mid a_k \mid, \mid b_k' \mid, \mid b_k \mid)\} \qquad \text{for} \quad (k = 1, \cdots, n) \end{array}\right.$$

and if $\mid P_n(z_i{}^{(\nu)}) \mid \leq \delta_n$, then convergence is assumed to have occurred.
Iterations are continued only for unconverged $z_i(\nu)$, and the calculation ends when all values are considered to have converged.
The iteration initial values are determined by the following three steps method.

(1) Determine the Aberth initial value.
For a center at:

$$\beta = \frac{\alpha_1 + \alpha_2 + \cdots + \alpha_n}{n} = -\frac{a_1}{na_0}$$

obtain the radius $r$ of a circle that includes all roots. Using derived division, obtain the coefficients $c_0, \cdots, c_n$ such that:

$$P_n(Z) = P_n(\beta + \zeta) = c_0\zeta^n + c_1\zeta^{n-1} + \cdots + c_{n-1}\zeta + c_n$$

Let $m$ be the number of nonzero coefficients among $c_1, \cdots, c_n$ and obtain $r^+$ as follows:

$$r^+ = \max_{k=1,\cdots,n} (m\frac{\mid c_k \mid}{\mid c_0 \mid})^{1/k}$$

Let $r^+$ be the initial value of $r$, and obtain $x$ such that:

$$Q_n(x) = \mid c_0 \mid x^n - \mid c_1 \mid x^{n-1} - \cdots - \mid c_{n-1} \mid x - \mid c_n \mid = 0$$

Let this value of $x$ be $r$.

(2) Decrease the radius of the circle that includes all roots.
Let $z$ be represented by:

$$z = \beta + rw$$

Then $\varphi_\ell(w)$ and $\varphi_{\ell+1}(w)$ can be defined as follows:

$$\begin{aligned} P_n(\beta + rw) &= (c_0 r^n)w^n + (c_1 r^{n-1})w^{n-1} + \cdots + (c_{n-1}r)w + c_n \\ &= d_0{}^{(0)}w^n + d_1{}^{(0)}w^{n-1} + \cdots + d_{n-1}{}^{(0)}w + d_n{}^{(0)} \\ &= \varphi_0(w) \\ \varphi_\ell(w) &= d_0{}^{(\ell)}w^{n\ell} + d_1{}^{(\ell)}w^{n\ell-1} + \cdots + d_{n\ell-1}{}^{(\ell)}w + d_{n\ell}{}^{(\ell)} \qquad (d_0^{(\ell)} \neq 0) \end{aligned}$$

(a) When $\mid a_0{}^{(\ell)} \mid < \mid a_{n\ell}{}^{(\ell)} \mid$

$$\varphi_{\ell+1}(w) = \varphi_\ell(w) - \frac{d_0{}^{(\ell)}}{d_{n\ell}{}^{(\ell)}}\tilde{\varphi}_\ell(w)$$

(b) When $\mid a_0{}^{(\ell)} \mid \geq \mid a_{n\ell}{}^{(\ell)} \mid$

$$\varphi_{\ell+1}(w) = \left\{\varphi_\ell(w) - \frac{d_{n\ell}{}^{(\ell)}}{d_0{}^{(\ell)}}\tilde{\varphi}_\ell(w)\right\} / w$$

where:

$$\tilde{\varphi}_\ell(w) = \overline{d_{n\ell}{}^{(\ell)}}w^{n\ell} + \overline{d_{n\ell-1}{}^{(\ell)}}w^{n\ell-1} + \cdots + \overline{d_1{}^{(\ell)}}w + \overline{d_0{}^{(\ell)}}$$

until $\varphi_n$ = constant.
The number of roots outside of the circle having radius $r$ is equal to the number of times $\mid a_0{}^{(\ell)} \mid < \mid a_{n\ell}{}^{(\ell)} \mid$ had occurred when $\varphi_n$ was obtained. The roots inside this circle are the remaining roots.
The radius of the minimum circle that includes all roots is obtained by a bisection method beginning with the Aberth initial value $r$ according to the condition that $\mid a_0{}^{(\ell)} \mid > \mid a_{n\ell}{}^{(\ell)} \mid$ occurs for all roots.

(3) Find the radius that minimizes the sum of the squares of the distances from the various roots.
Use the method described in (b) to obtain the numbers of roots inside and outside the circle of each radius that is obtained by having the radius $r$ obtained in (b). In $j$ iterations, the ring $D_j$ of width $r2^{-j}$ is obtained. If we let the average of the interior radius and exterior radius of ring $D_j$ be $r_j{}^*$ and the number of roots contained in $D_j$ be $N_j$, then the desired radius $r$ is obtained by:

$$r = \frac{\displaystyle\sum_{j=1}^{m} r_j{}^* N_j}{n}$$

The initial value for the iteration is obtained from this $r$ by using the following equation:

$$z_j{}^{(0)} = \beta + r\exp\left[i(\frac{2\pi(j-1)}{n} + \frac{3}{2n})\right] \quad \text{for} \quad (j = 1, \cdots, n; i = \sqrt{-1})$$

#### 4.1.2.3 The roots of real functions (initial value specified; derivative definition required)

Assume that the given nonlinear equation is $f(x) = 0$ and that the derivative of $f(x)$ is $f'(x)$. This algorithm is based on Newton's method, which is given by:

$$x^{(\nu+1)} = x^{(\nu)} - \frac{f(x^{(\nu)})}{f'(x^{(\nu)})}$$

However, since a root is not obtained by this method if the function oscillates or if the root update amount is too large, the algorithm has been improved as follows.

(1) Newton's iteration is performed as follows:

$$x^{(\nu+1)} = x^{(\nu)} - \frac{f(x^{(\nu)})}{f'(x^{(\nu)})}$$

(2) If $f'(x^{(\nu)}) = 0$ or $\mid f(x^{(\nu+1)}) \mid \geq \mid f(x^{(\nu)}) \mid$, then skip to (c); otherwise, let $x^{(\nu)} = x^{(\nu+1)}$ and return to (a).

(3) If $\nu = 1$ or $f'(x^{(\nu-1)}) \geq 0$, let $IS = -1$; otherwise let $IS = 1$.
Let $R = 1$, $P = 0$, $IOLD = \text{sign}\left\{1, IS \times f(x^{(\nu)})\right\}$, and $I1 = -IOLD$.

(4) Let $I2 = \text{sign}\left\{1, IS \times f(x^{(\nu)})\right\}$.
If $I2 \times I1 > 0$, then change is accelerated by letting $P = P + 1$; otherwise, it is decelerated by letting $R = R+1$. (For $I1$: Old update direction and $I2$: New update direction, + indicates update in the positive direction and − indicates update in the negative direction.)
$x^{(\nu)}$ is updated by the following iteration.
Let $\varepsilon$ be the unit for determining error, and let $x^{(\nu+1)}$ be as follows:

$$x^{(\nu+1)} = x^{(\nu)} + IS \times \sinh^{-1}(f(x^{(\nu)})) \times 2^{\{(p-3)/3-R\}} + \mid x^{(\nu)} + 1 \mid \times I2 \times \varepsilon$$

(For $IS$: Global function slope estimate, + indicates decreasing to the right and − indicates increasing to the right.)
This update is repeated at least three times.

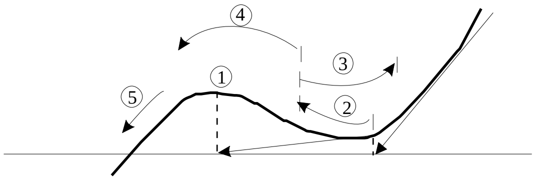
(5) For $f(x^{(\nu-1)}) \gg f(x^{(\nu)}) \gg f(x^{(\nu+1)})$, if $\mid f(x^{(\nu+1)}) \mid$ becomes sufficiently small, return to Newton's method (a).

(6) For $\mid f(x^{(\nu+1)}) \mid > \mid f(x^{(\nu)}) \mid$, when $f(x^{(\nu+1)})$ and $f(x^{(\nu)})$ have the same sign and $I2 = IOLD$, the function slope and search direction are changed by letting $IS = -IS, IOLD = IOLD$ and $I1 = IOLD$.
When this condition occurs for the first time,
Let the left end of the searched interval $XP = x^{(\nu+1)}$ and $FP = f(x^{(\nu+1)})$
Let the right end of the searched interval $XQ = XP$ and $FQ = FP$
When this condition occurs for the second or subsequent time,
If $I2 > 0$ (Update direction is positive)
Update the right end of the searched interval according to
$XQ = x^{(\nu+1)}, FQ = f(x^{(\nu+1)})$
Begin from the left end of the searched interval according to
$x^{(\nu)} = XP, f(x^{(\nu)}) = FP$.
If $I2 \leq 0$ (Update direction is negative)
Update the left end of the searched interval according to
$XP = x^{(\nu+1)}, FP = f(x^{(\nu+1)})$
Begin from the right end of the searched interval according to
$x^{(\nu)} = XQ, f(x^{(\nu)}) = FQ$.

Set $P = 0$. If the condition described above in (vi) occurs repeatedly several times, set $R = 2$ to accelerate the change so that the point emerges from a trough; otherwise, set $R = 0$. Then return to (d).

(7) If the condition described in (f) does not occur, then let $I1 = I2, x^{(\nu)} = x^{(\nu+1)}$ and return to (d). Figure 4−1 shows an example of the movements described above in (a) through (g).

Figure 4−1




(a) According to a Newton iteration, the following relationship occurs $| f(x^{(\nu+1)}) |>| f(x^{(\nu)}) |$.

(b) The $x^{(\nu+1)}$ update is performed according to the update expression shown in (d).

(c) Since the update direction was not change due to $| f(x^{(\nu+1)}) |>| f(x^{(\nu)}) |$, the left end of the searched interval is determined according to (d) and the search direction is reversed.

(d) The search starting point and search direction are changed repeatedly in a similar manner as described in 3.

(e) Since $| f(x^{(\nu+1)}) |<| f(x^{(\nu)}) |$ occurs, the root is obtained by the Newton's method of (a) or (d).

Convergence is determined as follows. If $e_r$ is assumed to be the required precision, then convergence is considered to have occurred when:

$$(| x^{(\nu+1)} - x^{(\nu)} |< e_r \max(1, | x^{(\nu+1)} |) \text{ and } | f(x^{(\nu+1)}) |< e_r + 64\varepsilon | x^{(\nu+1)} |)$$
$$\text{or } f(x^{(\nu+1)}) = 0$$

#### 4.1.2.4 The roots of real functions (initial value specified; derivative definition not required)

This algorithm basically uses the secant method indicated by:

$$x^{(\nu+1)} = x^{(\nu)} - f(x^{(\nu)}) \frac{x^{(\nu)} - x^{(\nu-1)}}{f(x^{(\nu)}) - f(x^{(\nu-1)})}$$

which has been improved in a manner similar to that described in (3). However, when approaching $f(x) = 0$, an extremum search is performed for a location if a trough occurs or the bisection method is used if the sign of the root is found to reverse. The algorithm is explained in more detail below.

(1) Update the solution by using the secant method.

(2) If both the function value and the update amount are large, then since $x^{(\nu)}$ is far from a root perform the following to be safe:

$$\left.\begin{array}{l} XP = XQ = x^{(\nu)} \\ FP = FQ = f(x^{(\nu)}) \end{array}\right\} \left(\begin{array}{l} \text{Set the left and right ends of the searched interval;} \\ XP \text{ is the left end and } XQ \text{ is the right end} \end{array}\right)$$

Let $x^{(\nu+1)} = x^{(\nu)}$ and $f(x^{(\nu+1)}) = f(x^{(\nu)})$ and skip to (c)

If $\mid f(x^{(\nu+1)}) \mid \geq \mid f(x^{(\nu)}) \mid$

If $\mid \dfrac{x^{(\nu)} - x^{(\nu-1)}}{f(x^{(\nu-1)})} \mid > 0.125$ (a root is expected to be at a location close to $x^{(\nu)}$)

Let $XP = XQ = x^{(\nu-1)}, FP = FQ = f(x^{(\nu-1)}), x^{(\nu+1)} = x^{(\nu-1)}$ and $f(x^{(\nu+1)}) = f(x^{(\nu-1)})$ and go to (c)

If $\mid \dfrac{(x^{(\nu)} - x^{(\nu-1)}}{f(x^{(\nu-1)})} \mid \leq 0.125$,

If the value was updated in the positive direction,

Let $XP = x^{(\nu-1)}, XQ = x^{(\nu+1)}, FP = f(x^{(\nu-1)})$ and $FQ = f(x^{(\nu+1)})$ (set searched interval) and perform an extremum search in the interval from $x^{(\nu-1)}$ to $x^{(\nu+1)}$.

If the value was updated in the negative direction,

Let $XP = x^{(\nu+1)}, XQ = x^{(\nu-1)}, FP = f(x^{(\nu+1)})$ and $FQ = f(x^{(\nu-1)})$ (set searched interval) and perform an extremum search in the interval from $x^{(\nu+1)}$ to $x^{(\nu-1)}$.

If a root is not found by the extremum search then skip to (c).

If none of the above conditions occurs

set $x^{(\nu)} = x^{(\nu+1)}$ and return to (a).

(3) If the function increases to the right in the interval from $x^{(\nu-1)}$ to $x^{(\nu+1)}$, let $IS = -1$; if it increases to the left, let $IS = 1$.
Let $R = 1, P = 0, IOLD = \text{sign}\left\{1, IS \times f(x^{(\nu+1)})\right\}$, and $I1 = -IOLD$.

(4) Let $I2 = \text{sign}\left\{1, IS \times f(x^{(\nu+1)})\right\}$.
If $I2 \times I1 > 0$, then change is accelerated by letting $P = P + 1$; otherwise, it is decelerated by letting $R = R+1$. (For $I1$: Old update direction and $I2$ : New update direction, $+$ indicates update in the positive direction and $-$ indicates update in the negative direction.)
Let $x^{(\nu)} = x^{(\nu+1)}, f(x^{(\nu)}) = f(x^{(\nu+1)}), x^{(\nu-1)} = x^{(\nu)}$ and $f(x^{(\nu-1)}) = f(x^{(\nu)})$.
$x^{(\nu)}$ is updated by the following iteration.
Let $\varepsilon$ be the unit for determining error, and let $x^{(\nu+1)}$ be as follows:

$$x^{(\nu+1)} = x^{(\nu)} + IS \times \sinh^{-1}(f(x^{(\nu)})) \times 2^{\{(P-3)/3-R\}} + \mid x^{(\nu)} + 1 \mid \times I2 \times \varepsilon$$

(For $IS$: Global function slope estimate, $+$ indicates decreasing to the right and $-$ indicates increasing to the right.)
This update is repeated at least three times.

(5) For $f(x^{(\nu-1)}) \gg f(x^{(\nu)}) \gg f(x^{(\nu+1)})$, if $\mid f(x^{(\nu+1)}) \mid$ becomes sufficiently small, return to the secant method shown in (a).

(6) If the signs of $f(x^{(\nu-1)})$ through $f(x^{(\nu+1)})$ are the same when $\mid f(x^{(\nu-1)}) \mid < \mid f(x^{(\nu)}) \mid < \mid f(x^{(\nu+1)}) \mid$, then:

If $I2 > 0$ (update direction is positive)

Update the right end of the searched interval by setting
$XQ = x^{(\nu+1)}$ and $FQ = f(x^{(\nu+1)})$

If $I2 \leq 0$ (update direction is negative)

Update the right end of the searched interval by setting
$XP = x^{(\nu+1)}$ and $FP = f(x^{(\nu+1)})$

Then, perform an extremum search in the interval from $x^{(\nu+1)}$ to $x^{(\nu-1)}$.
If a root was not found by the extremum search, then set $x^{(\nu+1)}$ and $f(x^{(\nu+1)})$ for the endpoint of the searched interval in the direction opposite to the direction that the solution had been updated, let:

$$IS = -IS, IOLD = -IOLD, I2 = IOLD, P = 0, R = 1$$

so that the value is updated in the opposite direction, and return to (d).

(7) If the signs of $f(x^{(\nu-1)})$ through $f(x^{(\nu+1)})$ are the same when $\mid f(x^{(\nu-1)}) \mid < \mid f(x^{(\nu)}) \mid < \mid f(x^{(\nu+1)}) \mid$, then set $x^{(\nu+1)}$ and $f(x^{(\nu+1)})$ for the endpoint of the searched interval in the direction opposite to the direction that the solution had been updated, let:

$$IS = -IS, IOLD = -IOLD, I2 = IOLD, P = 0, R = 1$$

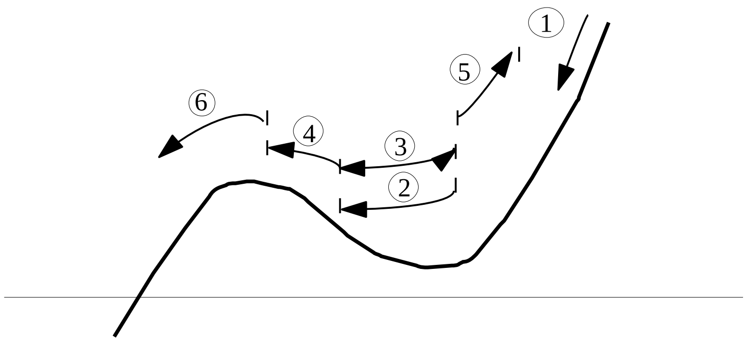
so that the value is updated in the opposite direction, and return to (d).

(8) If none of the conditions shown in (e) through (g) occurs, then let $I1 = I2$ and return to (d).

The extremum search portion of the algorithm obtains the minimum point by combining a golden section search with sequential parabolic interpolation. (See Section 5.1.2)
If a reversal of the sign of the equation value is seen during the extremum search, then the bisection method is performed to obtain the root that is on the initial value side of that point.

Figure 4−2 shows an example of the movements described above in (a) through (h).

Figure 4−2




(1) Perform the secant method iteration.

(2) $\mid f(x^{(\nu+1)}) \mid > \mid f(x^{(\nu)}) \mid$ occurs.

(3) An extremum search is performed, but the extremum is not a root.

(4) The solution is updated according to the update expression shown in (d) and the condition described in (g) occurs.

(5) A similar situation as described in 4 occurs even though the movement is in the opposite direction.

(6) Since $| f(x^{(\nu+1)}) | < | f(x^{(\nu)}) |$ occurs, the root is obtained by the secant method of (a) or (d).

Convergence is determined as follows. If $e_r$ is assumed to be the required precision, then convergence is considered to have occurred when:

$$\Big(| x^{(\nu+1)} - x^{(\nu)} | < e_r \max(1, | x^{(\nu+1)} |) \text{ and } | f(x^{(\nu+1)}) | < e_r + 64\varepsilon \, | x^{(\nu+1)} |\Big)$$
$$\text{or } f(x^{(\nu+1)}) = 0$$

Also, for an extremum search or for the bisection method, convergence is considered to have occurred when:

$$\text{(Search reduction interval)} < e_r \max(1, | x |) \text{ and } | f(x) | < e_r + 64\varepsilon \, | x |$$

#### 4.1.2.5 The roots of real functions (interval specification; derivative definition not required)

Assume that $f(x)$ is continuous on the interval $[a, b]$ and that the signs of the values $f(a)$ and $f(b)$ are opposite.

(1) Let $c = a$ and $d = e = b - a$.

(2) If $| f(c) | < | f(b) |$, exchange $b$ and $c$ and let $a$ be the $c$ after the exchange. In this way, the relationship $| f(b) | \le | f(c) |$ always will occur and the root is searched for in the interval $[b, c]$.

(3) Let $m = \dfrac{c-b}{2}$. Also, for:
$\varepsilon$ = Unit for determining error,
$e_r$ = Required precision
let:

$$\delta = \frac{\varepsilon \, | b | + e_r}{2}$$

(4) If $| e | \ge \delta$ and $| f(b) | < | f(a) |$, then
If $a = c$, the linear interpolation method is applied in the interval between $c$ and $b$.

$$P = (c-b)\frac{f(b)}{f(c)}$$
$$Q = 1 - \frac{f(b)}{f(c)}$$

If $a \ne c, (| f(a) | \le | f(c) |)$, the inverse quadratic interpolation method is applied in the intervals between $a, b$ and $c$.

$$\text{For } r_1 = \frac{f(a)}{f(c)}, r_2 = \frac{f(b)}{f(c)}, r_3 = \frac{f(b)}{f(a)}$$
$$R = r_3 \{(c-b) r_1 (r_1 - r_2) - (b-a)(r_2 - 1)\}$$
$$Q = (r_1 - 1)(r_2 - 1)(r_3 - 1)$$

If $| \dfrac{P}{Q} | > \dfrac{3}{4} | c - b | - | \delta |$ or $| \dfrac{P}{Q} | > | \dfrac{e}{2} |$, then skip to (e).
Otherwise, set $e = d$ and $d = -\dfrac{P}{Q}$.
Set $a = b$ and let:

$$b = b + \max(d, \operatorname{sign}(\delta, c - b))$$

(5) If the condition described in (d) does not occur, then set $a = b$, use the bisection method to set $b$ to the midpoint of the interval $[b, c]$, and let $d = e = m$.

(6) If the signs of the values $f(b)$ and $f(c)$ are the same, then set $c = a$ and let $d = e = b - a$.
After the above operation, return to (b).
Convergence is considered to have occurred if $f(b) = 0$ or if $\mid c - b \mid \leq e_r + 2\varepsilon \mid b \mid$.

#### 4.1.2.6 All the roots of real functions (interval specification; derivative definition not required)

This algorithm basically uses the same algorithm as the one described in (4) that is for finding a single root of a nonlinear equation when the derivative definition is not required. However, this algorithm uses the fact that $IS$ can be used in the expression shown in (d) to keep the search direction fixed so that roots are always obtained by moving inward from the endpoints of the interval. That is, this algorithm differs from the one in (4), and the following points have been changed.

- Intervals where a search for the solution will be excluded are taken from both ends of the interval.
- First, a root is sought beginning from the right end of the interval. If a root is found, the right end of the interval is changed to that point. Next, a root is sought beginning from the left end of the interval. If a root is found, the left end of the interval is changed to that point.
- If the root update value leaves the interval, it is moved back into the interval and processing continues.
- If the sign of the equation value changes during an extremum search, the roots to both the left and right of that point are obtained by the bisection method.
- If the interval size becomes less than or equal to $\delta$, then processing ends.

When a root is found, the interval size is reduced. However, at this time, the endpoint is shifted by a distance of the following value of $\delta$ towards the interior of the interval from that root.

$$\delta = \max(2e_r, (\mid A \mid + \mid B \mid)\varepsilon, \sqrt[3]{\varepsilon})$$

Where, $e_r$ : Required precision

$\varepsilon$ : Unit for determining error

$A, B$ : Left and right ends of the initially set interval

#### 4.1.2.7 The roots of complex functions (initial value specified; derivative definition not required)

This algorithm finds a solution according to Muller's method.
If the initial value $z = 0$, then assume that the starting values are:

$$\begin{cases} z_1 = -1.0 \\ z_2 = 1.0 \\ z_3 = 0.0 \end{cases}$$

If the initial value $z \neq 0$, then assume that the starting values are:

$$\begin{cases} z_1 = 0.9z \\ z_2 = 1.1z \\ z_3 = z \end{cases}$$

Let $f_i = f(z_i)$ and define $q, q', a, b$ and $c$ as follows:

$$\begin{aligned} q &= \frac{z_3 - z_2}{z_2 - z_1} \\ q' &= q + 1 \\ a &= qf_3 - qq'f_2 + q^2 f_1 \\ b &= (q + q')f_3 - q'^2 f_2 + q^2 f_1 \\ c &= q' f_3 \end{aligned}$$

Define $r$ as follows:

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

and take the $r$ having the smallest absolute value. (If the denominator of $r$ is zero and $f_1 = f_2 = f_3$, then set $r = 1$.)
Let $z = z_3 + r(z_3 - z_2)$, and then set $z_3, z_2$ and $z_1$ as follows:

$$\begin{cases} z_3 = z \\ z_2 = z_3 \\ z_1 = z_2 \end{cases}$$

and repeatedly iterate these steps.
Convergence is determined as follows. If $e_r$ is assumed to be the required precision and $\varepsilon$ is assumed to be the unit for determining error, then convergence is considered to have occurred when:

$$(| z - z_3 | < e_r \max(1, | z |) \text{ and } | f(z) | < e_r + 64\varepsilon | z |) \text{ or } f(z) = 0$$

#### 4.1.2.8 The roots of a set of simultaneous nonlinear equations (Jacobian matrix definition optional)

This algorithm obtains a solution according to Marquardt's method, which is explained below.

(1) Let $\lambda = 0.1$.

(2) Let the Jacobian matrix of $f(\boldsymbol{x})$ be $A$.
For $f(\boldsymbol{x})$ and $\boldsymbol{x}$ defined as follows:

$$\begin{aligned} f(\boldsymbol{x}) &= (f_1, f_2, \cdots, f_n)^T \\ \boldsymbol{x} &= (x_1, x_2, \cdots, x_n)^T \end{aligned}$$

where $n$ is the number of order, $A$ is defined as follows:

$$A = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \cdots & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

Let $D$ be a matrix formed by setting non-diagonal elements of $A^T A$ to zero.

(3) Obtain $\Delta x$ by solving the simultaneous linear equation:

$$\left(A^T A + \lambda D\right) \Delta \boldsymbol{x} = -A^T f(\boldsymbol{x})$$

(The expression is equivalent to applying Marquardt's method in which $x$ is scaled by $D^{1/2}$ and the $\lambda I$ term is added.)

(4) Assume $\boldsymbol{y} = \boldsymbol{x} + \Delta \boldsymbol{x}$.
If $\|f(\boldsymbol{y})\| \geq \|f(\boldsymbol{x})\|$, (where the symbol $\|\cdots\|$ indicates the square norm), then:
If $\lambda = 0$, set $\lambda = 0.001$
and:
If $\lambda \neq 0$, set $\lambda = 10\lambda$
and then return to (c).
If $\|f(\boldsymbol{y})\| < \|f(\boldsymbol{x})\|$, then if $\|f(\boldsymbol{y})\| < \|f(\boldsymbol{x})\|$ also occurred during the previous iteration, set $\lambda = \dfrac{\lambda}{10}$.
Set $\boldsymbol{x} = \boldsymbol{y}$ and return to (b).

Since the above Marquardt's method explicitly forms a system of normal equations, it has the shortcoming that it may not converge due to error. Therefore, if $e_r$ is assumed to be the required precision and $\varepsilon$ is assumed to be the unit for determining error, then if the following condition is reached:

$$(\|\Delta \boldsymbol{x}\|_\infty \leq e' \max(1, \|\boldsymbol{y}\|_\infty) \text{ and } \|f(\boldsymbol{y})\|_\infty \leq e') \text{ or } \|f(\boldsymbol{y})\|_\infty = 0$$
$$e' = {e_r}^{0.2}$$

where the symbol $\|\cdots\|_\infty$ indicates the maximum of the absolute values of the elements

or if $\|f(\boldsymbol{y})\|$ becomes only 0.75 times the value of $\|f(\boldsymbol{y})\|$ from the $(4n)$-th previous iteration, then processing shifts to the following Newton's method with scaling.
Newton's method with scaling
Define $g_i$ as follows:

$$g_i = \max(\varepsilon, a_{i1}, a_{i2}, \cdots, a_{in}) \text{ for } (i = 1, \cdots, n)$$

and perform the following scaling operations:

$$(a_{i1}, a_{i2}, \cdots, a_{in}) = \frac{(a_{i1}, a_{i2}, \cdots, a_{in})}{g_i}$$
$$f_i = \frac{f_i}{g_i}$$

Solve the simultaneous linear equations $A\Delta\boldsymbol{x} = -f(\boldsymbol{x})$ using the Jacobian matrix $A$ and function values $f(\boldsymbol{x})$ that were scaled in this way, and update the solution by using $\Delta\boldsymbol{x}$ as the correction vector.
Convergence is assumed to have occurred when the following relationships hold:

$$(\|\Delta \boldsymbol{x}\|_\infty < e_r \max(1, \|\boldsymbol{y}\|_\infty) \text{ and } \|f(\boldsymbol{y})\|_\infty < e_r + 64\varepsilon\|\boldsymbol{y}\|_\infty)$$
$$\text{or } \|f(\boldsymbol{y})\|_\infty = 0$$

#### 4.1.2.9 The roots of a set of simultaneous nonlinear equations (Jacobian matrix definition not required)

(1) Correction vector $\Delta\boldsymbol{x}$ calculation
Assume that $f(\boldsymbol{x})$, $A$ and $G$ are as follows:

$f(\boldsymbol{x})$ : Function value $f(\boldsymbol{x})$ at variable value $\boldsymbol{x}$

$A$ : Jacobian matrix $\dfrac{\partial f}{\partial \boldsymbol{x}}$
$G$ : Inverse matrix of the Jacobian matrix

First, let the Gauss-Newton solution be represented by $\Delta\boldsymbol{x}_G$ and the steepest descent solution be represented by $\Delta\boldsymbol{x}_S$ as follows:

$$\begin{aligned} \Delta\boldsymbol{x}_G &= -Gf(\boldsymbol{x}) \\ \Delta\boldsymbol{x}_S &= \left(\frac{\|\boldsymbol{b}\|^2}{\|A\boldsymbol{b}\|^2}\right)\boldsymbol{b} \end{aligned}$$

where $\boldsymbol{b} = -A^T f(\boldsymbol{x})$.
At first, let $\Delta\boldsymbol{x} = \Delta\boldsymbol{x}_S$ and $d = \|\Delta\boldsymbol{x}_S\|$.
For the second and subsequent iteration, let $d$ be the step size.

(a) If $d \le \|\Delta\boldsymbol{x}_S\|$, then:

$$\Delta\boldsymbol{x} = d\frac{\Delta\boldsymbol{x}_S}{\|\Delta\boldsymbol{x}_S\|}$$

(b) If $\|\Delta\boldsymbol{x}_S\| < d < \|\Delta\boldsymbol{x}_G\|$, then:

$$\Delta\boldsymbol{x} = \alpha\Delta\boldsymbol{x}_S + \beta\Delta\boldsymbol{x}_G \quad \text{for} \quad (\alpha > 0, \beta > 0, \|\Delta\boldsymbol{x}\| = d)$$

(c) If $\|\Delta\boldsymbol{x}_G\| \le d$

$$\Delta\boldsymbol{x} = \Delta\boldsymbol{x}_G$$

(2) Step size $d$ modification
Let $\Delta s$ be the difference of the sum of the squares of the linearized model function values and let $\Delta T$ be the difference of the sum of the squares of the actual function values. Estimate the degree of nonlinearity by using the ratio $r$ of these values.

$$\begin{aligned} \Delta s &= \|f(\boldsymbol{x}) - A\Delta\boldsymbol{x}\|^2 - \|f(\boldsymbol{x})\|^2 \\ \Delta T &= \|f(\boldsymbol{x} + \Delta\boldsymbol{x})\|^2 - \|f(\boldsymbol{x})\|^2 \\ r &= \frac{\Delta T}{\Delta s} \end{aligned}$$

(a) If $r < 0.1$, then reduce $d$ by half and let $\tau = 1.0$.

(b) If $r \ge 0.1$, then calculate $\lambda$, which is the rate of increase of $d$, as follows:

$$\lambda = \sqrt{1 - \frac{(r-0.1)\Delta s}{s_p + \sqrt{{s_p}^2 - (r-0.1)s_s\Delta s}}}$$

where, for $\delta f$ defined as:

$$\delta f = f(\boldsymbol{x} + \Delta\boldsymbol{x}) - \{f(\boldsymbol{x}) + A\Delta\boldsymbol{x}\}$$

$s_p$ and $s_s$ are as follows:

$$\begin{aligned} s_p &= \sum_{i=1}^{n} \mid f_i(\boldsymbol{x} + \Delta\boldsymbol{x})\delta f_i \mid \\ s_s &= \|\delta f\|^2 \end{aligned}$$

Actually, to prevent the value of $d$ from oscillating, $d$ is increased only when an increase is required two times consecutively. Also, the rate of increase is held to at most 2. The actual rate of increase $\mu$ is calculated as follows:

$$
\begin{aligned}
\mu &= \min(2, \lambda, \tau) \\
\tau &= \frac{\lambda}{\mu}
\end{aligned}
$$

where the initial value for $\tau$ is assumed to be 1.
In addition, an upper limit $d_{\max}$ and lower limit $d_{\min}$ are set for $d$ and $d$ is controlled so that it falls between these values.

(3) Calculations of the Jacobian matrix $A$ and its inverse matrix $G$
The Jacobian matrix is obtained according to a difference only for the first iteration, and thereafter, it is sequentially updated. Similarly, its inverse matrix is obtained from the Jacobian matrix only the first iteration, and thereafter, it is sequentially updated. The sequential updates are calculated according to the following formulas:

$$
\begin{aligned}
A &= A + \delta f \frac{\Delta \boldsymbol{x}^T}{\|\Delta \boldsymbol{x}\|^2} \\
G &= G + (\Delta \boldsymbol{x} - G\Delta f)\Delta \boldsymbol{x}^T \frac{G}{\Delta \boldsymbol{x}^T} G\Delta f
\end{aligned}
$$

where:

$$\Delta f = f(\boldsymbol{x} + \Delta \boldsymbol{x}) - f(\boldsymbol{x})$$

(4) Correction vector independence check
To correct the Jacobian matrix efficiently, the correction vectors that are taken sequentially must be nearly mutually orthogonal. Therefore, an original independence concept is defined according to a hybrid method, and the correction vectors are controlled so that they are taken in directions that are as independent as possible. A vector $\alpha$ is said to be independent of the $j$ vectors $(\alpha_1, \alpha_2, \cdots, \alpha_j)$ if $\alpha$ forms an angle of at least 30 degrees with an arbitrary vector of the space defined by these $j$ vectors. The calculation for this independence test conceived by Powell is as follows.
$n$ mutually independent vectors from the vectors used to correct the Jacobian matrix during the past $(2 \times n)$ iterations are made to be orthogonal and are stored in $\Omega = (\omega_1, \omega_2, \cdots, \omega_n)$. The array $\ell$ of size $n$ is used to store information indicating the number of iterations earlier in which $\omega_i$ was the correction vector. That is, this information indicates that $\omega_i$ is the vector that was taken $\ell_i$ iterations earlier. $\Omega$ is initialized as the unit matrix, and $\ell$ is initialized with values $\ell_i = n - i + 1 \;\; \text{for} \;\; (i = 1, \cdots, n)$
When a solution is corrected, the following steps are performed.

(a) When $\Delta \boldsymbol{x} = \Delta \boldsymbol{x}_G$
Regardless of its independence, $\Delta \boldsymbol{x}$ is taken as the correction vector.

(b) When $\Delta \boldsymbol{x} = \Delta \boldsymbol{x}_G$ does not occur
If $\ell_1 < 2n$ or if $\Delta \boldsymbol{x}$ is independent of $(\omega_2, \omega_3, \cdots, \omega_n)$, that is, if $| \, {\omega_1}^T \Delta \boldsymbol{x} \, | > \dfrac{\|\Delta \boldsymbol{x}\|}{2}$, then $\Delta \boldsymbol{x}$ is taken as the correction vector. Otherwise, the solution is not corrected.
When the Jacobian matrix is corrected, the following steps are performed.
If $\|\Delta \boldsymbol{x}\| < d_{\min}$ or if $\ell_1 = 2n$ and $\Delta \boldsymbol{x}$ is not independent of $(\omega_2, \omega_3, \cdots, \omega_n)$, then $\Delta \boldsymbol{x} = d_{\min}\omega_1$ is set.

Otherwise, $\Delta \boldsymbol{x}$ is taken.

Next, $\Omega$ and $\ell$ are revised. When $\Delta \boldsymbol{x} = d_{\min}\omega_1$ has been set, the following values should be set:

$$\begin{aligned} \omega_i &= \omega_{i+1} && \text{for } (i = 1, \cdots, n-1) \\ \omega_n &= \omega_1 \\ \ell_i &= \ell_{i+1} + 1 && \text{for } (i = 1, \cdots, n-1) \\ \ell_n &= 1 \end{aligned}$$

Otherwise, the following is performed.

The minimum value $k$ is obtained for which $(\omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta \boldsymbol{x})$ are mutually independent. $(\omega_1, \cdots, \omega_{k-1}, \omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta \boldsymbol{x})$ are made to be orthogonal, and these are again assumed to be $(\omega_1, \omega_2, \cdots, \omega_n)$. The following values are then set:

$$\begin{aligned} \ell_i &= \ell_i + 1 && \text{for } (i = 1, \cdots, k-1) \\ \ell_i &= \ell_{i+1} + 1 && \text{for } (i = k, \cdots, n-1) \\ \ell_n &= 1 \end{aligned}$$

In this way, the relationship $\ell_1 \leq 2n$ always holds.

(5) Switch of processing to Newton's method with scaling

If the various equations of the given problem have not been scaled, then trouble may occur during the solution modification process and this algorithm will not converge. Therefore, the following condition is used to detect the trouble during the solution modification process and cause processing to switch to Newton's method with scaling.

$$\frac{\|f(\boldsymbol{x} + \Delta \boldsymbol{x})\|^2 - \|f(\boldsymbol{x})\|^2}{d} > 10^{10}$$

The algorithm of Newton's method with scaling is as follows.

(a) Jacobian matrix calculation

The Jacobian matrix is obtained according to a difference during every iteration.

(b) Correction vector calculation

Assume that the $(i, j)$ component of the Jacobian matrix is $a_{ij}$. First, the values $g_i$ are obtained as follows.

$g_i = \max(\text{Unit for determining error}, a_{i1}, a_{i2}, \cdots, a_{iN})$ for $(i = 1, 2, \cdots, N)$

Then the following scaling operations are performed:

$$(a_{i1}, a_{i2}, \cdots, a_{iN}) \leftarrow \frac{(a_{i1}, a_{i2}, \cdots, a_{iN})}{g_i}$$

$$f_i \leftarrow \frac{f_i}{g_i}$$

for $(i = 1, 2, \cdots, N)$

The simultaneous linear equations $A\Delta \boldsymbol{x} = -f(\boldsymbol{x})$ obtained by using the Jacobian matrix $A$ and function values $f(\boldsymbol{x})$ that were scaled in this way are solved according to the Crout method. This $\Delta \boldsymbol{x}$ becomes the correction vector.

(6) Convergence decision

Convergence is assumed to have occurred when the following relationships hold:

$$(\|\Delta \boldsymbol{x}\|_\infty < e_r \max(1, \|\boldsymbol{x} + \Delta \boldsymbol{x}\|_\infty) \text{ and } \|f(\boldsymbol{x} + \Delta \boldsymbol{x})\|_\infty < e_r + 64\varepsilon\|\boldsymbol{x} + \Delta \boldsymbol{x}\|)$$
$$\text{or } f(\boldsymbol{x} + \Delta \boldsymbol{x}) = 0$$

where $e_r$ is the required relative precision, and:

$$\begin{aligned}
\|\boldsymbol{x}\|_\infty &= \max_i \mid x_i \mid \\
\|\Delta\boldsymbol{x}\|_\infty &= \max_i \mid \Delta x_i \mid \\
\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|_\infty &= \max_i \mid f_i \mid
\end{aligned}$$

### 4.1.3 Reference Bibliography

(1) Forsythe, G. E. , Malcolm, M. A. and Moler, C. B. , "Computer Methods for Mathematical Computations", Prentice-Hall Inc. , (1978).

## 4.2 ALGEBRAIC EQUATIONS

### 4.2.1 DLARHA, RLARHA
### The Roots of Real Coefficient Algebraic Equations

(1) **Function**

DLARHA or RLARHA solves an algebraic equation having real coefficients. (The roots are stored in real arrays.)

(2) **Usage**

Double precision:
CALL DLARHA (A, N, XR, XI, WK, IERR)
Single precision:
CALL RLARHA (A, N, XR, XI, WK, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N+1 | Input | Coefficients of the algebraic equation stored in the descending powers of $x$; CA(1) is the coefficient of $x^{\text{N}}$, and CA(N+1) is the constant term. |
| 2 | N | I | 1 | Input | Degree of algebraic equation |
| 3 | XR | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N | Output | Real parts of the roots of the algebraic equation |
| 4 | XI | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N | Output | Imaginary parts of the roots of the algebraic equation |
| 5 | WK | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | $4 \times (\text{N} + 1)$ | Work | Work area (not used when $\text{N} \leq 4$) |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) At least one of $\text{A(i)}, \text{i} = 1, 2, \cdots, \text{N}$ must satisfy the condition $|\text{A(i)}| >$ Unit for determining error.

(b) $\text{N} \geq 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 2000+$i$ | The first $i$ roots were obtained correctly, but the subsequent roots were obtained with bad precisions. | Processing terminates with the precision of the $(i+1)$th root remaining bad. |
| 2500+$j$ | $\lvert\mathrm{A(i)}\rvert \leq$Unit for determining error, $\mathrm{i} = 1, 2, \cdots, \mathrm{j}$ | Processing is performed on condition that $\mathrm{A(i)} = 0.0, \mathrm{i} = 1, 2, \cdots, \mathrm{j}$ |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000+$i$ | The first $i$ roots were obtained, but the subsequent roots could not be obtained. | The first $i$ roots are calculated, and then processing is aborted. |

(6) **Notes**

(a) If there are more than four roots, then except for the last four roots, this subroutine tends to obtain roots in ascending order of their distance from 0.0.

(b) If there is a multiple root, then the relative precision for a root having multiplicity $n$ will be on the order of $\sqrt[n]{\text{Unit for determining error}}$.

(7) **Example**

(a) Problem
Solve the following equation:

$$x^{10} - 55x^8 + 1023x^6 - 7645x^4 + 21076x^2 - 14400 = 0$$

(b) Input data
Array A which contains coefficients of the algebraic equations:
A(1)=1.0, A(2)=0.0, A(3)= -55.0,
A(4)=0.0, A(5)=1023.0, A(6)= 0.0,
A(7)=-7645.0, A(8)=0.0, A(9)=21076.0,
A(10)=0.0, A(11)=14400.0.
N=10.

(c) Main program

```
      PROGRAM BLARHA
! *** EXAMPLE OF DLARHA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER N,IERR,I
      DIMENSION A(11),XR(10),XI(10),WK(44)
!
      READ(5,*) N
      READ(5,*) (A(I),I=1,N+1,1)
      WRITE(6,1000)
      WRITE(6,2000) N
      WRITE(6,3000)
      WRITE(6,4000) (A(I),I=1,N+1,1)
      CALL DLARHA(A,N,XR,XI,WK,IERR)
      WRITE(6,5000)
      WRITE(6,6000) IERR
      WRITE(6,7000)
      WRITE(6,8000) (XR(I),XI(I),I=1,N,1)
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLARHA ***',/,&
      7X,'* ALGEBRAIC EQUATION *',/,&
      9X,'X**10-55*(X**8)+1023*(X**6)-7645*(X**4)+21076*(X**2)',&
      '-14400=0',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'DEGREE OF ALGEBRAIC EQUATION = ',I2)
 3000 FORMAT(9X,'COEFFICIENTS OF ALGEBRAIC EQUATION',/)
 4000 FORMAT(9X,F8.1)
```

```
5000 FORMAT(' ',/,/,6X,'** OUTPUT **')
6000 FORMAT(9X,'IERR = ',I4)
7000 FORMAT(9X,'ROOTS',/,&
     9X,'REAL PART',13X,'IMAGINARY PART',/)
8000 FORMAT(9X,D17.10,5X,D17.10)
     END
```

(d) Output results

```
*** DLARHA ***
  * ALGEBRAIC EQUATION *
    X**10-55*(X**8)+1023*(X**6)-7645*(X**4)+21076*(X**2)-14400=0
 ** INPUT **
    DEGREE OF ALGEBRAIC EQUATION = 10
    COEFFICIENTS OF ALGEBRAIC EQUATION

            1.0
            0.0
          -55.0
            0.0
         1023.0
            0.0
        -7645.0
            0.0
        21076.0
            0.0
       -14400.0


 ** OUTPUT **
    IERR =    0
    ROOTS
    REAL PART             IMAGINARY PART

     0.1000000000D+01      0.0000000000D+00
    -0.1000000000D+01      0.0000000000D+00
     0.2000000000D+01      0.0000000000D+00
    -0.2000000000D+01      0.0000000000D+00
     0.3000000000D+01      0.0000000000D+00
    -0.3000000000D+01      0.0000000000D+00
     0.4000000000D+01      0.0000000000D+00
     0.5000000000D+01      0.0000000000D+00
    -0.5000000000D+01      0.0000000000D+00
    -0.4000000000D+01      0.0000000000D+00
```

### 4.2.2 ZLACHA, CLACHA The Roots of Complex Coefficient Algebraic Equations

(1) **Function**

ZLACHA or CLACHA solves an algebraic equation having complex coefficients. (The coefficients and the roots are stored in complex arrays.)

(2) **Usage**

Double precision:
CALL ZLACHA (CA, N, NEV, CX, WK, CWK, IERR)
Single precision:
CALL CLACHA (CA, N, NEV, CX, WK, CWK, 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 | CA | {Z, C} | N+1 | Input | Coefficients of the algebraic equation stored in the descending powers of $x$; CA(1) is the coefficient of $x^{\mathrm{N}}$, and CA(N+1) is the constant term. |
| 2 | N | I | 1 | Input | Degree of algebraic equation |
| 3 | NEV | I | 1 | Input | Maximum number of function evaluations (Default value: 100) |
| | | | | Output | Actual number of function evaluations |
| 4 | CX | {Z, C} | N | Output | Roots of the algebraic equation |
| 5 | WK | {D, R} | $\mathrm{N}+1$ | Work | Work area |
| 6 | CWK | {Z, C} | $4 \times (\mathrm{N}+1)$ | Work | Work area |
| 7 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{CA}(1) \neq (0,0)$

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

(c) NEV $> 0$ (except when 0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (c) was not satisfied. | Processing is performed with the default value set for NEV. |
| 2000+$i$ | After the first $i$ roots were obtained, convergence did not occur within the maximum number of function evaluations. | Processing terminates with roots after the first $i$ roots not converging. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) If 0 is entered for the argument NEV, then the default value will be set.

(b) If there is a multiple root, then the relative precision for a root having multiplicity $n$ will be on the order of $\sqrt[n]{\text{Unit for determining error}}$.

(7) **Example**

(a) Problem
Solve the following equation:

$$x^{10} - 100\sqrt{-1}x^9 - 17x^6 + 1700\sqrt{-1}x^5 + 16x^2 - 1600\sqrt{-1}x = 0$$

(b) Input data
Array CA which contain coefficients of the algebraic equations:
CA(1)=(1.0, 0.0), CA(2)=(0.0, -100.0),
CA(2)=(0.0, 0.0), CA(4)=(0.0, 0.0),
CA(5)=(-17.0, 0.0), CA(6)=(0.0, 1700.0),
CA(7)=(0.0, 0.0), CA(8)=(0.0, 0.0),
CA(9)=(16.0, 0.0), CA(10)=(0.0, -1600.0),
CA(11)=(0.0, 0.0).
N=10 and NEV=0.

(c) Main program

```
      PROGRAM ALACHA
! *** EXAMPLE OF ZLACHA ***
      IMPLICIT COMPLEX(8) (A-H,O-V,X-Z)
      IMPLICIT REAL(8) (W)
      INTEGER N,NEV,IERR,I
      DIMENSION CA(11),CX(10),WK(11),CWK(44)
!
      READ(5,*) N
      READ(5,*) NEV
      READ(5,*) (CA(I),I=1,N+1,1)
      WRITE(6,1000)
      WRITE(6,1500) N
      WRITE(6,2000) NEV
      WRITE(6,2500)
      WRITE(6,3000) (CA(I),I=1,N+1,1)
      CALL ZLACHA(CA,N,NEV,CX,WK,CWK,IERR)
      WRITE(6,3500)
      WRITE(6,4000) IERR
      WRITE(6,4500) NEV
      WRITE(6,5000)
      WRITE(6,5500) (CX(I),I=1,N,1)
      STOP
!
 1000 FORMAT(' ',/,5X,'*** ZLACHA ***',/,&
      7X,'* ALGEBRAIC EQUATION *',/,&
      9X,'X**10-100I*(X**9)-17*(X**6)+1700I*(X**5)+16*(X**2)',&
      '-1600I*X=0',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'DEGREE OF ALGEBRAIC EQUATION = ',I2)
```

```
2000 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTION EVALUATIONS = ',I3)
2500 FORMAT(9X,'COEFFICIENTS OF ALGEBRAIC EQUATION',/,&
     9X,'REAL PART',5X,'IMAGINARY PART',/)
3000 FORMAT(9X,F7.1,7X,F7.1)
3500 FORMAT(' ',/,/,6X,'** OUTPUT **')
4000 FORMAT(9X,'IERR = ',I4)
4500 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTION EVALUATIONS = ',I3)
5000 FORMAT(9X,'ROOTS',/,&
     9X,'REAL PART',13X,'IMAGINARY PART',/)
5500 FORMAT(9X,D17.10,5X,D17.10)
     END
```

(d) Output results

```
*** ZLACHA ***
  * ALGEBRAIC EQUATION *
    X**10-100I*(X**9)-17*(X**6)+1700I*(X**5)+16*(X**2)-1600I*X=0
 ** INPUT **
    DEGREE OF ALGEBRAIC EQUATION = 10
    MAXIMUM NUMBER OF FUNCTION EVALUATIONS =   0
    COEFFICIENTS OF ALGEBRAIC EQUATION
    REAL PART     IMAGINARY PART

        1.0            0.0
        0.0         -100.0
        0.0            0.0
        0.0            0.0
      -17.0            0.0
        0.0         1700.0
        0.0            0.0
        0.0            0.0
       16.0            0.0
        0.0        -1600.0
        0.0            0.0


 ** OUTPUT **
    IERR =    0
    PRACTICAL NUMBER OF FUNCTION EVALUATIONS =  17
    ROOTS
    REAL PART              IMAGINARY PART

     0.0000000000D+00      0.1000000000D+03
     0.0000000000D+00     -0.2000000000D+01
     0.2000000000D+01      0.6483734377D-18
    -0.2869859255D-41      0.2000000000D+01
    -0.2000000000D+01      0.7826070366D-19
     0.0000000000D+00     -0.1000000000D+01
    -0.1000000000D+01      0.8029504743D-18
     0.0000000000D+00      0.1000000000D+01
     0.1000000000D+01      0.6074219118D-19
     0.0000000000D+00      0.0000000000D+00
```

# 4.3 NONLINEAR EQUATIONS

## 4.3.1 DLNRDS, RLNRDS

### A Root of a Real Function (Initial Value Specified; Derivative Definition Required)

(1) **Function**

DLNRDS or RLNRDS obtains a root of a nonlinear equation from an initial value, when the derivative of the nonlinear equation is defined.

(2) **Usage**

Double precision:

CALL DLNRDS (F, DF, X, ER, NEV, IERR)

Single precision:

CALL RLNRDS (F, DF, X, ER, NEV, IERR)

(3) **Arguments**

D:Double precision real    Z:Double precision complex    I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real    C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | − | Input | Name F(X) of function subprogram that defines the equation |
| 2 | DF | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | − | Input | Name DF(X) of function subprogram that defines the derivative |
| 3 | X | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Input | Initial value of root |
| | | | | Output | Root |
| 4 | ER | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Input | Required precision (Default value: (Unit for determining error)× 64) |
| 5 | NEV | I | 2 | Input | NEV(1): Maximum number of function evaluations (Default value: 100) NEV(2): Maximum number of derivative evaluations (Default value: 100) |
| | | | | Output | NEV(1): Actual number of function evaluations NEV(2): Actual number of derivative evaluations |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NEV(1) $>$ 0 (except when 0 is entered in order to set NEV(1) to the default value)

(b) NEV(2) $>$ 0 (except when 0 is entered in order to set NEV(2) to the default value)

(c) ER $\geq$ Unit for determining error
(except when 0 is entered in order to set ER to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a), (b) or (c) was not satisfied. | Processing is performed with the default value set for NEV(1), NEV(2) or ER. |
| 4000 | The root could not be obtained. | Processing is aborted. |
| 5000 | The root could not be obtained before the maximum number of function evaluations or maximum number of derivative evaluations was reached. | The value of X at that time is returned. |

(6) **Notes**

(a) The actual name in the first argument DF must be declared using an EXTERNAL statement in the user program, and a function subprograms having the actual name specified for F and DF must be created. (See Section 4.1.1) This function subprogram (in double-precision) should be created as follows.

Function subprogram creation method

```
REAL(8) FUNCTION F(X)
REAL(8) X
F =~
RETURN
END
REAL(8) FUNCTION DF(X)
REAL(8) X
DF =~
RETURN
END
```

(b) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(c) The subroutine algorithm has global convergency so that the problem can be solved even if the initial value is distant from the root.

(d) Convergence is considered to have occurred when the following relationships hold:

$$|\text{ (Root update amount) }| < \text{ER} \times \max(1, |\text{ Root value }|)$$

and

$$|\text{ Function value }| < \text{ER} + 64 \times (\text{Unit for determining error}) \times |\text{ Root value }|$$

(7) **Example**

(a) Problem

Obtain one root of the following equation:

$$\begin{cases} \mathrm{f} = \mathrm{x}^7 + 28\mathrm{x}^4 - 483.809683 = 0 \\ \mathrm{f}' = 7\mathrm{x}^6 + 112\mathrm{x}^3 \end{cases}$$

(b) Input data

Function subprogram name corresponding to function $f(x)$: FLNRDS

Function subprogram name corresponding to derivative $f'(x)$: FLNRDD

X = −1.0, ER=0.0, NEV(1)=0 and NEV(2)=0.

(c) Main program

```
      PROGRAM BLNRDS
! *** EXAMPLE OF DLNRDS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      IMPLICIT INTEGER (I-N)
      EXTERNAL FLNRDS,FLNRD2
      DIMENSION NEV(2)
!
      READ(5,*) (NEV(I),I=1,2,1)
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1000)
      WRITE(6,1500) NEV(1)
      WRITE(6,2000) NEV(2)
      WRITE(6,2500) ER
      WRITE(6,3000)
      WRITE(6,3500) X
      CALL DLNRDS(FLNRDS,FLNRD2,X,ER,NEV,IERR)
      WRITE(6,4000)
      WRITE(6,4500) IERR
      WRITE(6,5000) NEV(1)
      WRITE(6,5500) NEV(2)
      WRITE(6,6000)
      WRITE(6,6500) X
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLNRDS ***',/,&
      7X,'* NONLINEAR EQUATION *',/,&
      9X,'X**7+28*(X**4)-483.809683=0',/,&
      7X,'* DERIVATIVE *',/,&
      9X,'7*(X**6)+112*(X**3)',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTION EVALUATIONS = ',I3)
 2000 FORMAT(9X,'MAXIMUM NUMBER OF DERIVATIVE EVALUATIONS = ',I3)
 2500 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 3000 FORMAT(9X,'INITIAL VALUE OF ROOT')
 3500 FORMAT(9X,F4.1)
 4000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4500 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTION EVALUATIONS = ',I3)
 5500 FORMAT(9X,'PRACTICAL NUMBER OF DERIVATIVE EVALUATIONS = ',I3)
 6000 FORMAT(9X,'ROOT')
 6500 FORMAT(9X,D17.10)
      END


      REAL(8) FUNCTION FLNRDS(X)
      REAL(8) X
!
      FLNRDS = X**7+(28.0D0)*(X**4)-(483.809683D0)
      RETURN
      END


      REAL(8) FUNCTION FLNRD2(X)
      REAL(8) X
!
      FLNRD2 = 7.0D0*(X**6)+112.0D0*(X**3)
      RETURN
      END
```

(d) Output results

```
    *** DLNRDS ***
      * NONLINEAR EQUATION *
        X**7+28*(X**4)-483.809683=0
      * DERIVATIVE *
        7*(X**6)+112*(X**3)
     ** INPUT **
        MAXIMUM NUMBER OF FUNCTION EVALUATIONS =   0
        MAXIMUM NUMBER OF DERIVATIVE EVALUATIONS =   0
```

```
   REQUIRED ACCURACY = 0.0D+00
   INITIAL VALUE OF ROOT
   -1.0


** OUTPUT **
   IERR =    0
   PRACTICAL NUMBER OF FUNCTION EVALUATIONS =  36
   PRACTICAL NUMBER OF DERIVATIVE EVALUATIONS =   1
   ROOT
    0.1926185353D+01
```

### 4.3.2 DLNRIS, RLNRIS
### A Root of a Real Function (Initial Value Specified; Derivative Definition Not Required)

(1) **Function**

DLNRIS or RLNRIS obtains a root of nonlinear equation from an initial value, when the derivative of the nonlinear equation is not given.

(2) **Usage**

Double precision:
  CALL DLNRIS (F, X, ER, NEV, IERR)
Single precision:
  CALL RLNRIS (F, X, ER, NEV, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | – | Input | Name F(X) of function subprogram that defines the equation |
| 2 | X | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Input | Initial value of root |
| | | | | Output | Root |
| 3 | ER | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Input | Required precision (Default value: (Unit for determining error)× 64) |
| 4 | NEV | I | 1 | Input | Maximum number of function evaluations (Default value: 100) |
| | | | | Output | Actual number of function evaluations |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NEV > 0 (except when 0 is entered in order to set NEV to the default value)

(b) ER ≥ Unit for determining error
(except when 0 is entered in order to set ER to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a), (b) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 4000 | The root could not be obtained. | Processing is aborted. |
| 5000 | The root could not be obtained before the maximum number of function evaluations or maximum number of derivative evaluations was reached. | The value of X at that time is returned. |

(6) **Notes**

(a) The actual name in the first argument DF must be declared using an EXTERNAL statement in the user program, and a function subprograms having the actual name specified for F and DF must be created. (See Section 4.1.1) This function subprogram (in double-precision) should be created as follows.

Function subprogram creation method

```
REAL(8) FUNCTION F(X)
REAL(8) X
F =~
RETURN
END
```

(b) If a default value is shown for an argument in the “Contents” column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(c) The subroutine algorithm has global convergency so that the problem can be solved even if the initial value is distant from the root.

(d) Convergence is considered to have occurred when the following relationships hold:

$$| \text{(Root update amount)} | < \text{ER} \times \max(1, | \text{Root value} |)$$

and

$$| \text{Function value} | < \text{ER} + 64 \times \text{(Unit for determining error)} \times | \text{Root value} |$$

(7) **Example**

(a) Problem

Obtain one root of the following equation:

$$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$$

(b) Input data

Function subprogram name corresponding to function $f(x)$: FLNRIS

X=0.0, ER=0.0 and NEV=0.

(c) Main program

```
      PROGRAM BLNRIS
! *** EXAMPLE OF DLNRIS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FLNRIS
      INTEGER NEV,IERR
!
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1000)
      WRITE(6,1500) NEV
      WRITE(6,2000) ER
      WRITE(6,2500)
      WRITE(6,3000) X
      CALL DLNRIS(FLNRIS,X,ER,NEV,IERR)
      WRITE(6,3500)
      WRITE(6,4000) IERR
      WRITE(6,4500) NEV
      WRITE(6,5000)
      WRITE(6,5500) X
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLNRIS ***',/,&
      7X,'* NONLINEAR EQUATION *',/,&
      9X,'EXP(0.01*X)+3-(X-231)*(X-597)=0',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTION EVALUATIONS = ',I3)
 2000 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 2500 FORMAT(9X,'INITIAL VALUE OF ROOT')
 3000 FORMAT(9X,F3.1)
 3500 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 4500 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTION EVALUATIONS = ',I3)
 5000 FORMAT(9X,'ROOT')
 5500 FORMAT(9X,D17.10)
      END


      REAL(8) FUNCTION FLNRIS(X)
      REAL(8) X
!
      FLNRIS = EXP((0.01D0)*X)+(3.0D0)-(X-(231.0D0))*(X-(597.0D0))
      RETURN
      END
```

(d) Output results

```
     *** DLNRIS ***
       * NONLINEAR EQUATION *
         EXP(0.01*X)+3-(X-231)*(X-597)=0
      ** INPUT **
         MAXIMUM NUMBER OF FUNCTION EVALUATIONS =   0
         REQUIRED ACCURACY = 0.0D+00
         INITIAL VALUE OF ROOT
         0.0


      ** OUTPUT **
         IERR =    0
         PRACTICAL NUMBER OF FUNCTION EVALUATIONS =  10
         ROOT
          0.2309642908D+03
```

### 4.3.3 DLNRSS, RLNRSS A Root of a Real Function (Interval Specified; Derivative Definition Not Required)

(1) **Function**

DLNRSS or RLNRSS obtains a root of a nonlinear equation within an interval in which the function sign changes.

(2) **Usage**

Double precision:

CALL DLNRSS (F, AX, BX, ER, X, IERR)

Single precision:

CALL RLNRSS (F, AX, BX, ER, X, 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 | F | {D, R} | - | Input | Name of function subprogram F(X) that defines the equation |
| 2 | AX | {D, R} | 1 | Input | Left end of interval that encloses root |
| 3 | BX | {D, R} | 1 | Input | Right end of interval that encloses root |
| 4 | ER | {D, R} | 1 | Input | Required precision (Default value:(Unit for determining error) ×64) |
| 5 | X | {D, R} | 1 | Output | Root |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(b) AX $\neq$ BX

(c) F(AX) $\times$ F(BX) $\leq$ 0.0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a) was not satisfied. | Processing is performed with the default value set for ER. |
| 3000 | Restriction (b) or (c) was not satisfied. | Processing is aborted. |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. (See Section 4.1.1) This function subprogram (in double-precision) should be created as follows.

```
REAL(8) FUNCTION F(X)
REAL(8) X
F =~
RETURN
END
```

(b) If 0.0 is entered for argument ER, then the default value will be set.

(c) The problem can be solved even if AX is the right end of the interval and BX is the left end.

(d) Convergence is considered to have occurred when the following relationship holds:
$|$ Interval that encloses root $|<$ ER $+ 2\times$ (Unit for determining error) $\times$Root value

(7) **Example**

(a) Problem
Obtain one root of the following equation:

$$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$$

(b) Input data
Function subprogram name corresponding to function $f(x)$: FLNRSS
AX $= -200.0$, BX=240.0 and ER=0.0.

(c) Main program

```
      PROGRAM BLNRSS
! *** EXAMPLE OF DLNRSS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FLNRSS
      INTEGER IERR
!
      READ(5,*) ER
      READ(5,*) AX
      READ(5,*) BX
      WRITE(6,1000)
      WRITE(6,2000) ER
      WRITE(6,3000)
      WRITE(6,4000) AX,BX
      CALL DLNRSS(FLNRSS,AX,BX,ER,X,IERR)
      WRITE(6,5000)
      WRITE(6,6000) IERR
      WRITE(6,7000)
      WRITE(6,8000) X
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLNRSS ***',/,&
      7X,'* NONLINEAR EQUATION *',/,&
      9X,'EXP(0.01*X)+3-(X-231)*(X-597)=0',/,&
      6X,'** INPUT **')
 2000 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
```

```
 3000 FORMAT(9X,'INTERVAL IN WHICH ROOTS EXIST')
 4000 FORMAT(9X,'(',F6.1,2X,F6.1,')')
 5000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 6000 FORMAT(9X,'IERR = ',I4)
 7000 FORMAT(9X,'ROOT')
 8000 FORMAT(9X,D17.10)
      END

      REAL(8) FUNCTION FLNRSS(X)
      REAL(8) X
!
      FLNRSS = EXP((0.01D0)*X)+(3.0D0)-(X-(231.0D0))*(X-(597.0D0))
      RETURN
      END
```

(d) Output results

```
   *** DLNRSS ***
     * NONLINEAR EQUATION *
       EXP(0.01*X)+3-(X-231)*(X-597)=0
    ** INPUT **
       REQUIRED ACCURACY = 0.0D+00
       INTERVAL IN WHICH ROOTS EXIST
       (-200.0   240.0)


    ** OUTPUT **
       IERR =    0
       ROOT
        0.2309642908D+03
```

## 4.3.4 DLNRSA, RLNRSA All Roots of a Real Function (Interval Specified; Derivative Definition Not Required)

(1) **Function**

DLNRSA or RLNRSA obtains all roots of a nonlinear equation within an interval.

(2) **Usage**

Double precision:

CALL DLNRSA (F, AX, BX, ER, NEV, X, M, IERR)

Single precision:

CALL RLNRSA (F, AX, BX, ER, NEV, X, M, 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 | F | {D / R} | - | Input | Name of function subprogram F(X) that defines the equation |
| 2 | AX | {D / R} | 1 | Input | Left end of interval that encloses root |
| 3 | BX | {D / R} | 1 | Input | Right end of interval that encloses root |
| 4 | ER | {D / R} | 1 | Input | Required precision (Default value:(Unit for determining error) ×64) |
| 5 | NEV | I | 1 | Input | Maximum number of function evaluations (Default value: 100) |
| | | | | Output | Actual number of function evaluation |
| 6 | X | {D / R} | M | Output | Root |
| 7 | M | I | 1 | Input | Maximum number of roots to be obtained |
| | | | | Output | Number of roots obtained |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(b) NEV $> 0$
(except when 0 is entered in order to set NEV to the default value)

(c) M $> 0$

(d) AX $\neq$ BX

(e) AX < BX

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1200 | Restriction (e) was not satisfied. | Processing is performed with AX and BX being switched. |
| 1500 | Restriction (a) or (b) was not satisfied. | Processing is performed with the default value set for ER or NEV. |
| 2000 | The maximum number of function evaluations was reached. | The M roots that were obtained are output, and processing is aborted. |
| 2500 | The number of roots exceeded the maximum number of roots M. | |
| 3000 | Restriction (c) or (d) was not satisfied. | Processing is aborted. |
| 4000 | The root could not be obtained. | |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. (See Section 4.1.1) This function subprogram (in double-precision) should be created as follows.

```
REAL(8) FUNCTION F(X)
REAL(8) X
F =∼
RETURN
END
```

(b) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(c) Convergence is considered to have occurred when the following relationships hold:

$$\{|\ \mathrm{Rootupdateamount}\ | < \mathrm{ER} \times \max(1, |\ \mathrm{Rootvalue}\ |)\}$$

and

$$\{|\ \mathrm{Functionvalue}\ | < \mathrm{ER} + 64 \times (\text{Unit for determining error}) \times |\ \mathrm{Rootvalue}\ |\}$$

(d) Since two roots that are closer together than

$$\max(2 \times (\text{Required precision}),$$
$$(|\ \mathrm{A}\ | + |\ \mathrm{B}\ |) \times (\text{Unit for determining error}),$$
$$\sqrt[3]{(\text{Unit for determining error})})$$

cannot be isolated, one of these roots may not be obtained.

(7) **Example**

(a) Problem

Obtain all roots of the following equation:

$$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$$

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FLNRSA

AX $= -200.0$, BX=800.0, ER=0.0, NEV=0 and M=15.

(c) Main program

```
      PROGRAM BLNRSA
! *** EXAMPLE OF DLNRSA ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FLNRSA
      INTEGER NEV,M,IERR,I
      DIMENSION X(10)
!
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) M
      READ(5,*) AX
      READ(5,*) BX
      WRITE(6,1000)
      WRITE(6,1500) NEV
      WRITE(6,2000) ER
      WRITE(6,2500) M
      WRITE(6,3000)
      WRITE(6,3500) AX,BX
      CALL DLNRSA(FLNRSA,AX,BX,ER,NEV,X,M,IERR)
      WRITE(6,4000)
      WRITE(6,4500) IERR
      WRITE(6,5000) NEV
      WRITE(6,5500) M
      WRITE(6,6000)
      WRITE(6,6500) (X(I),I=1,M,1)
      STOP
!
 1000 FORMAT(' ',/,/,5X,'*** DLNRSA ***',/,&
      7X,'* NONLINEAR EQUATION *',/,&
      9X,'EXP(0.01*X)+3-(X-231)*(X-597)=0',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTION EVALUATIONS = ',I3)
 2000 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 2500 FORMAT(9X,'MAXIMUM NUMBER OF ROOTS = ',I2)
 3000 FORMAT(9X,'INTERVAL IN WHICH ROOTS EXIST')
 3500 FORMAT(9X,'(',F6.1,2X,F6.1,')')
 4000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4500 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTION EVALUATIONS = ',I3)
 5500 FORMAT(9X,'NUMBER OF ROOTS = ',I2)
 6000 FORMAT(9X,'ROOTS',/)
 6500 FORMAT(9X,D17.10)
      END


      REAL(8) FUNCTION FLNRSA(X)
      REAL(8) X
!
      FLNRSA = EXP((0.01D0)*X)+(3.0D0)-(X-(231.0D0))*(X-(597.0D0))
      RETURN
      END
```

(d) Output results

```
     *** DLNRSA ***
       * NONLINEAR EQUATION *
         EXP(0.01*X)+3-(X-231)*(X-597)=0
      ** INPUT **
         MAXIMUM NUMBER OF FUNCTION EVALUATIONS =   0
         REQUIRED ACCURACY = 0.0D+00
         MAXIMUM NUMBER OF ROOTS = 15
         INTERVAL IN WHICH ROOTS EXIST
         (-200.0   800.0)


      ** OUTPUT **
         IERR =    0
         PRACTICAL NUMBER OF FUNCTION EVALUATIONS =  98
         NUMBER OF ROOTS =  2
         ROOTS

          0.2309642908D+03
          0.5980863437D+03
```

### 4.3.5 ZLNCIS, CLNCIS A Root of a Complex Function (Initial Value Specified; Derivative Definition Not Required)

(1) **Function**

ZLNCIS or CLNCIS obtains a root of a complex nonlinear equation from an initial value, when the derivative of the nonlinear equation is not given.

(2) **Usage**

Double precision:

CALL ZLNCIS (F, CX, ER, NEV, IERR)

Single precision:

CALL CLNCIS (F, CX, ER, NEV, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{array}{c}Z\\C\end{array}\right\}$ | - | Input | Function name of complex function subprogram F(X) that defines the equation |
| 2 | CX | $\left\{\begin{array}{c}Z\\C\end{array}\right\}$ | 1 | Input | Initial value of root |
| | | | | Output | Root |
| 3 | ER | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Input | Required precision (Default value:(Unit for determining error) $\times 64$) |
| 4 | NEV | I | 1 | Input | Maximum number of function evaluations (Default value: 100) |
| | | | | Output | Actual number of function evaluation |
| 5 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NEV $> 0$
(except when 0 is entered in order to set NEV to the default value)

(b) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a) or (b) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 4000 | A zero-division error occurred. | Processing is aborted. |
| 5000 | The root could not be obtained before the maximum number of function evaluations was reached. | The value of CX at that time is returned. |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. (See Section 4.1.1) This function subprogram (in double-precision) should be created as follows.

COMPLEX(8) FUNCTION F(X)
COMPLEX(8) X
F =∼
RETURN
END

(b) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(c) Convergence is considered to have occurred when the following relationships hold:

$$\{|\ \mathrm{Rootupdateamount}\ | < \mathrm{ER} \times \max(1, |\ \mathrm{Rootvalue}\ |)\}$$

and

$$\{|\ \mathrm{Functionvalue}\ | < \mathrm{ER} + 64 \times (\text{Unit for determining error}) \times |\ \mathrm{Rootvalue}\ |\}$$

(7) **Example**

(a) Problem
Obtain one root of the following equation:

$$f = x^{10} - 1 = 0$$

(b) Input data
Function subprogram name corresponding to integrand $f(x)$: FLNCIS
CX = (1.0, −1.0), ER = 1.0D − 10 and NEV = 100.

(c) Main program

```
      PROGRAM ALNCIS
! *** EXAMPLE OF ZLNCIS ***
      IMPLICIT COMPLEX(8) (A-D,F-H,O-Z)
      EXTERNAL FLNCIS
      INTEGER NEV,IERR
      REAL(8) ER
!
      READ(5,*) NEV
```

```
      READ(5,*) ER
      READ(5,*) CX
      WRITE(6,1000)
      WRITE(6,1500) NEV
      WRITE(6,2000) ER
      WRITE(6,2500)
      WRITE(6,3000) CX
      CALL ZLNCIS(FLNCIS,CX,ER,NEV,IERR)
      WRITE(6,3500)
      WRITE(6,4000) IERR
      WRITE(6,4500) NEV
      WRITE(6,5000)
      WRITE(6,5500) CX
      STOP
!
 1000 FORMAT(' ',/,/,5X,'*** ZLNCIS ***',/,&
      7X,'* NONLINEAR EQUATION *',/,&
      9X,'Z**10-1=0',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTION EVALUATIONS = ',I3)
 2000 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 2500 FORMAT(9X,'INITIAL VALUE OF ROOT',/,&
      9X,'REAL PART',5X,'IMAGINARY PART')
 3000 FORMAT(9X,F3.1,11X,F3.1)
 3500 FORMAT(' ',/,/,9X,'** OUTPUT **')
 4000 FORMAT(9X,'IERR = ',I4)
 4500 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTION EVALUATIONS = ',I3)
 5000 FORMAT(9X,'ROOT',/,&
      9X,'REAL PART',13X,'IMAGINARY PART')
 5500 FORMAT(9X,D17.10,5X,D17.10)
      END

      COMPLEX(8) FUNCTION FLNCIS(X)
!
      COMPLEX(8) X
!
      FLNCIS = X**10-(1.0D0,0.0D0)
      RETURN
      END
```

(d) Output results

```
     *** ZLNCIS ***
       * NONLINEAR EQUATION *
         Z**10-1=0
      ** INPUT **
         MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 100
         REQUIRED ACCURACY = 0.1D-09
         INITIAL VALUE OF ROOT
         REAL PART     IMAGINARY PART
         1.0           1.0

         ** OUTPUT **
         IERR =    0
         PRACTICAL NUMBER OF FUNCTION EVALUATIONS =  11
         ROOT
         REAL PART             IMAGINARY PART
          0.8090169944D+00      0.5877852523D+00
```

## 4.4 SETS OF SIMULTANEOUS NONLINEAR EQUATIONS

### 4.4.1 DLSRDS, RLSRDS
### A Root of a Set of Simultaneous Nonlinear Functions (Jacobian Matrix Optional)

(1) **Function**

DLSRDS or RLSRDS obtains a root of a set of simultaneous nonlinear equations, either when the Jacobian matrix is defined or when it is not defined.

(2) **Usage**

Double precision:

CALL DLSRDS (SUB, SUBJ, X, N, ER, NEV, ISW, IWK, WK, DWK, IERR)

Single precision:

CALL RLSRDS (SUB, SUBJ, X, N, ER, NEV, ISW, IWK, WK, DWK, 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 | SUB | – | – | Input | Name of subroutine SUB(X, N, F) that defines the N nonlinear equations. |
| 2 | SUBJ | – | – | Input | Name of subroutine SUBJ(X, N, A) that defines the Jacobian matrix. |
| 3 | X | {D, R} | N | Input | Initial value of root |
| | | | | Output | Root |
| 4 | N | I | 1 | Input | Number of simultaneous nonlinear equations |
| 5 | ER | {D, R} | 1 | Input | Required precision<br>(Default value: (Unit for determining error) × 64) |
| 6 | NEV | I | 2 | Input | NEV(1): Maximum number of function evaluations (Default value: 100×N)<br>NEV(2): Maximum number of Jacobian matrix evaluations (Default value: 100×N) |
| | | | | Output | NEV(1): Actual number of function evaluations<br>NEV(2): Actual number of Jacobian matrix evaluations |
| 7 | ISW | I | 1 | Input | Input switch<br>ISW = 0: Jacobian matrix is automatically calculated<br>ISW ≠ 0: User-defined subroutine that creates Jacobian matrix is used<br>(See Notes (b)) |
| 8 | IWK | I | N | Work | Work area |
| 9 | WK | {D, R} | See Contents | Work | Work area<br>**Size**: N × (N + 3) |
| 10 | DWK | D | See Contents | Work | Work area. This array is double precision real.<br>**Size**: N × (2 × N + 2) |
| 11 | IERR | I | 1 | Output | Error indicator |

## (4) Restrictions

(a) NEV(1) > 0
(except when 0 is entered in order to set NEV(1) to the default value)

(b) NEV(2) > 0
(except when 0 is entered in order to set NEV(2) to the default value)

(c) ER ≥ Unit for determining error
(except when 0 is entered in order to set ER to the default value)

(d) N > 0

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a), (b) or (c) was not satisfied. | Processing is performed with the default value set for NEV(1), NEV(2) or ER. |
| 3000 | Restriction (d) was not satisfied. | Processing is aborted. |
| 4000 | An error occurred when solving a set of simultaneous linear equations. | |
| 4500 | The solution could not be updated because nonlinearity was too strong. | |
| 5000 | The solution could not be obtained before the maximum number of function evaluations or maximum number of Jacobian matrix evaluations was reached. | The value of X at that time is returned. |

(6) **Notes**

(a) The actual names in the first argument SUB and second argument SUBJ must be declared using an EXTERNAL statement in the user program, and subroutine subprograms having the actual names specified for SUB and SUBJ must be created. (See Section 4.1.1) These subroutine subprograms should be created as follows. If the set of simultaneous nonlinear equations is as follows:

$$\begin{cases} f_1(x_1, \cdots, x_N) = 0 \\ \vdots \\ f_N(x_1, \cdots, x_N) = 0 \end{cases}$$

the subroutineSUB is written as follows for single-precision calculations:

```
SUBROUTINE SUB(X, N, F)
REAL X(N), F(N)
F(1) = f1(x1, ···, xN)
⋮
F(N) = fN(x1, ···, xN)
RETURN
END
```

If the Jacobian matrix $A$ is defined as follows:

$$\mathrm{A} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix} = \begin{bmatrix} \mathrm{A}(1,1), & \cdots, & \mathrm{A}(1,\mathrm{N}) \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \mathrm{A}(\mathrm{N},1), & \cdots, & \mathrm{A}(\mathrm{N},\mathrm{N}) \end{bmatrix},$$

the subroutine SUBJ is written as follows for single-precision calculations:

```
SUBROUTINE SUBJ(X, N, A)
REAL X(N), A(N, *)
```

```
A(1,1) = ∂f1/∂x1
   ⋮
A(N,1) = ∂fN/∂x1
   ⋮
A(1,N) = ∂f1/∂xN
   ⋮
A(N,N) = ∂fN/∂xN
RETURN
END
```

Example: Assume that the set of simultaneous nonlinear equations is as follows:

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1x_2 + x_2x_3 + x_3x_1 = 0 \\ x_1x_2x_3 = 0 \end{cases}$$

and the Jacobian matrix A is as follows:

$$\mathrm{A} = \begin{bmatrix} 1, & 1, & 1 \\ x_2 + x_3, & x_1 + x_3, & x_1 + x_2 \\ x_2x_3, & x_1x_3, & x_1x_2 \end{bmatrix}$$

Then the subroutines are written as follows:

```
SUBROUTINE SUB(X, N, F)
REAL X(N), F(N)
F(1) = X(1) + X(2) + X(3)
F(2) = X(1) * X(2) + X(2) * X(3) + X(3) * X(1)
F(3) = X(1) * X(2) * X(3)
RETURN
END

SUBROUTINE SUBJ(X, N, A)
REAL X(N), A(N, *)
A(1,1) = 1.0
A(1,2) = 1.0
A(1,3) = 1.0
A(2,1) = X(2) + X(3)
A(2,2) = X(1) + X(3)
A(2,3) = X(1) + X(2)
A(3,1) = X(2) * X(3)
A(3,2) = X(1) * X(3)
```

A(3,3) = X(1) * X(2)
RETURN
END

(b) If ISW = 0, then the SUBJ argument is a dummy argument and no corresponding subroutine is required. Also, NEV(2) need not be entered.
If ISW $\neq$ 0, then a subroutine having the actual name specified for SUBJ must be created and NEV(2) must be entered.

(c) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) Convergence is considered to have occurred for all roots or all function values when the following relationships hold:

$$\{\| \text{Rootupdateamount} \|_\infty < \text{ER} \times \max(1, \| \text{Rootvalue} \|_\infty)\}$$

and

$$\{\| \text{Functionvalue} \|_\infty < \text{ER} + 64 \times \varepsilon \times \| \text{Rootvalue} \|_\infty\}$$

(7) **Example**

(a) Problem
Solve the following set of simultaneous nonlinear equations:

$$\begin{cases} 10(\mathrm{x}_2 - \mathrm{x}_1^2) = 0 \\ 1 - \mathrm{x}_1 = 0 \end{cases}$$

(b) Input data
Subroutine name defining function $\boldsymbol{f}(x)$ corresponding to nonlinear equations: FLSRDS
X(1) = −1.2, X(2) = 1.0, N = 2, ER = 0.0, NEV(1) = 200, NEV(2) = 200 and ISW = 0.

(c) Main program

```
      PROGRAM BLSRDS
! *** EXAMPLE OF DLSRDS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      IMPLICIT INTEGER (I-N)
      EXTERNAL FLSRDS
      DIMENSION NEV(2),IWK(2),X(2),WK(10),DWK(12)
!
      READ(5,*) (NEV(I),I=1,2,1)
      READ(5,*) N
      READ(5,*) ER
      READ(5,*) (X(I),I=1,N,1)
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1500) NEV(1)
      WRITE(6,2000) NEV(2)
      WRITE(6,2500) N
      WRITE(6,3000) ER
      WRITE(6,3500)
      WRITE(6,4000) (X(I),I=1,N,1)
      CALL DLSRDS(FLSRDS,NONSUB,X,N,ER,NEV,ISW,IWK,WK,DWK,IERR)
      WRITE(6,4500)
      WRITE(6,5000) IERR
      WRITE(6,5500) NEV(1)
      WRITE(6,6000) NEV(2)
      WRITE(6,7000)
      WRITE(6,7500) (X(I),I=1,N,1)
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLSRDS ***',/,&
      7X,'* SYSTEM OF NONLINEAR EQUATIONS *',/,&
      9X,'10*(X2-X1*X1)=0',/,&
      9X,'1-X1         =0',/,&
      6X,'** INPUT **')
```

```
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTIONS EVALUATIONS = ',I3)
 2000 FORMAT(9X,'MAXIMUM NUMBER OF JACOBIAN EVALUATIONS = ',I3)
 2500 FORMAT(9X,'NUMBER OF NONLINEAR EQUATIONS = ',I2)
 3000 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 3500 FORMAT(9X,'INITIAL VALUE OF ROOT',/)
 4000 FORMAT(9X,F4.1)
 4500 FORMAT(' ',/,/,6X,'** OUTPUT **')
 5000 FORMAT(9X,'IERR = ',I4)
 5500 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTIONS EVALUATIONS = ',I3)
 6000 FORMAT(9X,'PRACTICAL NUMBER OF JACOBIAN EVALUATIONS = ',I3)
 7000 FORMAT(9X,'ROOT',/)
 7500 FORMAT(9X,D17.10)
      END

      SUBROUTINE FLSRDS(X,N,F)
      INTEGER N
      REAL(8) X,F
      DIMENSION X(N),F(N)
!
      F(1) = 10.0D0*(X(2)-X(1)*X(1))
      F(2) = 1.0D0-X(1)
      RETURN
      END
```

(d) Output results

```
    *** DLSRDS ***
      * SYSTEM OF NONLINEAR EQUATIONS *
        10*(X2-X1*X1)=0
        1-X1         =0
     ** INPUT **
        MAXIMUM NUMBER OF FUNCTIONS EVALUATIONS = 200
        MAXIMUM NUMBER OF JACOBIAN EVALUATIONS = 200
        NUMBER OF NONLINEAR EQUATIONS =  2
        REQUIRED ACCURACY = 0.0D+00
        INITIAL VALUE OF ROOT

        -1.2
         1.0


     ** OUTPUT **
        IERR =    0
        PRACTICAL NUMBER OF FUNCTIONS EVALUATIONS =  66
        PRACTICAL NUMBER OF JACOBIAN EVALUATIONS =   0
        ROOT

         0.1000000000D+01
         0.1000000000D+01
```

## 4.4.2 DLSRIS, RLSRIS A Root of a Set of Simultaneous Nonlinear Functions (Jacobian Matrix Definition Not Required)

(1) **Function**

DLSRIS or RLSRIS obtains a root of a set of simultaneous nonlinear equations, when the Jacobian matrix is not defined.

(2) **Usage**

Double precision:
CALL DLSRIS (SUB, X, N, ER, NEV, IWK, WK, IERR)
Single precision:
CALL RLSRIS (SUB, X, N, ER, NEV, IWK, WK, 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 | SUB | – | – | Input | Name of subroutine SUB(X, N, F) that defines the N nonlinear equations. |
| 2 | X | {D/R} | N | Input | Initial value of root |
| | | | | Output | Root |
| 3 | N | I | 1 | Input | Number of simultaneous nonlinear equations |
| 4 | ER | {D/R} | 1 | Input | Required precision (Default value: (Unit for determining error)× 64) |
| 5 | NEV | I | 1 | Input | Maximum number of function evaluations (Default value: 100×N) |
| | | | | Output | Actual number of function evaluations |
| 6 | IWK | I | $2 \times N$ | Work | Work area |
| 7 | WK | {D/R} | See Contents | Work | Work area **Size**: $N \times (3 \times N + 7)$ |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) NEV $> 0$
(except when 0 is entered in order to set NEV to the default value)

(b) ER $\geq$ Unit for determining error
(except when 0 is entered in order to set ER to the default value)

(c) N $> 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a) or (b) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 3000 | Restriction (c) was not satisfied. | Processing is aborted. |
| 4000+ i | The determinant of the Jacobian matrix is 0. That is, $i$ indicates the first $(i, i)$ component of the Jacobian matrix to be 0. | |
| 4500 | A zero-division error occurred during the calculation of the inverse matrix of the Jacobian matrix or during the calculation of the step size. | |
| 5000 | The solution could not be obtained before the maximum number of function evaluations was reached. | The value of X at that time is returned. |

(6) **Notes**

(a) The actual names in the first argument SUB must be declared using an EXTERNAL statement in the user program, and subroutine having the actual names specified for SUB must be created. (See Section 4.1.1) The subroutine should be created as follows. If the set of simultaneous nonlinear equations is as follows:

$$\begin{cases} f_1(x_1, \cdots, x_N) = 0 \\ \vdots \\ f_N(x_1, \cdots, x_N) = 0 \end{cases}$$

the subroutineSUB is written as follows for single-precision calculations:

```
SUBROUTINE SUB(X, N, F)
REAL X(N), F(N)
F(1) = f1(x1, ..., xN)

      :

F(N) = fN(x1, ..., xN)
RETURN
END
```

(b) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(c) Although a solution can be obtained even if scaling is not performed for each of the equations in the set of simultaneous nonlinear equations, scaling should be performed to increase efficiency.

(d) Convergence is considered to have occurred for all roots or all function values when the following relationships hold:

$$\{\| \text{Rootupdateamount} \|_\infty < \text{ER} \times \max(1, \| \text{Rootvalue} \|_\infty)\}$$

and

$$\{\| \text{Functionvalue} \|_\infty < \text{ER} + 64 \times \varepsilon \times \| \text{Rootvalue} \|_\infty\}$$

(7) **Example**

(a) Problem

Solve the following set of simultaneous nonlinear equations:

$$\begin{cases} 10(\mathrm{x}_2 - \mathrm{x}_1^2) = 0 \\ 1 - \mathrm{x}_1 = 0 \end{cases}$$

(b) Input data

Subroutine name defining function $\boldsymbol{f}(x)$ corresponding to nonlinear equations: FLSRIS

X(1) = −1.2, X(2)=1.0, N=2, ER=0.0 and NEV=200.

(c) Main program

```
      PROGRAM BLSRIS
! *** EXAMPLE OF DLSRIS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      IMPLICIT INTEGER (I-N)
      EXTERNAL FLSRIS
      DIMENSION IWK(4),X(2),WK(26)
!
      READ(5,*) NEV
      READ(5,*) N
      READ(5,*) ER
      READ(5,*) (X(I),I=1,N,1)
      WRITE(6,1000)
      WRITE(6,1500) NEV
      WRITE(6,2000) N
      WRITE(6,2500) ER
      WRITE(6,3000)
      WRITE(6,3500) (X(I),I=1,N,1)
      CALL DLSRIS(FLSRIS,X,N,ER,NEV,IWK,WK,IERR)
      WRITE(6,4000)
      WRITE(6,4500) IERR
      WRITE(6,5000) NEV
      WRITE(6,5500)
      WRITE(6,6000) (X(I),I=1,N,1)
      STOP
!
 1000 FORMAT(' ',/,5X,'*** DLSRIS ***',/,&
      7X,'* SYSTEM OF NONLINEAR EQUATIONS *',/,&
      9X,'10*(X2-X1*X1)=0',/,&
      9X,'1-X1          =0',/,&
      6X,'** INPUT **')
 1500 FORMAT(9X,'MAXIMUM NUMBER OF FUNCTIONS EVALUATIONS = ',I3)
 2000 FORMAT(9X,'NUMBER OF NONLINEAR EQUATIONS = ',I2)
 2500 FORMAT(9X,'REQUIRED ACCURACY = ',D7.1)
 3000 FORMAT(9X,'INITIAL VALUE OF ROOT',/)
 3500 FORMAT(9X,F4.1)
 4000 FORMAT(' ',/,/,6X,'** OUTPUT **')
 4500 FORMAT(9X,'IERR = ',I4)
 5000 FORMAT(9X,'PRACTICAL NUMBER OF FUNCTIONS EVALUATIONS = ',I3)
 5500 FORMAT(9X,'ROOT',/)
 6000 FORMAT(9X,D17.10)
      END

      SUBROUTINE FLSRIS(X,N,F)
      INTEGER N
      REAL(8) X,F
      DIMENSION X(N),F(N)
!
      F(1) = 10.0D0*(X(2)-X(1)*X(1))
      F(2) = 1.0D0-X(1)
      RETURN
      END
```

(d) Output results

```
*** DLSRIS ***
  * SYSTEM OF NONLINEAR EQUATIONS *
    10*(X2-X1*X1)=0
    1-X1         =0
 ** INPUT **
    MAXIMUM NUMBER OF FUNCTIONS EVALUATIONS = 200
    NUMBER OF NONLINEAR EQUATIONS =  2
    REQUIRED ACCURACY = 0.0D+00
    INITIAL VALUE OF ROOT

    -1.2
     1.0


 ** OUTPUT **
    IERR =    0
    PRACTICAL NUMBER OF FUNCTIONS EVALUATIONS =  34
    ROOT

     0.1000000000D+01
     0.1000000000D+01
```

Chapter 5

# EXTREMUM PROBLEMS AND OPTIMIZATION

## 5.1 INTRODUCTION

This chapter explains subroutines for performing minimization of a function of one variable without constraints, minimization of a function of many variables without constraints, minimization of the sum of the squares of a function without constraints, minimization of a constrained function of one variable, minimization of a constrained linear function of many variables (linear programming, mixed 0-1 programming minimal-cost flow problem, project scheduling problem and transportation problem), minimization of a quadratic function of several variables (quadratic programming), minimization of a constrained function of several variables (nonlinear programming) and distance minimization on a graph (shortest path problem).
This library provides the following subroutine for minimizing a function of one variable with no constraints.

(1) Minimization of a function of one variable

This subroutine searches for the minimum value of a function of one variable by starting from a given initial point.

This library provides the following subroutines for minimizing a function of many variables with no constraints.

(1) Minimization of a function of many variables (derivative definition unnecessary)

(2) Minimization of a function of many variables (derivative definition required)

These subroutines search for the minimum value of a function of many variables by starting from a given initial point. If the user cannot provide a subroutine that calculates the gradient vector of the function, then the user can use the subroutine for which the derivative definition is unnecessary. If the user can provide a subroutine that calculates the gradient vector of the function, then the subroutine for which the derivative definition is required will obtain better results.

This library provides the following subroutine for minimizing the sum of the squares of a function with no constraints.

(1) Nonlinear least squares method (derivative function unnecessary)

This subroutine searches for the minimum value of the sum of the squares of a function of $m$ variables, by starting from a given initial point. The user need not provide a subroutine that calculates the Jacobian matrix of the function.

This library provides the following subroutine for minimizing a function of one variable with constraints.

(1) Minimization of a function of one variable (interval specified)

This subroutine searches for the minimum value of a function of one variable within a given interval.

This library provides the following subroutines for minimizing a constrained linear function of several variables.

(1) Minimization of a constrained linear function of several variables (linear constraints)

(2) Minimization of a constrained linear function of several variables including 0-1 variables (linear constraints)

(3) Minimization of cost for flow in a network

(4) Minimization of cost for project scheduling

(5) Minimization of cost for transportation from supply place to demand place

These subroutines searches for the minimum value of a linear function of several variables within the given linear constraints (Linear programming, mixed 0-1 programming and minimal-cost flow problem).

This library provides the following subroutines for minimizing a quadratic function of several variables.

(1) Minimization of a constrained convex quadratic function of several variables (linear constraints)

(2) Minimization of a constrained generalized convex quadratic function of several variables (linear constraints)

(3) Minimization of an unconstrained 0-1 quadratic function of several variables

These subroutines search for the minimum value of a quadratic function of several variables within the given linear constraints (Quadratic programming).

This library provides the following subroutine for minimizing a constrained function of several variables.

(1) Minimization of a constrained function of several variables (nonlinear constraints)

This subroutine searches for the minimum value of a function of several variables within the given nonlinear constraints (Nonlinear programming).

This library provides the following subroutines for distance minimization on a graph.

(1) Distance minimization for a given node to the other nodes on a graph

(2) Distance minimization for all sets of two nodes on a graph

(3) Distance minimization for two nodes on a graph

These subroutines search minimum distance for the nodes on a graph. (Shortest path problem).

### 5.1.1 Notes

(1) The search starting point should be located as close as possible to exact solution.

(2) An appropriate value for the required precision is on the order of the square root of the unit for determining error.

(3) When minimizing a function of many variables, scaling should be performed so that the contribution to the function value from each of the variables is on the same order. For example, if the problem is similar to $f(x) = 10000x_1^2 + x_2^3$, then better results will be obtained by transforming the variables according to $y_1 = 100x_1, y_2 = x_2$ so that the problem becomes $h(y) = y_1^2 + y_2^3$.

### 5.1.2 Algorithms Used

#### 5.1.2.1 Minimization of a function of one variable

This algorithm uses a combination of the golden section search method and successive parabolic interpolation. The search begins with a golden section search and switches to successive parabolic interpolation when possible. This algorithm is based on algorithms created by Brent (1973).

(1) Golden section search
The algorithm for reducing the interval according to a golden section is described below.
Assume that the interval to be reduced is $[a, b]$ and that the function value $f(x)$ is calculated at a single point $x$ within the interval $(a < x < b)$. If $c = \dfrac{a+b}{2}$ is the midpoint of the interval, then for $x < c$, the function value $f(u)$is calculated at the point $u = x + r(b-x)$ where $r = \dfrac{3-\sqrt{5}}{2}$. If $f(x) < f(u)$, then $b = u$ is set; otherwise, $a = x$ and $x = u$ are set. For $x \geq c$, a similar procedure is followed. In this way, since the previously calculated function value can be used, the function should be calculated once for each reduction of the interval. If $x = a + r(b - a)$ is taken initially, then the interval will continue to be reduced at a fixed rate of $1 - r$.
This algorithm converges with order 1.

(2) Successive parabolic interpolation
The algorithm for reducing the interval according to successive parabolic interpolation is described below.
Assume that the interval to be reduced is $[a, b]$. If function values have been calculated at three points $v, w$ and $x$, then a parabola is created that passes through the three points $(v, f(v)), (w, f(w))$ and $(x, f(x))$, and the X coordinate of its vertex is assumed to be $u$. For $x < u$, if $f(x) < f(u)$, then $b = u$ is set; otherwise $a = x$ and $x = u$ are set. For $x \geq u$, a similar procedure is followed.
If $f''(x) > 0$ at the extremum and if the interpolation begins with a sufficiently good approximation, then it has been proven that the algorithm converges with order $1.324\cdots$.

(3) Switch from the golden section search to successive parabolic interpolation
When a new function value is calculated, $v, w$ and $x$ are updated to be the three points having the smallest function values among the function values calculated up to that time. If possible, the X coordinate $u$ of the vertex of the parabola described above is calculated, and if the following condition is satisfied, then the algorithm switches from the golden section search to sequential parabolic interpolation.

$$a < u < b$$
$$\mid x - u \mid < \frac{b-a}{2}$$

(4) Convergence decision
If the required precision is $e_r$, then convergence is determined according to the following condition:

$$\max(b - x, x - a) \leq 2e_r \max(1, \mid x \mid)$$

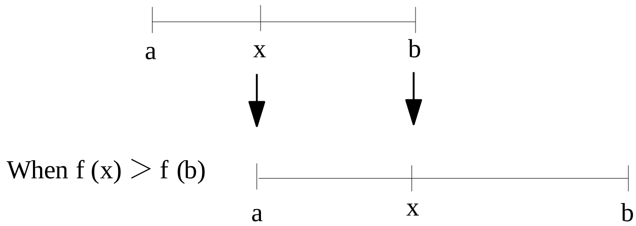
(5) Enclosure
Minimization without constraints is achieved by an enclosure operation that obtains an interval including the minimum point by beginning from an arbitrary starting point.
This algorithm is explained below.

(a) Obtain the direction of inclination at the starting point $a$.

(b) Let the enclosure point $b$ be at a distance $D = \max(\sqrt{\varepsilon} \mid x \mid, 1.0, 2e_r \mid x \mid)$ in the direction of inclination.

(c) If the function value at point $b$ is less than the function value at point $a$, then a decision is made that there is no minimum point, point $b$ is assumed to be point $x$, and a new point $b$ is set to enclose a wider interval extending to a width of $(3 - r)(x - a)$ from point $a$. In this way, the ratio $b - a : x - a$ becomes the golden section ratio, that is, $b - a : x - a = 1 : r$. If the function value at point $b$ is greater than the function value at point $a$, then since there is a minimum point in the interval $[a, b]$, the search will begin in this interval.

(d) If the function value at point $b$ is less than the function value at point $x$, then point $x$ is assumed to be point $a$, point $b$ is assumed to be point $x$, and a new point $b$ is taken at a distance of $(3 - r)(x - a)$ from point $a$. The actions described in iv. are performed repeatedly in this way.
If the function value at point $b$ is greater than the function value at point $x$, then the search will begin in the interval $[a, b]$. At this time, a function value obtained in the enclosure can be used by letting point $x$ be the point where the function value is calculated in the golden section search.




#### 5.1.2.2 Minimization of a function of many variables

Given a function $f(\boldsymbol{x})$ of $n$ variables, the problem is to obtain a value $\boldsymbol{x}$ for which $f(\boldsymbol{x})$ is a minimum. This value of $\boldsymbol{x}$ will be the solution of the equation $\nabla f(\boldsymbol{x}) = 0$. To solve this equation, the algorithm assumes that $\boldsymbol{x}_0$ is the starting point and sequentially corrects an approximate solution.
Now, the search proceeds to the solution value $\boldsymbol{x}$ by repeatedly attempting to obtain a next corrected solution $\boldsymbol{x}'$. The correction vector $\boldsymbol{d}$ of Newton's method is given by the following equation:

$$\boldsymbol{d} = -H\boldsymbol{g}$$

where $\boldsymbol{g}$ is the gradient vector $\boldsymbol{g} = \nabla^T f(\boldsymbol{x})$ and $H$ is the inverse matrix of the Hessian matrix $B = \nabla^2 f(\boldsymbol{x})$. (The notation $^T$ indicates the transpose.) Actually, $\boldsymbol{d}$ is taken to be the direction vector and the solution is corrected by performing a rectilinear search according to the following equation:

$$\boldsymbol{x}' = \boldsymbol{x} + \alpha\boldsymbol{d}$$

This method that sequentially updates an approximate solution without directly calculating $H$ is called a quasi-Newton method. The value $\alpha$ is determined by the rectilinear search described in (b).

(1) Calculating the inverse matrix $H$ of the Hessian matrix
Although various formulas for updating $H$ have been proposed, this subroutine algorithm uses the BFGS formula proposed by Broyden, Fletcher, Goldfarb, and Shanno.
The unit matrix $E$ is taken as the initial value of $H$, and this is then updated by the following formula:

$$H' = \left\{E - \frac{\delta\gamma^T}{\delta^T\gamma}\right\} H \left\{E - \frac{\gamma\delta^T}{\delta^T\gamma}\right\} + \frac{\delta\delta^T}{\delta^T\gamma}$$

where:

$$\begin{aligned}\delta &= \boldsymbol{x}' - \boldsymbol{x} \\ \gamma &= \nabla^T f(\boldsymbol{x}') - \nabla^T f(\boldsymbol{x})\end{aligned}$$

(2) Rectilinear search
Since a precise rectilinear search is not very efficient, Armijo's method is used. This method obtains the smallest nonnegative integer $m$ for which the following condition is satisfied:

$$f(\boldsymbol{x} + \beta^m \alpha_0 \boldsymbol{d}) - f(\boldsymbol{x}) \leq \beta^m \alpha_0 \mu \nabla f(\boldsymbol{x}) \boldsymbol{d}$$

and sets $\alpha = \beta^m \alpha_0$, where $\alpha_0 > 0, 0 < \beta < 1$ and $0 < \mu < 1$.

(3) Convergence decision
Convergence is determined according to the following condition, where $\boldsymbol{x}'$ is assumed to be the solution.

$$\begin{aligned}&\|\nabla^T f(\boldsymbol{x}')\|_\infty \leq \varepsilon \text{ or} \\ &(\|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq \varepsilon \text{ and a problem occurred during the previous modification}) \\ &\text{or} \\ &\|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq e_r \max(1, \|\boldsymbol{x}'\|_\infty) \quad \text{and} \quad \|\nabla^T f(\boldsymbol{x}')\|_\infty \leq 2e_r\end{aligned}$$

where $\varepsilon$ is the unit for determining error, $e_r$ is the required precision, and $\|\boldsymbol{x}\|_\infty = \max\limits_i \mid \boldsymbol{x}_i \mid$.

#### 5.1.2.3 Nonlinear least square method

Given $m$ functions $f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})$ of $n$ variables, the problem is to obtain a value $\boldsymbol{x}$ for which the sum of the squares of the functions:

$$S(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(\boldsymbol{x})^2 = \|\boldsymbol{f}(\boldsymbol{x})\|^2$$

is a minimum, where $\boldsymbol{f}(\boldsymbol{x})$ is a vector value function having components $f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})$ and $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T \boldsymbol{x}}$ (the notation $^T$ indicates the transpose).
To solve this problem, it is necessary to solve the equations $\dfrac{\partial S}{\partial \boldsymbol{x}} = 0$. To obtain a solution, a search is begun from an initial value $\boldsymbol{x}_0$, and Powell's hybrid method is used to sequentially correct the solution value.
The hybrid method blends a steepest descent solution and the Gauss-Newton solution of a linearized model depending on the nonlinearity of the function. An independence check always is performed for the correction vector so that approximate solutions do not end up being confined within a subspace. Also, function information is used to make corrections without directly calculating the Jacobian matrix.

(1) Correction vector $\Delta \boldsymbol{x}$ calculation
The model in which $\boldsymbol{f}$ is linearized is as follows:

$$S_L(\boldsymbol{x} + \Delta \boldsymbol{x}) = \|\boldsymbol{f}(\boldsymbol{x}) + A \Delta \boldsymbol{x}\|^2$$

where $A$ is the Jacobian matrix $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ of $\boldsymbol{f}$.
The steepest descent solution of this model $\Delta \boldsymbol{x}_S$ is given by:

$$\Delta \boldsymbol{x}_S = \frac{\|\boldsymbol{b}\|^2}{\|A\boldsymbol{b}\|^2} \boldsymbol{b}$$

where $\boldsymbol{b} = -A^T \boldsymbol{f}(\boldsymbol{x})$.
The Gauss-Newton solution $\Delta \boldsymbol{x}_G$ is obtained by solving the normal equation:

$$A^T A \Delta \boldsymbol{x} = \boldsymbol{b}$$

This subroutine uses the linear least squares method's QR decomposition algorithm to solve this equation. It has been provided that, in general, the following relationship holds:

$$\|\Delta \boldsymbol{x}_S\| \leq \|\Delta \boldsymbol{x}_G\|$$

These two solutions are blended as follows according to the step size $d$, which is determined depending to the nonlinearity of the function.

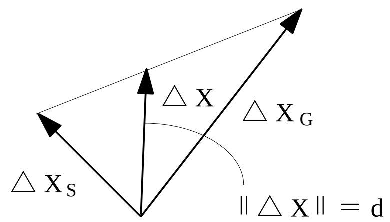
(a) If $d \leq \|\Delta \boldsymbol{x}_S\|$, then:

$$\Delta \boldsymbol{x} = d \frac{\Delta \boldsymbol{x}_S}{\|\Delta \boldsymbol{x}_S\|}$$

(b) If $\|\Delta \boldsymbol{x}_S\| < d < \|\Delta \boldsymbol{x}_G\|$, (See Figure 5−1) then:

$$\Delta \boldsymbol{x} = \alpha \Delta \boldsymbol{x}_S + \beta \Delta \boldsymbol{x}_G \quad (\alpha > 0, \beta > 0, \|\Delta \boldsymbol{x}\| = d)$$

Figure 5−1




(c) If $\|\Delta \boldsymbol{x}_G\| \leq d$

$$\Delta \boldsymbol{x} = \Delta \boldsymbol{x}_G$$

(2) Step size $d$ modification
The initial value of the step size is assumed to be $d = \|\Delta \boldsymbol{x}_S\|$. Thereafter, if the nonlinearity of the function is strong, then the step size is decreased; if the function is nearly linear, then the step size is increased.
To measure the degree of nonlinearity, the ratio $r = \dfrac{\Delta S}{\Delta S_L}$ is used where the amount of change for the linear model is represented by:

$$\Delta S_L = \Delta S_L(\boldsymbol{x} + \Delta \boldsymbol{x}) - S_L(\boldsymbol{x})$$

and the actual amount of change is represented by:

$$\Delta S = S(\boldsymbol{x} + \Delta \boldsymbol{x}) - S(\boldsymbol{x})$$

(a) If $r < 0.1$, then nonlinearity is considered to be strong and $d$ is reduced by half.

(b) If $r \geq 0.1$, then $\lambda$, which is the rate of increase of $d$, is calculated as follows:

$$\lambda^2 = 1.0 - (r - 0.1)\frac{\Delta S_L}{S_P + \sqrt{(S_P^2 - S_S(r - 0.1)\Delta S_L)}}$$

where, for $\delta \boldsymbol{f}$ defined as:

$$\delta \boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x} + \Delta \boldsymbol{x}) - (\boldsymbol{f}(\boldsymbol{x}) + A\Delta \boldsymbol{x})$$

$S_P$ and $S_S$ are as follows:

$$\begin{aligned} S_P &= \sum_{i=1}^{n} \mid f_i(\boldsymbol{x} + \Delta \boldsymbol{x})\delta f_i \mid \\ S_S &= \|\delta \boldsymbol{f}\|^2 \end{aligned}$$

Actually, to prevent the value of $d$ from oscillating, $d$ is increased only when an increase is required two times consecutively. Also, the rate of increase is held to at most 2. The actual rate of increase $\mu$ is calculated as follows:

$$\begin{aligned} \mu &= \min(2, \lambda, \tau) \\ \tau &= \frac{\lambda}{\mu} \end{aligned}$$

where the initial value for $\tau$ is 1, and 1 is reset whenever a reduction of $d$ is required.
In addition, an upper limit $d_{\max}$ and lower limit $d_{\min}$ are set for $d$ and $d$ is controlled so that it falls between these values.

(3) Correction vector independence check
To correct the Jacobian matrix efficiently, the correction vectors that are taken sequentially must be nearly mutually orthogonal. Therefore, an original independence concept is defined according to a hybrid method, and the correction vectors are controlled so that they are taken in directions that are as independent as possible. A vector $\boldsymbol{p}$ is said to be independent of the $i$ vectors $(\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_i)$ if $\boldsymbol{p}$ forms an angle of at least 30 degrees with an arbitrary vector of the space defined by these $i$ vectors. The calculation for this independence test conceived by Powell is as follows.
$n$ mutually independent vectors from the vectors used to correct the Jacobian matrix during the past $2 \times n$ iterations are made to be orthogonal and are stored in $\Omega = (\omega_1, \omega_2, \cdots, \omega_n)$. The array $\boldsymbol{j}$ of size $n$ is used to store information indicating the number of iterations earlier in which $\omega_i$ was the correction vector. That is, this information indicates that $\omega_i$ is the vector that was taken $j_i$ iterations earlier. $\Omega$ is initialized as the unit matrix, and $\boldsymbol{j}$ is initialized with values $j_i = n - i + 1$ for $(i = 1, \cdots, n)$.
When a solution is corrected, the following steps are performed.

(a) When $\Delta \boldsymbol{x} = \Delta \boldsymbol{x}_G$
Regardless of its independence, $\Delta \boldsymbol{x}$ is taken as the correction vector.

(b) When $\Delta \boldsymbol{x} = \Delta \boldsymbol{x}_G$ does not occur
If $j_1 < 2n$ or if $\Delta \boldsymbol{x}$ is independent of $(\omega_2, \omega_3, \cdots, \omega_n)$, then $\Delta \boldsymbol{x}$ is taken as the correction vector. Otherwise, the solution is not corrected.

(4) Calculation of Jacobian matrix $A$
The Jacobian matrix is obtained according to a difference only for the first iteration, and thereafter, it is sequentially updated. This method, which was devised by Broyden, calculates the Jacobian matrix according to the following equation:

$$A' = A + \delta \boldsymbol{f}\frac{\Delta \boldsymbol{x}^T}{\|\Delta \boldsymbol{x}\|^2}$$

However, if $\|\Delta \boldsymbol{x}\| < d_{\min}$ or if $j_1 = 2n$ and $\Delta \boldsymbol{x}$ is not independent of $(\omega_2, \omega_3, \cdots, \omega_n)$, then $\Delta \boldsymbol{x} = d_{\min}\omega_1$ is set.

(5) Revision of $\Omega$ and $\boldsymbol{j}$
$\Omega$ and $\boldsymbol{j}$ are revised as follows.
When $\Delta \boldsymbol{x} = d_{\min}\omega_1$ has been set, the following values should be set:

$$\begin{aligned} \omega_i &= \omega_{i+1} \ \text{ for } \ (i = 1, \cdots, n-1) \\ \omega_n &= \omega_1 \\ j_i &= j_{i+1} + 1 \ \text{ for } \ (i = 1, \cdots, n-1) \\ j_n &= 1 \end{aligned}$$

Otherwise, the following is performed.
The minimum value $k$ is obtained for which $(\omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta \boldsymbol{x})$ are mutually independent.
$(\omega_1, \cdots, \omega_{k-1}, \omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta \boldsymbol{x})$ are made to be orthogonal, and these are again assumed to be $(\omega_1, \omega_2, \cdots, \omega_n)$. The following values are then set:

$$\begin{aligned} j_i &= j_i + 1 \ \text{ for } \ (i = 1, \cdots, k-1) \\ j_i &= j_{i+1} + 1 \ \text{ for } \ (i = k, \cdots, n-1) \\ j_n &= 1 \end{aligned}$$

In this way, the relationship $j_1 \leq 2n$ always holds.

(6) Convergence decision
Convergence is assumed to have occurred when the following relationship holds and $\boldsymbol{x} + \Delta \boldsymbol{x}$ is assumed to be the solution:

$$\|\Delta \boldsymbol{x}\|_\infty \leq e_r \max(1, \|\boldsymbol{x} + \Delta \boldsymbol{x}\|_\infty)$$

where $e_r$ is the required relative precision, and:

$$\|\boldsymbol{x}\|_\infty = \max_i \mid x_i \mid$$

#### 5.1.2.4 Minimization of a constrained linear function of several variables (linear constraints)

When a linear function $f(\boldsymbol{x})$ of several variables related to a vector $\boldsymbol{x}$ of dimension $n$ is given as follows together with $m$ linear constraints and the domain of vector $\boldsymbol{x}$, this algorithm deals with a problem (linear programming problem) that obtains the vector $\boldsymbol{x}$ of dimension $n$ that satisfies the linear constraints and minimizes $f(\boldsymbol{x})$.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} \rightarrow \min$$

$$\begin{array}{lll} \text{Linear constraints:} & \boldsymbol{a}_i^T \boldsymbol{x} = b_i & \text{for} \ \ (i = 1, 2, \cdots, m_e) \\ & \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i & \text{for} \ \ (i = m_e + 1, m_e + 2, \cdots, m_{ne}) \\ & \boldsymbol{a}_i^T \boldsymbol{x} \geq b_i & \text{for} \ \ (i = m_{ne} + 1, m_{ne} + 2, \cdots, m) \quad 0 \leq m_e \leq m_{ne} \leq m \\ \text{Domain of } \boldsymbol{x}\text{:} & \boldsymbol{d} \leq \boldsymbol{x} \leq \boldsymbol{u} & \end{array}$$

Here, $\boldsymbol{c}, \boldsymbol{a}_i, \boldsymbol{d}$ and $\boldsymbol{u}$ are each constant vectors of dimension $n$, and $(b_i)$ is a constant vector of degree $m$. These are fixed by the problem.
Although an inequality is used for vectors such as $\boldsymbol{d} \leq \boldsymbol{x}$, this is used to mean that the inequality holds for each

element of the vector.
To solve a linear programming problem, this library provides subroutines that use the modified simplex method to handle cases in which the matrix $A = (a_{ij})$ that assigns the constraints is a dense matrix and subroutines that use the interior point method to handle cases in which $A$ is a sparse matrix.

First, the modified simplex method is explained. By adding new variables $X_i$, the inequality constraints $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$ can be converted to $\boldsymbol{a}_i^T \boldsymbol{x} + X_i = b_i$ and $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$ can be converted to $\boldsymbol{a}_i^T \boldsymbol{x} - X_i = b_i$. These variables are called a slack variables. By introducing slack variables, the constraints can all be represented by an equality as follows.

$$\begin{aligned} \text{Linear constraints:} &\quad A\boldsymbol{x}' = \boldsymbol{b} \\ \text{Domain of } \boldsymbol{x}\text{:} &\quad \boldsymbol{d}' \leq \boldsymbol{x}' \leq \boldsymbol{u}' \end{aligned}$$

Here, $A$ is an $n \times (n + m - m_e)$ matrix and $\boldsymbol{x}'$, $\boldsymbol{b}$, $\boldsymbol{d}'$ and $\boldsymbol{u}'$ are vectors of dimension $n + m - m_e$. Therefore, only equality constraints are considered subsequently. To simplify the notation, let's express $\boldsymbol{x}'$, $\boldsymbol{d}'$ and $\boldsymbol{u}'$ as $\boldsymbol{x}$, $\boldsymbol{d}$ and $\boldsymbol{u}$.

(1) Basic feasible solutions
The optimal solution exists among the basic feasible solutions. The simplex method obtains the optimal solution from among the basic feasible solutions, which are a finite set. Basic feasible solutions are explained below.
Select $m$ linearly independent columns from the columns of matrix $A$ and consider an $m \times m$ regular matrix $B$ having the selected columns as elements. Determine a transformation matrix $P$ of order $n + m - m_e$ so that the following are satisfied:

$$\begin{aligned} AP &= [B|L|U] \\ \left(P^{-1}\boldsymbol{x}\right)^T &= \left[\boldsymbol{x}_B^T \mid \boldsymbol{x}_L^T \mid \boldsymbol{x}_U^T\right], \\ \left(P^{-1}\boldsymbol{d}\right)^T &= \left[\boldsymbol{d}_B^T \mid \boldsymbol{d}_L^T \mid \boldsymbol{d}_U^T\right], \\ \left(P^{-1}\boldsymbol{u}\right)^T &= \left[\boldsymbol{u}_B^T \mid \boldsymbol{u}_L^T \mid \boldsymbol{u}_U^T\right] \end{aligned}$$

and represent the constraints in the following form:

$$\begin{aligned} B\boldsymbol{x}_B + L\boldsymbol{x}_L + U\boldsymbol{x}_U &= \boldsymbol{b} \\ \boldsymbol{d}_B \leq \boldsymbol{x}_B \leq \boldsymbol{u}_B & \\ \boldsymbol{d}_L \leq \boldsymbol{x}_L \leq \boldsymbol{u}_L & \\ \boldsymbol{d}_U \leq \boldsymbol{x}_U \leq \boldsymbol{u}_U & \end{aligned}$$

Here, $\boldsymbol{x}_B, \boldsymbol{d}_B$ and $\boldsymbol{u}_B$ represent vectors of dimension $m$; $L$ represents an $m \times \ell$ matrix; $\boldsymbol{x}_L, \boldsymbol{d}_L$ and $\boldsymbol{u}_L$ represent vectors of dimension $\ell$; $U$ represents an $m \times (n - m_e - \ell)$ matrix; and $\boldsymbol{x}_U, \boldsymbol{d}_U$ and $\boldsymbol{u}_U$ represent vectors of dimension $n - m_e - \ell$. However, $\ell$ is an integer that satisfies $0 \leq \ell \leq n - m_e$. Now, when $\boldsymbol{x}_B, \boldsymbol{x}_L, \boldsymbol{x}_U$ satisfy the following conditions, then:

$$\boldsymbol{x} = P \begin{bmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_L \\ \boldsymbol{x}_U \end{bmatrix}$$

is called the basic feasible solution.

$$\boldsymbol{d}_B \leq \quad \boldsymbol{x}_B \quad \leq \boldsymbol{u}_B$$

$$
\begin{aligned}
\boldsymbol{x}_L &= \boldsymbol{d}_L \\
\boldsymbol{x}_U &= \boldsymbol{u}_U
\end{aligned}
$$

$B$ is called a basic matrix, $\boldsymbol{x}_B$ is called a basic variable, $L$ and $U$ are called nonbasic matrices, and $\boldsymbol{x}_L$ and $\boldsymbol{x}_U$ are called nonbasic variables.

(2) Simplex method
The simplex method procedure is as follows.

(a) Obtain the initial feasible solution.

(b) With other nonbasic variables fixed, change a certain single variable $X_{IN}$ (element of $\boldsymbol{x}_L$ or $\boldsymbol{x}_U$) until a certain variable $x$ (element of $\boldsymbol{x}_B$ or $X_{IN}$) becomes the upper or lower bound value.

(c) When $\boldsymbol{x}_B$ has become the upper or lower bound value, replace the basic variable with a nonbasic variable.

(d) As long as the function value can get smaller, perform these operations repeatedly while selecting various nonbasic variables $x$ as $X_{IN}$.

This is explained concretely below.
First, obtain the initial feasible solution.
Next, obtain the nonbasic variable to be changed, $X_{IN}$. Using basic and nonbasic variables, the function value $f(\boldsymbol{x})$ can be represented in a form that includes the constraints as follows:

$$
\begin{aligned}
f(\boldsymbol{x}) &= \boldsymbol{c}_B^T B^{-1}\boldsymbol{b} + \boldsymbol{g}_L^T \boldsymbol{x}_L + \boldsymbol{g}_U^T \boldsymbol{x}_U \\
\boldsymbol{g}_L^T &= \boldsymbol{c}_L^T - \boldsymbol{c}_B^T B^{-1} L \\
\boldsymbol{g}_U^T &= \boldsymbol{c}_U^T - \boldsymbol{c}_B^T B^{-1} U
\end{aligned}
$$

Here, in $(P\boldsymbol{c})^T = (\boldsymbol{c}_B^T \;\; \boldsymbol{c}_L^T \;\; \boldsymbol{c}_U^T)$, the vectors $\boldsymbol{c}_B, \boldsymbol{c}_L$ and $\boldsymbol{c}_U$ are vectors of dimension $m, \ell$ and $n - m_e - \ell$ respectively.
Since $\boldsymbol{x}_L = \boldsymbol{d}_L$ and $\boldsymbol{x}_U = \boldsymbol{u}_U$, elements of $\boldsymbol{x}_L$ can only be changed in the positive direction and elements of $\boldsymbol{x}_U$ can only be changed in the negative direction. Therefore, the function value $f(\boldsymbol{x})$ can be reduced only when a certain element of $\boldsymbol{g}_L$ is negative or a certain element of $\boldsymbol{g}_U$ is positive. When there is no such element, the current value $\boldsymbol{x}$ is the optimal solution. The element of $\boldsymbol{g}_U$ or $\boldsymbol{g}_L$ that satisfies this condition and has the largest absolute value becomes the element $IN$ to be changed.
At this time, the variable $x$ (element of $\boldsymbol{x}_B$ or $X_{IN}$) to be changed to the upper or lower bound value is as follows. When the nonbasic variable $X_{IN}$ is changed, if $X_{IN}$ is assumed to be an element of $X_L$, the following relationships hold while the constraints are satisfied. Here, $\boldsymbol{a}_{IN}$ is the $IN$-th column of $A$ (this differs from the $\boldsymbol{a}$ mentioned earlier), and $\Delta$ is the amount of change in $X_{IN}$ ($\Delta \geq 0$).

$$
\begin{aligned}
\boldsymbol{d}_B \leq \quad & \boldsymbol{x}_B - B^{-1}\boldsymbol{a}_{IN}\Delta & \leq \boldsymbol{u}_B \\
\boldsymbol{d}_{IN} \leq \quad & X_{IN} + \Delta & \leq \boldsymbol{u}_{IN}
\end{aligned}
$$

When $\Delta$ is changed, the first variable $x$ (element of $\boldsymbol{x}_B$ or $X_{IN}$) for which the above relationship is not satisfied becomes the variable to be changed to the upper or lower bound value. If $\boldsymbol{x}_{IN}$ is an element of $\boldsymbol{x}_U$, the following relationship holds instead of the one shown above.

$$
\begin{aligned}
\boldsymbol{d}_B \leq \quad & \boldsymbol{x}_B + B^{-1}\boldsymbol{a}_{IN}\Delta & \leq \boldsymbol{u}_B \\
d_{IN} \leq \quad & X_{IN} - \Delta & \leq \boldsymbol{u}_{IN}
\end{aligned}
$$

The amount of decrease of $f(\boldsymbol{x})$ at this time is as follows. Assume the updated $f(\boldsymbol{x})$ is $f(\boldsymbol{x}')$.

$$
\begin{aligned}
f(\boldsymbol{x}') &= \quad f(\boldsymbol{x}) + g_{IN}\Delta \quad (\text{When } X_{IN} \text{ is an element of } \boldsymbol{x}_L, \quad g_{IN} < 0) \\
f(\boldsymbol{x}') &= \quad f(\boldsymbol{x}) - g_{IN}\Delta \quad (\text{When } X_{IN} \text{ is an element of } \boldsymbol{x}_U, \quad g_{IN} > 0)
\end{aligned}
$$

Finally, replace elements of $B, L$ and $U$ to match the change in the variable value.
If the above operations are performed repeatedly until there is no more $X_{IN}$ that can be selected, the optimal solution is obtained.

(3) Revised simplex method
The revised simplex method improves the handling of the basic matrix and the method of selecting the variable to be entered in the basis so that computations can be performed efficiently.
The greatest distinguishing feature of the revised simplex method is that uses an eta matrix $E_k$, which is defined in the following form, when updating the basic matrix $B_{k-1}$ to $B_k$.

$$
E_k = \begin{bmatrix}
1 & & & B_{k-1}^{-1} & \vdots & \boldsymbol{a}_{IN} & & & & \\
& \ddots & & & \vdots & & & & 0 & \\
& & 1 & & \vdots & & & & & \\
& & & \ddots & \vdots & & & & & \\
& & & B_{k-1}^{-1} & & \boldsymbol{a}_{IN}(OUT) & & & & \\
& & & & \vdots & & \ddots & & & \\
& & & & \vdots & & & 1 & & \\
& 0 & & & \vdots & & & & \ddots & \\
& & & B_{k-1}^{-1} & \vdots & \boldsymbol{a}_{IN} & & & & 1
\end{bmatrix}
$$

($OUT$: Subscript of basic variable to be changed to the upper or lower bound value)
($\boldsymbol{a}_{IN}(OUT)$ : $OUT$-th element of $\boldsymbol{a}_{IN}$)
Updating is performed according to $B_k = B_{k-1}E_k$.

Next, the interior point method is explained.

(1) Form of problems handled by the interior point method

The interior point method handles problems that were formulated as follows so that the lower limit value of $\boldsymbol{x}$ is 0.

$$
\begin{aligned}
& f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} \to \min \\
& \text{Linear constraint} \quad A\boldsymbol{x} = \boldsymbol{b} \\
& \text{Domain of } \boldsymbol{x} \qquad \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u}
\end{aligned}
\tag{5.1}
$$

Here, $A$ is an $m \times n$ matrix that assigns $m$ constraints for the $n$-dimensional vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$. The general linear programming problem that was handled by the modified simplex method can be expressed in the form shown in (5.1) by a translation in the $\boldsymbol{x}$ space and the introduction of slack variables. Although there are numerous types of algorithms that are classified as interior point methods, this library uses a method in which an affine transformation is performed.

(2) Optimal solution search in the interior point method

While the simplex method searches for an optimal solution from within the basic feasible solutions corresponding to the vertices of the feasible region $S$, the interior point method starts from the interior of the feasible region and searches for an optimal solution while varying the solution in the direction that reduces the objective function.

When $\boldsymbol{x}^{(k)}$ is a feasible solution, the interior point method searches in the direction of the direction vector $\boldsymbol{d}^{(k)}$ for which the reduction rate of the objective vector is the greatest among the vectors $\boldsymbol{d}^{(k)}$ for which $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}$ is feasible and $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k+1)} < \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k)}$ is satisfied. Here, $t$ is called the step size. If the constraint is not taken into account, the direction that reduces the objective function value $\boldsymbol{c} \cdot \boldsymbol{x}$ the most is the $-\boldsymbol{c}$ direction. Actually, since the constraint $A\boldsymbol{x}^{(k+1)} = A(\boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}) = \boldsymbol{b}$ must be satisfied, the direction for which $-\boldsymbol{c}$ is projected onto the subspace $\{x|A\boldsymbol{x} = \boldsymbol{0}\}$ of $S$ should be taken as the $\boldsymbol{d}^{(k)}$ direction.

(3) Big-M method

To search for the optimal solution according to the interior point method, an interior point of the feasible region, that is, an initial solution $\boldsymbol{x}^{(0)}$ which is inside the feasible region and not on the boundary, is required as the search starting point. However, the method of obtaining this kind of initial solution is not self-evident. Therefore, this library uses a method called the Big-M method to calculate the feasible solutions and optimal solution simultaneously. First, introduce $m$ artificial variables as the vector

$$\boldsymbol{x'} = (x_{n+1}, x_{n+2}, \ldots, x_{n+m})^{\mathrm{T}}$$

and formulate a linear programming problem of $n + m$ variables as follows.

$$\begin{array}{ll} \bar{f}(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{c}^T\boldsymbol{x} + M\boldsymbol{x'} \to \min & \\ \text{Linear constraint} & A\boldsymbol{x} + \bar{A}\boldsymbol{x'} = \boldsymbol{b} \\ \text{Domain of } \boldsymbol{x} & \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u} \\ \text{Domain of } \boldsymbol{x'} & \boldsymbol{0} \le \boldsymbol{x'} \end{array} \tag{5.2}$$

If we define the $m \times m$ matrix $\bar{A} = (\bar{a}_{ij})$ $(1 \le i, j \le m)$ for an arbitrary $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^{\mathrm{T}}$ by

$$\bar{a}_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik}x_k^{(0)} & (\text{when } i = j) \\ 0 & (\text{when } i \ne j) \end{cases}$$

then

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{bmatrix} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}, 1, 1, \ldots, 1)^{\mathrm{T}}$$

clearly is a feasible solution of (5.2). If a sufficiently large positive number is taken for the parameter $M$, then as the objective function $\bar{f}(\boldsymbol{x}, \boldsymbol{x'})$ of (5.2) with the artificial variables $x' = (x'_1, x'_2, \ldots, x_m)^{\mathrm{T}}$ is reduced according to the interior method, the artificial variable term $M\boldsymbol{x'}$ quickly approaches zero. This means that each artificial variable rapidly approaches zero. When the various artificial variables are sufficiently close to zero, the portion of the feasible solution of (5.2) excluding the artificial variables is viewed as a feasible solution of (5.1). As the objective function $\bar{f}(\boldsymbol{x}, \boldsymbol{x'})$ is further reduced, the $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$ term will be reduced. Therefore, the portion of the optimal solution of (5.2) excluding the artificial variables ultimately will be an optimal feasible solution of (5.1). If the artificial variables do not become sufficiently close to zero in the optimal solution of (5.2), that is if the following relationship holds for a given small positive number $\epsilon_A$,

$$\max_{1 \le j \le m} \{|x'_{n+j}|\} > \epsilon_A$$

the problem in (5.1) is considered to be infeasible.

In the following explanations, artificial variables are considered to already have been incorporated in (5.1).

(4) Determining the search direction according to an affine transformation

In the method of determining the search direction of the optimal solution described above, when $\boldsymbol{x}^{(k)}$ is near the center of $S$, since the step size $t$ is taken sufficiently large, the value of the objective function can be significantly reduced. However, when $\boldsymbol{x}^{(k)}$ is near the boundary of S, since a large step size cannot be taken, the objective function is not sufficiently reduced. Therefore, a transformation is performed for the variable $\boldsymbol{x}$ so that the current solution $\boldsymbol{x}^{(k)}$ becomes sufficiently distant from the boundary of the feasible region $S'$ in the post-transformation space. Then, the solution is updated so that the objective function is significantly reduced, and the inverse transformation is performed for the new solution in the pre-transformation space that was obtained to get $\boldsymbol{x}^{(k+1)}$. For this purpose, the affine transformation

$$y = (D^{(k)})^{-1}x$$

is performed according to the following diagonal matrix that is defined for the current solution $\boldsymbol{x}^{(k)}$.

$$D^{(k)} = \begin{bmatrix} x_1^{(k)} & & & 0 \\ & x_2^{(k)} & & \\ & & \ddots & \\ 0 & & & x_n^{(k)} \end{bmatrix}$$

It is clear that $\boldsymbol{x}^{(k)}$ in the pre-transformation space is shifted to$(1, 1, \ldots, 1)^{\mathrm{T}}$. The original linear programming problem is expressed as follows in the $\boldsymbol{y}$ space.

$$f(\boldsymbol{y}) = \hat{\boldsymbol{c}}^{(k)\mathrm{T}}\boldsymbol{y} \to \min$$
$$\text{Linear constraint} \quad \hat{A}^{(k)}\boldsymbol{y} = \boldsymbol{b}$$
$$\text{Domain of } \boldsymbol{y} \quad \boldsymbol{0} \leq \boldsymbol{y} \leq \hat{\boldsymbol{u}}^{(k)}$$

Here,

$$\hat{A}^{(k)} = AD^{(k)}$$

and

$$\hat{\boldsymbol{c}}^{(k)} = D^{(k)}\boldsymbol{c}$$

The direction $\hat{\boldsymbol{d}}^{(k)}$ that maximizes the reduction rate of the objective function in this problem is the direction for which $-\hat{\boldsymbol{c}}^{(k)}$ is projected onto the subspace $\{\boldsymbol{y}|\hat{A}^{(k)}\boldsymbol{y} = \boldsymbol{0}\}$. Since this projection is given by the transformation matrix expressed as follows

$$\hat{P}^{(k)} = I - (\hat{A}^{(k)})^{\mathrm{T}}(\hat{A}^{(k)}(\hat{A}^{(k)})^{\mathrm{T}})^{-1}\hat{A}^{(k)}$$

the search direction will be

$$\hat{\boldsymbol{d}}^{(k)} = -\hat{P}^{(k)}\hat{\boldsymbol{c}}^{(k)}$$

However, since this search direction has the following relationship with the search direction $\boldsymbol{d}^{(k)}$ for the original variable

$$\boldsymbol{d}^{(k)} = D^{(k)}\hat{\boldsymbol{d}}^{(k)}$$

the search direction in the original space is given by

$$\boldsymbol{d}^{(k)} = -D^{(k)}\hat{P}^{(k)}\hat{\boldsymbol{c}}^{(k)} = -(D^{(k)})^2(I - A^{\mathrm{T}}(A(D^{(k)})^2A^{\mathrm{T}})^{-1}A(D^{(k)})^2)\boldsymbol{c}$$

Although the inverse matrix calculation is included in this formula, the actual calculations first solve the following simultaneous linear equations to obtain $\boldsymbol{w}^{(k)}$

$$(A(D^{(k)})^2A^{\mathrm{T}})\boldsymbol{w}^{(k)} = A(D^{(k)})^2\boldsymbol{c}$$

and the search direction $\boldsymbol{d}^{(k)}$ can be determined as

$$\boldsymbol{d}^{(k)} = -(D^{(k)})^2(\boldsymbol{c} - A^{\mathrm{T}}\boldsymbol{w}^{(k)})$$

(5) Determining the step size

Points on the straight line $\boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}$ $(t \geq 0)$ given by the search direction $\boldsymbol{d}^{(k)}$ that was determined as described above will satisfy the constraint,

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

and the objective function will decrease monotonically for increases in the step size $t$. However, since the variable $\boldsymbol{x}$ is not permitted to leave the variable domain $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u}$, the maximum value that can be taken for the step size $t$ will be as follows.

$$t_{\max} = \min\left\{\min\left\{\frac{x_j^{(k)}}{-d_j^{(k)}}\,\middle|\, d_j^{(k)} < 0\right\}, \min\left\{\frac{u_j - x_j^{(k)}}{d_j^{(k)}}\,\middle|\, d_j^{(k)} > 0\right\}\right\}$$

Actually, since the search cannot continue if the boundary of the feasible region is reached, the value $t^{(k)}$ given by where the parameter $\alpha$ satisfying $0 < \alpha < 1$ will be the step size for $\boldsymbol{x}^{(k)}$, and the new solution $\boldsymbol{x}^{(k+1)}$ will be given by

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + t^{(k)}\boldsymbol{d}^{(k)}$$

(6) Replacing the upper and lower limits of the variable

If any variable approaches the lower limit during the search for the optimal solution, the affine transformation described earlier is effective. However, if the variable approaches the upper limit, the same method cannot be used. Therefore, if the variable $x_j$ approaches its upper limit $u_j$, this subroutine replaces the upper and lower limits of the variable by performing the following transformations.

$$\begin{array}{rcll} x_j & \leftarrow & u_j - x_j & \\ u_j & \leftarrow & u_j & \\ c_j & \leftarrow & -c_j & \\ b_i & \leftarrow & b_i - a_{ij}u_j & (i = 1, 2, \ldots, m) \\ a_{ij} & \leftarrow & -a_{ij} & (i = 1, 2, \ldots, m) \end{array}$$

(7) Checking the residual for the constraint

While repeating the iterative calculations for finding the optimal solution, the residual $||A\boldsymbol{x} - \boldsymbol{b}||$ for the constraint may become large because of calculation error. Once the residual becomes large, subsequent calculations are meaningless. To prevent this, a check is performed every $\ell$-th iteration to determine if the following relationship is satisfied

$$||A\boldsymbol{x}^{(k)} - \boldsymbol{b}|| \leq \epsilon_r$$

for a given positive number $\epsilon_r$. If it is not satisfied, the search for the optimal solution is started over again from the beginning by recalculating the matrix $\bar{A}$ of problem (5.2) with

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{bmatrix} = (x_1^{(k-\ell)}, x_2^{(k-\ell)}, \ldots, x_n^{(k-\ell)}, 1, 1, \ldots, 1)^{\mathrm{T}}$$

as the initial solution.

(8) Determining convergence

When the following relationship is satisfied

$$\frac{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k-1)} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k)}}{1 + ||\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k)}||} \leq \epsilon_C$$

the $\boldsymbol{x}^{(k)}$ values are considered to converge to the optimal solution. $\epsilon_C$ is the parameter for determining convergence.

#### 5.1.2.5 Minimization of a constrained linear function of several variables including 0-1 variables

This section deals with a linear programming problem having the added condition that several of the variables are 0-1 variables, that is, variables that take only 0 or 1 as the value (mixed 0-1 programming problem).

$$\begin{array}{llcll}
\text{Objective function} & : & f(\boldsymbol{x}) & = & \boldsymbol{c}^T\boldsymbol{x} \quad \rightarrow min \\
\text{Linear constraints} & : & a_{ij}x_j & = & b_i \quad (i = 1, \cdots, m; j = 1, \cdots, n) \\
\text{Domain of } \boldsymbol{x} & : & d_j \leq x_j \leq u_j & & (j = n_{01}+1, \cdots, n) \\
\text{0-1 variable condition} & : & x_j = 0 \; or \; 1 & & (j = 1, \cdots, n_{01}) \\
 & & & & (1 \leq n_{01} \leq n)
\end{array}$$

The explanation below assumes that the required number of slack variables have been introduced in advance so that only equality constraints are included as constraints.
This library uses the branch-and-bound method to solve the mixed 0-1 programming problem. he branch-and-bound method decomposes the original problem into several partial problems and obtains the solutions of the original problem by solving all of these partial problems. A partial problem of a mixed 0-1 programming problem is one in which the values of several 0-1 variables are fixed at 0 or 1. This is represented as follows.

$$\begin{array}{llcll}
\text{Objective function} & : & f(\boldsymbol{x}) & = & \boldsymbol{c}^T\boldsymbol{x} \quad \rightarrow min \\
\text{Linear constraints} & : & a_{ij}x_j & = & b_i \quad (i = 1, \cdots, m; j = 1, \cdots, n) \\
\text{Domain of } \boldsymbol{x} & : & d_j \leq x_j \leq u_j & & (j \notin S^0 \cup S^1 \cup F) \\
\text{0-1 variable condition} & : & x_j = 0 & & (j \in S^0) \\
 & & x_j = 1 & & (j \in S^1) \\
 & & x_j = 0 \; or \; 1 & & (j \in F)
\end{array}$$

Here, $S^0$, $S^1$, and $F$ are the set of subscripts of 0-1 variables having values fixed at 0, the set of subscripts of 0-1 variables having values fixed at 1, and the set of subscripts of 0-1 variables having values that are not fixed, respectively, in partial problem $P_k$. Now, 0-1 variables having values fixed at 0 or 1 are called fixed variables, and other 0-1 variables are called free variables. A partial problem is a mixed 0-1 programming problem in itself. In particular, the partial problem having $S^0 \cup S^1 = \phi$, that is, not having fixed variables, is the original mixed 0-1 programming problem (hereafter called $P_1$) itself. Also, the linear programming problem represented as follows is called the relaxation problem for the original partial problem.

| | | | | |
|---|---|---|---|---|
| Objective function | : | $f(\boldsymbol{x})$ | $=$ | $\boldsymbol{c}^T\boldsymbol{x} \quad \to min$ |
| Linear constraints | : | $a_{ij}x_j$ | $=$ | $b_i \quad (i = 1, \cdots, m; j = 1, \cdots, n)$ |
| Domain of $\boldsymbol{x}$ | : | $d_j \leq x_j \leq u_j$ | | $(j \notin S^0 \cup S^1 \cup F)$ |
| 0-1 variable condition | : | $x_j = 0$ | | $(j \in S^0)$ |
| | | $x_j = 1$ | | $(j \in S^1)$ |
| | | $0 \leq x_j \leq 1$ | | $(j \in F)$ |

The method of obtaining the solutions of $P_1$ by the branch-and-bound method is explained below.

(1) Initial settings

(a) Partial problem list
The branch-and-bound method uses the partial problem list PLIST. In the initial state, the members of PLIST consist only of $P_1$, and the number of members of PLIST $L$ is 1. $L$ increases and decreases as the branch-and-bound method calculations proceed.

(b) Incumbent
In the initial state, since not even one feasible solution of $P_1$ is found, the incumbent $\boldsymbol{x}^*$ is indeterminate, and a sufficiently large positive value is set in advance as the objective function value $z^*$ for the incumbent $\boldsymbol{x}^*$.

(c) Pseudo cost
The initial values of the pseudo costs $h_{up}(j)$ and $h_{down}(j)$ $(j \in F)$ to be used for updating PLIST are set as follows. First, the revised simplex method is used to solve the relaxation problem for $P_1$. At this time, if the optimal solution that was obtained is represented by $\bar{\boldsymbol{x}}$, the set of subscripts of basic variables within the optimal solution is represented by $B$, the set of subscripts of nonbasic variables that take the upper bound of the domain of definition is represented by $N_{up}$, and the set of subscripts of nonbasic variables that take the lower bound of the domain of definition is represented by $N_{down}$, the various basic variables $\bar{x}_{B_i}$ and the objective function value $f(\bar{\boldsymbol{x}})$ are represented as follows, using the nonbasic variables.

$$
\begin{aligned}
f(\bar{\boldsymbol{x}}) &= \bar{z}_0 + \sum_{j \in N_{down}} \bar{c}_j \bar{x}_j - \sum_{j \in N_{up}} \bar{c}_j \bar{x}_j \\
\bar{x}_{B_i} &= \bar{b}_i - \sum_{j \in N_{down}} y_{ij} \bar{x}_j - \sum_{j \in N_{up}} y_{ij} \bar{x}_j \quad (B_i \in B)
\end{aligned}
$$

Here, $\bar{z}_0$, $\bar{b}_i$, $\bar{c}_j$ and $y_{ij}$ are quantities obtained by the revised simplex method calculation process when obtaining the optimal solution of the relaxation problem of $P_1$, and the following relationship is satisfied.

$$\bar{c}_j \quad \geq \quad 0 \quad (j \in N_{up} \cup N_{down})$$

These quantities are used to set the initial values of $h_{up}(p)$ and $h_{down}(p)$.
When $p \in N_{up}$ :

$$
\begin{aligned}
h_{up}(p) &= 0 \\
h_{down}(p) &= -\bar{c}_p
\end{aligned}
$$

When $p \in N_{down}$ :

$$\begin{aligned} h_{up}(p) &= \bar{c}_p \\ h_{down}(p) &= 0 \end{aligned}$$

When $p = B_i \in B$ :

$$\begin{aligned} h_{up}(p) &= \min\{-\bar{c}_j/y_{ij} | y_{ij} < 0, j \in N_{down} - F \text{ or } y_{ij} > 0, j \in N_{up} - F\} \\ h_{down}(p) &= \min\{\bar{c}_j/y_{ij} | y_{ij} > 0, j \in N_{down} - F \text{ or } y_{ij} < 0, j \in N_{up} - F\} \end{aligned}$$

However, when $p = B_i \in B$, if there is no $j$ that satisfies the condition on the right-hand side for either $h_{up}(p)$ or $h_{down}(p)$ (but not both), a sufficiently large positive number is set as the value. If there is no $j$ that satisfies the condition on the right-hand side for both $h_{up}(p)$ and $h_{down}(p)$, then $h_{up}(p)$ and $h_{down}(p)$ are as follows:

$$\begin{aligned} h_{up}(p) &= \frac{\sum_{j \in N_{up}} -\bar{c}_j/y_{ij}}{|M_{up}|} \\ h_{down}(p) &= \frac{\sum_{j \in N_{up}} \bar{c}_j/y_{ij}}{|M_{down}|} \end{aligned}$$

where, $M_{up}(p)$ and $M_{down}(p)$ are defined as follows.

$$\begin{aligned} M_{up}(p) &= \{j | y_{ij} < 0, j \in N_{down} \cap F \text{ or } y_{ij} > 0, j \in N_{up} \cap F\} \\ M_{down}(p) &= \{j | y_{ij} > 0, j \in N_{down} \cap F \text{ or } y_{ij} < 0, j \in N_{up} \cap F\} \end{aligned}$$

(2) Relaxation problem
If any relaxation problem corresponding to a partial problem included in PLIST has not been solved, the revised simplex method is used to solve it. If the optimal value $g(P_i)$ for the solutions of the relaxation problem corresponding to partial problem $P_i$ satisfies $g(P_i) > z^*$, $P_i$ is removed from PLIST and $L \leftarrow L - 1$ is performed. Also, when the optimal solution is represented by $\bar{\boldsymbol{x}}$, if the value of $\bar{\boldsymbol{x}}_j$ is 0 or 1 for all $j \in F$, the solutions of the relaxation problem are solutions of the original partial problem, and the optimal value of the partial problem is equal to the optimal value $g(P_i)$ of the relaxation problem. At this time, if $f(P_i) < z^*$, $\boldsymbol{x}^*$ and $z^*$ are replaced by $\bar{\boldsymbol{x}}$ and $f(P_i)$.

Now, a new relaxation problem must be solved when the initial settings described above are made and when $L$ is increased by a branching operation, which is described later. In the latter case, the last two partial problems in PLIST are the pair of partial problems $P_{i0}$, for which the branching variable $x_{j_0}$ is fixed at 0, and $P_{i1}$, for which it is fixed at 1. At this time, the pseudo costs are updated as follows by using the optimal values of the relaxation problems obtained as described above $g(P_{i0})$ and $g(P_{i1})$, the optimal value of the partial problem $P_i$ before the branching operation $g(P_i)$, and the optimal solution value $\bar{x}_{j_0}$.

$$\begin{aligned} h_{up}(j_0) &= (g(P_{i1}) - g(P_i))/(1 - \bar{x}_{j_0}) \\ h_{down}(j_0) &= (g(P_{i0}) - g(P_i))/\bar{x}_{j_0} \end{aligned}$$

(3) Optimal value estimate
If the branching operation, which is described later, has been performed, let $K' = \min\{K+1, L\}$. Otherwise, let $K' = \min\{K, L\}$. Here, $K$, which is called the depth of search in the branch-and-bound method, is a positive integer parameter that has been assigned in advance. For the last $K'$ partial problems $P_i$ in PLIST, the optimal value of the relaxation problem $g(P_i)$, the optimal solution $\bar{\boldsymbol{x}}$, and the pseudo costs $h_{up}(j)$

and $h_{down}(j)$ are used to calculate the estimate $h(P_i)$ of the optimal value $f(P_k)$ of the partial problem as follows:

$$h(P_i) = g(P_i) + \sum_{j \in F} \min\{h_{up}(j)(1-\bar{x}_j), h_{down}(j)\bar{x}_j\}$$

and the last $K'$ entries of PLIST are rearranged in descending order of $h(P_i)$.

(4) Partial problem test
For the last partial problem $P_k$ in PLIST, if the basic variables $\bar{x}_{B_i}$ $(B_i \in B)$ of the optimal solution of its relaxation problem are represented by the nonbasic variables $\bar{x}_j$ $(j \in N_{up} \cup N_{down})$, quantities are obtained that correspond to the quantities $\bar{z}_0$, $y_{ij}$, $\bar{c}_j$, and $\bar{b}_i$, which were obtained earlier when calculating the initial values of the pseudo costs. These quantities are used to defined $Z_{max}(i)$ and $Z_{min}(i)$ as follows:

$$\begin{aligned} Z_{max}(i) &= \bar{b}_i - \sum_{j \in N_{down}-S^0} \min\{0, y_{ij}\}(u_j - d_j) + \sum_{j \in N_{up}-S^1} \max\{0, y_{ij}\}(u_j - d_j) \\ Z_{min}(i) &= \bar{b}_i - \sum_{j \in N_{down}-S^0} \max\{0, y_{ij}\}(u_j - d_j) + \sum_{j \in N_{up}-S^1} \min\{0, y_{ij}\}(u_j - d_j) \end{aligned}$$

and the following eight rules can be used to fix several of the free variables $x_j (j \in F)$ at 0 or 1.

- $Z_{max}(i) < 1$ and $B_i \in F$, then $x_{B_i} = 0$
- $Z_{min}(i) > 0$ and $B_i \in F$, then $x_{B_i} = 1$
- $Z_{max}(i) - \max\{0, y_{ij}\} < d_{B_i}$ and $j \in N_{down} \cap F$, then $x_j = 0$
- $Z_{max}(i) + \min\{0, y_{ij}\} < d_{B_i}$ and $j \in N_{up} \cap F$, then $x_j = 1$
- $Z_{min}(i) - \min\{0, y_{ij}\} > u_{B_i}$ and $j \in N_{down} \cap F$, then $x_j = 0$
- $Z_{min}(i) + \max\{0, y_{ij}\} > u_{B_i}$ and $j \in N_{up} \cap F$, then $x_j = 1$
- $|\bar{c}_j| \geq z^* - g(P_k)$ and $j \in N_{down} \cap F$, then $x_j = 0$
- $|\bar{c}_j| \geq z^* - g(P_k)$ and $j \in N_{up} \cap F$, then $x_j = 1$

Now, $F$, $S^0$, and $S^1$ are updated each time a variable is fixed. When all free variables have been fixed, the partial problem $P_k$ is solved, $P_k$ is removed from PLIST, and $L \leftarrow L-1$ is performed. At this time, if $f(P_k) < z^*$, $\boldsymbol{x}^*$ and $z^*$ are replaced by the solution of $P_k$ and $f(P_k)$. Then, processing proceeds with step (6) according to the updated PLIST. If there are any remaining free variables that have not been fixed, processing proceeds with the next branching operation.

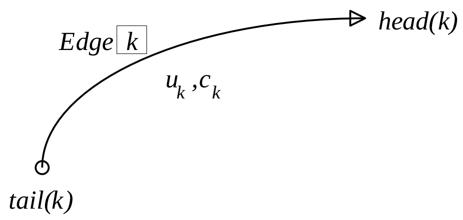
(5) Branching operation
$P_k$ is removed from PLIST. Then, one branching variable $x_{j_0}$ is selected from the free variables that have not been fixed by the partial problem test described above, the partial problems $P_{k0}$, for which $j_0$ was added to $S^0$, and $P_{k1}$, for which $j_0$ was added to $S^1$, are added at the end of PLIST, and $L \leftarrow L+1$ is performed. The branching variable $x_{j_0}$ is determined here as follows.
When incumbent $\boldsymbol{x}^*$ has not been obtained:
For $j_0$, set the $j \in F \cap B$ that maximizes the value of $|h_{up}(j)(1-\bar{x}_j) - h_{down}(j)\bar{x}_j|$. Here, $\bar{\boldsymbol{x}}$ is the optimal solution of relaxation problem of the partial problem $P_k$.
When incumbent $\boldsymbol{x}^*$ has been obtained:
For $j_0$, set the $j \in F \cap B$ that maximizes the value of $\min\{h_{up}(j)(1-\bar{x}_j), h_{down}(j)\bar{x}_j\}$.

(6) Termination condition
If $L = 0$, processing for the branch-and-bound method terminates, and the incumbent at that time $x^*$ is the solution of $P_1$, that is, of the original mixed 0-1 programming problem. Otherwise, processing returns to step (2).

#### 5.1.2.6 Minimization of cost for flow in a network

Consider a network having $n$ vertices and $m$ edges in which it is assumed that no loops (edges for which the tail of the edge and head of the edge are the same vertex) exist. Directed edge $k$, which is represented by its tail $tail(k)$ and head $head(k)$, has a nonnegative capacity $u_k$ and cost coefficient per unit flow $c_k$. The minimal-cost

Figure 5−2 Edge $k$ and its Capacity $u_k$ and Cost Coefficient $c_k$

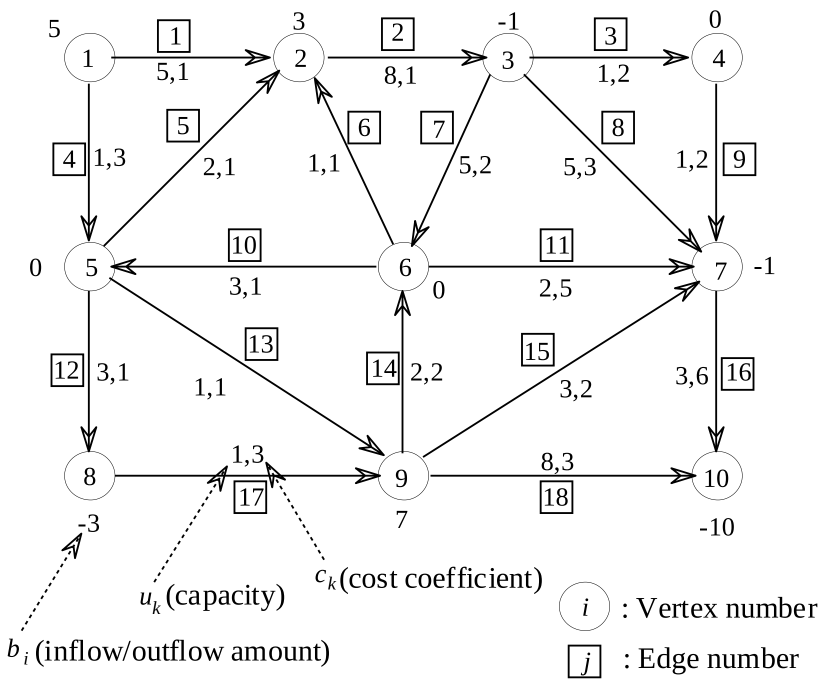



flow problem obtains the nonnegative flows $x_k$ $(k = 1, \cdots, m)$ that satisfy the vertex $i$ inflow/outflow amount $b_i$ and edge $k$ capacity $u_k$ constraints and minimize the sum of the costs of all edges in this kind of network.

$$\begin{aligned} \text{Objective function} \quad &: \quad \sum_{k=1}^{m} c_k x_k \longrightarrow min \\ \text{Constraints} \quad &: \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k = b_i, \quad (i = 1, \cdots, n) \\ & \quad\;\; 0 \le x_k \le u_k, \quad (k = 1, \cdots, m) \\ & \quad\;\; \sum_{i=1}^{n} b_i = 0 \end{aligned}$$

Although this is a linear programming problem to which the simplex method can be applied, a data structure representing the tableau graphically can be created by taking advantage of the characteristics of the constraints, and calculations involving pivoting arithmetic can be executed more efficiently.

Figure 5−3 Network Example




(1) **Initial feasible solution**

Add vertex $n+1$ to the network to create edge $k = (i, n+1)$ from vertex $i$ if $b_i \geq 0$ or edge $k = (n+1, i)$ to vertex $i$ if $b_i < 0$. Here, a sufficiently large positive value $L$ is set for the cost coefficient of the added edge, $\infty$ (specifically, a sufficiently large positive value $U$) is set for the capacity, and $b_{n+1}$ is set equal to zero. At this time, the modified minimal-cost flow problem is as follows.

$$\begin{aligned}
\text{Objectivefunction} \quad &: \quad \sum_{k=1}^{m'} c_k x_k \longrightarrow \min \\
\text{Constraints} \quad &: \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k + sign(b_i) \cdot x_{m+i} = b_i, \quad (i = 1, \cdots, n) \\
& \quad\;\; 0 \leq x_k \leq u_k, \quad (k = 1, \cdots, m'(= m+n)) \\
& \quad\;\; \sum_{i=1}^{n} b_i = 0
\end{aligned}$$

where,

$$\begin{array}{rcl} c_k & = & L \\ u_k & = & U \end{array} \quad , k = m+1, \cdots, m'$$

$$sign(b_i) = \begin{cases} 1, & b_i \geq 0 \\ -1, & b_i < 0 \end{cases}$$

From this, the initial feasible solution should be as follows:

$$x_k = \begin{cases} 0 & (k = 1, \cdots, m) \\ sign(b_{k-m}) \cdot b_{k-m} & (k = m+1, \cdots, m') \end{cases}$$

(2) **Selection of a column as the pivoting column and pivoting arithmetic**
If $A_k$ is the $k$-th column of the coefficient matrix of the constraints and $B$ is a basic matrix determined by selecting an appropriate $n \times n$ regular submatrix from the coefficient matrix, then $\bar{c_k}$ defined as follows:

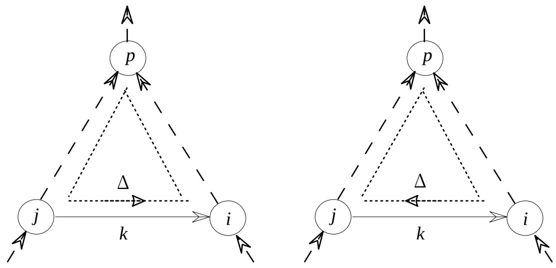
$$\bar{c_k} \quad = \quad c_k - c_B^T B^{-1} A_k$$

is obtained for nonbasic variable $x_k$, and the column that satisfies the following condition:

$$\begin{array}{ll} \bar{c_k} < 0, & \text{for } x_k = 0 \\ \bar{c_k} > 0, & \text{for } x_k = u_k \end{array} \qquad (5.3)$$

is selected for one pivot column. When the pivot column $k$ is determined, the upper bound $\Delta$ of the variation of $x_k$ is equal to the maximum possible flow when flowing along the circuit $j \longrightarrow i \longrightarrow p \longrightarrow j$ passing through vertex $p$, which is the vertex having the maximum depth among the common ancestors of end vertices $i$ and $j$ of edge $k$. The minimum value among the $\Delta_\ell$, which are determined as follows for each

Figure 5−4 Circulation of Flow Δ




(a) When $x_k = 0$ (b) When $x_k = u_k$

edge $\ell$ within this circuit, should be set for $\Delta$.

- On edge $k$, $\Delta_k = u_k$
- If the direction of edge $\ell$ on the path from $i$ to $p$ is downward, $\Delta_\ell = u_\ell - x_\ell$; if it is upward, $\Delta_\ell = x_\ell$
- If the direction of edge $\ell$ on the path from $j$ to $p$ is downward, $\Delta_\ell = x_\ell$; if it is upward, $\Delta_\ell = u_\ell - x_\ell$

Next, make flow amount $\Delta$ flow along this circuit. That is, for each edge $\ell$ within the circuit, perform the following:

$$x_\ell \quad := \quad \begin{cases} x_\ell + \Delta, & \text{when directions of edge } \ell \text{ and circuit match} \\ x_\ell - \Delta, & \text{when directions of edge } \ell \text{ and circuit are opposite} \end{cases}$$

and do not change the flow amount of edges not on the circuit. This is pivoting arithmetic. Also, let the new flow amount $x_\ell$ of edge $\ell$ for which $\Delta_\ell = \Delta$ had been satisfied within the circuit be zero or let $u_\ell$ be a new nonbasic variable $x_r$ (if multiple nonbasic variables exist, select any one of them). Repeat the processing described. Processing terminates when there are no columns left that satisfy condition (5.3)).

#### 5.1.2.7 Minimization of cost for project scheduling

The project which consists of two or more tasks is considered. Each task has some of the precedence tasks, and if they all are not completed, it cannot start it. A project network expresses each task with an effective branch, and a precedence relation is shown by minding a node.

Each task $k$ has a normal task time $t_N(k)$ and a crash task time $t_C(k)$, $t_N(k)$ and $t_C(k)$ are related as follows.

$$t_N(k) \geq t_C(k)$$

Also, the normal task cost $c_N(k)$ and crash task cost $c_C(k)$ for accomplishing task $k$ are related as follows.

$$c_N(k) \leq c_C(k)$$

When $t_N(k) \geq t_C(k)$, the cost $c(k)$ to accomplish task $k$ according to an intermediate task time $t(k)$ can be given by the equation:

$$c(k) = a(k) - b(k) \times t(k)$$

where,

$$a(k) = \frac{c_C(k) \times t_N(k) - c_N(k) \times t_C(k)}{t_N(k) - t_C(k)}, \quad b(k) = \frac{c_C(k) - c_N(k)}{t_N(k) - t_C(k)}$$

Now, when each task $k$ is done by task time $t(k)$, for each node $i$, the time for which all tasks that enter $i$ are completed and the tasks for branches leaving $i$ can start at any time $E(i)$, which is called the earliest node time, is obtained by the following equations:

$$E(1) = 0$$
$$E(i) = \max\{E(tail(k)) + t(k) \mid head(k) = i\}, \quad i = 2, 3, \cdots, n$$

Since $n$ represents the number of nodes, $tail(k)$ represents the starting point of each task $k$ and $head(k)$ represented the endpoint. Also, the project completion time $T$ is obtained by the following equation:

$$T = E(n)$$

And the time for which to complete the project at time $T$, by what time must the tasks that enter each node $i$ be completed even if they are delayed $L(i)$, which is called the latest node time, is obtained by the following equations:

$$L(n) = T$$
$$L(i) = \min\{L(head(k)) - t(k) \mid tail(k) = i\}, \quad i = n-1, n-2, \cdots, 1$$

To devise a concrete scheduling plan, first, set the scheduled completion time $T_S$. $T_S$ is normally set in the following range:

$$T_C \leq T_S \leq T_N$$

The problem of completing a project within the scheduled completion time $T_S$ according to a minimum cost can be formulated as the following linear programming problem by setting the required time for task $k$ as the variable $t(k)$ and the node time of node $i$ as the variable $\tau(i)$.

$$\begin{aligned}
\text{Objective function} \quad &: \quad \sum_{k=1}^{n} (a(k) - b(k) \times t(k)) \to \min \\
\text{Constraints} \quad &: \quad t(k) + \tau(tail(k)) - \tau(head(k)) \leq 0, \quad k = 1, 2, \cdots, n \\
&\quad\;\; \tau(1) = 0 \\
&\quad\;\; \tau(n) \leq T_S \\
&\quad\;\; t(k) \leq t_N(k), \quad k = 1, 2, \cdots, n \\
&\quad\;\; -t(k) \leq -t_C(k), \quad k = 1, 2, \cdots, n
\end{aligned}$$

When the node time $\tau(i)$ is obtained, the required time $t(k)$ of task $k$ can be obtained. $t(k)$ should be set as follows.

$$t(k) = \min\{t_N(k), \tau(head(k)) - \tau(tail(k))\}$$

Furthermore, the earliest node time $E(i)$ and latest node time $L(i)$ of each node $i$ can be obtained.

At this time, the earliest time $ES(k)$ that work can begin for task $k$, which is called the earliest start time of task $k$, and the last possible time $LS(k)$ by which work for task $k$ must begin, even if it is delayed, in order for the project to be completed by $T_S$, which is called the latest start time of task $k$ are given as follows:

$$ES(k) = E(tail(k))$$
$$LS(k) = L(head(k)) - t(k)$$

Also, if the start of task $k$ is delayed within a range $TF(k)$, the project can be completed within $T_S$ as long as the schedule of the remaining tasks is properly adjusted, which is called the total float of task $k$, and within the total float, even if the start of task $k$ is delayed in a range $FF(k)$, the subsequent work plan is not affected at all, which is called the free float of task $k$ are given as follows:

$$TF(k) = LS(k) - ES(k)$$
$$FF(k) = E(head(k)) - E(tail(k)) - t(k)$$

#### 5.1.2.8 Minimization of cost for transportation from supply place to demand place

The transportation problem is a special problem among linear programming problems. This problem requires us to find a route by which an article, which is a problem caused along with moving goods, is transported from a supply site to a demand site at minimum transportation costs, and also requires us to find out the expense to be presumed in that case. To solve this problem, settle on an initial plan in the first approximate solution and then improve it to get to the final plan (optimal solution). This library provides the northwestern corner rule and Houthakker's Method as first approximate solution, and Revised Simplex Method as method of improvement. (See "Algorithms Used" (5.1.2.4).) Here, supply quantity of supply place $i$ is $a_i$, demand quantity of demand place $j$ is $b_j$ and $x_{ij}$ is the volume transported from supply place $i$ to demand place $j$.

Constraint

$$\sum_{j=1}^{n} x_{ij} = a_i \quad (i = 1, \cdots, m)$$

$$\sum_{i=1}^{m} x_{ij} = b_i \quad (j = 1, \cdots, n)$$

$$x_{ij} \geq 0 \quad (\text{ For all } i \text{ and } j \text{ numbers})$$

Therefore, the total transportation cost is obtained by finding $x_{ij}$ that minimizes the following function.

$$Z = \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}x_{ij}$$

(1) First Approximate Solution(The Northwest Corner Rule, Houthakker's Method)
The northwest corner rule distributes a minimum of resources one by one after comparing the transportation cost, supply quantity, and demand quantity for a given unit quantity with one another from upper left of the matrix. Houthakker's Method distributes resources in a multiple way to a minimum of a unit transportation cost that extends from each supply place to each demand place.

(2) Unbalanced Transportation Problem
The total supply volume $\sum a_i$ is sometimes less than the total demand volume $\sum b_j$ because of a transportation problem. In this case, even if all demand volumes cannot be satisfied, a certain amount of the volume can be distributed from supply places to demand places by a method that minimizes the total handling cost. In this case, assume that we have a fictitious supply places that handles a total volume of $\sum b_j$-$\sum a_i$. Assume that the cost that covers delivering one unit from the fictitious supply places to a destination is zero. If the original shape problem is the NS$\times$ND problem, this problem, therefore, is supposed to be solved in the same way as a transportation problem that handles the (NS $+$ 1)$\times$ND problem. If the total supply volume of the problem is larger than the total demand volume, set up a fictitious problem in the same way. In this case, assume a fictitious destination that satisfies $\sum a_i$-$\sum b_j$. Assume that this delivery cost is zero. Then, the following NS$\times$ND problem is supposed to be solved for a NS$\times$(ND $+$ 1) problem.

#### 5.1.2.9 Minimization of a constrained quadratic function of several variables (linear constraints)

This algorithm deals with a problem (quadratic programming problem) that minimizes the $n$ variable objective function:

$$M(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x}$$

under the constraints:

$$\begin{aligned} \boldsymbol{a}_i^T\boldsymbol{x} = b_i \qquad & \text{for} \ \ (i = 1, 2, \cdots, m_e) \\ \boldsymbol{a}_i^T\boldsymbol{x} \geq b_i \qquad & \text{for} \ \ (i = m_e + 1, \cdots, m) \end{aligned}$$

Here, $\boldsymbol{a}_i$ and $\boldsymbol{c}$ are $n$ dimensional column vectors, $b_i$ are constants, and $G$ is an $n \times n$ positive symmetric matrix. This library uses the GI method to solve this problem. This method was proposed by D. Goldfarb and A. Idnani. Let $J$ be a set consisting of several of the constraint subscripts, let the solution $\boldsymbol{x}$ that minimizes the objective function $M(\boldsymbol{x})$ under all of the constraints corresponding to elements of $J$ be called the $J$-optimal solution, and let the solution of the original problem be called simply the optimal solution. Also, denote the number of elements of $J$ by $\mid J \mid$ and denote the $n\times \mid J \mid$ matrix having $\boldsymbol{a}_i \ \ (i \in J)$ as columns by $A_J$. Furthermore, let $K$ be the set of all subscripts of inequality constraints and $E$ be the set of all subscripts of equality constraints. Therefore, when $J = E \cup K$, the $J$ -optimal solution is the optimal solution itself. As the procedure for obtaining the optimal solution, first let $J = E \cup K$ and let the $J$-optimal solution at this time be the initial solution. Add inequality constraints that must be satisfied one at a time to $J$ and continue to revise the $J$-optimal solution so that the added constraint is satisfied. Repeatedly revise the solution until finally $J = E \cup K$.

(1) Calculate the initial solution
Obtain the $J$-optimal solution when $J = E$. When the object function is convex, the problem the obtains the optimal solution is equivalent to obtaining $\boldsymbol{x}$ and $\boldsymbol{v}_J$ satisfying the Kuhn-Tucker condition:

$$\begin{aligned} G\boldsymbol{x} - A_J\boldsymbol{v}_J &= -\boldsymbol{c} \\ A_J^T\boldsymbol{x} &= \boldsymbol{b}_J \\ \boldsymbol{v}_i &\geq 0 \quad (i \in J) \end{aligned}$$

Here, $\boldsymbol{v}_J$ and $\boldsymbol{b}_J$ are $\mid J \mid$ dimensional vectors having $v_i \ \ (i \in J)$ and $b_i \ \ (i \in J)$ as components, respectively. Now, if we let:

$$\begin{aligned} A_J^\dagger &= (A_J^T G^{-1} A_J)^{-1} A_J^T G^{-1} \\ H_J &= G^{-1}(I - A_J A_J^\dagger) \end{aligned}$$

then $\boldsymbol{x}$ and $\boldsymbol{v}_J$ are calculates as follows:

$$\begin{aligned}\boldsymbol{x} &= (A_J^\dagger)^T b_J - H_J \boldsymbol{c} \\ \boldsymbol{v}_J &= (A_J^T G^{-1} A_J)^{-1} \boldsymbol{b}_J + A_J^\dagger \boldsymbol{c}\end{aligned}$$

(2) Update $J$

If the $J$-optimal solution has been obtained for a certain $J \subset K$ and that solution is not the optimal solution, determine the constraint to be added to $J$ as follows. First, if we let:

$$c_i(x) = \boldsymbol{a}_i^T \boldsymbol{x} - \boldsymbol{b}_i$$

then there exists at least one $s \notin J$ for which $\boldsymbol{c}_s(x) < 0$ is satisfied. Therefore, determine $s$ according to:

$$c_s(x) = \min \{c_i(x) | i \notin J\} < 0$$

Then, if $\boldsymbol{a}_s$ is independent of the $\boldsymbol{a}_i \quad (i \in J)$ that form the columns of $A_J$, let $J \cup \{s\}$ be the new $J$. Otherwise, recalculate the $J$-optimal solution by eliminating one element from $J$.

(3) Update the $J$-optimal solution

After updating $J$ by adding the following inequality constraint determined in step (b) to $J$:

$$\boldsymbol{a}_s^T \boldsymbol{x} - \boldsymbol{b}_s \geq 0$$

obtain the $J$-optimal solution as follows. Let $\boldsymbol{x}$ be the $J$-optimal solution before $J$ was updated and let $c_i = 0 \quad (i \in J)$. At this time, if we set:

$$\bar{\boldsymbol{x}} = \boldsymbol{x} + tH_J \boldsymbol{a}_s$$

then the following relationships hold for the $c_i \quad (i \in J)$ and $c_s$ defined in (b) and for $v_i \quad (i \in J)$ :

$$\begin{aligned}c_i(\bar{\boldsymbol{x}}) &= c_i(\boldsymbol{x}) = 0 \quad (i \in J) \\ c_s(\bar{\boldsymbol{x}}) &= c_s(\boldsymbol{x}) + t\boldsymbol{a}_s^T H_J \boldsymbol{a}_s \\ v_i(\bar{\boldsymbol{x}}) &= v_i(\boldsymbol{x}) - t \cdot r_i\end{aligned}$$

where, $r_i \quad (i \in J)$ is the $i$-th component of:

$$\boldsymbol{r} = A_J^\dagger \boldsymbol{a}_s$$

now, if we let:

$$\begin{aligned}t_1 &= \min \left\{ \frac{v_i(\boldsymbol{x})}{r_i} \middle| r_i > 0, \quad i \in J \cap K \right\} \\ t_2 &= -\frac{c_s(\boldsymbol{x})}{\boldsymbol{a}_s^T H_J \boldsymbol{a}_s}\end{aligned}$$

then in the range $0 \leq t \leq t_1$ :

$$v_i(\bar{\boldsymbol{x}}) \geq 0 \quad (i \in \bar{J} \cap K)$$

and when $t = t_2$ :

$$c_s(\bar{\boldsymbol{x}}) = 0$$

Therefore, if $t_1 \geq t_2$, the $\bar{\boldsymbol{x}}$ when $t = t_2$ satisfies the Kuhn-Tucker condition used in (a) for constraints corresponding to $J \cup \{s\}$, and this becomes the $J$-optimal solution after $J$ is updated. On the other hand, if $t_1 \geq t_2$, one element is removed from $J$ and the $J$-optimal solution is recalculated.

(4) Termination condition
If the $J$-optimal solution $\boldsymbol{x}$ satisfies:

$$c_i(\boldsymbol{x}) \geq 0$$

for all $i \in \bar{J} \cap K$, the optimal solution is assumed to be $\boldsymbol{x}$, and the calculation ends.

#### 5.1.2.10 Minimization of a generalized convex quadratic function of several variables (linear constraints)

Here we will deal with the problem of obtaining $\boldsymbol{x}^*$ that minimizes the generalized convex quadratic function of $n$ variables

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad (G : \text{positive semi-definite symmetric matrix})$$

which is the objective function under the constraints

$$\begin{aligned}
\sum_{j=1}^{n}(a_{ij}x_j) &= b_i \quad \text{for} \quad (i = 1, \cdots, m_e) \\
\sum_{j=1}^{n}(a_{ij}x_j) &\geq b_i \quad \text{for} \quad (i = m_e + 1, \cdots, m) \\
x_i &\geq 0 \quad \text{for} \quad (i = 1, \cdots, n)
\end{aligned}$$

and to obtain the function value $f(\boldsymbol{x}^*)$ at that time. To solve this problem, the subroutines in this library convert the given problem to an equivalent linear complementarity problem and solve that problem by using the Lemke method.
The actual procedure is shown below.

(1) Creation of linear complementarity problem
First, eliminate $m_e$ variables by using the equality constraints and convert the original quadratic programming problem to a quadratic programming problem for the remaining variables. Then, create an equivalent linear complementarity problem for the new quadratic programming problem. From $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$, use the equality constraints to represent $x_1, x_2, \cdots, x_{m_e}$ in terms of $x_{m_e+1}, \cdots, x_n$ as follows.

$$x_i = \sum_{j=m_e+1}^{n}(p_{ij}x_j) + r_i \quad \text{for} \quad (i = 1, \cdots, m_e)$$

Using this expression, convert the given quadratic programming problem to a quadratic programming problem for $(n - m_e)$ variables $\boldsymbol{x}' = [x_{m_e+1}, \cdots, x_n] = [x'_1, \cdots, x'_{n-m_e}]$

$$\begin{aligned}
\text{Constraints} \quad &: \quad \sum_{j=1}^{n-m_e}(a'_{ij}x'_j) \geq b'_i \quad \text{for} \quad (i = 1, \cdots, n - m_e) \\
& \quad\;\; x'_i \geq 0 \quad \text{for} \quad (i = 1, \cdots, n - m_e) \\
\text{Objective function} \quad &: \quad f(\boldsymbol{x}') = \frac{1}{2}(\boldsymbol{x}')^T G'\boldsymbol{x}' + \boldsymbol{c}'^T\boldsymbol{x}'
\end{aligned}$$

For

$$
\begin{aligned}
G'_{i-m_e,j-m_e} &= G_{i,j} + \sum_{k=1}^{m_e}(G_{i,k}p_{k,j}) + \sum_{k=1}^{m_e}(G_{k,j}p_{k,i}) + \sum_{s=1}^{m_e}\sum_{t=1}^{m_e}(p_{s,i}G_{s,t}p_{t,j}) \\
c'_{i-m_e} &= c_i + \sum_{j=1}^{m_e}(c_j p_{j,i}) + \sum_{k=1}^{m_e}(G_{i,k}r_k) + \sum_{s=1}^{m_e}\sum_{t=1}^{m_e}(G_{s,t}p_{s,i}r_t) \\
a'_{i\ j-m_e} &= \begin{cases} p_{i,j} & \text{for } \ (i=1,\cdots,m_e; j=m_e+1,\cdots,n) \\ a_{i,j} + \displaystyle\sum_{k=1}^{m_e}(a_{i,k}p_{k,j}) & \text{for } \ (i=m_e+1,\cdots,m; j=m_e+1,\cdots,n) \end{cases} \\
b'_i &= \begin{cases} -r_i & \text{for } \ (i=1,\cdots,m_e) \\ b_i - \displaystyle\sum_{k=1}^{m_e}(a_{i,k}r_k) & \text{for } \ (i=m_e+1,\cdots,m) \end{cases}
\end{aligned}
$$

let $A' = (a'_{i,j}), G' = (G'_{i,j}), (\boldsymbol{c}')^T = [c'_1, \cdots, c'_{n-m_e}]$ and $(\boldsymbol{b}')^T = [b'_1, \cdots, b'_{n-m_e}]$ to create the following linear complementarity problem where, $\boldsymbol{v}$ is the Lagrange multiplier vector for the inequality constraints.

$$
\begin{aligned}
\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{w} \end{bmatrix} &= \begin{bmatrix} G' & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix} + \begin{bmatrix} \boldsymbol{c}' \\ -\boldsymbol{b}' \end{bmatrix} \\
y_i &\geq 0 \ \text{ for } \ (i=1,\cdots,n-m_e) \\
w_i &\geq 0 \ \text{ for } \ (i=1,\cdots,m) \\
x_i &\geq 0 \ \text{ for } \ (i=1,\cdots,n-m_e) \\
v_i &\geq 0 \ \text{ for } \ (i=1,\cdots,m) \\
[\boldsymbol{y}^T \ \boldsymbol{w}^T] \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix} &= 0
\end{aligned}
$$

(2) Linear complementarity problem

Solve the linear complementarity problem to obtain solutions of the new quadratic programming problem obtained by eliminating the equality constraints.

For the given $n \times n$ matrix

$$
T = \begin{bmatrix} G' & -A^T \\ A & 0 \end{bmatrix}
$$

obtain the $n$-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfying

$$
\begin{aligned}
\boldsymbol{y} &= T\boldsymbol{x} + \boldsymbol{q} && (5.4) \\
x_i &\geq 0 \ \text{ for } \ (i=1,\cdots,n) && (5.5) \\
y_i &\geq 0 \ \text{ for } \ (i=1,\cdots,n) && (5.6)
\end{aligned}
$$

First, for the linear complementarity problem in (5.4), (5.5) and (5.6), define the following equation for the $(2n+1)$ variables $(\boldsymbol{y}, \boldsymbol{x}, \xi)$

$$
\boldsymbol{y} - T\boldsymbol{x} - \boldsymbol{d}\xi = \boldsymbol{q}
$$

where, $\boldsymbol{d}$ is an $n$-dimensional constant vector for which all components are positive. Within $(\boldsymbol{y}, \boldsymbol{x}, \xi)$, the nonzero components are called basic variables and the others are called nonbasic variables.

Next, let the initial solution for the equation in (5.4) be $(\boldsymbol{y}, \boldsymbol{x}, \xi) = (\boldsymbol{q} + \boldsymbol{d}\xi_0, 0, \xi_0)$, where,

$$\begin{aligned} \xi_0 &= \max\{-\frac{q_i}{d_i} | i = 1, 2, \cdots, n\} \\ \eta &= \max\{-\frac{q_i}{d_i} | i = 1, 2, \cdots, n\} \end{aligned}$$

When the value of a certain basic variable becomes 0, if the basic variable at this time is $y_s$ remove $y_s$ from the basis, insert $\eta$ in the basis, and let $\eta = x_s$. If $x_s$ is a basis variable, remove $x_s$ from the basis, insert $\eta$ in the basis, and let $\eta = y_s$. Repeat this processing, and terminate the calculation when that basis variable becomes $\xi$.

Also, if the value of $\eta$ for which the expressions in (5.5) and (5.6) are satisfied can get infinitely large, terminate the calculation with no solution existing.

(3) Transformation of the solution
The solution that was obtained is transformed to a solution of the original quadratic programming problem before the equality constraints were removed.

#### 5.1.2.11 Minimization of an unconstrained 0-1 quadratic function of several variables

This section describes the problem of minimizing quadratic functions (unconstrained 0-1 quadratic programming problem) where every variables assumes a 0-1 variables, namely, only 0 or 1, as the value.

$$\begin{array}{lllll} \text{Objective function} & : & f(\boldsymbol{x}) & = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \to \min \\ \text{0-1 condition} & : & x_j = 0 \ or \ 1 & (j = 1, \cdots, n) & \end{array}$$

This library uses the branch-and-bound method to solve the unconstrained 0-1 quadratic programming problem.

For an unconstrained 0-1 quadratic programming problem, we use a following partial problem.

$$\begin{array}{lllll} \text{The objective function} & : & f(\boldsymbol{x}) & = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \to \min \\ \text{The 0-1 condition} & : & x_j = 0 & (j \in S^0) & \\ & & x_j = 1 & (j \in S^1) & \\ & & x_j = 0 \ or \ 1 & (j \in F) & \end{array}$$

Here, $S^0$, $S^1$, and $F$ are the set of subscripts of 0-1 variables having values fixed at 0, the set of subscripts of 0-1 variables having values fixed at 1, and the set of subscripts of 0-1 variables having values that are not fixed, respectively, in partial problem $P_k$. Now, 0-1 variables having values fixed at 0 or 1 are called fixed variables, and other 0-1 variables are called free variables. A partial problem is an unconstrained 0-1 quadratic programming problem in itself. In particular, the partial problem having $S^0 \cup S^1 = \phi$, that is, not having fixed variables, is the original mixed 0-1 programming problem (hereafter called $P_1$) itself.

In addition, the relaxation problem for partial problems is defined as follows:

$$\begin{array}{lllll} \text{The objective function} & : & f(\boldsymbol{x}) & = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \to \min \\ \text{The 0-1 condition} & : & x_j = 0 & (j \in S^0) & \\ & & x_j = 1 & (j \in S^1) & \\ & & 0 \le x_j \le 1 & (j \in F) & \end{array}$$

The method of obtaining the solutions of $P_1$ by the branch-and-bound method is explained below.

(1) Initial setting

(a) Preconditioning for coefficient matrix

To successfully solve the relaxation problem described bellow, the coefficient matrix of an objective function must be a positive definite matrix. Using the characteristic of 0-1 variable having

$$x_i^2 = x_i \qquad (i = 1, \ldots, n)$$

in the unconstrained 0-1 quadratic programming problem, the objective function can be rewritten into a quadratic function with symmetric matrix in its coefficient as follows:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G' \boldsymbol{x}$$

Where, $G'$ is as follows:

$$G' = \frac{1}{2}(G + G^T) + 2\mathrm{diag}(c_1, \ldots, c_n)$$

If $G'$ is a positive semi-definite matrix, $P_1$ has a self-evident optimal solution, as in the following.

$$x^* = (0, 0, \ldots, 0)^T$$

If $G'$ is not a positive semi-definite matrix, namely, $G'$ has a negative eigen value, the objective function can be rewritten into a quadratic function with a positive definite matrix $G''$ as the coefficient, as in the following,

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G'' \boldsymbol{x} + (\boldsymbol{c}'')^T \boldsymbol{x}$$

by defining the following with the minimal eigen value $\lambda_{min}$ of $G'$ and positive number parameter $\beta(> 0)$:

$$G'' = G' + (|\lambda_{min}| + \beta)\mathrm{diag}(1, 1, \ldots, 1)$$

$$\boldsymbol{c}'' = -\frac{1}{2}(|\lambda_{min}| + \beta)(1, 1, \ldots, 1)^T$$

where, $\beta$ is a minimal eigen value of matrix $G''$. In the following explanation, the coefficient matrix $G$ is assumed to have been converted into a positive definite matrix in advance through this kind of preconditioning.

(b) Partial problem list

The branch-and-bound method uses the partial problem list PLIST. In the initial state, the members of PLIST consist only of $P_1$, and the number of members of PLIST $L$ is 1. $L$ increases and decreases as the branch-and-bound method calculations proceed.

(c) Incumbent solution

In contrast to the case of a mixed 0-1 programming problem, the way to use a solution whose objective function is as small as possible is more efficient in searching the optimal solution by the branch-and-bound method, although the existence of an executable solution is self-evident. In this library, we use a solution obtained by two heuristic search methods, simulated annealing and tabu search, as an initial incumbent solution.

(d) Simulated annealing method

① Select an initial solution $\boldsymbol{x} = \boldsymbol{x}_0$ at random.

② Set the initial temperature to $T = T_0$.

③ Assume $z \leftarrow f(\boldsymbol{x}_0)$.

④ Assume $y \leftarrow f(\boldsymbol{x})$.

⑤ Assume $T \leftarrow \alpha \times T$ $(0 < \alpha \leq 1)$.

⑥ Select one subscript $i (1 \leq i \leq n)$ at random.

⑦ Assume $x_i \leftarrow 1 - x_i$.

⑧ If $y < f(\boldsymbol{x})$, select a real number $r \in [0, 1]$ randomly. If $r > \exp(-1/T)$, assume $x_i \leftarrow 1 - x_i$.

⑨ Assume $z \leftarrow \min\{y, z\}$.

⑩ Repeat processing from ④ to ⑨ for the specified count. Assume that $z$, which is ultimately obtained, is the objective function value of the initial incumbent solution by the branch-and-bound method.

(e) Tabu search method

① Select an initial solution $\boldsymbol{x} = \boldsymbol{x}_0$ at random.

② Assume $K = \phi$ tabu list.

③ Assume $z \leftarrow f(\boldsymbol{x}_0)$.

④ Assume $U(\boldsymbol{x}) = \{\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(n)}\}$ where $\boldsymbol{u}^{(i)} = (x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n)^T$.

⑤ Assume $\boldsymbol{x} \leftarrow \min_{U \setminus K} \boldsymbol{u}_i$.

⑥ Assume $K \leftarrow K \cup \{\boldsymbol{x}\}$. If the number of $K$'s elements exceeds the specified number, remove the element the most previously added to $K$.

⑦ Assume $z \leftarrow \min\{f(\boldsymbol{x}), z\}$.

⑧ Repeat processing from ④ to ⑦ for the specified count. Assume that $z$, which is ultimately obtained, is the objective function value of the initial incumbent solution by the branch-and-bound method.

By the tabu search method, you can obtain a workable solution, although considerable time is needed. The computation time by the simulated annealing method is relatively short. In addition to those search methods, this library supports a method by which a primary solution obtained by the simulated annealing method is improved by the tabu search. If $n$, the number of variables, is relatively large, it is hard to obtain an exact optimal solution by the branch-and-bound method described later. However, in most cases, a solution obtained by a heuristic search has ample approximate accuracy. Therefore, this library also supports a method by which any branch-and-bound method calculation is omitted because a solution obtained by a heuristic search is used as an approximate solution as is.

(2) Lower bound value calculation

Calculate the lower bound value of an optimal value using the corresponding relaxation problem as follows:

(a) Assume that the eigen values of coefficient matrix $G$ are $\lambda_1, \ldots, \lambda_n$ and the corresponding orthonormalized eigen vectors $\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(n)}$.

(b) Solve the convex quadratic programming problem, which is a relaxation problem, using the G-I method, and assume that its optimal solution is $\bar{\boldsymbol{x}}$.

(c) If free variable $x_i (i \in F)$ satisfies the following, $x_i$ is fixed as zero.

$$f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{\bar{x}_i^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right| \leq z^* < f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{(1-\bar{x}_i)^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right|$$

Then, assume $S_0 \leftarrow S_0 \cup \{i\}, F \leftarrow F - \{i\}$.

(d) If free variable $x_i (i \in F)$ satisfies the following, $x_i$ is fixed as one.

$$f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{(1-\bar{x}_i)^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right| \leq z^* < f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{\bar{x}_i^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right|$$

Then, assume $S_1 \leftarrow S_1 \cup \{i\}, F \leftarrow F - \{i\}$.

If some free variables are fixed by the above operation, solve the relaxation problem again and repeat it until there are no more free variables to be newly fixed.

(e) Coordinate transformation

Define the $U_i^0$ and $U_i^1$ sub sets of the space of variable $\hat{\boldsymbol{x}}$ defined with

$$x_i - \bar{x}_i \leftarrow \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j$$

as follows:

$$U_i^0 = \{\hat{\boldsymbol{x}} | \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j = 0\}$$

$$U_i^1 = \{\hat{\boldsymbol{x}} | \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j = 1\}$$

As for the set of subscripts for an arbitrary free variable $\{i_1, i_2, \ldots, i_r\}$, if

$$D = \min\{|\hat{\boldsymbol{x}}| | \boldsymbol{x} \in \cap_{k=1}^{r} (U_{i_k}^0 \cup U_{i_k}^1)\}$$

is assumed,

$$g(\bar{\boldsymbol{x}}) = f(\bar{\boldsymbol{x}}) + \frac{D^2}{2}$$

provides the lower bound value of a partial problem. In this library, $r$ portions from a larger one of those among free variables obtained form the following formula are used to calculate $D$.

$$\frac{\min\{\bar{x}_i, 1 - \bar{x}_i\}^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}$$

If the optimal value $g(P_i)$ for the solutions of the relaxation problem corresponding to partial problem $P_i$ satisfies $g(P_i) > z^*$, $P_i$ is removed from PLIST and $L \leftarrow L - 1$ is performed. Also, when the optimal solution is represented by $\bar{\boldsymbol{x}}$, if the value of $\bar{\boldsymbol{x}}_j$ is 0 or 1 for all $j \in F$, the solutions of the relaxation problem are solutions of the original partial problem, and the optimal value of the partial problem is equal to the optimal value $g(P_i)$ of the relaxation problem. At this time, if $f(P_i) < z^*$, $\boldsymbol{x}^*$ and $z^*$ are replaced by $\bar{\boldsymbol{x}}$ and $f(P_i)$.

Now, a new relaxation problem must be solved when the initial settings described above are made and when $L$ is increased by a branching operation, which is described later. In the latter case, the last two partial problems in PLIST are the pair of partial problems $P_{i0}$, for which the branching variable $x_{j_0}$ is fixed at 0, and $P_{i1}$, for which it is fixed at 1.

(3) Branching operation

$P_k$ is removed from PLIST. Then, one branching variable $x_{j_0}$ is selected from the free variables that have not been fixed by the partial problem test described above, the partial problems $P_{k0}$, for which $j_0$ was added to $S^0$, and $P_{k1}$, for which $j_0$ was added to $S^1$, are added at the end of PLIST, and $L \leftarrow L + 1$ is performed. The branching variable $x_{j_0}$ is determined here as follows.

$$j_0 = \mathrm{argmin}_{j \in F} \left| \bar{x}_j - \frac{1}{2} \right|$$

After the branching operation, calculate the lower bound of the optimal value of each partial problem for $P_{k0}$ and $P_{k1}$.

(4) PLIST sorting
Assume $K' = \min\{K, L\}$. Here, $K$ is called the "search depth in the branch-and-bound method" and is a positive integer parameter. The last $K'$ portions of partial problems $P_i$in PLIST are sorted in descending order of the lower bound value $g(P_i)$.

(5) Partial problem test
For the last partial problem $P_k$ in PLIST, when all free variables have been fixed, the partial problem is solved, $P_k$ is removed from PLIST, and $L \leftarrow L-1$ is performed. At this time, if $f(P_k) < z^*$, $\boldsymbol{x}^*$ and $z^*$ are replaced by the solution of $P_k$ and $f(P_k)$. Then, processing proceeds with step (6) according to the updated PLIST. If there are any remaining free variables that have not been fixed, processing proceeds with the next branching operation.

(6) Termination condition
If $L = 0$, processing for the branch-and-bound method terminates, and the incumbent at that time $x^*$ is the solution of $P_1$, that is, of the original mixed 0-1 programming problem. Otherwise, processing returns to step (2).

#### 5.1.2.12 Minimization of a constrained function of several variables

Here, given a function $f(\boldsymbol{x})$ of $n$ variables, we will deal with the problem of obtaining the point $\boldsymbol{x}^*$ (optimal solution) that minimizes it based on the $(m+\ell)$ given constraints

$$\begin{cases} g_i(\boldsymbol{x}) & \leq & 0 & \text{for} \ \ (i=1,\cdots,m) \\ h_j(\boldsymbol{x}) & = & 0 & \text{for} \ \ (j=1,\cdots,\ell) \end{cases} \tag{5.7}$$

Global minimization generally is difficult to achieve. This library deals only with local minimization. Therefore, in the explanation below, the term "minimization" is used only in the local sense. Similarly, the local optimal solution $\boldsymbol{x}^*$ is referred to simply as the optimal solution.
Now, if we define the penalty function $\theta_\delta(\boldsymbol{x})$ as

$$\theta_\delta(\boldsymbol{x}) \quad = \quad f(\boldsymbol{x}) + \delta \max\left(0, g_1(\boldsymbol{x}), \cdots, g_m(\boldsymbol{x}), \mid h_1(\boldsymbol{x}) \mid, \cdots, \mid h_\ell(\boldsymbol{x}) \mid\right) \ \ \text{for} \ \ (\delta > 0)$$

then the following holds for an arbitrary $\boldsymbol{x}$

$$\theta_\delta(\boldsymbol{x}) \quad \geq \quad f(\boldsymbol{x})$$

and, in particular, when $\boldsymbol{x}$ is a point that satisfies the constraints given above, the following relationship holds

$$\theta_\delta(\boldsymbol{x}) \quad = \quad f(\boldsymbol{x})$$

Therefore, the problem of minimizing $f(\boldsymbol{x})$ under constraints (5.7) is reduced to the problem of minimizing $\theta_\delta(\boldsymbol{x})$ without constraints.
$\theta_\delta(\boldsymbol{x})$ is a strict penalty function, and if the quasi-Newton method is applied to it, the optimal solution can be obtained theoretically. However, since $\theta_\delta(\boldsymbol{x})$ mathematically has unstable properties, this library uses an algorithm called the sequential quadratic programming method to perform iterative improvement of a solution, which resembles the quasi-Newton method for the Lagrange function

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad = \quad f(\boldsymbol{x}) + \sum_{i=1}^{m}(\lambda_i g_i(\boldsymbol{x})) + \sum_{j=1}^{\ell}(\mu_j h_j(\boldsymbol{x}))$$

$\theta_\delta(\boldsymbol{x})$ is used as the linear search evaluation function. The computation procedure is as follows.

(1) Partial quadratic programming problem
when the $k$-th approximation $\boldsymbol{x}_k$ of optimal solution $\boldsymbol{x}^*$ is obtained, the following quadratic programming problem

$$\begin{array}{lll} \text{Objective function} & : & \frac{1}{2}\boldsymbol{d}^T B_k \boldsymbol{d} + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} \\ \text{Inequality constraints} & : & g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{d} \leq 0 \qquad \text{for} \quad (i = 1, \cdots, m) \\ \text{Equality constraints} & : & h_j(\boldsymbol{x}_k) + \nabla h_j(\boldsymbol{x}_k)^T \boldsymbol{d} = 0 \qquad \text{for} \quad (j = 1, \cdots, \ell) \end{array}$$

which is obtained by expanding the objective function and constraints around $\boldsymbol{x}_k$ is solved, and that solution is denoted as $\boldsymbol{d}_k$. However, since calculating the Hessian $\nabla^2 f(\boldsymbol{x}_k)$ is difficult mathematically, it is replaced by its approximation matrix $B_k$ in a similar manner as in the quasi-Newton method.
The Goldfarb-Idnani method is used as the algorithm for solving this quadratic programming problem.
If $\boldsymbol{d}_k = 0$, the calculation terminates with $\boldsymbol{x}^* = \boldsymbol{x}_k$. The Lagrange multiplier vectors obtained here are set to $\boldsymbol{\lambda}_{k+1}$ and $\boldsymbol{\mu}_{k+1}$.

(2) Linear search
Search for $\boldsymbol{x}_{k+1}$ with $\boldsymbol{d}_k$ as the search direction. Start with $\alpha_k = 1$, and if the following condition is satisfied

$$\theta_\delta(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \quad \leq \quad \theta_\delta(\boldsymbol{x}_k) - \omega \alpha_k \boldsymbol{d}_k^T B_k \boldsymbol{d}_k$$

proceed to the processing in (c). If the condition is not satisfied, repeatedly replace $\alpha_k$ so that $\tau\alpha_k \to \alpha_k$ until the condition is satisfied.

(3) Updating of $\boldsymbol{x}_k$

$$\boldsymbol{x}_{k+1} \quad = \quad \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

(4) Updating of $B_k$
Since the positive definite property of $B_k$ is not maintained in the original BFGS formula, $B_k$ is updated by the modified BFGS formula as shown below.

$$B_{k+1} \quad = \quad B_k - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k} + \frac{\boldsymbol{\eta}_k \boldsymbol{\eta}_k^T}{\boldsymbol{s}_k^T \eta_k}$$

where,

$$\begin{array}{rcl} \boldsymbol{s}_k & = & \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \\ \boldsymbol{y}_k & = & \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}) - \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_k, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}) \\ \phi & = & \begin{cases} 1 & \text{for} \quad \boldsymbol{s}_k^T \boldsymbol{y}_k \geq 0.2 \boldsymbol{s}_k^T \\ \dfrac{0.8 \boldsymbol{s}_k^T B_k \boldsymbol{s}_k}{\boldsymbol{s}_k^T (B_k \boldsymbol{s}_k - \boldsymbol{y}_k)} & \text{otherwise} \end{cases} \\ \boldsymbol{\eta}_k & = & \phi \boldsymbol{y}_k + (1 - \phi) B_k \boldsymbol{s}_k \end{array}$$

(5) Termination of updating
The updating of $\boldsymbol{x}_k$ is repeated until the Karush-Kuhn-Tucker condition

$$\begin{array}{rcl} \nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} (\lambda_i \nabla g_i(\boldsymbol{x})) + \sum_{j=1}^{\ell} (\mu_j \nabla h_j(\boldsymbol{x})) & = & 0 \\ g_i(\boldsymbol{x}) & = & 0 \quad \text{for} \quad (i = 1, \cdots, m) \end{array}$$

$$
\begin{aligned}
h_j(\boldsymbol{x}) &= 0 \text{ for } (j = 1, \cdots, \ell) \\
\sum_{i=1}^{m} (\lambda_i g_i(\boldsymbol{x})) &= 0 \\
\lambda_i &\geq 0 \text{ for } (i = 1, \cdots, m)
\end{aligned}
$$

is satisfied.

#### 5.1.2.13 Minimization of the distance between two nodes in a network

(1) Calculating the shortest path from a given node to all other nodes
The Dijkstra's method is used to obtain the shortest path from a given specified node init on graph $G = (V, E)$ to all other nodes and the corresponding distance. However, the branch weights are assumed to be nonnegative.

(a) For each vertex $i \in V$, let $Distance(i) = \infty$, $Path(i) = init$, and $P = \phi$. For the starting point $init$, let $Distance(init) = 0$. Also, let $next = init$.

(b) Let $P = P \cup \{next\}$. For each node $j$ that is connected to node next, if $Distance(j) > Distance(next) + Weight(next, j)$, update $Distance(j) = Distance(next) + Weight(next, j)$ and $Path(j) = next$.

(c) For each vertex $i \in P$, select node $v$ for which $Distance(i)$ is the minimum. Let $next = v$ for that node.

(d) Repeat steps (ii) and (iii) until $P = V$.

(2) Calculating the shortest path between all sets of two nodes The Floyd's method is used to obtain the shortest path between all sets of two nodes on graph $G = (V, E)$ and the corresponding distance.
For undirected graphs:Assume that no negative weighted branches are included.

(a) For each pair of nodes $i, j \in V$ , let $Distance(i, j) = \infty$ and $Path(i, j) = i$.

(b) For each pair of nodes $i, j$, if $Distance(i, j) > Distance(i, k) + Distance(k, j)$, update $Distance(i, j) = Distance(i, k) + Distance(k, j)$ and $Path(i, j) = k$.

(c) Repeat step (ii) for $k = 1, \ldots, n$.

For directed graphs:Assume that negative weighted branches are included but cycles with negative lengths are not included.

(a) For each pair of nodes $i, j \in V$, let $Distance(i, j) = \infty$ and $Path(i, j) = 0$.

(b) For each pair of nodes $i, j \in V$, if $Distance(i, j) > Distance(i, k) + Distance(k, j)$, update $Distance(i, j) = Distance(i, k) + Distance(k, j)$ and $Path(i, j) = k$.

(c) If $Distance(i, i)$ is negative, interrupt processing since no solution exists. Otherwise, repeat step (ii) for $k = 1, \ldots, n$.

(3) Calculating the shortest path between two nodes
The Dijkstra's method described above is applied to obtain the shortest path between two nodes on graph $G = (V, E)$ and the corresponding distance. However, the branch weights are assumed to be nonnegative.

(a) For each vertex $i \in V$, let $Distance(i) = \infty$, $Path(i) = 0$, and $P = j$. For the starting point $init$, let $Distance(init) = 0$. Also, let $next = init$.

(b) Let $P = P \cup next$. For each node $j$ that is connected to node next, if $Distance(j) > D(next) + Weight(next, j)$, update $Distance(j) = D(next) + Weight(next, j)$ and $Path(j) = next$.

(c) For each vertex $i \in P$, select node $v$ for which $Distance(i)$ is the minimum. Let $next = v$ for that node.

(d) Repeat steps (ii) and (iii) until $next = end$.

### 5.1.3 Reference Bibliography

(1) Forsythe, G. E. , Malcolm, M. A. and Moler, C. B. , “Computer method for mathematical computations”, Prentice-Hall Inc. , (1978).

(2) Brent, R. P. , “Algorithms for minimization without derivatives”, Englewood Cliffs, N. J. , Prent-Hall, (1973).

(3) Powell, M. J. D. , “A Hybrid Method for Nonlinear Equations”, Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowits, ed. , Gordon and Breach, pp.87-161, (1970).

# 5.2 MINIMIZATION OF A FUNCTION OF ONE VARIABLE WITHOUT CONSTRAINTS

## 5.2.1 DMUUSN, RMUUSN
## Minimization of a Function of One Variable

(1) **Function**

DMUUSN or RMUUSN searches for the minimum value of a function $f(x)$ of one variable.

(2) **Usage**

Double precision:

CALL DMUUSN (F, X, ER, NEV, Y, IERR)

Single precision:

CALL RMUUSN (F, X, ER, NEV, Y, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\begin{cases}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{cases}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | – | Input | Name of function subprogram that defines the function $f(x)$. (See Notes (a)) |
| 2 | X | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Search starting point $x_0$ |
| | | | | Output | Final destination $x^*$ |
| 3 | ER | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{\text{(Unit for determining error)}}$) |
| 4 | NEV | I | 1 | Input | Maximum number of evaluations of function $f(x)$ (Default value: 100) |
| | | | | Output | Actual number of function evaluations |
| 5 | Y | $\begin{Bmatrix}D\\R\end{Bmatrix}$ | 1 | Output | Function value $y = f(x^*)$ at final destination $x^*$ |
| 6 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(b) NEV $> 3$
(except when 0.0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (a) or (b) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 4000 | The enclosure operation failed. | Processing is aborted. |
| 5000 | The root could not be obtained before maximum number of function evaluations was reached. | The values of X and Y at that time are output and processing is aborted. |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. This function subprogram (in double-precision) should be created as follows.
· Function subprogram creation method:

```
REAL(8)  FUNCTION   F   (X)
REAL(8)  X

    F = f(x)

RETURN
END
```

(b) If the search interval becomes $[a, b]$ due to interval reduction, then convergence is determined according to the following condition:

$$\max(b - x, x - a) \le 2 \times \mathrm{ER} \times \max(1, \mid x \mid).$$

A value on the order of the default value should be taken for ER.

(c) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) If there are multiple minimum values, you cannot guarantee to which minimum value the subroutine will converge.

(e) The function must continuously first-order differentiable.

(f) To search for a maximum value, set the function value $f(x)$ so that the sign will be reverse. At this time, the value of the fifth argument Y will be output with the sign reversed.

(7) **Example**

(a) Problem
Search for the minimum value of the function:

$$f(x) = x(x^2 - 2) - 5$$

(b) Input data
Function subprogram name corresponding to function $f(x)$: FMUUSN
X = 1.0, ER=0.0 and NEV=0.

(c) Main program

```
      PROGRAM BMUUSN
! *** EXAMPLE OF DMUUSN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FMUUSN
!
      WRITE(6,1000)
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1100) NEV,ER,X
      CALL DMUUSN(FMUUSN,X,ER,NEV,Y,IERR)
      WRITE(6,1200) IERR,NEV,X,Y
      STOP
!
 1000 FORMAT(' ',/,' *** DMUUSN ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'NEV  =',I5,/,&
             5X,'ER   =',D18.10,/,&
             5X,'(( INITIAL VALUE ))',/,&
             5X,'   X =',F5.1)
 1200 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR =',I5,/,&
             5X,'NEV  =',I5,/,&
             5X,'(( SOLUTION ))',/,&
             5X,'   X =',D18.10,/,&
             5X,'(( FUNCTION VALUE ))',/,&
             5X,'   Y =',D18.10)
      END


      REAL(8) FUNCTION FMUUSN(X)
      REAL(8) X
!
      FMUUSN = X*(X*X-2.0D0)-5.0D0
      RETURN
      END
```

(d) Output results

```
*** DMUUSN ***
 ** INPUT **
    NEV  =     0
    ER   =  0.0000000000D+00
    (( INITIAL VALUE ))
       X =  1.0
 ** OUTPUT **
    IERR =     0
    NEV  =    23
    (( SOLUTION ))
       X =  0.8090169944D+00
    (( FUNCTION VALUE ))
       Y = -0.6088525492D+01
```

## 5.3 MINIMIZATION OF A FUNCTION OF MANY VARIABLES WITHOUT CONSTRAINTS

### 5.3.1 DMUMQN, RMUMQN
### Minimization of a Function of Many Variables (Derivative Definition Unnecessary)

(1) **Function**

DMUMQN or RMUMQN searches for the minimum value of a function $f(\boldsymbol{x})$ of $n$ variables.

(2) **Usage**

Double precision:

CALL DMUMQN (F, X, N, ER, NEV, Y, WK, IERR)

Single precision:

CALL RMUMQN (F, X, N, ER, NEV, Y, WK, IERR)

(3) **Arguments**

D:Double precision real   Z:Double precision complex   I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$

R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | − | Input | Name of function subprogram F(X) that defines the function $f(\boldsymbol{x})$. (See Notes (a)) |
| 2 | X | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Input | Search starting point $\boldsymbol{x}_0$ |
| | | | | Output | Final destination $\boldsymbol{x}^*$ |
| 3 | N | I | 1 | Input | Number of components $n$ of independent variable $\boldsymbol{x}$ |
| 4 | ER | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{(\text{Unit for determining error})}$) |
| 5 | NEV | I | 1 | Input | Maximum number of evaluations of function $f(\boldsymbol{x})$ (Default value: $400 \times \text{N}$) |
| | | | | Output | Actual number of function evaluations |
| 6 | Y | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Function value $y = f(\boldsymbol{x}^*)$ at final destination $\boldsymbol{x}^*$ |
| 7 | WK | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | See Contents | Work | Work area **Size**: $\text{N} \times (3 \times \text{N} + 7)$ |
| 8 | IERR | I | 1 | Output | Error indicator |

**(4) Restrictions**

(a) $N > 0$

(b) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(c) NEV $> 0$
(except when 0.0 is entered in order to set NEV to the default value)

**(5) Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (b) or (c) was not satisfied. | Processing is performed with the default value set for NEV or ER |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 5000 | The root could not be obtained before maximum number of function evaluations was reached. | The values of X and Y at that time are output and processing is aborted. |

**(6) Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. This function subprogram (in double-precision) should be created as follows.

```
REAL(8)  FUNCTION   F   (X)
REAL(8)  X
      F = f(x)
RETURN
END
```

(b) Convergence is determined according to the following condition, and the solution is assumed to be $\boldsymbol{x} + \Delta\boldsymbol{x}$:

$$\|\Delta\boldsymbol{x}\| \leq \mathrm{ER} \times \max(1, \|\boldsymbol{x} + \Delta\boldsymbol{x}\|)$$

and

$$\|\nabla f(\boldsymbol{x})\| \leq 2 \times \mathrm{ER}$$

or

$$\|\nabla f(\boldsymbol{x})\| \leq \text{Unit for determining error}$$

where $\Delta\boldsymbol{x}$ is the correction vector for $\boldsymbol{x}$ and $\|\boldsymbol{x}\| = \max_i \mid x_i \mid$. Also, $\nabla f(\boldsymbol{x})$, which is the gradient vector of $f(\boldsymbol{x})$, has components $\partial f(\boldsymbol{x})/\partial x_i$. A value on the order of the default value should be taken for ER.

(c) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) If the gradient vector can be calculated analytically, then it is more efficient to use the subroutine 5.3.2 $\left\{\begin{matrix}\text{DMUMQG}\\\text{RMUMQG}\end{matrix}\right\}$.

(e) Scaling should be performed so that the contribution to the function value from each of the variables is on the same order. (See Section 5.1.1)

(f) If the gradient vector is 0 at the initial point, then that point is output as the solution.

(g) If there is no minimum value, then processing will be continued until the maximum number of function evaluations is reached and IERR = 5000 will be output.

(h) If there are multiple minimum values, you cannot guarantee to which minimum value the subroutine will converge.

(i) The function must continuously second-order differentiable.

(j) To search for a maximum value, set the function value $f(\boldsymbol{x})$ so that the sign will be reversed. At this time, the value of the sixth argument Y will be output with the sign reversed.

(7) **Example**

(a) Problem
Search for the minimum value of the function

$$f(\boldsymbol{x}) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2,$$

using $\boldsymbol{x} = [-1.2, 1.0]^T$ as the initial value.

(b) Input data
Function subprogram name corresponding to function $f(\boldsymbol{x})$: FMUMQN
X(1) = −1.2, X(2)=1.0, ER=0.0 and NEV=0.

(c) Main program

```
      PROGRAM BMUMQN
! *** EXAMPLE OF DMUMQN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N = 2)
      DIMENSION X(N),WK(N*(3*N+7))
      EXTERNAL  FMUMQN
!
      WRITE(6,1000)
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1100) N,NEV,ER,(I,X(I),I=1,N)
      CALL DMUMQN(FMUMQN,X,N,ER,NEV,Y,WK,IERR)
      WRITE(6,1200) IERR,NEV
      WRITE(6,1300) (I,X(I),I=1,N)
      WRITE(6,1400) Y
      STOP
!
 1000 FORMAT(' ',/,' *** DMUMQN ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'N     =',I5,/,&
             5X,'NEV   =',I5,/,&
             5X,'ER    =',D18.10,/,&
             5X,'(( INITIAL VALUE ))',/,&
            (5X,'   X(',I2,') =',F5.1))
 1200 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR  =',I5,/,&
             5X,'NEV   =',I5)
 1300 FORMAT(5X,'(( SOLUTION ))',/,&
            (5X,'   X(',I2,') =',D18.10))
 1400 FORMAT(5X,'(( FUNCTION VALUE ))',/,&
             5X,'   Y     =',D18.10)
      END

      REAL(8) FUNCTION FMUMQN(X)
      REAL(8) W,X,Y
      DIMENSION X(*),Y(2)
!
      Y(1) = 10.0D0*(X(2)-X(1)*X(1))
```

```
      Y(2) = 1.0D0-X(1)
      W = 0.0D0
      DO 100 I=1,2
         W = W+Y(I)*Y(I)
  100 CONTINUE
      FMUMQN = W
      RETURN
      END
```

(d) Output results

```
*** DMUMQN ***
 ** INPUT **
    N     =    2
    NEV   =    0
    ER    =  0.0000000000D+00
    (( INITIAL VALUE ))
       X( 1) = -1.2
       X( 2) =  1.0
 ** OUTPUT **
    IERR  =    0
    NEV   =  140
    (( SOLUTION ))
       X( 1) =  0.1000000000D+01
       X( 2) =  0.1000000000D+01
    (( FUNCTION VALUE ))
       Y     =  0.5453357277D-23
```

## 5.3.2 DMUMQG, RMUMQG Minimization of a Function of Many Variables (Derivative Definition Required)

(1) **Function**

DMUMQG or RMUMQG searches for the minimum of a function $f(\boldsymbol{x})$ of $n$ variables.

(2) **Usage**

Double precision:

CALL DMUMQG (F, SUBG, X, N, ER, NEV, Y, WK, IERR)

Single precision:

CALL RMUMQG (F, SUBG, X, N, ER, NEV, Y, WK, IERR)

(3) **Arguments**

D:Double precision real　Z:Double precision complex　I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

R:Single precision real　C:Single precision complex

| No. | Argument | Type | Size | Input/Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | – | Input | Name of function subprogram F(X) that defines the function $f(\boldsymbol{x})$. (See Notes (a)) |
| 2 | SUBG | – | – | Input | Name of subroutine SUBG(X, G) that calculates the gradient vector $\nabla f(\boldsymbol{x})$. (See Notes (b)) |
| 3 | X | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | N | Input | Search starting point $\boldsymbol{x}_0$ |
| | | | | Output | Search point final destination $\boldsymbol{x}^*$ |
| 4 | N | I | 1 | Input | Number of components $n$ of independent variable $\boldsymbol{x}$ |
| 5 | ER | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{\text{(Unit for determining error)}}$) |
| 6 | NEV | I | 2 | Input | NEV(1): Maximum number of evaluations of function F (Default value: 100 × N)<br>NEV(2): Maximum number of evaluations of subroutineSUBG (Default value: 100 × N) |
| | | | | Output | Actual number of function evaluations |
| 7 | Y | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | 1 | Output | Function value $y = f(\boldsymbol{x}^*)$ at final destination $\boldsymbol{x}^*$ |
| 8 | WK | $\left\{\begin{array}{c}D\\R\end{array}\right\}$ | See Contents | Work | Work area<br>**Size**: $N \times (3 \times N + 7)$ |
| 9 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N > 0

(b) ER ≥ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(c) NEV(i) > 0  $(\mathrm{i} = 1, 2)$
(except when 0.0 is entered in order to set NEV(i)  $(\mathrm{i} = 1, 2)$to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (b) or (c) was not satisfied. | Processing is performed with the default value set for NEV(1), NEV(2) or ER |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 5000 | The root could not be obtained before maximum number of function evaluations was reached. | The values of X and Y at that time are output and processing is aborted. |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. This function subprogram (in double-precision) should be created as follows.

```
REAL(8)  FUNCTION    F    (X)
REAL(8)  X
DIMENSION X(*)

      F = f(x)

RETURN
END
```

(b) The actual name in the second argument SUBG must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for SUBG must be created. This function subprogram (in double-precision) should be created as follows.

```
SUBROUTINE    SUBG (X, G)
REAL(8) X, G
DIMENSION    X(*), G(*)

      G(1)  = 1st component of ∇f(x) = ∂f(x)∂x_i
             ⋮
      G(N)  = n-th component of ∇f(x) = ∂f(x)∂x_n

RETURN
END
```

(c) Convergence is determined according to the following condition, and the solution is assumed to be $\boldsymbol{x} + \Delta\boldsymbol{x}$:
$\|\Delta\boldsymbol{x}\| \leq \mathrm{ER} \times \max(1, \|\boldsymbol{x} + \Delta\boldsymbol{x}\|)$ and $\|\nabla f(\boldsymbol{x})\| \leq 2 \times \mathrm{ER}$ or $\|\nabla f(\boldsymbol{x})\| \leq$ Unit for determining error where $\Delta\boldsymbol{x}$ is the correction vector for $\boldsymbol{x}$ and $\|\boldsymbol{x}\| = \max_i \mid x_i \mid$. A value on the order of the default value should be taken for ER.

(d) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(e) Scaling should be performed so that the contribution to the function value from each of the variables is on the same order. (See Section 5.1.1)

(f) If the gradient vector is 0 at the initial point, then that point is output as the solution.

(g) If there is no minimum value, then processing will be continued until the maximum number of function evaluations is reached and IERR = 5000 will be output.

(h) If there are multiple minimum values, you cannot guarantee to which minimum value the subroutine will converge.

(i) The function must continuously second-order differentiable.

(j) To search for a maximum value, set the function value $f(\boldsymbol{x})$ so that the sign will be reversed. At this time, the value of the seventh argument Y will be output with the sign reversed.

(7) **Example**

(a) Problem
Search for the minimum value of the function.

$$f(\boldsymbol{x}) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2$$

using $\boldsymbol{x} = [-1.2, 1.0]^T$ as the initial value.

(b) Input data
Function subprogram name corresponding to function $f(\boldsymbol{x})$: FMUMQG
Name of subroutine which calculates the gradient vector $\nabla f(\boldsymbol{x})$: GMUMQG
X(1) = −1.2, X(2)=1.0, N=2, ER=0.0, NEV(1)=0 and NEV(2)=0.

(c) Main program

```
      PROGRAM BMUMQG
! *** EXAMPLE OF DMUMQG ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N = 2)
      DIMENSION NEV(2),X(N),WK(N*(3*N+7))
      EXTERNAL  FMUMQG,FMUMQ2
!
      WRITE(6,1000)
      READ(5,*) (NEV(I),I=1,2)
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1100) N,NEV(1),NEV(2),ER,(I,X(I),I=1,N)
      CALL DMUMQG(FMUMQG,FMUMQ2,X,N,ER,NEV,Y,WK,IERR)
      WRITE(6,1200) IERR,NEV(1),NEV(2)
      WRITE(6,1300) (I,X(I),I=1,N)
      WRITE(6,1400) Y
      STOP
!
 1000 FORMAT(' ',/,' *** DMUMQG ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'N      =',I5,/,&
             5X,'NEV(1)=',I5,/,&
             5X,'NEV(2)=',I5,/,&
             5X,'ER     =',D18.10,/,&
             5X,'(( INITIAL VALUE ))',/,&
            (5X,'   X(',I2,') =',F5.1))
 1200 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR  =',I5,/,&
```

```
             5X,'NEV(1)=',I5,/,&
             5X,'NEV(2)=',I5)
 1300 FORMAT(5X,'(( SOLUTION ))',/,&
            (5X,'    X(',I2,') =',D18.10))
 1400 FORMAT(5X,'(( FUNCTION VALUE ))',/,&
             5X,'    Y      =',D18.10)
      END

      REAL(8) FUNCTION FMUMQG(X)
      REAL(8) W,X,Y
      DIMENSION X(*),Y(2)
!
      Y(1) = 10.0D0*(X(2)-X(1)*X(1))
      Y(2) = 1.0D0-X(1)
      W = 0.0D0
      DO 100 I=1,2
         W = W+Y(I)*Y(I)
  100 CONTINUE
      FMUMQG = W
      RETURN
      END

      SUBROUTINE FMUMQ2(X,G)
      REAL(8) G,X,Y
      DIMENSION X(*),G(*),Y(2)
!
      Y(1) = 10.0D0*(X(2)-X(1)**2)
      Y(2) = 1.0D0-X(1)
      G(1) =-2.0D0*(20.0D0*X(1)*Y(1)+Y(2))
      G(2) = 20.0D0*Y(1)
      RETURN
      END
```

(d) Output results

```
*** DMUMQG ***
 ** INPUT **
    N     =    2
    NEV(1)=    0
    NEV(2)=    0
    ER    =  0.0000000000D+00
    (( INITIAL VALUE ))
       X( 1) = -1.2
       X( 2) =  1.0
 ** OUTPUT **
    IERR  =    0
    NEV(1)=   55
    NEV(2)=   36
    (( SOLUTION ))
       X( 1) =  0.1000000000D+01
       X( 2) =  0.1000000000D+01
    (( FUNCTION VALUE ))
       Y     =  0.7664560722D-24
```

## 5.4 MINIMIZATION OF THE SUM OF THE SQUARES OF A FUNCTION WITHOUT CONSTRAINTS

### 5.4.1 DMUSSN, RMUSSN
### Nonlinear Least Squares Method (Derivative Definition Unnecessary)

(1) **Function**

DMUSSN or RMUSSN searches for the minimum value of the sum of the squares $s = \sum_{i=1}^{m} f_i(\boldsymbol{x})^2$ of a function of $n$ variables, where $f_i(\boldsymbol{x})$ is the $i$-th component of the vector function $\boldsymbol{f}(\boldsymbol{x})$ of $n$ variables ($i = 1, \cdots, m$).

(2) **Usage**

Double precision:

CALL DMUSSN (SUB, X, N, ER, NEV, Y, M, S, IWK, WK, IERR)

Single precision:

CALL RMUSSN (SUB, X, N, ER, NEV, Y, M, S, IWK, WK, IERR)

## (3) **Arguments**

| | | |
|---|---|---|
| D:Double precision real | Z:Double precision complex | I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$ |
| R:Single precision real | C:Single precision complex | |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | SUB | – | – | Input | Name of subroutine SUB(X, Y) that calculates the function value $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. (See Notes (a)) |
| 2 | X | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N | Input | Search starting point $\boldsymbol{x}_0$ |
| | | | | Output | Search point final destination $\boldsymbol{x}^*$ |
| 3 | N | I | 1 | Input | Number of components $n$ of independent variable $\boldsymbol{x}$ |
| 4 | ER | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{(\text{Unit for determining error})}$) |
| 5 | NEV | I | 1 | Input | Maximum number of evaluations of function $\boldsymbol{f}(\boldsymbol{x})$ (Default value: 100 ×N) |
| | | | | Output | Actual number of function evaluations |
| 6 | Y | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | M | Output | Function value $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}^*)$ at final destination $\boldsymbol{x}^*$ |
| 7 | M | I | 1 | Input | Number of components $m$ of dependent variable $\boldsymbol{y}$ |
| 8 | S | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | The sum of the squares of the function at the final destination $\boldsymbol{x}^*$: $\text{s}=\sum_{i=1}^{m} f_i(\boldsymbol{x}^*)^2$ |
| 9 | IWK | I | $4 \times \text{N}$ | Work | Work area |
| 10 | WK | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | See Contents | Work | Work area **Size**: $\text{M} \times (2 \times \text{N} + 1) + \text{N} \times (\text{N} + 4)$ |
| 11 | IERR | I | 1 | Output | Error indicator |

## (4) **Restrictions**

(a) $0 < \text{N} \leq \text{M}$

(b) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(c) NEV $> 0$
(except when 0.0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (b) or (c) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | The linear least squares method could not be solved. | The value of X, Y, and S at that time are output and processing is aborted. |
| 4100 | The steepest descent could not be calculated. | |
| 4200 | The solution could not be corrected 2×N times consecutively. | |
| 5000 | The values did not converge before the given maximum number of evaluations was reached. | |

(6) **Notes**

(a) The actual name in the first argument SUB must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for SUB must be created. This function subprogram (in double-precision) should be created as follows.

SUBROUTINE SUB(X, Y)
REAL(8) X, Y
DIMENSION X(∗), Y(∗)

$$\begin{aligned} \mathrm{Y(1)} &= f_1(\boldsymbol{x}) \\ &\vdots \\ \mathrm{Y(M)} &= f_m(\boldsymbol{x}) \end{aligned}$$

RETURN
END

(b) Convergence is determined according to the following condition, and the solution is assumed to be $\boldsymbol{x} + \Delta\boldsymbol{x}$:

$$\|\Delta\boldsymbol{x}\| \leq \mathrm{ER} \times \max(1, \|\boldsymbol{x} + \Delta\boldsymbol{x}\|)$$

where $\Delta\boldsymbol{x}$ is the correction vector for $\boldsymbol{x}$ and $\|\boldsymbol{x}\| = \max_i \mid x_i \mid$. A value on the order of the default value should be taken for ER.

(c) If a default value is shown for an argument in the "Contents" column of the table in the argument section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) Scaling should be performed so that the contribution to the function value from each of the variables is on the same order. (See Section 5.1.1)

(e) If there are multiple minimum values, you cannot guarantee to which minimum value the subroutine will converge.

(f) The function must continuously first-order differentiable.

(7) **Example**

(a) Problem
Minimize $s = f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2$ for the functions:

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} 10(x_2 - x_1^2) \\ 1 - x_2 \end{bmatrix}$$

using $\boldsymbol{x} = [-1.2, 1.0]^T$ as the initial value.

(b) Input data
Name of subroutine that calculates the function: FMUSSN
X(1) = $-1.2$, X(2)=1.0, N=2, ER=0.0, NEV=0 and M=2.

(c) Main program

```
      PROGRAM BMUSSN
! *** EXAMPLE OF DMUSSN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N = 2, M = 2)
      DIMENSION IWK(3*N)
      DIMENSION X(N),Y(M),WK(M*(2*N+1)+N*(N+4))
      EXTERNAL  FMUSSN
!
      WRITE(6,1000)
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) X
      WRITE(6,1100) N,M,NEV,ER
      WRITE(6,1200) (I,X(I),I=1,N)
      CALL DMUSSN(FMUSSN,X,N,ER,NEV,Y,M,SY,IWK,WK,IERR)
      WRITE(6,1300) IERR,NEV
      WRITE(6,1400) (I,X(I),I=1,N)
      WRITE(6,1500) SY
      WRITE(6,1600) (I,Y(I),I=1,M)
      STOP
!
 1000 FORMAT(' ',/,' *** DMUSSN ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'N     =',I5,/,&
             5X,'M     =',I5,/,&
             5X,'NEV   =',I5,/,&
             5X,'ER    =',D18.10)
 1200 FORMAT(5X,'(( INITIAL VALUE ))',/,&
            (5X,'   X(',I2,') =',F5.1))
 1300 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR  =',I5,/,&
             5X,'NEV   =',I5)
 1400 FORMAT(5X,'(( SOLUTION ))',/,&
            (5X,'   X(',I2,') =',D18.10))
 1500 FORMAT(5X,'(( LEAST SQUARES ))',/,&
             5X,'   S     =',D18.10)
 1600 FORMAT(5X,'(( FUNCTION VALUE ))',/,&
            (5X,'   Y(',I2,') =',D18.10))
      END

      SUBROUTINE FMUSSN(X,Y)
      REAL(8) X,Y
      DIMENSION X(*),Y(*)
!
      Y(1) = 10.0D0*(X(2)-X(1)*X(1))
      Y(2) = 1.0D0-X(1)
      RETURN
      END
```

(d) Output results

```
*** DMUSSN ***
 ** INPUT **
    N     =    2
    M     =    2
    NEV   =    0
    ER    =  0.0000000000D+00
    (( INITIAL VALUE ))
       X( 1) = -1.2
       X( 2) =  1.0
 ** OUTPUT **
    IERR  =    0
    NEV   =   30
    (( SOLUTION ))
       X( 1) =  0.1000000000D+01
```

```
   X( 2) =  0.1000000000D+01
(( LEAST SQUARES ))
   S     =  0.0000000000D+00
(( FUNCTION VALUE ))
   Y( 1) =  0.0000000000D+00
   Y( 2) =  0.0000000000D+00
```

# 5.5 MINIMIZATION OF A FUNCTION OF ONE VARIABLE WITH CONSTRAINTS

## 5.5.1 DMCUSN, RMCUSN
## Minimization of a Function of One Variable (Interval Specified)

(1) **Function**

DMCUSN or RMCUSN searches for the minimum value of a function $f(x)$ of one variable, within the interval $[a, b]$.

(2) **Usage**

Double precision:
CALL DMCUSN (F, AX, BX, ER, NEV, X, Y, IERR)
Single precision:
CALL RMCUSN (F, AX, BX, ER, NEV, X, Y, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | − | Input | Name of function subprogram that defines the function $f(x)$ (See Notes (a)) |
| 2 | AX | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Initial value $a$ of left end of search interval |
| 3 | BX | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Initial value $b$ of right end of search interval |
| 4 | ER | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{(\text{Unit for determining error})}$) |
| 5 | NEV | I | 1 | Input | Maximum number of evaluations of function $\boldsymbol{f(x)}$ (Default value: 100 ×N) |
| | | | | Output | Actual number of function evaluations |
| 6 | X | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Final destination $x^*$ |
| 7 | Y | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Function value $y = f(x^*)$ at final destination $x^*$ |
| 8 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) AX $<$ BX

(b) AX $\neq$ BX

(c) ER $\geq$ Unit for determining error
(except when 0.0 is entered in order to set ER to the default value)

(d) NEV $>$ 0
(except when 0.0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1200 | Restriction (a) was not satisfied. | Processing is performed with AX and BX switched. |
| 1500 | Restriction (c) or (d) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 3000 | Restriction (b) was not satisfied. | Processing is aborted. |
| 5000 | The value did not converge before the given maximum number of function evaluations was reached. | The value of X and Y at that time are output and processing is aborted. |

(6) **Notes**

(a) The actual name in the first argument F must be declared using an EXTERNAL statement in the user program, and a function subprogram having the actual name specified for F must be created. This function subprogram (in double-precision) should be created as follows.

```
REAL(8) FUNCTION F(X)
REAL(8) X

    F = f(x)

RETURN
END
```

(b) If the search interval becomes $[a, b]$ due to interval reduction, then convergence is determined according to the following condition:

$$\max(b-a, x-a) \leq 2 \times \mathrm{ER} \times \max(1, \mid x \mid)$$

A value on the order of the default value should be taken for ER.

(c) If a default value is shown for an argument in the "Contents" column of the table in the section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) If there is no minimum value, then the end point is assumed to be the solution and 0 is output for IERR.

(e) If there are multiple minimum values, you cannot guarantee to which minimum value the subroutine will converge.

(f) The function must continuously first-order differentiable.

(g) To search for a maximum value, set the function value $f(x)$ so that sign will be reversed. At this time, the value of the seventh argument Y will be output with the sign reversed.

(7) **Example**

(a) Problem
Search for the minimum value of the function

$$f(x) = x(x^2 - 2) - 5$$

in the interval $[0, 1]$.

(b) Input data
Function subprogram name corresponding to function $f(x)$: FMCUSN
AX=0.0, BX=1.0, ER=0.0 and NEV=0.

(c) Main program

```
      PROGRAM BMCUSN
! *** EXAMPLE OF DMCUSN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FMCUSN
!
      WRITE(6,1000)
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) AX,BX
      WRITE(6,1100) NEV,ER,AX,BX
      CALL DMCUSN(FMCUSN,AX,BX,ER,NEV,X,Y,IERR)
      WRITE(6,1200) IERR,NEV,X,Y
      STOP
!
 1000 FORMAT(' ',/,' *** DMCUSN ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'NEV   =',I5,/,&
             5X,'ER    =',D18.10,/,&
             5X,'(( SEARCH SECTION ))',/,&
             5X,'   AX =',F5.1,/,&
             5X,'   BX =',F5.1)
 1200 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR  =',I5,/,&
             5X,'NEV   =',I5,/,&
             5X,'(( SOLUTION ))',/,&
             5X,'   X  =',D18.10,/,&
             5X,'(( FUNCTION VALUE ))',/,&
             5X,'   Y  =',D18.10)
      END

      REAL(8) FUNCTION FMCUSN(X)
      REAL(8) X
!
      FMCUSN = X*(X*X-2.0D0)-5.0D0
      RETURN
      END
```

(d) Output results

```
*** DMCUSN ***
 ** INPUT **
    NEV   =    0
    ER    =  0.0000000000D+00
    (( SEARCH SECTION ))
       AX =  0.0
       BX =  1.0
 ** OUTPUT **
    IERR  =    0
    NEV   =   11
    (( SOLUTION ))
       X  =  0.8164965811D+00
    (( FUNCTION VALUE ))
       Y  = -0.6088662108D+01
```

## 5.6 MINIMIZATION OF A CONSTRAINED LINEAR FUNCTION OF SEVERAL VARIABLES (LINEAR PROGRAMMING)

### 5.6.1 DMCLSN, RMCLSN
### Minimization of a Linear Function of Several Variables (Linear Constraints)

(1) **Function**

DMCLSN or RMCLSN obtains the $\boldsymbol{x}$ that minimized a linear function of several variables $f(\boldsymbol{x})$ of the $n$ dimensional vector $\boldsymbol{x} = [x_1, \cdots, x_n]^T$.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x}$$

$m$ constraints: Any of the following for $i = 1, 2, \cdots, m$

- $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$

Domain of $x$ :

$$d_j \leq x_j \leq u_j \quad (j = 1, 2, \cdots, n)$$

where $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ and $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ are vectors of dimension $n$ and $b_i$, $dj$ and $u_j$ are constants ($i = 1, 2, \cdots, m;\ \ j = 1, 2, \cdots, n$).

(2) **Usage**

Double precision:

CALL DMCLSN (A, LMA, NM, B, M, XUP, XLOW, C, ITYPE, ER, NEV, X, Y, ISW, IWK, WK, IERR)

Single precision:

CALL RMCLSN (A, LMA, NM, B, M, XUP, XLOW, C, ITYPE, ER, NEV, X, Y, ISW, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | LMA, NM | Input | When ISW=0, Matrix $A = (a_{i,j})$ ($i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n$) corresponding to the constant coefficients of constraints. (See Notes (a) and (f)) |
| | | | | Output | Matrix corresponding to the constant coefficients of constraints modified by using slack variables. |
| 2 | LMA | I | 1 | Input | Adjustable dimension of array A |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 3 | NM | I | 1 | Input | When ISW=0: $n + 2 \times m$, where $n$ is the number of variables and $m$ is the number of constraints. (See Notes (f)) |
| | | | | Output | Working variable. |
| 4 | B | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | M | Input | When ISW=0, right-hand side of constraints $\boldsymbol{b} = (b_i) \quad (i = 1, 2, \cdots, m)$ (See Notes (f)) |
| | | | | Output | Right-hand side of constraints modified by using slack variables. |
| 5 | M | I | 1 | Input | Number of constraints $m$ |
| 6 | XUP | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NM | Input | When ISW=0, variable upper bounds $\boldsymbol{u} = (u_j) \quad (j = 1, 2, \cdots, n)$ (See Notes (c) and (f)) |
| | | | | Output | variable upper bounds modified by using slack variables. |
| 7 | XLOW | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NM | Input | When ISW=0, variable lower bounds $\boldsymbol{d} = (d_j) \quad (j = 1, 2, \cdots, n)$ (See Notes (c) and (f)) |
| | | | | Output | variable lower bounds modified by using slack variables. |
| 8 | C | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NM | Input | When ISW=0, function coefficients $\boldsymbol{c} = (c_j) \quad (j = 1, 2, \cdots, n)$ (See Notes (f)) |
| | | | | Output | function coefficients modified by using slack variables. |
| 9 | ITYPE | I | M | Input | Distinction between equality and inequality constraints<br>0: $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$<br>1: $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$<br>$-1$: $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$ |
| 10 | ER | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Required precision (Default value : $2 \times \sqrt{(\text{Unit for determining error})}$) |
| 11 | NEV | I | 1 | Input | Maximum number of evaluations of function $f(\boldsymbol{x})$ (Default value: 10× M) |
| | | | | Output | Actual number of function evaluations |
| 12 | X | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | NM | Output | Final destination $\boldsymbol{x}$ (See Notes (b)) |
| 13 | Y | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Output | Function value $f(\boldsymbol{x})$ at final destination $\boldsymbol{x}$ |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 14 | ISW | I | 1 | Input | Processing switch (See Notes (i))<br>0: First processing<br>1: Continuation processing |
| 15 | IWK | I | NM + M | Work | Work area |
| 16 | WK | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | See Contents | Work | Work area<br>**Size**: $2 \times \mathrm{M}^2 + \mathrm{NM} \times (7 + \mathrm{M})$ |
| 17 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

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

(b) ISW = 0 or ISW = 1

(c) ER $\geq$ Unit for determining error (except when 0.0 is entered in order to set ER to the default value)

(d) NEV > 0 (except when 0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1500 | Restriction (c) or (d) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 4000 | A basic feasible solution was not obtained. | |
| 5000 | The values did not converge before the given maximum number of function evaluations was reached. | The values of X and Y at that time are output and processing is aborted. (See Notes (h) and (i)) |

(6) **Notes**

(a) When ISW=0, coefficients $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ corresponding to the constants of constraints must be set in array A as follows. Where, $n$ is number of variables and $m$ is number of constraints.

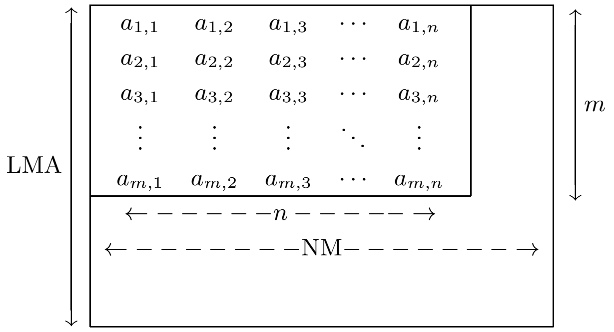
(b) Values of final destination $\boldsymbol{x}$ are set in X(1) through X($n$). The remainder of array X is filled with slack variable values and so on.

(c) If the variable upper and lower bounds have not been specifically determined, positive or negative numbers having appropriately large absolute values must be set for the upper and lower bounds, respectively. Since the optimal solution may not be obtained when the value of variable at destination matches a value of upper or lower bounds set here, numbers having even larger absolute values may have to be set for the upper and lower bounds and the calculation must be performed again.

(d) If a default value is shown for an argument in the "Contents" column of the table in the arguments section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(e) Scaling should be performed so that the contribution to the function value from each variable is of the same order. For example, if $f(\boldsymbol{x}) = 100x_1 + x_2$ with the constraint $200x_1 + 5x_2 = 3$, the best result

Storage status within array A(LMA, NM)




**Remarks**

a. NM $= n + 2 \times m$ must hold.

Figure 5−5

is obtained by performing the variable transformations $y_1 = 100x_1, y_2 = x_2$ to set $h(\boldsymbol{y}) = y_1 + y_2$ with the constraint $2y_1 + 5y_2 = 3$.

(f) Value after the constraints have been transformed are entered in A, B, XUP, XLOW, C and NM for output.

(g) To search for the maximum value, perform a search for the minimum value of $-f(\boldsymbol{x})$. At this time, the value of Y is the maximum value with the sign reversed.

(h) Although the value of $\boldsymbol{x}$ when IERR = 5000 is not the optimal solution, the constraint is satisfied.

(i) If IERR=5000 is returned and the number of iterations is less than the specified convergence count, the calculation can be continued using the information calculated up to the intermediate point. To perform this processing set 1 for the ISW value, set a sufficient value for the NEV value, and use the output values of the previous execution for all other input values. Also, use the work information from the previous execution. (See the example)

## (7) Example

(a) Problem

Minimize the function:

$$f(\boldsymbol{x}) = -x_1 - 3x_2 + 2x_3 + 3x_4 - 4x_5 - 2x_6$$

based on:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 - 3x_4 - 2x_5 - x_6 &\leq 20 \\ 2x_1 - 3x_2 - x_3 + 2x_4 + 4x_5 + x_6 &\geq 28 \\ -3x_1 + 2x_2 + 2x_3 + x_4 + 5x_5 + 2x_6 &= 40 \\ 5x_1 - x_2 + 3x_3 + 3x_4 - 2x_5 + 4x_6 &= 50 \end{aligned}$$

$$0.0 \leq x_i \leq 1000.0 \ (i = 1, 2, 3, 4, 5, 6)$$

In this example, to illustrate the continuation processing described in Note (i), the maximum number of evaluations NEV=6 is set small enough so that IERR=5000 is output.

(b) Input data

(First time): NM=14, M=4, ER=0.0, NEV=6, ISW=0, LMA=11, arrays A, B, XUP, XLOW, C and ITYPE.

(Second and subsequent times): NEV=6 and ISW=1.

(For other arguments, the value obtained after the previous calculation is used directly as the input value.)

(c) Main program

```
      PROGRAM BMCLSN
! *** EXAMPLE OF DMCLSN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( NMO = 14 )
      PARAMETER ( MO = 4 )
      PARAMETER ( LMA = 11 )
      PARAMETER ( NWK = 2*MO*MO+NMO*(7+MO) )
      PARAMETER ( NIWK = NMO+MO )
      DIMENSION A(LMA,NMO),B(MO),XUP(NMO),XLOW(NMO),C(NMO),X(NMO)
      DIMENSION WK(NWK),ITYPE(MO),IWK(NIWK)
!
      WRITE(6,1000)
      READ(5,*) NM,M,ER,NEV,ISW
      WRITE(6,1100) NM , M
      NN = NM-2*M
      WRITE(6,1300)
      DO 10 I = 1,M
        READ(5,*) ( A(I,J),J=1,NN )
        WRITE(6,1200) ( A(I,J),J=1,NN )
   10 CONTINUE
      WRITE(6,1400)
      READ(5,*) ( B(I),I=1,M )
      WRITE(6,1200) ( B(I),I=1,M )
      WRITE(6,1500)
      READ(5,*) ( XUP(I),I=1,NN )
      WRITE(6,1200) ( XUP(I),I=1,NN )
      WRITE(6,1600)
      READ(5,*) ( XLOW(I),I=1,NN )
      WRITE(6,1200) ( XLOW(I),I=1,NN )
      WRITE(6,1700)
      READ(5,*) ( C(I),I=1,NN )
      WRITE(6,1200) ( C(I),I=1,NN )
      WRITE(6,1800)
      READ(5,*) ( ITYPE(I),I=1,M )
      WRITE(6,1250) ( ITYPE(I),I=1,M )
      CALL DMCLSN&
       (A,LMA,NM,B,M,XUP,XLOW,C,ITYPE,ER,NEV,X,Y,ISW,IWK,WK,IERR)
      WRITE(6,1900)
      WRITE(6,2000) IERR
      WRITE(6,2100)
      WRITE(6,2400) ( I,X(I),I=1,NN )
      WRITE(6,2200) Y
!
   20 READ(5,*) NEV,ISW
      CALL DMCLSN&
       (A,LMA,NM,B,M,XUP,XLOW,C,ITYPE,ER,NEV,X,Y,ISW,IWK,WK,IERR)
      WRITE(6,2300)
      WRITE(6,2000) IERR
      WRITE(6,2100)
      WRITE(6,2400) ( I,X(I),I=1,NN )
      WRITE(6,2200) Y
      IF (IERR.EQ.5000) GOTO 20
      STOP
!
 1000 FORMAT(' ',/,6X,'*** DMCLSN ***',/,&
                  7X,'** INPUT **')
 1100 FORMAT(9X,'NM =',I3,8X,'M =',I3)
 1200 FORMAT(8X,6(F7.1))
 1250 FORMAT(8X,4I4)
 1300 FORMAT(8X,'** MATRIX A **')
 1400 FORMAT(8X,'** VECTOR B **')
 1500 FORMAT(8X,'** VECTOR XUP **')
 1600 FORMAT(8X,'** VECTOR XLOW **')
 1700 FORMAT(8X,'** VECTOR C **')
 1800 FORMAT(8X,'** VECTOR ITYPE **')
 1900 FORMAT(7X,'** OUTPUT **',/,&
             8X,'** FIRST RESULT (ISW.EQ.0) **')
 2000 FORMAT(9X,'IERR =',I5)
 2100 FORMAT(8X,'** VECTOR X **')
 2200 FORMAT(9X,'Y =',D18.10)
 2300 FORMAT(7X,'** OUTPUT **',/,&
             8X,'** IMPROVED RESULT (ISW.NE.0) **')
 2400 FORMAT(8X,'X(',I2,') =',D18.10)
      END
```

(d) Output results

```
      *** DMCLSN ***
       ** INPUT **
         NM = 14        M =  4
        ** MATRIX A **
            1.0    2.0    3.0   -3.0   -2.0   -1.0
            2.0   -3.0   -1.0    2.0    4.0    1.0
           -3.0    2.0    2.0    1.0    5.0    2.0
            5.0   -1.0    3.0    3.0   -2.0    4.0
        ** VECTOR B **
           20.0   28.0   40.0   50.0
        ** VECTOR XUP **
```

```
  1000.0 1000.0 1000.0 1000.0 1000.0 1000.0
 ** VECTOR XLOW **
     0.0    0.0    0.0    0.0    0.0    0.0
 ** VECTOR C **
    -1.0   -3.0    2.0    3.0   -4.0   -2.0
 ** VECTOR ITYPE **
    1  -1   0   0
** OUTPUT **
 ** FIRST RESULT (ISW.EQ.0) **
  IERR = 5000
 ** VECTOR X **
 X( 1) =  0.5162689805D+01
 X( 2) =  0.0000000000D+00
 X( 3) =  0.9843817787D+01
 X( 4) =  0.0000000000D+00
 X( 5) =  0.6412147505D+01
 X( 6) =  0.1869848156D+01
  Y = -0.1486334056D+02
** OUTPUT **
 ** IMPROVED RESULT (ISW.NE.0) **
  IERR =    0
 ** VECTOR X **
 X( 1) =  0.1811320755D+02
 X( 2) =  0.1415094340D+02
 X( 3) =  0.0000000000D+00
 X( 4) =  0.0000000000D+00
 X( 5) =  0.1320754717D+02
 X( 6) =  0.0000000000D+00
  Y = -0.1133962264D+03
```

## 5.6.2 DMCLAF, RMCLAF Minimization of a Function of Many Variables (Linear Constraint Given by a Real Irregular Sparse Matrix)

(1) **Function**

DMCLAF or RMCLAF subroutine obtains the value $\boldsymbol{x}$ that minimizes the function of many variables

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x}$$

of the $n$-dimensional vector $\boldsymbol{x} = [x_1, \cdots, x_n]^T$. There are $m$ constraints, which are given by one of the following for $i = 1, 2, \cdots, m$.

- $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$

Also, the domain of $\boldsymbol{x}$ is defined as

$$d_j \leq x_j \leq u_j \quad (j = 1, 2, \cdots, n)$$

Here, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$, $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ are $n$-dimensional vectors, and $b_i$, $dj$ and $u_j$ are constants $(i = 1, 2, \cdots, m; \quad j = 1, 2, \cdots, n)$.

(2) **Usage**

Double precision:

CALL DMCLAF (AVAL, NA, JCN, IA, NM, B, M, XUP, XLOW, C, ITYPE, ETB, AP, BM,
NCK, NEV, X, Y, ISW1, ISW2, IW, W, NW, IERR)

Single precision:

CALL RMCLAF (AVAL, NA, JCN, IA, NM, B, M, XUP, XLOW, C, ITYPE, ETB, AP, BM,
NCK, NEV, X, Y, ISW1, ISW2, IW, W, NW, 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 | AVAL | {D, R} | NA + 2 × M | Input | When ISW=0, Matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ corresponding to coefficients of constraints (See Note (e)) |
| | | | | Output | Matrix corresponding to coefficients of constraints that were modified by using slack variables |
| 2 | NA | I | 1 | Input | Number of nonzero elements of matrix $A$ |
| 3 | JCN | I | NA + 2 × M | Input | Column numbers of nonzero elements of matrix $A$ (See Notes (e) and (f)) |
| 4 | IA | I | M+1 | Input | Information about nonzero elements of matrix $A$ (See Notes (e) and (f)) |
| 5 | NM | I | 1 | Input | Value of $n + 2 \times m$ when ISW2=0, where $n$ the number of variables and $m$ is the number of constraints (See Note (f)) |
| | | | | Output | Work variable |
| 6 | B | {D, R} | M | Input | Right-hand side $\boldsymbol{b} = (b_i)$ $(i = 1, 2, \cdots, m)$ of constraints when ISW2=0 (See Note (f)) |
| | | | | Output | Right-hand side of constraints that were modified by using slack variables |
| 7 | M | I | 1 | Input | Number of constraints $m$ |
| 8 | XUP | {D, R} | NM | Input | Upper limit values $\boldsymbol{u} = (u_j)$ $(j = 1, 2, \cdots, n)$ of variables when ISW2=0 (See Notes (b) and (f).) |
| | | | | Output | Upper limit values of variables that were modified by using slack variables |
| 9 | XLOW | {D, R} | NM | Input | Lower limit values $\boldsymbol{d} = (d_j)$ $(j = 1, 2, \cdots, n)$ of variables when ISW2=0 (See Notes (b) and (f).) |
| | | | | Output | Lower limit values of variables that were modified by using slack variables |
| 10 | C | {D, R} | NM | Input | Coefficients $\boldsymbol{c} = (c_j)$ $(j = 1, 2, \cdots, n)$ of functions when ISW2=0 (See Note (f)) |
| | | | | Output | Coefficients of functions that were modified by using slack variables |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 11 | ITYPE | I | M | Input | Distinction among symbols for equality and inequality of constraints<br>0: $\boldsymbol{a}_i^T\boldsymbol{x} = b_i$<br>1: $\boldsymbol{a}_i^T\boldsymbol{x} \leq b_i$<br>$-1$: $\boldsymbol{a}_i^T\boldsymbol{x} \geq b_i$ |
| 12 | ETB | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 3 | Input | ETB(1) :<br>Unit for determining convergence $\epsilon_C$ (Default value : $10 \times \sqrt{\text{(unit for determining error)}}$)<br>ETB(2) : Unit for testing residual $\epsilon_r$ (Default value : $\sqrt{\text{(unit for determining error)}}$)<br>ETB(3) : Unit for testing feasibility $\epsilon_A$ (Default value :<br>$\sqrt{\text{(unit for determining error)}}$)<br>(See Note (i)) |
| 13 | AP | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 1 | Input | Step size parameter $\alpha$<br>(Default value:0.99)(See Note (h)) |
| 14 | BM | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 1 | Input | Parameter $M$ for Big-M method<br>(Default value : $10 \times \max_i \lvert c_i \rvert$) |
| 15 | NCK | I | 1 | Input | Spacing for performing residual check for constraints (Default value : 10)<br>(See Note (i)) |
| 16 | NEV | I | 1 | Input | Maximum number of evaluations of function $f(\boldsymbol{x})$ (Default value : $10 \times$ M) |
| | | | | Output | Actual number of function evaluations |
| 17 | X | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | NM | Output | Final destination $\boldsymbol{x}$ (See Note (a)) |
| 18 | Y | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 1 | Output | Function value $f(\boldsymbol{x})$ at final destination $\boldsymbol{x}$ |
| 19 | ISW1 | I | 1 | Input | Processing switch (Default value:0) (See Note (j))<br>0: Automatic selection<br>1: Treat coefficient matrix of simultaneous linear equations as a dense matrix.<br>2: Treat coefficient matrix of simultaneous linear equations as a sparse matrix. |
| | | | | Output | Selected switch when ISW1=0 |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 20 | ISW2 | I | 1 | Input | Processing switch (Default value:0) (See Note (k))<br>0: Initial processing<br>1: Continuation processing |
| 21 | IW | I | See Contents | Work | Work area<br>**Size**: $\mathrm{NW}+\mathrm{NA}+14\times\mathrm{M}+4\times\mathrm{NM}+30$ |
| 22 | W | $\begin{Bmatrix}\mathrm{D}\\ \mathrm{R}\end{Bmatrix}$ | NW | Work | Work area |
| 23 | NW | I | 1 | Input | Size of array W<br>(See Note (l)) |
| 24 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{XLOW(J)} \leq \mathrm{XUP(J)}\ \mathrm{J}=1,2,\ldots,\mathrm{NM}-2\times\mathrm{M}$

(b) $\mathrm{ISW1}=0$ or $\mathrm{ISW1}=1$

(c) $\mathrm{ISW2}=0$ or $\mathrm{ISW2}=1$

(d) $\mathrm{ETB(I)} \geq$ unit for determining error $(\mathrm{I}=1,2,3)$ (except when 0.0 is entered to set default value)

(e) $\mathrm{NEV}>0$ (except when 0 is entered to set default value)

(f) $\mathrm{NCK}>0$ (except when 0 is entered to set default value)

(g) $\mathrm{AP}>0$ (except when 0.0 is entered to set default value)

(h) $\mathrm{BM}>0$ (except when 0.0 is entered to set default value)

(i) $0<\mathrm{NM}, 0<\mathrm{M}$ (except when 0 is entered to set default value)

(j) IA(1)=1
$0<\mathrm{IA(i+1)}-\mathrm{IA(i)}\leq\mathrm{N}\ (\mathrm{i}=1,2,\cdots,\mathrm{N}-1)$
$0<\mathrm{NA}-\mathrm{IA(N)}+1\leq\mathrm{N}$

(k) $\mathrm{M}\leq\mathrm{NA}\leq\mathrm{M}\times\mathrm{N}$

(l) 0< Column number of nonzero element of matrix $A$ ≤N
The column numbers of the nonzero elements of matrix $A$, which are stored in array JCN , must be in ascending order for each row. Also, each column must have at least one nonzero element.

(m) $\mathrm{NW}\geq\mathrm{NA}+17\times\mathrm{M}+13+$ (size of area required to perform LU decomposition of $A(D^{(k)})^2A^{\mathrm{T}}$).

Here, $\mathrm{N}=\mathrm{NM}-2\times\mathrm{M}$. Also, $D^{(k)}$ is a diagonal matrix having $x_i^{(k)}$ in element $(i,i)$ for the solution $\boldsymbol{x}^{(k)}=(x_1^{(k)},x_2^{(k)},\ldots,x_n^{(k)})^{\mathrm{T}}$ at the time of the $k$-th iteration.

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | For some $j$ ($j = 1, 2, \ldots$,NM$-2\times$M), XLOW($j$) and XUP($j$) did not satisfy Restriction (a). | The values of XLOW($j$) and XUP($j$) are swapped and processing is performed. |
| 1500 | One of restrictions (b) to (h) was not satisfied. | Processing is performed with the default value set. |
| 3000 | Restriction (i) was not satisfied. | Processing is aborted. |
| 3010 | One of restrictions (j) to (l) was not satisfied. | |
| 3100 | Restriction (m) was not satisfied. | |
| 4000 | A basic feasible solution could not be obtained. | |
| 4100 | $rank(A) < m$ for the matrix $A$ that assigns the constraints. ($m$ is the number of constraints.) | |
| 4200 | The residual for the constraints did not satisfy the required precision. (See Note (i)) | |
| 5000 | The sequence did not converge even though the maximum assigned iteration count was reached. | The values of X and Y at that time were output, and processing is aborted. (See Notes (h) and (i)) |

(6) **Notes**

(a) The value of the final destination $\boldsymbol{x}$ is set in X(1) to X($n$). The remainder of array X contains values such as the slack variables.

(b) If the upper and lower limit values of the variable have not specifically been defined, they are set to positive and negative numbers having suitably large absolute values. If the variable value at the final destination matches the upper or lower limit value defined here, an optimal value may not be obtained. Therefore, a number having a larger absolute value must be set for the upper or lower limit value and the calculation must be executed again.

(c) If a default value appears in the "Contents" column of the argument table and 0 is entered for an integer type argument or 0.0 is entered for a real type argument, the default value is set.

(d) Scaling should be performed so that each variable participates to an equal degree in the function value. For example, if $f(\boldsymbol{x}) = 100x_1 + x_2$ with constraint : $200x_1 + 5x_2 = 3$ the transformation $y_1 = 100x_1, y_2 = x_2$ should be performed so that the result is obtained using , $h(\boldsymbol{y}) = y_1 + y_2$ with constraint : $2y_1 + 5y_2 = 3$.

(e) When ISW2=0, only nonzero coefficients among the coefficients $a_{i,j}$ ($i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n$) corresponding to constraints are stored in array AVAL. Here, $m$ represents the number of constraints and $n$ represents the number of variables. For example, if the matrix $A = (a_{ij})$ representing the constraints is given by

$$A = \begin{bmatrix} 0.0 & a_{12} & a_{13} & 0.0 & 0.0 & a_{16} \\ a_{21} & 0.0 & 0.0 & 0.0 & a_{25} & 0.0 \\ 0.0 & a_{32} & 0.0 & 0.0 & 0.0 & a_{36} \\ 0.0 & 0.0 & 0.0 & a_{44} & 0.0 & a_{46} \end{bmatrix}$$

the storage conditions of arrays AVAL, JCN and IA are as follows.

StorageConditionsofArraysAVAL, JAandIA

| | AVAL | JCN |
|---|---|---|
| NA | $a_{1,2}$ | 2 |
| | $a_{1,3}$ | 3 |
| | $a_{1,6}$ | 6 |
| | $a_{2,1}$ | 1 |
| | $a_{2,5}$ | 5 |
| | $a_{3,2}$ | 2 |
| | $a_{3,6}$ | 6 |
| | $a_{4,4}$ | 4 |
| | $a_{4,6}$ | 6 |
| 2×M | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |
| | $*$ | $*$ |

| | IA |
|---|---|
| M+1 | 1 |
| | 4 |
| | 6 |
| | 8 |
| | $*$ |

**Remark:** The portions indicated by $*$ need not be set as input values.

(f) The values after the constraints and other items were transformed are entered for the output of A, B, XUP, XLOW, C and NM.

(g) To search for the maximum value, you should search for the minimum value of $-f(\boldsymbol{x})$. At this time, the maximum value will be the value of Y with the plus or minus sign reversed.

(h) In theory, as the step size parameter $\alpha$ approaches 1, fewer iterations are required to converge to the optimal solution. However, in some cases, if $\alpha$ is too close to 1, the error gets large.

(i) To prevent a situation in which the constraint cannot be satisfied with sufficient precision due to calculation error, this subroutine performs the check described below when the iteration count $k$ satisfies any of the following conditions.

   i. $k$ is 1
   ii. $k$ is divisible by the value of the argument NCK
   iii. $k$ is equal to the value of the argument NEV

   When any of these conditions is satisfied, the subroutine checks whether or not the residual for the constraint of the solution $\boldsymbol{x}^{(k)}$ at that time satisfies the condition

   $$||A\boldsymbol{x}^{(k)} - b|| \leq \epsilon_r$$

   where $\epsilon_r$ is the value that is set for the argument ETB(2). If this condition is not satisfied, the initial solution is recalculated based on the previous solution for which this condition was satisfied, and the iterations are repeated. If the condition is not satisfied again after NCK iterations from the recalculation of the initial value, the value of NCK is replaced by , max(NCK/2, 1) and the iterations are repeated further. If this condition is still not satisfied even when NCK=1, IERR=4200 is output, and processing is stopped.

(j) This subroutine solves simultaneous linear equations to determine the search direction for the optimal solution. The value of the argument ISW1 can be used to select whether the coefficient matrix of the simultaneous linear equations is to be treated as a dense matrix or as a sparse matrix. If the coefficient matrix is to be treated as a dense matrix, the subroutine 2.2.2 $\left\{\begin{matrix}\text{DBGMSL}\\ \text{RBGMSL}\end{matrix}\right\}$ (described in <Basic Functions Vol. 2> , Section (2.2.2)) is used. If the coefficient matrix is to be treated as a sparse matrix, the subroutine 2.21.1 $\left\{\begin{matrix}\text{DBMFSL}\\ \text{RBMFSL}\end{matrix}\right\}$ (described in <Basic Functions Vol. 2> , Section (2.21.1)) is used. If the coefficient matrix $A$ of the constraints is a sparse matrix and if $AA^{\mathrm{T}}$ is also a sparse matrix, the calculation time will be shorter if the coefficient matrix of the simultaneous linear equations is treated as a sparse matrix. Otherwise, ISW1=0 should be set.

(k) If the specified convergence count is small and IERR=5000 is returned, the calculation can be continued using information that was calculated up to that time. To perform this processing, set 1 for the value of ISW1, set a sufficiently large value for NEV, and use the output values from the previous execution directly for the other input values. Also use the work area information from the previous execution (See the example).

(l) If the maximum value of the absolute values of the artificial variables is greater than the value that was set for ETB(3) when the iterative calculation ends, the assigned problem is considered to be infeasible, and IERR=4000 is output.

(m) If $rank(A) < m$ for the coefficient matrix $A$ that assigns the constraints, this subroutine cannot calculate the optimal solution, and IERR=4100 is output. In this case, the subroutine 5.6.1 $\left\{\begin{matrix}\text{DMCLSN}\\ \text{RMCLSN}\end{matrix}\right\}$ should be used.

(n) The size NW of the array W must be estimated in advance. To prevent IERR=3100 from being output because NW is not sufficiently large, NW = NA + 13 × NM + 2 × M × (M + 1) + 13 should be set.

(7) **Example**

(a) ProblemMinimize

$$f(\boldsymbol{x}) = 2x_1 + x_2 + x_3 - 3x_4 + x_5$$

based on

$$\begin{aligned} x_1 + 2x_4 + 3x_5 &\geq -7 \\ 2x_2 - x_3 &= 0 \\ x_1 + 2x_5 &\leq 8 \end{aligned}$$

$$0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 1 \leq x_3 \leq 2, -3 \leq x_4 \leq 0, 2 \leq x_5 \leq 5.$$

In this example, to show the continuation processing described in Note (k), the maximum evaluation count NEV is set to a small value, NEV=5, so that IERR=5000 will be output.

(b) Input data
(First time): NM=14, M=4, AP=0.0, BM=0.0, NEV=3, ISW1=0, ISW2=0, LMA=11, arrays A, JCN, IA, B, XUP, XLOW, C, ETB and ITYPE.
(Second and subsequent times): NEV=20 and ISW=1.
(For the other arguments, the values that were output by the previous calculation are used directly as input values.)

(c) Main program

```
      PROGRAM BMCLAF
! *** EXAMPLE OF DMCLAF ***
      IMPLICIT NONE
      INTEGER  N,NA,M,NM,NCK,NW,LNW,LNA,LM,LNM,LA
      INTEGER  NEV,ISW1,ISW2
      PARAMETER (LNA=50,LM=5,LNM=20,LA=5,LNW=5000)
      INTEGER I
      INTEGER IA(LA),ITYPE(LM),IERR
      INTEGER JCN(LNA),IW(LNW+LNA+14*LM+4*LNM+30)
      REAL(8) B(LA),AVAL(LNA),ER(3),W(LNW)
      REAL(8) X(LNM),XUP(LNM),XLOW(LNM),C(LNM)
      REAL(8) AP,BM,Y
!
      READ (5,*) NA
      READ (5,*) NM
      READ (5,*) M
      READ (5,*) ER(1),ER(2),ER(3)
      READ (5,*) AP
      READ (5,*) BM
      READ (5,*) NCK
      READ (5,*) NEV
      READ (5,*) ISW1
      READ (5,*) ISW2
      READ (5,*) NW
      N = NM - 2 * M
      WRITE (6,1000) NA,NM,M,ER(1),ER(2),ER(3),AP,BM,&
                     NCK,NEV,ISW1,ISW2,NW
!
      READ (5,*) (AVAL(I),I=1,NA)
      WRITE (6,1100) (AVAL(I),I=1,NA)
!
      READ (5,*) (JCN(I),I=1,NA)
      WRITE (6,1130) (JCN(I),I=1,NA)
!
      READ (5,*) (IA(I),I=1,M)
      WRITE (6,1150) (IA(I),I=1,M)
!
      READ (5,*) (B(I),I=1,M)
      WRITE (6,1200) (B(I),I=1,M)
!
      READ (5,*) (XUP(I),I=1,N)
      WRITE (6,1210) (XUP(I),I=1,N)
!
      READ (5,*) (XLOW(I),I=1,N)
```

```
      WRITE (6,1220) (XLOW(I),I=1,N)
!
      READ (5,*) (C(I),I=1,N)
      WRITE (6,1230) (C(I),I=1,N)
!
      READ (5,*) (ITYPE(I),I=1,M)
      WRITE (6,1160) (ITYPE(I),I=1,M)
!
      WRITE (6,1300)
!
      CALL DMCLAF(AVAL,NA,JCN,IA,NM,B,M,XUP,XLOW,C,ITYPE,&
      ER,AP,BM,NCK,NEV,X,Y,ISW1,ISW2,IW,W,NW,IERR)
!
      WRITE (6,1900)
      WRITE (6,1400) 'DMCLAF',IERR
      WRITE (6,1450) NEV
      WRITE (6,1470) ISW1
      IF ((IERR .GE. 3000) .AND. (IERR .NE. 5000)) STOP
      WRITE (6,1500) (I,X(I),I=1,N)
      WRITE (6,1600) Y
!
      IF (IERR .EQ. 5000) THEN
        ISW2 = 1
        NEV = 100
        CALL DMCLAF(AVAL,NA,JCN,IA,NM,B,M,XUP,XLOW,C,ITYPE,&
        ER,AP,BM,NCK,NEV,X,Y,ISW1,ISW2,IW,W,NW,IERR)
!
        WRITE (6,2300)
        WRITE (6,1400) 'DMCLAF',IERR
        WRITE (6,1450) NEV
        WRITE (6,1470) ISW1
        IF (IERR .GE. 3000) STOP
        WRITE (6,1500) (I,X(I),I=1,N)
        WRITE (6,1600) Y
      ENDIF
      STOP
!
 1000 FORMAT(' ',/,/,&
              ' ***  DMCLAF  ***',/,&
              2X,'**  INPUT  **',/,&
              6X,'NA   =',I3,/,&
              6X,'NM   =',I3,/,&
              6X,'M    =',I3,/,&
              6X,'ER(1)=',F9.3,/,&
              6X,'ER(2)=',F9.3,/,&
              6X,'ER(3)=',F9.3,/,&
              6X,'AP   =',F9.3,/,&
              6X,'BM   =',F9.3,/,&
              6X,'NCK  =',I3,/,&
              6X,'NEV  =',I3,/,&
              6X,'ISW1 =',I3,/,&
              6X,'ISW2 =',I3,/,&
              6X,'NW   =',I3,/,&
              6X,'COEFFICIENT MATRIX AVAL')
 1100 FORMAT(' ' ,7(F9.3))
 1130 FORMAT(' ',/,/,&
              6X,'JCN (COLUMN NUMBER CORRESPONDING TO AVAL(K))',/,&
              7X,11(I4))
 1150 FORMAT(' ',/,/,&
              6X,'IA (STARTING POSITION OF THE I-TH ROW IN  ',/,&
              6X,' ARRAYS AVAL,JCN)',/,&
              7X,10(I4))
 1160 FORMAT(' ',/,/,&
              6X,'ITYPE (TYPE OF EACH CONSTRAINT)',/,&
              7X,10(I4))
 1200 FORMAT(' ',/,/,&
              6X,'CONSTANT VECTOR',/,(7X,F10.4))
 1210 FORMAT(' ',/,/,&
              6X,'UPPER BOUND OF EACH VARIABLE',/,(7X,F10.4))
 1220 FORMAT(' ',/,/,&
              6X,'LOWER BOUND OF EACH VARIABLE',/,(7X,F10.4))
 1230 FORMAT(' ',/,/,&
              6X,'COEFFICIENTS OF OBJECTIVE FUNCTION',/,(7X,F10.4))
 1300 FORMAT(' ',/,/,&
              2X,'**  OUTPUT  **',/,/)
 1400 FORMAT(6X,'IERR (',A6,') =',I5)
 1450 FORMAT(6X,'ITERATION NUMBER =',I5)
 1470 FORMAT(6X,'SELECTED ISW1 =',I5)
 1500 FORMAT(' ',/,&
              6X,'SOLUTION',/,(8X,'X(',I2,') =',D18.10))
 1600 FORMAT(' ',/,&
              6X,'FUNCTION VALUE ',/,(8X,D18.10))
 1900 FORMAT(' ',/,/,&
              7X,'** OUTPUT **',/,&
              8X,'** FIRST RESULT (ISW2.EQ.0) **',/)
 2300 FORMAT(' ',/,/,&
              7X,'** OUTPUT **',/,&
              8X,'** IMPROVED RESULT (ISW2.NE.0) **',/)
      END
```

(d) Output results

```
***  DMCLAF  ***
 **  INPUT  **
```

```
     NA   =  7
     NM   = 11
     M    =  3
     ER(1)=     0.000
     ER(2)=     0.000
     ER(3)=     0.000
     AP   =     0.000
     BM   =     0.000
     NCK  =  0
     NEV  =  5
     ISW1 =  0
     ISW2 =  0
     NW   =200
     COEFFICIENT MATRIX AVAL
    1.000    2.000    3.000    2.000   -1.000    1.000    2.000

     JCN (COLUMN NUMBER CORRESPONDING TO AVAL(K))
         1   4   5   2   3   1   5

     IA (STARTING POSITION OF THE I-TH ROW IN
      ARRAYS AVAL,JCN)
         1   4   6

     CONSTANT VECTOR
         -7.0000
          0.0000
          8.0000

     UPPER BOUND OF EACH VARIABLE
          1.0000
          1.0000
          2.0000
          0.0000
          5.0000

     LOWER BOUND OF EACH VARIABLE
          0.0000
          0.0000
          1.0000
         -3.0000
          2.0000

     COEFFICIENTS OF OBJECTIVE FUNCTION
          2.0000
          1.0000
          1.0000
         -3.0000
          1.0000

     ITYPE (TYPE OF EACH CONSTRAINT)
        -1   0   1

 **  OUTPUT  **


      ** OUTPUT **
       ** FIRST RESULT (ISW2.EQ.0) **

     IERR (DMCLAF) = 5000
     ITERATION NUMBER =    5
     SELECTED ISW1 =    1

     SOLUTION
       X( 1) =  0.3804293265D+00
       X( 2) =  0.7015472780D+00
       X( 3) =  0.1403143425D+01
       X( 4) = -0.2995978790D+01
       X( 5) =  0.2002974615D+01

     FUNCTION VALUE
         0.1385646034D+02

      ** OUTPUT **
       ** IMPROVED RESULT (ISW2.NE.0) **

     IERR (DMCLAF) = 1500
     ITERATION NUMBER =   20
     SELECTED ISW1 =    1

     SOLUTION
       X( 1) =  0.2194790042D-08
       X( 2) =  0.5000000006D+00
       X( 3) =  0.1000000001D+01
       X( 4) = -0.1322835175D-09
```

```
  X( 5) =  0.2000000000D+01

FUNCTION VALUE
    0.3500000007D+01
```

## 5.6.3 DMCLMZ, RMCLMZ Minimization of a Constrained Linear Function of Several Variables Including 0-1 Variables (Mixed 0-1 Programming)

(1) **Function**

DMCLMZ or RMCLMZ obtains $\boldsymbol{x} = (x_1, \ldots, x_n)$ that minimizes the objective function:

$$f(\boldsymbol{x}) = \sum_{j=1}^{n} c_{jx_j}$$

based on the constraints:

$$\begin{array}{rcccl} \sum_{j=1}^{n} a_{i,j}x_j & = & b_i & & (i = 1, \cdots, m_e; j = 1, \cdots, n) \\ \sum_{j=1}^{n} a_{i,j}x_j & \leq & b_i & & (i = m_e + 1, \cdots, m; j = 1, \cdots, n) \\ d_j & \leq & x_j & \leq \quad u_j & (j = 1, \cdots, n) \\ x_j & = & 0, 1 & & (j \in N_{01}) \end{array}$$

and the value of the objective function $f(\boldsymbol{x})$ for that $\boldsymbol{x}$. $N_{01}$ is the set of subscripts of the 0-1 variables.

(2) **Usage**

Double precision:

CALL DMCLMZ (A, MA, N, B, M, ME, XUP, XLOW, LZ, LN, C, MP, NP, ER, NEV, X, Y, ISW, IWK, WK, IERR)

Single precision:

CALL RMCLMZ (A, MA, N, B, M, ME, XUP, XLOW, LZ, LN, C, MP, NP, ER, NEV, X, Y, ISW, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | MA,N | Input | Coefficients of left-hand side of constraints $a_{i,j}$ (See Note (a)) |
| 2 | MA | I | 1 | Input | Adjustable dimension of array A |
| 3 | N | I | 1 | Input | Number of variables $n$ |
| 4 | B | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | M | Input | $m$-dimensional vector having constants of right-hand side of constraints $b_i$ as components |
| 5 | M | I | 1 | Input | Number of constraints $m$ |
| 6 | ME | I | 1 | Input | Number of equality constraints $m_e$ |
| 7 | XUP | I | N | Input | Variable $x_j$ upper bound $u_j$ (See Note (b)) |
| 8 | XLOW | I | N | Input | Variable $x_j$ lower bound $d_j$ (See Note (b)) |
| 9 | LZ | I | LN | Input | Set of subscripts of 0-1 variables $N_{01}$ (See Note (i)) |
| 10 | LN | I | 1 | Input | Number of 0-1 variables (See Note (h)) |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 11 | C | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | N | Input | Objective function coefficient Vector $c_j$ |
| 12 | MP | I | 1 | Input | Depth of search in branch-and-bound method $p$ When $p = 1$, a depth-first search is performed. As $p$ gets larger, the search gets closer to a heuristic search (See Note (f)) |
| 13 | NP | I | 1 | Input | Maximum length of partial problem list in branch-and-bound method |
| 14 | ER | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 1 | Input | Required precision (See Note (c)) |
| 15 | NEV | I | 1 | Input | Maximum number of iterations when solving the relaxation problem (See Note (d)) |
| 16 | X | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | N | Output | Optimal solution $\boldsymbol{x}$ |
| 17 | Y | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | 1 | Output | Objective function value $f(\boldsymbol{x})$ for optimal solution $\boldsymbol{x}$ |
| 18 | ISW | I | 1 | Input | Processing switch (See Notes (g) and (h)) ISW=0: Initial processing ISW=1: Continuation processing |
| 19 | IWK | I | See Contents | Work | Work area **Size :** $\mathrm{MP} + (\mathrm{MP}+3) \times (\mathrm{N}+\mathrm{M}-\mathrm{ME}) + \mathrm{M} + (\mathrm{LN}+2) \times \mathrm{NP} + 7$ |
| 20 | WK | $\begin{Bmatrix} \mathrm{D} \\ \mathrm{R} \end{Bmatrix}$ | See Contents | Work | Work area **Size :** $\mathrm{M}\times 4+(\mathrm{M}+4)\times(\mathrm{N}+2\times\mathrm{M}-\mathrm{ME})+(\mathrm{MP}+1)\times(\mathrm{M}+1)\times(\mathrm{N}-\mathrm{ME}+1)+(\mathrm{M}+1)\times(\mathrm{N}+\mathrm{M}-\mathrm{ME}+1)+2\times\mathrm{LN}+(\mathrm{N}+\mathrm{M}-\mathrm{ME}+2)\times(\mathrm{NP}+1)+2$ |
| 21 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{M}, \mathrm{LN}, \mathrm{N}, \mathrm{MP} > 0$

(b) $\mathrm{MA} \geq \mathrm{M}$

(c) $0 \leq \mathrm{ME} \leq \mathrm{N}, \mathrm{M}$

(d) $\mathrm{N} \geq \mathrm{LN}$

(e) $\mathrm{NP} \geq 2$

(f) $1 \leq \mathrm{LZ}(1) < \mathrm{LZ}(2) < \cdots < \mathrm{LZ}(\mathrm{LN}) \leq \mathrm{N}$

(g) $\mathrm{ER} > 0.0$ (except when 0.0 or a negative value is entered to use the default value)

(h) $\mathrm{NEV} > 0$ (except when 0 or a negative value is entered to use the default value)

(i) ISW=0 or ISW=1

(j) $\mathrm{XUP(i)} \geq \mathrm{XLOW(i)} \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{N})$

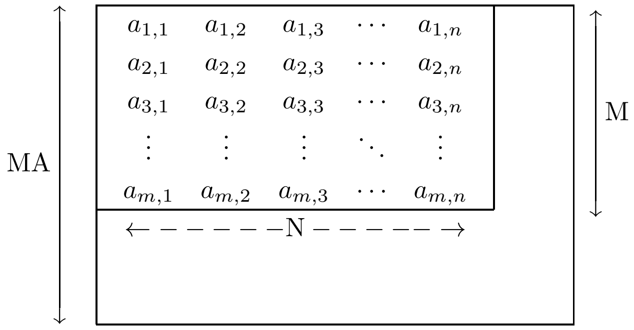
(k) When ISW=1 is set to perform continuation processing after IERR=5600 is output, the value of NP must be set larger than it was for the previous processing.

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Restriction (g) or (h) was not satisfied. | The default value is used, and processing continues. |
| 1100 | Restriction (i) was not satisfied. | ISW=0 is considered to have been specified, and processing continues. |
| 1200 | Restriction (j) was not satisfied. | The upper and lower bounds are switched, and processing continues. |
| 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. | |
| 3100 | Restriction (f) was not satisfied. | |
| 3500 | Restriction (k) was not satisfied. | |
| 4000 | No feasible solution exists. | |
| 5000 | The solution of the relaxation problem could not be obtained within the given number of iterations. The incumbent has been obtained. | The incumbent at that time is output, and processing is aborted. |
| 5100 | The solution of the relaxation problem could not be obtained within the given number of iterations. The incumbent has not been obtained. | Processing is aborted. |
| 5500 | The number of partial problems retained in memory without processing ending reached NP during calculations of the branch-and-bound method. The incumbent has been obtained. | The incumbent at that time is output, and processing is aborted. |
| 5600 | The number of partial problems retained in memory without processing ending reached NP during calculations of the branch-and-bound method. The incumbent has not been obtained. | Processing is aborted. |

(6) **Notes**

(a) The coefficients $a_{i,j}$ of the left-hand side of the constraints are stored as follows in array A.




(b) When $x_j$ is a 0-1 variable, the values of XUP($j$) and XLOW($j$) need not be set in advance.

(c) If a nonpositive value is entered for argument ER, the default value, which is $2 \times \sqrt{\text{(Units for determining error)}}$, is used as the required precision.

(d) A problem in which some of the 0-1 variables of a mixed 0-1 problem are fixed at 0 or 1 is called a partial problem. The linear programming problem in which the conditions for the 0-1 variables that have not been fixed in a partial problem have been weakened so that these variables can take real values greater than or equal to 0 and less than or equal to 1 is called the relaxation problem of that partial problem. When using the simplex method to solve the relaxation problem of a partial problem generated in the process for searching for the solutions of a mixed 0-1 programming problem, the value of argument NEV is used as the upper bound of the number of iterations. If a nonpositive value is entered for argument NEV, the default value, which is $10 \times$M is used as the upper bound of the number of iterations.

(e) During the search for the optimal solution by using the branch-and-bound method, the solution that yields the smallest objective function value among the solutions satisfying the constraints that have been obtained up to that time is called the incumbent. If IERR=5000 or 5500 is returned, the incumbent is output for X when processing is aborted. At this time, although the solution that was obtained satisfies the constraints, it does not necessarily match the true optimal solution.

(f) The depth of search MP is a parameter assigned by the user for setting the solution search method in the branch-and-bound method. The larger the value assigned for MP, the stronger the tendency for a good incumbent to be obtained quickly. However, as the value assigned for MP gets larger, the number of partial problems retained in memory without processing terminating tends to increase dramatically. Therefore, the value of NP must be set sufficient large in advance. When MP=1, the area required for calculations is smallest. To prevent IERR=5500 or 5600 from being returned because the size of NP is insufficient, the value of MP should be set to 1 and the value of NP should be set to LN+1.

(g) When IERR=5000 or 5100 is output and processing is aborted, ISW=1 can be set, NEV can be reset to a larger value, and calculations can be resumed using the previous calculation results. In this case, the contents of all arguments other than NEV that were used by the previous processing must be saved in advance.

(h) When IERR=5500 or 5600 is output and processing is aborted, ISW=1 can be set, NP can be reset to a larger value, and calculations can be resumed using the previous calculation results. In this case, the contents of all arguments other than NP, IWK and WK that were used by the previous processing must be saved in advance. In addition, for IWK and WK, which are work area arrays, suitably large areas must be reserved according to the value of NP, and the contents of IWK and WK at the time the previous processing terminated must be stored in advance at the beginnings of those areas.

(i) The subscripts of 0-1 variables among the variables $x_1, x_2, \cdots, x_n$ are stored in array LZ, and the number of 0-1 variables is stored in argument LN. For example, when dealing with the mixed 0-1 programming problem in which there are eight variables $x_1, x_2, \cdots, x_8$ of which $x_2, x_5$, and $x_8$ are 0-1 variables, set array LZ and argument LN as follows.

$$\begin{aligned} \mathrm{LN} &= 3 \\ \mathrm{LZ(1)} &= 2 \\ \mathrm{LZ(2)} &= 5 \\ \mathrm{LZ(3)} &= 8 \end{aligned}$$

(7) **Example**

(a) Problem
Minimize the following objective function:

$$\begin{aligned} f(\boldsymbol{x}) = & -13x_1 - 18x_2 - 10x_3 - 8x_4 - 8x_5 - 12x_6 \\ & -10x_7 - 8x_8 + 11x_9 - 9x_{10} - 30x_{11} - 10x_{12} \end{aligned}$$

under the following constraints:

$$\begin{aligned} x_2 + x_3 + x_{10} &= 1 \\ 6x_1 + 28x_2 + 6x_3 + 6x_4 + 3x_5 + 6x_6 & \\ +21x_7 + 8x_8 - 18x_9 + 12x_{10} + 20x_{11} + 23x_{12} &\leq 60 \\ 7x_1 + 7x_2 + 5x_3 + 2x_4 + 2x_5 + 5x_6 & \\ +4x_7 + 2x_8 - 3x_9 + 3x_{10} + 8x_{11} + 3x_{12} &\leq 20 \\ -x_4 - x_{11} &\leq -1 \\ -x_6 - x_7 - x_{12} &\leq -1 \end{aligned}$$

$$0 \leq x_1 \leq 2\ ,\ 0 \leq x_5 \leq 3\ ,\ 0 \leq x_8 \leq 1\ ,\ -2 \leq x_9 \leq 0$$
$$x_j = 0, 1 \qquad (j = 2, 3, 4, 6, 7, 10, 11, 12)$$

(b) Input data
MA= 6, N= 12, M= 5, ME= 1, MP= 4,
NP= 50, ER= 0.0 (Set to the default value), NEV= 0 (Set to the default value) and ISW= 0.
Arrays A and B for coefficients of constraints, array C for objective function coefficients and array LZ for subscripts of 0-1 variables.

(c) Main program

```
      PROGRAM BMCLMZ
      IMPLICIT NONE
!
      INTEGER MAO,NO,MO,LNO,MPO,NPO,MEO
      INTEGER NIWK,NWK
      PARAMETER ( MAO = 6 )
      PARAMETER ( NO = 12 )
      PARAMETER ( MO = 5 )
      PARAMETER ( LNO = 8 )
      PARAMETER ( NPO = 10 )
      PARAMETER ( MPO = 4 )
      PARAMETER ( MEO = 1 )
      PARAMETER ( NIWK=MPO+(MPO+3)*(NO+MO-MEO)+MO+(LNO+2)*NPO+7 )
      PARAMETER ( NWK=4*MO+(MO+4)*(NO+2*MO-MEO)&
                 +(MPO+1)*(MO+1)*(NO-MEO+1)&
                 +(MO+1)*(NO+MO-MEO+1)&
                 +2*LNO+(NO+MO-MEO+2)*(NPO+1)+2)
      INTEGER MA,ME,N,M,LN,MP,NP,NEV,IERR
      REAL(8) A(MAO,NO),B(MO),XUP(NO),XLOW(NO),C(NO),X(NO)
      REAL(8) ER,Y,WK(NWK)
      INTEGER LZ(LNO),IWK(NIWK)
```

```
      INTEGER I,J,ISW
!
      READ(5,*) MA,N,M,ME,LN,MP,NP,NEV,ISW,ER
      WRITE(6,6000) MA,N,M,ME,LN,MP,NP,NEV,ISW,ER
      WRITE(6,6010)
      DO 100 I = 1,M
         READ(5,*) (A(I,J),J=1,N),B(I)
  100 CONTINUE
      DO 110 I = 1,M
         WRITE(6,6020) (A(I,J),J=1,N),B(I)
  110 CONTINUE
      WRITE(6,6030)
      READ(5,*) (C(J),J=1,N)
      DO 120 J = 1,N
         WRITE(6,6040) J,C(J)
  120 CONTINUE
      WRITE(6,6070)
      READ(5,*) (LZ(I),I=1,LN)
      WRITE(6,6080) (LZ(I),I=1,LN)
      WRITE(6,6050)
      DO 130 J = 1,N
         DO 132 I = 1,LN
            IF(LZ(I).EQ.J) THEN
               GOTO 135
            ENDIF
  132    CONTINUE
         READ(5,*) XUP(J),XLOW(J)
         WRITE(6,6060) J,XUP(J),J,XLOW(J)
  135 CONTINUE
  130 CONTINUE
      CALL DMCLMZ&
      (A,MA,N,B,M,ME,XUP,XLOW,LZ,LN,C,MP,NP,ER,NEV,X,Y,ISW,IWK,WK,&
       IERR)
      WRITE(6,6090)
      WRITE(6,6100) IERR
      DO 150 J = 1,N
         WRITE(6,6110) J,X(J)
  150 CONTINUE
      WRITE(6,6120) Y
 6000 FORMAT(/,/,5X,'** INPUT **',/,/,/,&
                  11X,'MA =  ',I5,8X,',N   =',I5,/,&
                  11X,'M  =  ',I5,8X,',ME  =',I5,/,&
                  11X,'LN =  ',I5,8X,',MP  =',I5,/,&
                  11X,'NP =  ',I5,8X,',NEV =',I5,/,&
                  11X,'ISW = ',I5,8X,',ER  =',D7.1)
 6010 FORMAT(/,5X,'CONSTRAINTS',/,/,&
       32X,' A ',31X,'B',/)
 6020 FORMAT(12(F5.1),3X,F5.1)
 6030 FORMAT(/,5X,'COEFFICIENTS OF THE OBJECTIVE FUNCTION',/)
 6040 FORMAT(8X,'C(',I2,') = ',D15.5)
 6050 FORMAT(/,5X,'UPPER AND LOWER BOUNDS OF X',/)
 6060 FORMAT(8X,'XUP(',I2,') = ',D15.5,2X,'XLOW(',I2,') = ',D15.5)
 6070 FORMAT(/,5X,'INDICES OF 0-1 VARIABLES',/)
 6080 FORMAT(8X,12(1X,I2),/)
 6090 FORMAT(/,5X,'** OUTPUT **',/,/)
 6100 FORMAT(8X,'IERR = ',I5)
 6110 FORMAT(8X,'X(',I2,') = ',D15.5)
 6120 FORMAT(8X,'Y  =  ',D15.5)
      STOP
      END
```

(d) Output results

```
     ** INPUT **


           MA =      6        ,N   =   12
           M  =      5        ,ME  =    1
           LN =      8        ,MP  =    4
           NP =     10        ,NEV =   20
           ISW =     0        ,ER  =0.1D-11

     CONSTRAINTS

                                 A                                B

 0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0      1.0
 6.0 28.0  6.0  6.0  3.0  6.0 21.0  8.0-18.0 12.0 20.0 23.0     60.0
 7.0  7.0  5.0  2.0  2.0  5.0  4.0  2.0 -3.0  3.0  8.0  3.0     20.0
 0.0  0.0  0.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0 -1.0  0.0     -1.0
 0.0  0.0  0.0  0.0  0.0 -1.0 -1.0  0.0  0.0  0.0  0.0 -1.0     -1.0

     COEFFICIENTS OF THE OBJECTIVE FUNCTION

        C( 1) =    -0.13000D+02
        C( 2) =    -0.18000D+02
        C( 3) =    -0.10000D+02
        C( 4) =    -0.80000D+01
        C( 5) =    -0.80000D+01
        C( 6) =    -0.12000D+02
        C( 7) =    -0.10000D+02
        C( 8) =    -0.80000D+01
        C( 9) =     0.11000D+02
        C(10) =    -0.90000D+01
```

```
   C(11) =    -0.30000D+02
   C(12) =    -0.10000D+02

INDICES OF 0-1 VARIABLES

     2  3  4  6  7 10 11 12

UPPER AND LOWER BOUNDS OF X

   XUP( 1) =     0.20000D+01  XLOW( 1) =     0.00000D+00
   XUP( 5) =     0.30000D+01  XLOW( 5) =     0.00000D+00
   XUP( 8) =     0.10000D+01  XLOW( 8) =     0.00000D+00
   XUP( 9) =     0.00000D+00  XLOW( 9) =    -0.20000D+01

** OUTPUT **

   IERR =     0
   X( 1) =     0.00000D+00
   X( 2) =     0.00000D+00
   X( 3) =     0.00000D+00
   X( 4) =     0.10000D+01
   X( 5) =     0.30000D+01
   X( 6) =     0.10000D+01
   X( 7) =     0.00000D+00
   X( 8) =     0.10000D+01
   X( 9) =    -0.66667D+00
   X(10) =     0.10000D+01
   X(11) =     0.00000D+00
   X(12) =     0.00000D+00
   Y  =     -0.68333D+02
```

### 5.6.4 DMCLMC, RMCLMC Minimization of Cost for Flow in a Network (Minimal-Cost Flow Problem)

(1) **Function**

DMCLMC or RMCLMC obtains the nonnegative flows $x_k \quad (k = 1, 2, \cdots, m)$ that satisfy the vertex $i$ inflow/outflow amount $b_i$ and directed edge $k$ nonnegative capacity $u_k$ constraints and minimize the sum of the costs of all edges in a network having $n$ vertices and $m$ edges, and the subroutine also obtains the minimum value of $\sum_{k=1}^{m} c_k x_k$ at that time.

$$\text{Objective function} \quad : \quad \sum_{k=1}^{m} c_k x_k \rightarrow min$$

$$\text{Constraints} \quad : \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k = b_i, \qquad (i = 1, \cdots, n)$$

$$0 \le x_k \le u_k, \qquad (k = 1, \cdots, m)$$

$$\sum_{i=1}^{n} b_i = 0$$

(2) **Usage**

Double precision:

CALL DMCLMC (N, M, B, ITL, IHD, CAP, COST, NEV, X, Y, ISW, IWK, WK, IERR)

Single precision:

CALL RMCLMC (N, M, B, ITL, IHD, CAP, COST, NEV, X, Y, ISW, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real   Z:Double precision complex   I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real   C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Number of vertices $n$ |
| 2 | M | I | 1 | Input | Number of edges $m$ |
| 3 | B | $\left\{ \begin{array}{c} \text{D} \\ \text{R} \end{array} \right\}$ | N | Input | Vertex $i$ inflow/outflow amount $b_i$ |
| 4 | ITL | I | M+N | Input | Vertex number of edge $k$ tail, $tail(k)$ (See Note (a)) |
| | | | | Output | Vertex numbers of edge tail after graph modification |
| 5 | IHD | I | M+N | Input | Vertex number of edge $k$ head, $head(k)$ (See Note (a)) |
| | | | | Output | Vertex numbers of edge head after graph modification |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 6 | CAP | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | M+N | Input | Edge $k$ capacity $u_k$ (See Note (a)) |
| | | | | Output | Edge capacity after graph modification |
| 7 | COST | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | M+N | Input | Edge $k$ cost coefficient per unit flow $c_k$ (See Note (a)) |
| | | | | Output | Edge costs coefficient per unit flow after graph modification |
| 8 | NEV | I | 1 | Input | Maximum number of updates of basic tree (Default: N$\times$ 10)<br>(See Note (c)) |
| | | | | Output | Actual number of updates of basic tree |
| 9 | X | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | M | Output | Flow minimizing sum of costs of all edges $x_k$ |
| 10 | Y | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Output | Sum of costs at final destination X $\sum_{k=1}^{m} c_k x_k$ |
| 11 | ISW | I | 1 | Input | Processing switch (See Note (d))<br>ISW=0: Initial processing<br>ISW=1: Continuation processing |
| 12 | IWK | I | See Contents | Work | Work area<br>**Size :**$\mathrm{N}^2 + 11 \times \mathrm{N} + \mathrm{M} + 3$ |
| 13 | WK | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | See Contents | Work | Work area<br>**Size**: $2 \times \mathrm{N} + \mathrm{M} + 1$ |
| 14 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{N} \geq 2$

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

(c) $\mathrm{B(1)} + \mathrm{B(2)} + \cdots + \mathrm{B(N)} = 0.0$

(d) $1 \leq \mathrm{ITL(k)} \leq \mathrm{N}, \quad (k = 1, \cdots, \mathrm{M})$

(e) $1 \leq \mathrm{IHD(k)} \leq \mathrm{N}, \quad (k = 1, \cdots, \mathrm{M})$

(f) $\mathrm{ITL(k)} \neq \mathrm{IHD(k)}, \quad (k = 1, \cdots, \mathrm{M})$ (No loop, that is, edge for which the head and tail are the same, exists)

(g) $\mathrm{CAP(k)} \geq 0.0, \quad (k = 1, \cdots, \mathrm{M})$

(h) $\mathrm{NEV} > 0$ (Except when zero or a negative number is entered to set the default value)

(i) $\mathrm{ISW} = 0$ or $\mathrm{ISW} = 1$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Restriction (h) was not satisfied. | Processing is performed with the default value set. |
| 1100 | Restriction (i) was not satisfied. | Processing is performed with ISW=0 set. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (c) was not satisfied. | |
| 3200 | Restriction (d) or (e) was not satisfied. | |
| 3300 | Restriction (f) was not satisfied. | |
| 3400 | Restriction (g) was not satisfied. | |
| 4000 | No basic feasible solution could be obtained. | |
| 5000 | The problem could not be solved even though the assigned maximum number of updates of the basic tree was reached. | |

(6) **Notes**

(a) You should set the values of the leading M elements of ITL, IHD, CAP and COST respectively. The last N elements of these arrays are used for work areas.

(b) If the capacities have not been specifically determined, positive numbers having appropriately large absolute values are assumed for the capacities, respectively. Since the optimal solution may not be obtained when the destination matches a value set here, numbers having even larger absolute values may have to be set for the capacities.

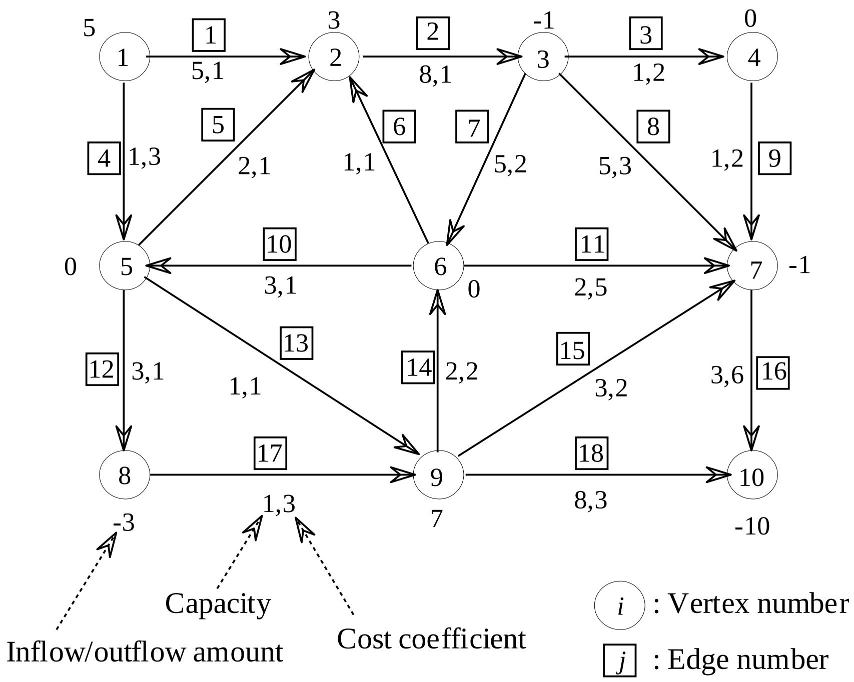
(c) If zero or a negative number is entered for NEV, the default value is set.

(d) If the specified value of the maximum number of updates of the basic tree was too small and IERR=5000 was returned, continuation processing can be performed using information that was calculated up to that point. To perform this processing, set ISW to 1, set a sufficiently large value for NEV, and use the values from the previous processing directly for the input values of other arguments. Also, use the Work information from the previous processing.

(7) **Example**

(a) Problem

For the network shown below, obtain the flow that satisfies the constraints for the inflow/outflow of each vertex and capacity of each edge and minimizes the sum of the costs of all edges.




(b) Input data

N=10, M=18, array B for inflow/outflow amount array CAP for capacity, arrays ITL and IHD for vertex number of edge, and array COST for edge cost coefficient per unit flow. NEV=0 (Set to the default value) and ISW=0.

(c) Main program

```
      PROGRAM BMCLMC
!
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER( NMAX = 10 )
      PARAMETER( MMAX = 18 )
      DIMENSION B(NMAX),ITAIL(MMAX+NMAX),IHEAD(MMAX+NMAX)
      DIMENSION CAP(MMAX+NMAX),COST(MMAX+NMAX),X(MMAX)
      DIMENSION IWK(NMAX**2+11*NMAX+MMAX+3),WK(2*NMAX+MMAX+1)
!
      READ(5,*) N,M
      DO 100 I=1,N
         READ(5,*) B(I)
  100 CONTINUE
      DO 110 K=1,M
         READ(5,*) ITAIL(K),IHEAD(K),CAP(K),COST(K)
  110 CONTINUE
!
      NEV = 0
      ISW = 0
      WRITE(6,6000) N,M,NEV,ISW
      WRITE(6,6010)
      DO 120 I=1,N
         WRITE(6,6020) B(I)
  120 CONTINUE
      WRITE(6,6030)
      DO 130 K=1,M
         WRITE(6,6040) ITAIL(K),IHEAD(K),CAP(K),COST(K)
  130 CONTINUE
!
      CALL DMCLMC(N,M,B,ITAIL,IHEAD,CAP,COST,NEV,X,Y,ISW,IWK,WK,IERR)
!
      WRITE(6,6050) IERR,NEV
      DO 140 K=1,M
         WRITE(6,6060) K,X(K)
```

```
  140 CONTINUE
      WRITE(6,6070) Y
!
 6000 FORMAT(/,&
      1X,'    ** INPUT **',/,/,&
      1X,'       N   = ',I5,/,&
      1X,'       M   = ',I5,/,&
      1X,'       NEV = ',I5,/,&
      1X,'       ISW = ',I5)
 6010 FORMAT(/,&
      1X,'       VALUES OF B')
 6020 FORMAT(1X,'         ',F7.2)
 6030 FORMAT(/,&
      1X,'       ITAIL    IHEAD     CAP     COST')
 6040 FORMAT(1X,'    ',I7,'  ',I7,'    ',F7.2,'  ',F7.2)
 6050 FORMAT(/,&
      1X,'    ** OUTPUT **',/,/,&
      1X,'       IERR = ',I5,/,/,&
      1X,'       NEV  = ', I5,/)
 6060 FORMAT(1X,'       X(',I2,') =',F10.2)
 6070 FORMAT(/,&
      1X,'       Y     =',F10.2)
      STOP
      END
```

(d) Output results

```
    ** INPUT **

       N   =    10
       M   =    18
       NEV =     0
       ISW =     0

       VALUES OF B
            5.00
            3.00
           -1.00
            0.00
            0.00
            0.00
           -1.00
           -3.00
            7.00
          -10.00

       ITAIL    IHEAD     CAP     COST
         1        2       5.00     1.00
         2        3       8.00     1.00
         3        4       1.00     2.00
         1        5       1.00     3.00
         5        2       2.00     1.00
         6        2       1.00     1.00
         3        6       5.00     2.00
         3        7       5.00     3.00
         4        7       1.00     2.00
         6        5       3.00     1.00
         6        7       2.00     5.00
         5        8       3.00     1.00
         5        9       1.00     1.00
         9        6       2.00     2.00
         9        7       3.00     2.00
         7       10       3.00     6.00
         8        9       1.00     3.00
         9       10       8.00     3.00

    ** OUTPUT **

       IERR =  1000

       NEV  =    13

       X( 1) =      4.00
       X( 2) =      7.00
       X( 3) =      0.00
       X( 4) =      1.00
       X( 5) =      0.00
       X( 6) =      0.00
       X( 7) =      3.00
       X( 8) =      3.00
       X( 9) =      0.00
       X(10) =      3.00
       X(11) =      0.00
       X(12) =      3.00
       X(13) =      1.00
       X(14) =      0.00
       X(15) =      0.00
       X(16) =      2.00
       X(17) =      0.00
       X(18) =      8.00

       Y     =     72.00
```

## 5.6.5 DMCLCP, RMCLCP
## Minimization of Cost for Project Scheduling (Project Scheduling Problem)

(1) **Function**

DMCLCP or RMCLCP solves the problem of completing a project within the scheduled completion time according to a minimum cost.

(2) **Usage**

Double precision:

CALL DMCLCP (N, M, ITL, IHD, TN, TC, CN, CC, TS, NEV, TMAX, TMIN, TIME, TTIME, ES, LS, TF, FF, CMAX, CMIN, COST, TCOST, IWK, WK, IERR)

Single precision:

CALL RMCLCP (N, M, ITL, IHD, TN, TC, CN, CC, TS, NEV, TMAX, TMIN, TIME, TTIME, ES, LS, TF, FF, CMAX, CMIN, COST, TCOST, IWK, WK, 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 nodes. |
| 2 | M | I | 1 | Input | Number of tasks. |
| 3 | ITL | I | M | Input | Tail of task $k$. |
| 4 | IHD | I | M | Input | Head of task $k$. |
| 5 | TN | {D/R} | M | Input | Normal time of task $k$. |
| 6 | TC | {D/R} | M | Input | Crash time of task $k$. |
| 7 | CN | {D/R} | M | Input | Normal cost of task $k$. |
| 8 | CC | {D/R} | M | Input | Crash cost of task $k$. |
| 9 | TS | {D/R} | 1 | Input | Scheduled completion time. |
| 10 | NEV | I | 1 | Input | Maximum number of updates of basic tree for Minimal-Cost Flow Problem. (Default:N×10) |
| | | | | Output | Actual number of updates of basic tree. |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 11 | TMAX | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Completion time with normal process times. |
| 12 | TMIN | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Completion time with crash process times. |
| 13 | TIME | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Scheduled process time of task $k$. |
| 14 | TTIME | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Total task time. |
| 15 | ES | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Earliest start time of task $k$. |
| 16 | LS | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Latest start time of task $k$. |
| 17 | TF | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Total float of task $k$. |
| 18 | FF | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Free float of task $k$. |
| 19 | CMAX | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Completion time with normal process costs. |
| 20 | CMIN | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Completion time with crash process costs. |
| 21 | COST | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | M | Output | Schedules process cost of task $k$. |
| 22 | TCOST | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Total task cost. |
| 23 | IWK | I | See Contents | Work | Work area.<br>**Size**:$\mathrm{N}^2 + 15 \times \mathrm{N} + 6 \times \mathrm{M} + 23$ |
| 24 | WK | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | See Contents | Work | Work area.<br>**Size**:$7 \times \mathrm{N} + 12 \times \mathrm{M} + 14$ |
| 25 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{N} > 1$

(b) $\mathrm{M} \geq \mathrm{N} - 1$

(c) $1 \leq \mathrm{ITL(k)} < \mathrm{N}, \quad (k = 1, \cdots, \mathrm{M})$

(d) $1 < \mathrm{IHD(k)} \leq \mathrm{N}, \quad (k = 1, \cdots, \mathrm{M})$

(e) $\mathrm{ITL(k)} < \mathrm{IHD(k)}, \quad (k = 1, \cdots, \mathrm{M})$

(f) $\mathrm{TN(k)} \geq 0.0, \quad (k = 1, \cdots, \mathrm{M})$

(g) $\mathrm{TC(k)} \geq 0.0, \quad (k = 1, \cdots, \mathrm{M})$

(h) $\mathrm{TN(k)} \geq \mathrm{TC(k)}, \quad (k = 1, \cdots, \mathrm{M})$

(i) $\mathrm{CN(k)} \leq \mathrm{CC(k)}, \quad (k = 1, \cdots, \mathrm{M})$

(j) NEV $> 0$ (Except when zero is entered to set the default value.)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | NEV=0. | Processing is performed with the default value set. |
| 1100 | TS≥TMAX. | Processing is performed with TS=TMAX. |
| 1200 | TS≤TMIN. | Processing is performed with TS=TMIN. |
| 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. | |
| 3400 | Restriction (e) was not satisfied. | |
| 3500 | Restriction (f) was not satisfied. | |
| 3600 | Restriction (g) was not satisfied. | |
| 3700 | Restriction (h) was not satisfied. | |
| 3800 | Restriction (i) was not satisfied. | |
| 3900 | Restriction (j) was not satisfied. | |
| 4000 | No basic feasible solution for Minimal-Cost Flow Problem could be obtained. | |
| 5000 | Minimal-Cost Flow Problem could not be solved even though the assigned maximum number of updates of the basic tree was reached. | |

(6) **Notes**

(a) If zero is entered for NEV, the default value is set.

(b) If large number than completion time with normal process times TMAX is entered for scheduled completion time TS, completion time with normal process times is set for scheduled completion time. And if large number than completion time with crash process times TMIN is entered for scheduled completion time TS, completion time with crash process times is set for scheduled completion time.

(7) **Example**

(a) Problem

Compute completion time with normal process times, completion time with crash process times, an optimal schedule, its cost, earliest start time, latest start time, total float, free float, total task time and total task cost for given project follows task lists.

| task | tail of task | head of task | normal task time | crash task time | normal task cost | crash task cost |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 5 | 3 | 3 | 6 |
| 2 | 1 | 3 | 8 | 4 | 2 | 3 |
| 3 | 2 | 4 | 5 | 2 | 1 | 5 |
| 4 | 2 | 5 | 9 | 4 | 1 | 3 |
| 5 | 4 | 5 | 0 | 0 | 0 | 0 |
| 6 | 2 | 6 | 3 | 3 | 4 | 4 |
| 7 | 3 | 6 | 7 | 3 | 2 | 3 |
| 8 | 5 | 7 | 5 | 1 | 1 | 5 |
| 9 | 6 | 7 | 0 | 0 | 0 | 0 |
| 10 | 6 | 8 | 12 | 6 | 7 | 11 |
| 11 | 7 | 8 | 7 | 4 | 2 | 7 |
| 12 | 5 | 9 | 15 | 8 | 5 | 10 |
| 13 | 7 | 9 | 8 | 2 | 5 | 11 |
| 14 | 8 | 9 | 2 | 1 | 3 | 4 |
| 15 | 8 | 10 | 8 | 5 | 2 | 5 |
| 16 | 9 | 10 | 3 | 2 | 1 | 3 |
| 17 | 8 | 11 | 13 | 10 | 10 | 12 |
| 18 | 10 | 11 | 8 | 4 | 6 | 10 |

(b) Input data

N=11,M=18,tail of task ITL,head of task IHD, normal task time TN,crash task time TC,normal task cost CN, crash task cost CC, TS=36 and NEV=N×10.

(c) Main program

```
      PROGRAM BMCLCP
! *** EXAMPLE OF DMCLCP ***
      IMPLICIT REAL(8) (A-H,O-Z)
!
      INTEGER NMAX,MMAX
      PARAMETER( NMAX = 11, MMAX = 18 )
      INTEGER N,M,ITAIL(MMAX),IHEAD(MMAX),NEV,IERR
      INTEGER IWK(NMAX*NMAX+15*NMAX+6*MMAX+23)
      REAL(8) TN(MMAX),TC(MMAX),CN(MMAX),CC(MMAX),TS
      REAL(8) TMIN,TMAX,TIME(MMAX),TTIME
      REAL(8) ES(MMAX),LS(MMAX),TF(MMAX),FF(MMAX)
      REAL(8) CMIN,CMAX,COST(MMAX),TCOST
      REAL(8) WK(7*NMAX+12*MMAX+14)
!
      DATA ITAIL /1,1,2,2,4,2,3,5,6,6,7,5,7,8, 8, 9, 8,10/
      DATA IHEAD /2,3,4,5,5,6,6,7,7,8,8,9,9,9,10,10,11,11/
      DATA TN&
     /5.0D0,8.0D0,5.0D0,9.0D0,0.0D0,&
      3.0D0,7.0D0,5.0D0,0.0D0,12.0D0,&
      7.0D0,15.0D0,8.0D0,2.0D0,8.0D0,&
      3.0D0,13.0D0,8.0D0/
      DATA TC&
     /3.0D0,4.0D0,2.0D0,4.0D0,0.0D0,&
      3.0D0,3.0D0,1.0D0,0.0D0,6.0D0,&
      4.0D0,8.0D0,2.0D0,1.0D0,5.0D0,&
      2.0D0,10.0D0,4.0D0/
      DATA CN&
```

```
     /3.0D0,2.0D0,1.0D0,1.0D0,0.0D0,&
      4.0D0,2.0D0,1.0D0,0.0D0,7.0D0,&
      2.0D0,5.0D0,5.0D0,3.0D0,2.0D0,&
      1.0D0,10.0D0,6.0D0/
     DATA CC&
     /6.0D0,3.0D0,5.0D0,3.0D0,0.0D0,&
      4.0D0,3.0D0,5.0D0,0.0D0,11.0D0,&
      7.0D0,10.0D0,11.0D0,4.0D0,5.0D0,&
      3.0D0,12.0D0,10.0D0/
!
     N = 11
     M = 18
     TS = 36.0D0
     NEV = N * 10
!
     WRITE(6,6000) N,M,NEV
     WRITE(6,6010)
     DO 100 K=1,M
        WRITE(6,6020) K,ITAIL(K),IHEAD(K),TN(K),TC(K),CN(K),CC(K)
 100 CONTINUE
     WRITE(6,6030) TS
!
     CALL DMCLCP&
     (N,M,ITAIL,IHEAD,TN,TC,CN,CC,TS,NEV,&
      TMAX,TMIN,TIME,TTIME,ES,LS,TF,FF,CMAX,CMIN,COST,TCOST,&
      IWK,WK,IERR)
     WRITE(6,6040) IERR
     IF( IERR .LT. 3000 ) THEN
        WRITE(6,6050) TMAX,TMIN
        WRITE(6,6060)
        DO 110 K=1,M
           WRITE(6,6070) K,TIME(K),COST(K),ES(K),LS(K),TF(K),FF(K)
 110    CONTINUE
        WRITE(6,6080) NEV
        WRITE(6,6090) TTIME,TCOST
     ENDIF
     STOP
!
6000 FORMAT(/,/,&
     1X,'*** DMCLCP ***',/,/,&
     1X,'     ** INPUT **',/,/,&
     1X,'       N = ',I5,'    M = ',I5,'    NEV = ',I5)
6010 FORMAT(/,/,&
     1X,'                         NORMAL    CRASH     NORMAL    CRASH',/,&
     1X,'       ',&
        'TASK TAIL HEAD TASK TIME TASK TIME TASK COST TASK COST',/)
6020 FORMAT(&
     1X,'       ',I4,2I5,4F10.3)
6030 FORMAT(/,&
     1X,'       REQUEST TASK TIME = ',F10.3)
6040 FORMAT(/,&
     1X,'     ** OUTPUT **',/,/,&
     1X,'       IERR = ',I4)
6050 FORMAT(/,&
     1X,'       NORMAL COMPLETION TIME = ',F10.3,/,&
     1X,'       CRASH  COMPLETION TIME = ',F10.3)
6060 FORMAT(/,/,&
     1X,'                                    EARLIEST    LATEST',/,&
     1X,'                 TASK      TASK     START     START',&
        '     TOTAL      FREE',/,&
     1X,'       TASK      TIME      COST      TIME      TIME',&
        '     FLOAT     FLOAT',/)
6070 FORMAT(&
     1X,'       ',I4,6F10.3)
6080 FORMAT(/,&
     1X,'       ITERATION COUNT = ',I3)
6090 FORMAT(/,&
     1X,'       TOTAL TASK TIME = ',F10.3,/,&
     1X,'       TOTAL TASK COST = ',F10.3)
     END
```

(d) Output results

```
*** DMCLCP ***

     ** INPUT **

       N =    11    M =    18    NEV =   110


                         NORMAL    CRASH     NORMAL    CRASH
       TASK TAIL HEAD TASK TIME TASK TIME TASK COST TASK COST

          1    1    2     5.000     3.000     3.000     6.000
          2    1    3     8.000     4.000     2.000     3.000
          3    2    4     5.000     2.000     1.000     5.000
          4    2    5     9.000     4.000     1.000     3.000
          5    4    5     0.000     0.000     0.000     0.000
          6    2    6     3.000     3.000     4.000     4.000
          7    3    6     7.000     3.000     2.000     3.000
          8    5    7     5.000     1.000     1.000     5.000
          9    6    7     0.000     0.000     0.000     0.000
         10    6    8    12.000     6.000     7.000    11.000
```

```
     11    7    8     7.000     4.000     2.000     7.000
     12    5    9    15.000     8.000     5.000    10.000
     13    7    9     8.000     2.000     5.000    11.000
     14    8    9     2.000     1.000     3.000     4.000
     15    8   10     8.000     5.000     2.000     5.000
     16    9   10     3.000     2.000     1.000     3.000
     17    8   11    13.000    10.000    10.000    12.000
     18   10   11     8.000     4.000     6.000    10.000

  REQUEST TASK TIME =     36.000

** OUTPUT **

  IERR =    0

  NORMAL COMPLETION TIME =     43.000
  CRASH  COMPLETION TIME =     23.000

                                EARLIEST    LATEST
            TASK      TASK       START       START     TOTAL      FREE
  TASK      TIME      COST       TIME        TIME      FLOAT     FLOAT

     1     5.000     3.000      0.000      0.000     0.000     0.000
     2     7.000     2.250      0.000      0.000     0.000     0.000
     3     5.000     1.000      5.000      5.000     0.000     0.000
     4     5.000     2.600      5.000      5.000     0.000     0.000
     5     0.000     0.000     10.000     10.000     0.000     0.000
     6     3.000     4.000      5.000      7.000     2.000     2.000
     7     3.000     3.000      7.000      7.000     0.000     0.000
     8     5.000     1.000     10.000     10.000     0.000     0.000
     9     0.000     0.000     10.000     15.000     5.000     5.000
    10    12.000     7.000     10.000     10.000     0.000     0.000
    11     7.000     2.000     15.000     15.000     0.000     0.000
    12    15.000     5.000     10.000     10.000     0.000     0.000
    13     8.000     5.000     15.000     17.000     2.000     2.000
    14     2.000     3.000     22.000     23.000     1.000     1.000
    15     6.000     4.000     22.000     22.000     0.000     0.000
    16     3.000     1.000     25.000     25.000     0.000     0.000
    17    13.000    10.000     22.000     23.000     1.000     1.000
    18     8.000     6.000     28.000     28.000     0.000     0.000

  ITERATION COUNT =  19

  TOTAL TASK TIME =      36.000
  TOTAL TASK COST =      59.850
```

### 5.6.6 DMCLTP, RMCLTP
### Minimization of Cost for Transportation from Supply Place to Demand Place (Transportation Problem)

(1) **Function**

Supply quantity of supply place $i$ is $a_i$ , demand quantity of demand place $j$ is $b_j$ and $x_{ij}$ is the volume transported from supply place $i$ to demand place $j$.

Constraint:

$$\sum_{j=1}^{n} x_{ij} = a_i \quad (i = 1, \cdots, m)$$

$$\sum_{i=1}^{m} x_{ij} = b_i \quad (j = 1, \cdots, n)$$

$$x_{ij} \geq 0 \quad (\text{ For all } i \text{ and } j \text{ numbers})$$

Therefore, the total transportation cost is obtained by finding $x_{ij}$ that minimizes the following function.

$$Z = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij}$$

(2) **Usage**

Double precision:
CALL DMCLTP (C, LMC, NS, ND, S, D, NEV, X, Z, ISW1, ISW2, IWK, WK, IERR)
Single precision:
CALL RMCLTP (C, LMC, NS, ND, S, D, NEV, X, Z, ISW1, ISW2, IWK, WK, 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 | C | {D, R} | See Contents | Input | Transportation Costs $c_{ij}(i = 1, \cdots, m : j = 1, \cdots, n)$<br>(See Note (a))<br>**Size**: LMC, (ND + 1) |
| 2 | LMC | I | 1 | Input | Adjustable dimension of array C |
| 3 | NS | I | 1 | Input | Number of supply places including imaginary supply places $m$<br>(See Note (b)) |
| 4 | ND | I | 1 | Input | Number of demand places including imaginary demand places $n$<br>(See Note (b)) |

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 5 | S | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | NS+1 | Input | Supply volume of each Supply places $a_i(i = 1, \cdots, m)$ |
| 6 | D | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | ND+1 | Input | Demand volume of each Demand places $b_j(j = 1, \cdots, n)$ |
| 7 | NEV | I | 1 | Input | Maximum number of evaluation of Transportation plan $x_{ij}$ |
| | | | | Output | Actual number of Transportation plan $x_{ij}$ evaluation |
| 8 | X | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | See Contents | Output | The improved transportation plan $x_{ij}(i = 1, \cdots, m : j = 1, \cdots, n)$<br>(See Note (c), (d))<br>**Size**: $\mathrm{LMC}, (\mathrm{ND} + 1)$ |
| 9 | Z | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | 1 | Output | Total transportation cost of The improved transportation plan $Z$ |
| 10 | ISW1 | I | 1 | Input | Processing switch (See Note (e))<br>0: First processing<br>1: Continuation processing |
| 11 | ISW2 | I | 1 | Input | Select switch of first approximate solution (See note (f))<br>0: Northwest corner rule<br>1: Houthakker's method |
| 12 | IWK | I | See Contents | Work | Work area<br>**Size**: $12 \times (\mathrm{NS} + \mathrm{ND} + 1) + 5$ |
| 13 | WK | $\left\{\begin{matrix}\mathrm{D}\\\mathrm{R}\end{matrix}\right\}$ | See Contents | Work | Work area<br>**Size**: $3 \times (\mathrm{NS} + \mathrm{ND} + 2)$ |
| 14 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{C(i,j)} \geq 0$ $(i = 1, \cdots, \mathrm{NS} : j = 1, \cdots, \mathrm{ND})$

(b) $\mathrm{S(i)} > 0$ $(i = 1, \cdots, \mathrm{NS}\ )$

(c) $\mathrm{D(i)} > 0$ $(i = 1, \cdots, \mathrm{ND}\ )$

(d) $\mathrm{ISW1}, \mathrm{ISW2} = 0$ or 1

(e) $\mathrm{NS} > 2, \mathrm{ND} > 2$

(f) $\mathrm{LMC} \geq \mathrm{NS} + 1$

(g) $\mathrm{NEV} \geq 0$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Total demand volume is greater than Total supply volume. | Processing continues by setting up an imaginary supply place that handles a surplus. |
| 1100 | Total demand volume is less than Total supply volume | Processing continues by setting up an imaginary demand place that handles a surplus. |
| 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. | |
| 3400 | Restriction (e) was not satisfied. | |
| 3500 | Restriction (f) was not satisfied. | |
| 3600 | Restriction (g) was not satisfied. | |
| 5000 | The values did not converge before the given maximum number of function evaluation was reached. | The values of X and Z at that time are output and processing is aborted. |

(6) **Notes**

(a) Store the transportation cost in Array C as in the following table so that it can correspond to each supply place and demand place. In the following table, $c_{2,1}$ indicates the unit cost for transportation between Supply Place 2 and Demand Place 1.

| | Demand place 1 | Demand place 2 | $\cdots$ $\cdots$ | Demand place $n$ | Supply volume |
|---|---|---|---|---|---|
| Supply place 1 | $c_{11}$ | $c_{12}$ | $\cdots$ | $c_{1,n}$ | $a_1$ |
| Supply place 2 | $c_{21}$ | $c_{22}$ | $\cdots$ | $c_{2,n}$ | $a_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| Supply place $m$ | $c_{m,1}$ | $c_{m,2}$ | $\cdots$ | $c_{m,n}$ | $a_m$ |
| Demand volume | $b_1$ | $b_2$ | $\cdots$ | $b_n$ | |

(b) For Array C and Array X, (ND+1) $\times$ (NS+1) units or more must be secured per each. For Array S and Array D, (NS+1) units or more and (ND+1) units or more must be secured, respectively. However, the values to be actually input are enough with ND $\times$ NS units for Array C, and NS and ND units for Array S and Array D, respectively.

(c) Store the improved transportation cost in Array X as in the following table so that it can correspond to each supply place and demand place. In the following table, $c_{2,1}$ indicates the transport volume for transportation between Supply Place 2 and Demand Place 1.

| | Demand place 1 | Demand place 2 | $\cdots$ | Demand place $n$ | ( Imaginary demand place $n+1$ ) |
|---|---|---|---|---|---|
| Supply place 1 | $c_{11}$ | $c_{12}$ | $\cdots$ | $c_{1,n}$ | $*$ |
| Supply place 2 | $c_{21}$ | $c_{22}$ | $\cdots$ | $c_{2,n}$ | $*$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| Supply place $m$ | $c_{m,1}$ | $c_{m,2}$ | $\cdots$ | $c_{m,n}$ | $*$ |
| ( Imaginary supply place $m+1$ ) | $*$ | $*$ | $\cdots$ | $*$ | |

(d) Although the value of $\boldsymbol{x}$ when IERR = 5000 is not the optimal solution, the constraint is satisfied.

(e) If IERR=5000 is returned and the number of iterations is less than the specified convergence count, the calculation can be continued using the information calculated up to the intermediate point. To perform this processing set 1 for the ISW1 value, set a sufficient value for the NEV value, and use the output values of the previous execution for all other input valued. Also, use the work information from the previous execution (See the example).

(f) In this subroutine, the northwestern corner rule or Houthakker's Method is applied. By Houthakker's Method, you can obtain as a primary solution an approximate solution value closer to the optimum solution than that by the northwest corner rule, which can shorten the time for improving the solution. However, calculating approximate solution by Houthakker's Method needs longer time compared with calculating it by the northwestern corner rule.

(7) **Example**

(a) Problem

Transportation cost:

$$C = \begin{bmatrix} 6.000 & 2.000 & 6.000 & 1.000 & 3.000 & 3.000 & 7.000 \\ 7.000 & 5.000 & 3.000 & 8.000 & 5.000 & 8.000 & 5.000 \\ 4.000 & 8.000 & 9.000 & 7.000 & 5.000 & 7.000 & 2.000 \\ 4.000 & 7.000 & 3.000 & 6.000 & 7.000 & 3.000 & 4.000 \\ 2.000 & 3.000 & 6.000 & 4.000 & 7.000 & 9.000 & 9.000 \\ 2.000 & 3.000 & 9.000 & 4.000 & 3.000 & 2.000 & 3.000 \end{bmatrix}$$

Demand volume of each Demand place:

$$D = \begin{bmatrix} 40.000 & 73.000 & 32.000 & 24.000 & 32.000 & 45.000 & 35.000 \end{bmatrix}$$

Supply volume of each Supply place:

$$S = \begin{bmatrix} 73.000 & 29.000 & 64.000 & 23.000 & 76.000 & 34.000 \end{bmatrix}$$

In this example, to illustrate the continuation processing described in Note (e), the maximum number of evaluations NEV=2 is set small enough so that IERR=5000 is output.

(b) Input data

(First time):array C, NS=7, LMC=7, ND=8, array D, array S, NEV=2, ISW1=0 and ISW2=1.
(Second and subsequent times):NEV=2 and ISW1=1.

(For other arguments, the value obtained after the previous calculation is used directly as the input value.)

(c) Main program

```
      PROGRAM BMCLTP
! *** EXAMPLE OF DMCLTP ***
      INTEGER I,J,LMC,MD,MWK
      INTEGER ND,NS,NEV,ISW1,ISW2,IERR
      PARAMETER( LMC=7,MD=8, MWK=12*(LMC+MD+1)+5, MDWK=3*(LMC+MD+2) )
      REAL(8) C(LMC,MD), X(LMC,MD),Z
      REAL(8) S(LMC), D(MD), WK(MDWK)
      INTEGER IWK(MWK)
      DATA ZERO/0.0D0/
!
      READ(5,*) ISW1
      READ(5,*) ISW2
      READ(5,*) NEV
      READ(5,*) NS
      READ(5,*) ND
!
      DO 100 I=1,NS
      DO 110 J=1,ND
         C(I,J)=ZERO
  110 CONTINUE
  100 CONTINUE
      DO 120 I=1,NS
         S(I) = ZERO
  120 CONTINUE
      DO 130 J=1,ND
         D(J) = ZERO
  130 CONTINUE
!
      DO 140 I=1,NS
         READ(5,*) ( C(I,J), J=1,ND )
  140 CONTINUE
      READ(5,*) ( S(I), I=1, NS )
      READ(5,*) ( D(J), J=1, ND )
!
      WRITE(6,6000)
      WRITE(6,6010) LMC, NS, ND, NEV, ISW1, ISW2
      WRITE(6,6020)
      DO 150 I=1,NS
         WRITE(6,6030) I,( C(I,J), J=1,ND )
  150 CONTINUE
      WRITE(6,6040) (I,I=1,ND)
      WRITE(6,6050)
      WRITE(6,6060)
      WRITE(6,6070) ( D(J), J=1,ND )
      WRITE(6,6080)
      WRITE(6,6070) ( S(I), I=1,NS )
!
      CALL DMCLTP&
           ( C, LMC, NS, ND, S, D, NEV,&
           X, Z, ISW1, ISW2, IWK, WK, IERR )
!
      WRITE(6,6090)
      WRITE(6,6100)
      WRITE(6,6110) IERR
      WRITE(6,6120) NEV
      WRITE(6,6130) Z
      WRITE(6,6140)
      DO 160 I=1,NS+1
         WRITE(6,6030) I,( X(I,J), J=1,ND+1 )
  160 CONTINUE
      WRITE(6,6040) (I,I=1,ND+1)
      WRITE(6,6050)
      IF( IERR .EQ. 5000 ) THEN
         NEV = 1000
         ISW1  = 1
!
         CALL DMCLTP&
              ( C, LMC, NS, ND, S, D, NEV,&
              X, Z, ISW1, ISW2, IWK, WK, IERR )
!
         WRITE(6,6150)
         WRITE(6,6110) IERR
         WRITE(6,6120) NEV
         WRITE(6,6130) Z
         WRITE(6,6140)
         DO 170 I=1,NS+1
            WRITE(6,6030) I,( X(I,J), J=1,ND+1 )
  170    CONTINUE
         WRITE(6,6040) (I,I=1,ND+1)
         WRITE(6,6050)
      ENDIF
!
 6000 FORMAT( 1X,/,&
              1X,' *** DMCLTP *** '                    )
 6010 FORMAT( 1X,/,&
              1X,'  ** INPUT **',/,/,&
              1X,'    LMC   = ',I5,/,&
              1X,'    NS    = ',I5,/,&
              1X,'    ND    = ',I5,/,&
              1X,'    NEV   = ',I5,/,&
              1X,'    ISW1  = ',I5,/,&
              1X,'    ISW2  = ',I5                     )
```

```
 6020 FORMAT( 1X,/,&
              1X,'    TRANSPORTATION COST = '          )
 6030 FORMAT( 1X,'     ',I2,'   ',8(2X,F5.2)           )
 6040 FORMAT( 1X,' SUPPLY-SITE ',8(I3,4X)              )
 6050 FORMAT( 1X,'         DEMAND-SITE'                )
 6060 FORMAT( 1X,/,&
              1X,'    DEMAND = '                       )
 6070 FORMAT( 1X,'     ',8(2X,F5.2)                    )
 6080 FORMAT( 1X,/,&
              1X,'    SUPPLY = '                       )
 6090 FORMAT( 1X,/,/,&
              1X,'  ** OUTPUT **'                      )
 6100 FORMAT( 1X,/,&
              1X,'   ** FIRST RESULT    (ISW1.EQ.0) **' )
 6110 FORMAT( 1X,'    IERR       = ',I5                )
 6120 FORMAT( 1X,'    NEV        = ',I5                )
 6130 FORMAT( 1X,/,&
              1X,'    TOTAL COST = ',F10.4,/)
 6140 FORMAT( 1X,'    TRANSPORTATION PLAN = '          )
 6150 FORMAT( 1X,/,&
              1X,'   ** IMPROVED RESULT (ISW1.EQ.1) **',/)
!
      STOP
      END
```

(d) Output results

```
 *** DMCLTP ***

  ** INPUT **

    LMC   =     7
    NS    =     6
    ND    =     7
    NEV   =     2
    ISW1  =     0
    ISW2  =     1

    TRANSPORTATION COST =
      1      6.00   2.00   6.00   1.00   3.00   3.00   7.00
      2      7.00   5.00   3.00   8.00   5.00   8.00   5.00
      3      4.00   8.00   9.00   7.00   5.00   7.00   2.00
      4      4.00   7.00   3.00   6.00   7.00   3.00   4.00
      5      2.00   3.00   6.00   4.00   7.00   9.00   9.00
      6      2.00   3.00   9.00   4.00   3.00   2.00   3.00
 SUPPLY-SITE   1      2      3      4      5      6      7
         DEMAND-SITE

    DEMAND =
       40.00  73.00  32.00  24.00  32.00  45.00  35.00

    SUPPLY =
       73.00  29.00  64.00  23.00  76.00  34.00


  ** OUTPUT **

   ** FIRST RESULT    (ISW1.EQ.0) **
    IERR       =  5000
    NEV        =     2

    TOTAL COST =   707.0000

    TRANSPORTATION PLAN =
      1      0.00  37.00   0.00  24.00   3.00   0.00   0.00   9.00
      2      0.00   0.00  20.00   0.00   0.00   0.00   0.00   9.00
      3      0.00   0.00   0.00   0.00  29.00   0.00  35.00   0.00
      4      0.00   0.00  12.00   0.00   0.00  11.00   0.00   0.00
      5     40.00  36.00   0.00   0.00   0.00   0.00   0.00   0.00
      6      0.00   0.00   0.00   0.00   0.00  34.00   0.00   0.00
      7      0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
 SUPPLY-SITE   1      2      3      4      5      6      7      8
         DEMAND-SITE

   ** IMPROVED RESULT (ISW1.EQ.1) **

    IERR       =  1100
    NEV        =     3

    TOTAL COST =   680.0000

    TRANSPORTATION PLAN =
      1      0.00  37.00   0.00  24.00  12.00   0.00   0.00   0.00
      2      0.00   0.00  29.00   0.00   0.00   0.00   0.00   0.00
      3      0.00   0.00   0.00   0.00  11.00   0.00  35.00  18.00
      4      0.00   0.00   3.00   0.00   0.00  20.00   0.00   0.00
      5     40.00  36.00   0.00   0.00   0.00   0.00   0.00   0.00
      6      0.00   0.00   0.00   0.00   9.00  25.00   0.00   0.00
      7      0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
 SUPPLY-SITE   1      2      3      4      5      6      7      8
         DEMAND-SITE
```

## 5.7 MINIMIZATION OF A QUADRATIC FUNCTION OF SEVERAL VARIABLES (QUADRATIC PROGRAMMING)

### 5.7.1 DMCQSN, RMCQSN
### Minimization of a Constrained Convex Quadratic Function of Several Variables (Linear Constraints)

(1) **Function**

DMCQSN or RMCQSN obtains the $\boldsymbol{x}$ that minimizes a convex quadratic function of several variables $f(\boldsymbol{x})$ of the $n$ dimensional vector $\boldsymbol{x} = [x_1, \cdots, x_n]^T$.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + (1/2)\boldsymbol{x}^T G\boldsymbol{x}$$

Constraints :

$$\boldsymbol{a}_i^T\boldsymbol{x} = b_i \quad (i = 1, 2, \cdots, m_e)$$

$$\boldsymbol{a}_i^T\boldsymbol{x} \geq b_i \quad (i = m_e + 1, m_e + 2, \cdots, m; \quad 0 \leq m_e \leq m)$$

where $G$ is an $n \times n$ positive definite matrix, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ and $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ are vectors of dimension $n$ and $b_i$ are constants $(i = 1, 2, \cdots, m)$.

(2) **Usage**

Double precision:

CALL DMCQSN (A, LNA, M, B, G, C, N, ME, ER, NEV, X, Y, ISW, IWK, WK, IERR)

Single precision:

CALL RMCQSN (A, LNA, M, B, G, C, N, ME, ER, NEV, X, Y, ISW, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real  Z:Double precision complex  I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real  C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | LNA, M | Input | When ISW=0, transposed matrix $\mathrm{A^T} = ((\mathrm{a_{j,i}})(\mathrm{i} = 1, 2, \cdots, \mathrm{m};\ \mathrm{j} = 1, 2, \cdots, \mathrm{n})$ corresponding to the constant coefficients of constraints. (See Notes (a) and (d)) |
| | | | | Output | Transposed matrix corresponding to the constant coefficients of constraints after transform. |
| 2 | LNA | I | 1 | Input | Adjustable dimension of array A |
| 3 | M | I | 1 | Input | Number of constraints $m$ |
| 4 | B | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | M | Input | When ISW=0, right-hand side of constraints $\boldsymbol{b} = (b_i)\ \ (i = 1, 2, \cdots, m)$ (See Notes (d)) |
| | | | | Output | Right-hand side of constraints after transform. |
| 5 | G | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | LNA,N | Input | Objective function second order coefficient matrix $G$ |
| 6 | C | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | N | Input | objective function first order coefficient vector $\boldsymbol{c}$ |
| 7 | N | I | 1 | Input | Number of variables $n$ |
| 8 | ME | I | 1 | Input | Number of equality constraints $m_e$ |
| 9 | ER | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Input | Required precision (Default value: $2 \times \sqrt{\text{(Unit for determining error)}}$) |
| 10 | NEV | I | 1 | Input | Maximum number of evaluations of function $f(\boldsymbol{x})$ (Default value: $10 \times \mathrm{N} + \mathrm{M}$) |
| | | | | Output | Actual number of function evaluations |
| 11 | X | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | N | Output | Final destination $\boldsymbol{x}$ |
| 12 | Y | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Output | Function value $f(\boldsymbol{x})$ at final destination $\boldsymbol{x}$ |
| 13 | ISW | I | 1 | Input | Processing switch (See Notes (f))<br>0: First processing<br>1: Continuation processing |
| 14 | IWK | I | 3 | Work | Work area |
| 15 | WK | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | See Contents | Work | Work area<br>**Size**: $2 \times \mathrm{N}^2 + 17 \times \mathrm{N} + 16 \times \mathrm{M} + 16$ |
| 16 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0 < \mathrm{N} \le \mathrm{LNA}, 0 < \mathrm{M}, 0 \le \mathrm{ME} \le \mathrm{M}$

(b) ISW = 0 or ISW = 1

(c) ER $\ge$ Unit for determining error (except when 0.0 is entered in order to set ER to the default value)

(d) NEV > 0 (except when 0 is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | N was equal to 1. | Processing continues. |
| 1500 | Restriction (c) or (d) was not satisfied. | Processing is performed with the default value set for NEV or ER. |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3500 | Equality constraints were not mutually independent. | |
| 4000 | The problem was not executable. | |
| 4000+$i$ | (1)The diagonal element became 0.0 in the processing of the $i$-th column during QR decomposition.<br>(2)The diagonal element became less than or equal to 0.0 in the processing of the $i$-th column during LL$^T$ decomposition. | |
| 5000 | The value did not converge before the given maximum number of function evaluations was reached. | The value of X and Y at that time is output and processing is aborted.<br>(See Notes (e) and (f)) |

(6) **Notes**

(a) When ISW=0, coefficients $a_{i,j}$ $(i = 1, 2, \cdots, m_e;\ j = 1, 2, \cdots, n)$ corresponding to the equality constraints and coefficients $a_{i,j}$ $(i = m_e+1, m_e+2, \cdots, m;\ j = 1, 2, \cdots, n)$ corresponding to the inequality constraints must be set in array A as follows, where $n$ is number of variables and $m$ is number of constraints.

Storage status within array A(LNA, M)

$$
\mathrm{LNA}\left\{
\begin{array}{ccc|ccc}
a_{1,1} & \cdots & a_{m_e,1} & a_{m_e+1,1} & \cdots & a_{m,1} \\
a_{1,2} & \cdots & a_{m_e,2} & a_{m_e+1,2} & \cdots & a_{m,2} \\
a_{1,3} & \cdots & a_{m_e,3} & a_{m_e+1,3} & \cdots & a_{m,3} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
a_{1,n} & \cdots & a_{m_e,n} & a_{m_e+1,n} & \cdots & a_{m,n} \\
\hline
\multicolumn{3}{c}{\leftarrow - - - m_e - - - \rightarrow} & & & \\
\multicolumn{6}{c}{\leftarrow - - - - - - - - - - m - - - - - - - - - - \rightarrow} \\
 & & & & & 
\end{array}
\right.
\quad n
$$

**Remarks**

a. LNA $\ge n$ must hold.

(b) The optimal solution is not obtained by this subroutine if the objective function second order coefficient matrix $G$ is not positive definite.

(c) If a default value is shown for an argument in the "Contents" column of the table in the Arguments section, then the default value will be set if 0 is entered for an integer-type argument or if 0.0 is entered for a real-type argument.

(d) Transformed values are entered in A and B for output.

(e) Usually, the value of $\boldsymbol{x}$ is not satisfied constraints when IERR = 5000 is returned.

(f) If IERR = 5000 is returned and the number of iterations is less than the specified convergence count, the calculation can be continued using the information calculated up to the intermediate point. To perform this processing set 1 for the ISW value, set a sufficient value for the NEV value, and use the output values of the previous execution for all other input values. Also use the work information from the previous execution. (See the example)

(7) **Example**

(a) Problem
Minimize the function:

$$f(\boldsymbol{x}) = 1/2(5{x_1}^2 + 6x_1x_2 + 5{x_2}^2) - 95x_1 - 105x_2$$

based on:

$$\begin{aligned} -x_1 - 2x_2 &\geq -10 \\ -3x_1 - x_2 &\geq -15 \\ -2x_1 - 3x_2 &\geq -30 \\ 15x_1 - 13x_2 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

In this example, to illustrate the continuation processing described in Notes (f), the maximum number of evaluations NEV=3 is set small enough so that IERR=5000 is output.

(b) Input data
(First time): N=2, M=6, ME=0, LNA=11, NEV=3, ISW=0, ER=0.0,
arrays A, B, G and C.
(Second and subsequent times): NEV=3 and ISW=1.
(For other arguments, the value obtained after the previous calculation is used directly as the input value.)

(c) Main program

```
      PROGRAM BMCQSN
! *** EXAMPLE OF DMCQSN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( MO = 6 )
      PARAMETER ( NO = 2 )
      PARAMETER ( LNA = 11 )
      PARAMETER ( NWK = (2*NO+17)*NO+16*(MO+1) )
      DIMENSION A(LNA,MO),B(MO),G(LNA,NO),C(NO),X(NO)
      DIMENSION WK(NWK),IWK(3)
!
      WRITE(6,1000)
      READ(5,*) N,M,ME,ER,NEV,ISW
      WRITE(6,1100) N,M,ME
      WRITE(6,1300)
      DO 10 I = 1,N
        READ(5,*) ( A(I,J),J=1,M )
        WRITE(6,1200) ( A(I,J),J=1,M )
```

```
   10 CONTINUE
      WRITE(6,1400)
      READ(5,*) ( B(I),I=1,M )
      WRITE(6,1200) ( B(I),I=1,M )
      WRITE(6,1500)
      DO 20 I = 1,N
        READ(5,*) ( G(I,J),J=1,N )
        WRITE(6,1200) ( G(I,J),J=1,N )
   20 CONTINUE
      WRITE(6,1600)
      READ(5,*) ( C(I),I=1,N )
      WRITE(6,1200) ( C(I),I=1,N )
      CALL DMCQSN&
       (A,LNA,M,B,G,C,N,ME,ER,NEV,X,Y,ISW,IWK,WK,IERR)
      WRITE(6,1700)
      WRITE(6,1800) IERR
      WRITE(6,1900)
      WRITE(6,2200) ( I,X(I),I=1,N )
!
   30 READ(5,*) NEV,ISW
      CALL DMCQSN&
       (A,LNA,M,B,G,C,N,ME,ER,NEV,X,Y,ISW,IWK,WK,IERR)
      WRITE(6,2100)
      WRITE(6,1800) IERR
      WRITE(6,1900)
      WRITE(6,2200) ( I,X(I),I=1,N )
      WRITE(6,2000) Y
      IF (IERR.EQ.5000) GOTO 30
      STOP
!
 1000 FORMAT(' ',/,6X,'*** DMCQSN ***',/,&
                 7X,'** INPUT **')
 1100 FORMAT(9X,'N =',I3,8X,'M =',I3,8X,'ME =',I3)
 1200 FORMAT(8X,6(F7.1))
 1300 FORMAT(8X,'** MATRIX A **')
 1400 FORMAT(8X,'** VECTOR B **')
 1500 FORMAT(8X,'** MATRIX G **')
 1600 FORMAT(8X,'** VECTOR C **')
 1700 FORMAT(7X,'** OUTPUT **',/,&
             8X,'** FIRST RESULT (ISW.EQ.0) **')
 1800 FORMAT(9X,'IERR =',I5)
 1900 FORMAT(8X,'** VECTOR X **')
 2000 FORMAT(9X,'Y =',D18.10)
 2100 FORMAT(7X,'** OUTPUT **',/,&
             8X,'** IMPROVED RESULT (ISW.NE.0) **')
 2200 FORMAT(8X,'X(',I2,') =',D18.10)
      END
```

(d) Output results

```
      *** DMCQSN ***
       ** INPUT **
         N =  2        M =  6        ME =  0
        ** MATRIX A **
           -1.0   -3.0   -2.0   15.0    1.0    0.0
           -2.0   -1.0   -3.0  -13.0    0.0    1.0
        ** VECTOR B **
          -10.0  -15.0  -30.0    0.0    0.0    0.0
        ** MATRIX G **
            5.0    3.0
            3.0    5.0
        ** VECTOR C **
          -95.0 -105.0
       ** OUTPUT **
        ** FIRST RESULT (ISW.EQ.0) **
         IERR = 5000
        ** VECTOR X **
        X( 1) =  0.2142857143D+01
        X( 2) =  0.8571428571D+01
       ** OUTPUT **
        ** IMPROVED RESULT (ISW.NE.0) **
         IERR =    0
        ** VECTOR X **
        X( 1) =  0.4000000000D+01
        X( 2) =  0.3000000000D+01
         Y = -0.5965000000D+03
```

### 5.7.2 DMCQLM, RMCQLM Minimization of a Generalized Convex Quadratic Function of Several Variables (Linear Constraints)

(1) **Function**

DMCQLM or RMCQLM obtains $\boldsymbol{x}$ that minimizes the objective function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$$

under the constraints

$$\sum_{j=1}^{n} a_{i,j}x_j = b_i \quad (i = 1, \cdots, m_e)$$

$$\sum_{j=1}^{n} a_{i,j}x_j \geq b_i \quad (i = m_e + 1, \cdots, m)$$

$$x_i \geq 0 \quad (i = 1, \cdots, n)$$

and obtains the objective function value $f(\boldsymbol{x})$ at that time. Where, $G$ is an $n \times n$ positive semidefinite matrix and $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ and $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ are $n$ dimensional vectors $(i = 1, \cdots, m)$.

(2) **Usage**

Double precision:

CALL DMCQLM (A, NA, N, B, M, ME, G, NG, C, NEV, X, Y, ISW, IW, W, IERR)

Single precision:

CALL RMCQLM (A, NA, N, B, M, ME, G, NG, C, NEV, X, Y, ISW, IW, W, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{ \begin{array}{l} \text{INTEGER(4) as for 32bit Integer} \\ \text{INTEGER(8) as for 64bit Integer} \end{array} \right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | A | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NA,N | Input | When ISW=0, matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ corresponding to the constant coefficients of constraints. |
| 2 | NA | I | 1 | Input | Adjustable dimension of array A |
| 3 | N | I | 1 | Input | Number of variables $n$ |
| 4 | B | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | M | Input | $m$ dimensional vector having the constants $b_i$ on the right-hand side of the constraint conditions as components |
| 5 | M | I | 1 | Input | Number of constraint conditions $m$ |
| 6 | ME | I | 1 | Input | Number of equality constraint conditions $m_e$ |
| 7 | G | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | NG,N | Input | Two-dimensional coefficient matrix $G$ of the objective function |
| 8 | NG | I | 1 | Input | Adjustable dimension of array G |
| 9 | C | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Input | One-dimensional coefficient vector $\boldsymbol{c}$ of the objective function |
| 10 | NEV | I | 1 | Input | Maximum number of iterations (See Notes (c)) |
| | | | | Output | Number of iterations required to terminate calculation |
| 11 | X | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Output | Optimal solution $\boldsymbol{x}$ |
| 12 | Y | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Output | Objective function value $f(\boldsymbol{x})$ for optimal solution $\boldsymbol{x}$ |
| 13 | ISW | I | 1 | Input | Processing switch (See Notes (d))<br>0:First processing<br>1:Continuation processing |
| 14 | IW | I | See Contents | Work | Work area<br>**Size**: $2 \times \text{N} - \text{ME} + \text{M} + 2$ |
| 15 | W | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | See Contents | Work | Work area<br>**Size**: $(\text{N} + \text{M} - \text{ME} + 2) \times (\text{N} + \text{M} - \text{ME}) \times 2 + (\text{M} + \text{ME}) \times (\text{N} + 1) + \text{N}$ |
| 16 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\mathrm{NA} \geq \mathrm{M}, \mathrm{NG} \geq \mathrm{N}$

(b) $\mathrm{NG}, \mathrm{NA}, \mathrm{N} > 0$

(c) $\mathrm{M}, \mathrm{ME} \geq 0$

(d) $\mathrm{M} \geq \mathrm{ME}$

(e) $\mathrm{N} \geq \mathrm{ME}$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | N was 1. | Processing continues. |
| 1500 | M was 0. | |
| 3000 | One of restrictions (a) through (e) was not satisfied. | Processing is aborted. |
| 3500 | No solution that satisfies the equality constraints exists. | |
| 4000 | It was determined that no optimal solution exists. | |
| 5000 | The solution could not be obtained within the given number of iterations. | |

(6) **Notes**

(a) When ISW=0, coefficients $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ corresponding to the constants of constraints must be set in array A as follows, where $n$ is number of variables and $m$ is number of constraints.

Storage status within array A(NA, K)

$$\mathrm{NA}\quad \underbrace{\underbrace{\left.\begin{matrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{matrix}\right\} m}_{n}}_{\mathrm{K}}$$

**Remarks**

a. $\mathrm{K} \geq n$ must hold.

(b) The optimal solution is not obtained by this function if the two-dimensional coefficient matrix $G$ of the objective function is not positive semidefinite.

(c) If a non-positive value is entered for argument NEV, then $\mathrm{NEV} = \mathrm{N} - \mathrm{M} - \mathrm{ME}$ is set as the default value.

(d) If the specified number of iterations is too small and IERR=5000 is returned, perform the calculation using the information calculated up that point. To perform this processing set ISW=1 and use the output values from the previous time as input values. The contents of work arrays W and IW must be saved in advance. If the number of iterations has already reached $\mathrm{N} + \mathrm{M} - \mathrm{ME}$, no solution will be obtained even if further iterations are performed.

(7) **Example**

(a) Problem

Minimize the function

$$f(\boldsymbol{x}) = \frac{1}{2}(5x_1^2 + 6x_1x_2 + 5x_2^2) - 95x_1 - 105x_2$$

under the constraints

$$\begin{aligned} -x_1 - 2x_2 &\geq -10 \\ -3x_1 - x_2 &\geq -15 \\ -2x_1 - 3x_2 &\geq -30 \\ 15x_1 - 13x_2 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

In this example, to indicate the continuation processing described in Notes (d), set a small value (NEV=2) for the maximum number of evaluations so that IERR=5000 will be output.

(b) Input data

(First time): N=2, M=4, ME=0, NA=11, NEV=0,
arrays A, B, G and C.
(Second and subsequent times): NEV=0 and ISW=1.
(For other values, directly set the values at the end of the previous calculation as input values.)

(c) Main program

```
      PROGRAM BMCQLM
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER ( NO = 2)
      PARAMETER ( MO = 4)
      PARAMETER ( MEO = 0)
      PARAMETER ( NIWK = 2*NO-MEO+MO+2)
      PARAMETER ( NWK = 2*(NO+MO-MEO+2)*(NO+MO-MEO)+(MO+MEO)*(NO+1)+NO)
      PARAMETER ( NA = 11 , NG = 8)
      DIMENSION A(NA,NO),B(MO),G(NG,NO),C(NO),X(NO)
      DIMENSION W(NWK),IW(NIWK)
!
      WRITE(6,1000)
      READ(5,*) N,M,ME,NEV,ISW
      WRITE(6,1100) N,M,ME,NEV,ISW
      WRITE(6,1300)
      DO 10 I = 1,M
        READ(5,*) ( A(I,J),J=1,N )
        WRITE(6,1200) ( A(I,J),J=1,N )
   10 CONTINUE
      WRITE(6,1400)
      READ(5,*) ( B(I),I=1,M )
      WRITE(6,1200) ( B(I),I=1,M )
      WRITE(6,1500)
      DO 20 I = 1,N
        READ(5,*) ( G(I,J),J=1,N )
        WRITE(6,1200) ( G(I,J),J=1,N )
   20 CONTINUE
      WRITE(6,1600)
      READ(5,*) ( C(I),I=1,N )
      WRITE(6,1200) ( C(I),I=1,N )
      CALL DMCQLM&
       (A,NA,N,B,M,ME,G,NG,C,NEV,X,Y,ISW,IW,W,IERR)
      WRITE(6,1700)
      WRITE(6,1800) IERR
      READ(5,*) NEV,ISW
      CALL DMCQLM&
       (A,NA,N,B,M,ME,G,NG,C,NEV,X,Y,ISW,IW,W,IERR)
      WRITE(6,1900)
      WRITE(6,1800) IERR
      WRITE(6,2000)
```

```
      WRITE(6,2200) ( I,X(I),I=1,N )
      WRITE(6,2100) Y
      STOP
!
 1000 FORMAT(' ',/,6X,'*** DMCQLM ***',/,&
                 7X,'** INPUT **')
 1100 FORMAT(6X,'N =',I3,5X,'M =',I3,5X,'ME =',I3&
      ,5X,'NEV =',I3,5X,'ISW =',I3)
 1200 FORMAT(8X,6(F8.2))
 1300 FORMAT(8X,'** MATRIX A **')
 1400 FORMAT(8X,'** VECTOR B **')
 1500 FORMAT(8X,'** MATRIX G **')
 1600 FORMAT(8X,'** VECTOR C **')
 1700 FORMAT(7X,'** OUTPUT **',/,&
              8X,'** FIRST RESULT (ISW .EQ. 0) **')
 1800 FORMAT(9X,'IERR =',I5)
 1900 FORMAT(7X,'** OUTPUT **',/,&
              8X,'** IMPROVED RESULT (ISW .EQ. 1) **')
 2000 FORMAT(8X,'** VECTOR X **')
 2100 FORMAT(9X,'Y =',D18.10)
 2200 FORMAT(8X,'X(',I2,') =',D18.10)
      END
```

(d) Output results

```
      *** DMCQLM ***
       ** INPUT **
      N =  2     M =  4     ME =  0     NEV =  2     ISW =  0
        ** MATRIX A **
           -1.00   -2.00
           -3.00   -1.00
           -2.00   -3.00
           15.00  -13.00
        ** VECTOR B **
          -10.00  -15.00  -30.00    0.00
        ** MATRIX G **
            5.00    3.00
            3.00    5.00
        ** VECTOR C **
          -95.00 -105.00
       ** OUTPUT **
        ** FIRST RESULT (ISW .EQ. 0) **
         IERR = 5000
       ** OUTPUT **
        ** IMPROVED RESULT (ISW .EQ. 1) **
         IERR =    0
        ** VECTOR X **
        X( 1) =  0.4000000000D+01
        X( 2) =  0.3000000000D+01
         Y = -0.5965000000D+03
```

### 5.7.3 DMCQAZ, RMCQAZ
### Minimization of an Unconstrained 0-1 Quadratic Function of Several Variables (Unconstrained 0-1 Quadratic Programming Problem)

(1) **Function**

DMCQAZ or RMCQAZ obtains the $\boldsymbol{x}$ that minimizes a quadratic function of several variables of the $n$ dimensional vector $\boldsymbol{x} = [x_1, \cdots, x_n]^T$ $(x_i \in \{0, 1\})$.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} + (1/2)\boldsymbol{x}^T G \boldsymbol{x}$$

where $G$ is a $n \times n$ matrix and $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ is a $n$ dimensional vector.

(2) **Usage**

Double precision:
CALL DMCQAZ (G, NA, N, C, IR1, IR2, IPM, RPM, IX, Y, IWK, WK, IERR)
Single precision:
CALL RMCQAZ (G, NA, N, C, IR1, IR2, IPM, RPM, IX, Y, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\\text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | G | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | NA,N | Input | Coefficient matrix $G$ of the quadratic term of the objective function |
| 2 | NA | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Input | Adjustable dimension of array G |
| 3 | N | I | 1 | Input | Number of variables $n$ |
| 4 | C | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | N | Input | Coefficient vector $\boldsymbol{c}$ of the linear term of the objective function |
| 5 | IR1 | I | 1 | Input | Random number initial value. (See Notes (a) and (b)) |
| | | | | Output | Next random number initial value |
| 6 | IR2 | I | 1 | Input | Random number initial value. (See Notes (a) and (b)) |
| | | | | Output | Next random number initial value |
| 7 | IPM | I | 12 | Input | Integer parameters |
| | | | | Output | IPM(2-11) : Parameter values actually used. IPM(12) : Actual number of solution evaluations by branch-and-bound method |
| 8 | RPM | I | 5 | Input | Real parameters |
| | | | | Output | RPM(2-5) : Parameter values actually used |
| 9 | IX | I | N | Input | Initial solution $\boldsymbol{x}^{(0)}$ (Only when IPM(2)$\neq$1) |
| | | | | Output | Final destination $\boldsymbol{x}$ |
| 10 | Y | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | 1 | Output | Function value $f(\boldsymbol{x})$ at final destination $\boldsymbol{x}$ |
| 11 | IWK | I | See Contents | Work | Work area<br>**Size**: $\text{N} \times (\text{MAX}(\text{IPM}(7), \text{IPM}(10)) + 3) + \text{IPM}(10) + 13$ |
| 12 | WK | $\left\{\begin{array}{c}\text{D}\\\text{R}\end{array}\right\}$ | See Contents | Work | Work area<br>**Size**: $3 \times \text{N}^2 + 12 \times \text{N} + \text{MAX}(\text{IPM}(10), \text{N}) + (\text{N}+1) \times \text{IPM}(10) + 9$ |
| 13 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $0 < \text{N} \leq \text{NA}, 0 < \text{N}$

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | The branch-and-bound search for the optimal solution failed because of the numerical errors. | The value of X and Y at that time is output and processing is aborted. |
| 5000 | The solution of the relaxation problem could not be obtained within the given number of iterations. | |
| 5500 | The number of active partial problems reached the given limit. | |
| 5600 | The branch-and-bound search did not complete within the given number of evaluations. | |

(6) **Notes**

(a) IX and IY should be odd numbers.

(b) A random number sequence that continues the previously generated random number sequence can be obtained by using the output values of the previous execution for IX and IY.

(c) To search for the maximum value, perform a search for the minimum value of $-f(\boldsymbol{x})$. At this time, the value of Y is the maximum value with the sign reversed.

(d) If IERR = 4000, 5000, 5500 or 5600 is returned and you try to continue the calculation, you can perform it efficiently by using the solution obtained in the previous execution as the initial solution. To perform this processing set the integer other than 1 for IPM(1) and IPM(2) values, and use the output values of the previous execution for all other input values.

(e) When $n$ is too large, it takes very long calculation time to search the optimal solution by the branch-and-bound method. In such a case, set the value other than 0 for IPM(6) and you can search the optimal solution only heuristically, which can be used as an approximate solution for the true optimal solution but has no guarantee for the accuracy.

(f) The accuracy of the lower bound tests for the branch-and-bound method gets better sometimes when the parameter IPM(8) is set a large value. But the calculation time for each lower bound test increases exponentially as the value of IPM(8) increases. Therefore, you should usually set 1 for the the values of it.

(g) As the value of the parameter RPM(2) increases the calculation time by the branch-and-bound method increases but the numerical stability gets better and it makes IERR harder to return 4000 or 5000.

Table 5−1 Parameter Table (Unconstrained Zero-One Quadratic Programming Problem)

| Argument name | Default value | Contents |
|---|---|---|
| IPM(1) | – | Setting of default values<br>0:Set default values of the integer parameters IPM(2-12)<br>Other than 0:The user sets the values of the integer parameters IPM(2-12) |
| IPM(2) | 1 | Initial solution setting selection switch<br>0:The initial solution $\boldsymbol{x}^{(0)}$ is set automatically<br>Other than 0:The user sets the initial solution $\boldsymbol{x}^{(0)}$ |
| IPM(3) | 0 | Heuristic search method selection switch<br>0:Obtain the initial solution by the simulated annealing and improve it by the tabu search.<br>1:Use the simulated annealing<br>2:Use the tabu search<br>Otherwise:The heuristic search is not performed |
| IPM(4) | 10000 | The evaluation number of search by the simulated annealing.<br>(When the value less than or equal to 0 is set, the default value is used.) |
| IPM(5) | 100 | The evaluation number of search by the tabu search.<br>(When the value less than or equal to 0 is set, the default value is used.) |
| IPM(6) | 0 | Selection switch if the user perform the search of the solution by the branch-and-bound method or not<br>0:Perform the search of the solution by branch-and-bound method<br>Other than 0:Does not perform the search of the solution by branch-and-bound method |
| IPM(7) | IPM(5) | The length of the tabu list<br>(When the value less than or equal to 0 is set, the default value is used.) |
| IPM(8) | 1 | The number of free variables which are used for the evaluating the lower bounds of partial problems in the branch-and-bound search<br>(When the value other than 0 to 10 is set, the default value is used.) |
| IPM(9) | 1 | The maximum iteration number for solving relaxation quadratic programming problems<br>(When the value less than or equal to 0 is set, the default value is used.) |
| IPM(10) | See Contents | The maximum number of active partial problems in the branch-and-bound search<br>Default value:MAX(N,IPM(7))<br>(When the value less than or equal to 0 is set, the default value is used.) |

Table 5−1 Parameter Table (Unconstrained Zero-One Quadratic Programming Problem) (cont'd)

| Argument name | Default value | Contents |
|---|---|---|
| IPM(11) | 1 | The depth of search for the branch-and-bound method (When the value less than or equal to 0 is set, the default value is used.) |
| IPM(12) | 100×N | The maximum evaluation number for the branch-and-bound method. (When the value less than or equal to 0 is set, the default value is used.) |
| RPM(1) | – | Setting of default values<br>0.0:Set default values of the real parameters RPM(2-5)<br>Other than 0.0:The user sets the values of the real parameters RPM(2-5) |
| RPM(2) | See Contents | Required precision for solving the relaxation quadratic programming problem<br>Default value: $\sqrt{\text{Unit for determining error}}$<br>(When the value less than or equal to 0.0 is set, the default value is used.) |
| RPM(3) | 1.0 | The minimum eigen value $\beta$ of the coefficient matrix which has been transformed into a positive definite matrix. (When the value less than or equal to 0.0 is set, the default value is used.) |
| RPM(4) | 50000.0 | The initial temperature $T_0$ for the simulated annealing search (When the value less than 0.0 is set, the default value is used.) |
| RPM(5) | 0.999 | The ratio of the temperature reduction $\alpha$ (When the value less than or equal to 0.0 or larger than 1.0 is set, the default value is used.) |

(7) **Example**

(a) Problem

Obtain the zero-one vector $\boldsymbol{x}^*$ that minimize the objective function

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x}$$

and the function value $f(\boldsymbol{x}^*)$. where the matrix $G$ and the vector $\boldsymbol{c}$ are given as follows.

$$G = \begin{pmatrix} 1 & -2 & -1 & 3 & 2 & 0 \\ -1 & 4 & 2 & 1 & 5 & -4 \\ 0 & 1 & 3 & 0 & -3 & 2 \\ 2 & 5 & 0 & -3 & 1 & 3 \\ 1 & 1 & 5 & 3 & -3 & 2 \\ 4 & 5 & -1 & -1 & 3 & 1 \end{pmatrix}$$

$$\boldsymbol{c} = (1, 1, 1, -1, -1, -1)^T$$

(b) Input data

NA=7, N=6, IR1=1, IR2=1, IPM(1)=0, RPM(1)=0.0, coefficient matrix $G$ and coefficient vector $\boldsymbol{c}$.

(c) Main program

```
      PROGRAM BMCLMZ
      IMPLICIT NONE
!
      INTEGER NA
      PARAMETER ( NA = 7 )
      REAL(8) G(NA,NA),C(NA),RPM(5),Y,WK(989)
      INTEGER N,IR1,IR2,IPM(12),IX(NA),IWK(731),IERR
      INTEGER I,J
!
      READ(5,*) N
      READ(5,*) IR1,IR2
      IPM(1) = 0
      RPM(1) = 0.0D0
      WRITE(6,6000) NA,N,IR1,IR2
      WRITE(6,6010)
      DO 100 I=1,N
        READ(5,*) (G(I,J),J=1,N)
  100 CONTINUE
      DO 110 I = 1,N
         WRITE(6,6020) (G(I,J),J=1,N)
  110 CONTINUE
      WRITE(6,6030)
      READ(5,*) (C(J),J=1,N)
      WRITE(6,6020) (C(J),J=1,N)
      CALL DMCQAZ(G,NA,N,C,IR1,IR2,IPM,RPM,IX,Y,IWK,WK,IERR)
      WRITE(6,6090)
      WRITE(6,6100) IERR
      DO 150 J = 1,N
         WRITE(6,6110) J,IX(J)
  150 CONTINUE
      WRITE(6,6120) Y
!
 6000 FORMAT(3X,'** DMCQAZ **',/,/,&
             5X,'** INPUT **',/,/,&
             7X,'NA  = ',I5,8X,' N   = ',I5,/,&
             7X,'IR1 = ',I5,8X,' IR2 = ',I5)
 6010 FORMAT(/,7X,'MATRIX G')
 6020 FORMAT(/,9X,6(F5.1))
 6030 FORMAT(/,7X,'VECTOR C')
 6090 FORMAT(/,5X,'** OUTPUT **',/)
 6100 FORMAT(8X,'IERR = ',I5,/)
 6110 FORMAT(8X,'IX(',I2,') = ',I5)
 6120 FORMAT(/,8X,'Y  =  ',D15.5)
!
      STOP
      END
```

(d) Output results

```
   ** DMCQAZ **

     ** INPUT **

       NA  =     7          N   =     6
       IR1 =     1          IR2 =     1

       MATRIX G

           1.0 -2.0 -1.0  3.0  2.0  0.0

          -1.0  4.0  2.0  1.0  5.0 -4.0

           0.0  1.0  3.0  0.0 -3.0  2.0

           2.0  5.0  0.0 -3.0  1.0  3.0

           1.0  1.0  5.0  3.0 -3.0  2.0

           4.0  5.0 -1.0 -1.0  3.0  1.0

       VECTOR C

           1.0  1.0  1.0 -1.0 -1.0 -1.0

     ** OUTPUT **

        IERR =     0

        IX( 1) =     0
        IX( 2) =     0
        IX( 3) =     0
        IX( 4) =     1
        IX( 5) =     1
        IX( 6) =     0

        Y  =     -0.30000D+01
```

## 5.8 MINIMIZATION OF A CONSTRAINED FUNCTION OF SEVERAL VARIABLES (NONLINEAR PROGRAMMING)

### 5.8.1 DMSQPM, RMSQPM
### Minimization of a Constrained Function of Several Variables (Nonlinear Constraints)

(1) **Function**

DMSQPM or RMSQPM obtains $\boldsymbol{x}^*$ that locally minimizes the function $f(\boldsymbol{x})$ of several variables under the $m$ inequality constraints

$$g_i(\boldsymbol{x}) \le 0, \quad i = 1, \cdots, m$$

and $\ell$ equality constraints

$$h_j(\boldsymbol{x}) = 0, \quad j = 1, \cdots, \ell$$

and obtains the function value $f(\boldsymbol{x}^*)$ at that time.

(2) **Usage**

Double precision:

CALL DMSQPM (F, GF, HF, M, ML, X, N, ER, NEV, Y, IWK, WK, IERR)

Single precision:

CALL RMSQPM (F, GF, HF, M, ML, X, N, ER, NEV, Y, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | F | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | – | Input | Name of function subprogram F(X) that defines the function $f(\boldsymbol{x})$ |
| 2 | GF | – | – | Input | Name of subroutine GF(X, N, G, MM) that gives inequality constraints. |
| 3 | HF | – | – | Input | Name of subroutine HF(X, N, H, ML) that gives equality constraints. |
| 4 | M | I | 1 | Input | Number of constraints $m+\ell$ |
| 5 | ML | I | 1 | Input | Number of equality constraints $\ell$ |
| 6 | X | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | N | Input | Initial value of search point $\boldsymbol{x}_0$ |
| | | | | Output | optimal solution $\boldsymbol{x}^*$ |
| 7 | N | I | 1 | Input | Number of components of $\boldsymbol{x}$ |
| 8 | ER | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Input | Unit for determining 0 (Default value: $2\times\sqrt{\text{(Unit for determining error)}}$) |
| 9 | NEV | I | 1 | Input | Maximum number of updates of $\boldsymbol{x}$ (Default value: 100) |
| | | | | Output | Actual number of updates of $\boldsymbol{x}$ |
| 10 | Y | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | 1 | Output | Function value $f(\boldsymbol{x}^*)$ at the final destination $\boldsymbol{x}^*$ |
| 11 | IWK | I | M+3 | Work | Work area |
| 12 | WK | $\left\{\begin{matrix}D\\R\end{matrix}\right\}$ | See Contents | Work | Work area<br>Size: $4\times\mathrm{N}\times\mathrm{N}+2\times\mathrm{M}\times\mathrm{N}+21\times\mathrm{N}+108\times\mathrm{M}$ |
| 13 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) M $>$ 0 (except when 0 is entered in order to solve a problem with no constraints)

(b) ML $\geq$ 0

(c) N $>$ 0

(d) N $\geq$ ML

(e) M $\geq$ ML

(f) ER $>$ 0.0 (except when 0.0 is entered in order to set ER to the default value)

(g) NEV $\geq$ 0(except when a negative value is entered in order to set NEV to the default value)

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 1000 | Restriction (f) or (g) was not satisfied. | Processing is performed with the default value set for ER or NEV. |
| 1500 | Mwas 0. | Processing continues. |
| 3000 | One of restrictions (a) through (e) was not satisfied. | Processing is aborted. |
| 3500 | Equality constraints of the partial quadratic programming problem were not mutually independent. | The values of X and Y at that time are output and processing is aborted. The value of X satisfies constraints. |
| 3600 | The partial quadratic programming problem was not executable. | |
| 3700 | (1)A diagonal element of the matrix, which is to be QR decomposed for solving the partial quadratic programming problem, became 0.0 in the processing of decomposition. (The matrix does not have the designed rank.)<br>(2) A diagonal element of the matrix, which is to be LL$^T$ decomposed for solving the partial quadratic programming problem, became less than or equal to 0.0 in the processing of decomposition. (The matrix is nearly singular.) | |
| 3800 | The partial quadratic programming problem did not converge. | |
| 3900 | The step-size became less than or equal to 0.0 in the processing of rectilinear search. | |
| 4000 | Equality constraints of the partial quadratic programming problem were not mutually independent. | |
| 4100 | The partial quadratic programming problem was not executable. | |

| IERR value | Meaning | Processing |
|---|---|---|
| 4200 | (1)A diagonal element of the matrix, which is to be QR decomposed for solving the partial quadratic programming problem, became 0.0 in the processing of decomposition. (The matrix does not have the designed rank.)<br>(2) A diagonal element of the matrix, which is to be $\mathrm{LL}^T$ decomposed for solving the partial quadratic programming problem, became less than or equal to 0.0 in the processing of decomposition. (The matrix is nearly singular.) | The values of X and Y at that time are output and processing is aborted. The value of X does not satisfy constraints. |
| 4300 | The partial quadratic programming problem did not converge. | |
| 4400 | The step-size became less than or equal to 0.0 in the processing of rectilinear search. | |
| 4500 | The problem could not be solved even though the given maximum number of updates of $\boldsymbol{x}$ was reached. | |
| 5000 | The problem could not be solved even though the given maximum number of updates of $\boldsymbol{x}$ was reached. | |

(6) **Notes**

(a) The actual name of first argument F must be declared by the EXTERNAL statement of the user program. A function subprogram of the actual name of F must be created beforehand.
The function subprogram (in double-precision) should be created as follows.

```
REAL(8) FUNCTION F(X)
REAL(8) X
DIMENSION X(N)
```

F = f($\boldsymbol{x}$)

```
RETURN
END
```

(b) The actual names of the second argument GF and the third argument HF must be declared by the EXTERNAL statement of a user program. Subroutine subprograms of the actual names of GF and HF must be created beforehand.
These subroutine subprograms (in double-precision) should be created as follows.

```
SUBROUTINE GF(X, N, G, MM)
INTEGER N, MM
REAL(8) X, G
DIMENSION X(N), G(MM)
```

G(1) = $g_1(\boldsymbol{x})$
G(2) = $g_2(\boldsymbol{x})$

$\vdots$

G(MM) $= g_{\mathrm{MM}}(\boldsymbol{x})$

```
RETURN
END

SUBROUTINE HF(X, N, H, ML)
INTEGER N, ML
REAL(8) X, H
DIMENSION X(N), H(ML)
```

H(1) $= h_1(\boldsymbol{x})$
H(2) $= h_2(\boldsymbol{x})$
$\vdots$
H(ML) $= h_{\mathrm{ML}}(\boldsymbol{x})$

```
RETURN
END
```

The value of argument MM here is the number of inequality constraints.

(c) If the specified maximum number of updates of $\boldsymbol{x}$ is too small and IERR=5000 is returned, perform the calculation using the information calculated up that point. When performing this processing, use the output value of X from the previous time as the input value.

(7) **Example**

(a) Problem
Minimize the function

$$f(\boldsymbol{x}) = -x_3$$

under the constraints

$$\begin{aligned} -x_1 + x_2 &\leq 0 \\ {x_1}^2 + {x_2}^2 + {x_3}^2 - 1.0 &= 0 \end{aligned}$$

(b) Input data
Name of function subprogram that calculates the objective function value: F
Name of subroutine that gives inequality constraints: GF
Name of subroutine that gives equality constraints: HF
N=3, M=2, ML=1, ER $= 1.0\text{D}-14$, NEV=100, X(1) $= -0.1$, X(2) $= 1.0$ and X(3) $= 0.1$.

(c) Main program

```
      PROGRAM BMSQPM
! *** EXAMPLE OF DMSQPM ***
      IMPLICIT NONE
      INTEGER MO,NO,NWK
      PARAMETER(MO=2,NO=3)
      PARAMETER(NWK=(4*NO+2*MO+21)*NO+108*MO)
      INTEGER          M,ML,N,NEV,IWK(MO+3),IERR,I
      REAL(8) F,X(NO),ER,Y,WK(NWK)
      EXTERNAL         F,GF,HF
!
      READ(5,*) M,ML,N
      READ(5,*) ER
      READ(5,*) NEV
      DO 10 I=1,N
         READ(5,*) X(I)
   10 CONTINUE
      CALL DMSQPM(F,GF,HF,M,ML,X,N,ER,NEV,Y,IWK,WK,IERR)
```

```
      WRITE(6,601)
      WRITE(6,602) M,ML,N
      WRITE(6,603) IERR
      IF( IERR .EQ. 3000 .OR. IERR .EQ. 4000 ) GOTO 9999
      WRITE(6,604) NEV
      DO 20 I=1,N
         WRITE(6,605) I,X(I)
   20 CONTINUE
      WRITE(6,606) Y
 9999 CONTINUE
      STOP
!
  601 FORMAT(' ',/,/,' *** DMSQPM ***',/,2X,'** INPUT **')
  602 FORMAT(6X,'M  = ',I4,/,6X,'ML = ',I4,/,6X,'N  = ',I4)
  603 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5)
  604 FORMAT(6X,'NEV  = ',I5,/)
  605 FORMAT(8X,'X(',I2,') = ',1PD18.10)
  606 FORMAT(8X,'Y     = ',1PD18.10)
      END

      REAL(8) FUNCTION F(X)
      REAL(8) X(3)
      F = -1.0D0 * X(3)
      RETURN
      END

      SUBROUTINE GF(X,N,G,MM)
      INTEGER          N,MM
      REAL(8) X(N),G(MM)
      G(1) = -1.0D0 * X(1) + 1.0D0 * X(2)
      RETURN
      END

      SUBROUTINE HF(X,N,H,ML)
      INTEGER          N,ML
      REAL(8) X(N),H(ML)
      H(1) = X(1)**2 + X(2)**2 + X(3)**2 - 1.0D0
      RETURN
      END
```

(d) Output results

```
 *** DMSQPM ***
  ** INPUT **
      M  =    2
      ML =    1
      N  =    3
  ** OUTPUT **
      IERR =     0
      NEV  =    15

        X( 1) =  -3.6562662107D-17
        X( 2) =  -3.7019577088D-16
        X( 3) =   1.0000000000D+00
        Y     =  -1.0000000000D+00
```

## 5.9 DISTANCE MINIMIZATION ON A GRAPH (SHORTEST PATH PROBLEM)

### 5.9.1 DMSP1M, RMSP1M
### Distance Minimization for a Given Node to the Other Node on a Graph

(1) **Function**

On a graph that has $n$ nodes and $m$ branches and for which all branches have nonnegative weights, this subroutine obtains the path $P = (v_1, v_2, \cdots, v_p)$ for which the sum $W(P)$ of the weights $w(k_j)$ $(v_j, v_{j+1})$ of the branches from a given node $v_1$ to the other nodes $v_p$ is the minimum and this subroutine also obtains the value of $W(P)$ (shortest distance) at that time.

$$\text{Objective function} \quad : \quad W(P) = \sum_{j=1}^{p-1} w(k_j) \rightarrow Minimum$$

(2) **Usage**

Double precision:

CALL DMSP1M (N, M, ITL, IHD, WGHT, INIT, D, IP, ISW, IWK, WK, IERR)

Single precision:

CALL RMSP1M (N, M, ITL, IHD, WGHT, INIT, D, IP, ISW, IWK, WK, IERR)

(3) **Arguments**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{INTEGER(4) as for 32bit Integer}\\ \text{INTEGER(8) as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

| No. | Argument | Type | Size | Input/ Output | Contents |
|---|---|---|---|---|---|
| 1 | N | I | 1 | Input | Number of nodes $n$ |
| 2 | M | I | 1 | Input | Number of branches $m$ |
| 3 | ITL | I | M | Input | Node number of starting point of branch k, tail(k) |
| 4 | IHD | I | M | Input | Node number of ending point of branch k, head(k) |
| 5 | WGHT | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | M | Input | Weight of branch $k$, $w(k)$ |
| 6 | INIT | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | 1 | Input | Node number of starting point $v_1$ |
| 7 | D | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Output | Shortest distance from the starting point $v_1$ to each node $v_i$. (See Note (a)) |
| 8 | IP | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | N | Output | New node number of each node $v_i$ on the shortest path. (See Note (b)) |
| 9 | ISW | I | 1 | Input | Processing switch. (See Note (c))<br>ISW=0:directed graph<br>ISW=1:undirected graph |
| 10 | IWK | I | See Contents | Work | Work area.<br>**Size**: $4 \times \text{N} + 2 \times \text{M}$ |
| 11 | WK | $\left\{\begin{array}{c}\text{D}\\ \text{R}\end{array}\right\}$ | $2 \times \text{M}$ | Work | Work area. |
| 12 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) $\text{N} \geq 2$

(b) $\text{M} \geq 1$

(c) $\text{WGHT}(k) \geq 0.0, \quad (k = 1, \cdots, \text{M})$

(d) $1 \leq \text{ITL}(k) \leq \text{N}, \quad (k = 1, \cdots, \text{M})$

(e) $1 \leq \text{IHD}(k) \leq \text{N}, \quad (k = 1, \cdots, \text{M})$

(f) ISW=0 or ISW=1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (c) was not satisfied. | |
| 3200 | Restriction (d) or (e) was not satisfied. | |
| 3300 | Restriction (f) was not satisfied. | |

(6) **Notes**

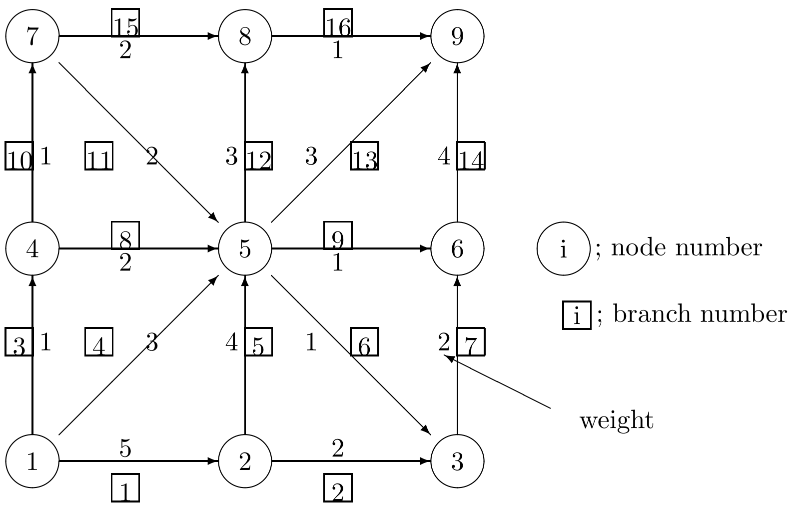
(a) When the value of $D(v_1, v_p)$ is negative, no path exists from node $v_1$ to node $v_p$.

(b) When the series of nodes on the shortest path from node $v_1$ to node $v_p$ is $(v_1, \delta, \cdots, \beta, \alpha, v_p)$, these nodes are obtained so that $\mathrm{IP}(v_p) = \alpha, \mathrm{IP}(\alpha) = \beta, \cdots, \mathrm{IP}(\delta) = v_1$ sequentially from the ending point to the starting point.

(c) For an undirected graph, since each branch is automatically replaced by two directed branches within the subroutine, the input data need not be duplicated in advance.

(7) **Example**

(a) Problem
Obtain the path for which the sum of the weights of the branches from starting point $v_1$ to node $v_p$ on the following kind of graph is the minimum. However, assume that the weights of all branches are nonnegative.




(b) Input data
N=9, M=16, array WGHT for storing the weight of each branch, arrays ITL and IHD for storing the node numbers of the branches, starting point INIT and ISW=0.

(c) Main program

```
      PROGRAM BMSP1M
!
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER ( NMAX=10 )
      PARAMETER ( MMAX=40 )
      DIMENSION ITL(MMAX),IHD(MMAX),IP(NMAX)
      DIMENSION WGHT(MMAX),D(NMAX)
```

```
      DIMENSION IWK(4*NMAX+2*MMAX),WK(2*MMAX)
!
      READ(5,*) N,M
      DO 100 I=1,M
         READ(5,*)ITL(I),IHD(I),WGHT(I)
 100  CONTINUE
!
      INIT=1
      ISW=0
      WRITE(6,6000)N,M,INIT,ISW
      DO 110 I=1,M
         WRITE(6,6010)ITL(I),IHD(I),WGHT(I)
 110  CONTINUE
      CALL DMSP1M(N,M,ITL,IHD,WGHT,INIT,D,IP,ISW,IWK,WK,IERR)
!
      WRITE(6,6020)IERR
      DO 120 I=1,N
         WRITE(6,6030)I,D(I),I,IP(I)
 120  CONTINUE
      STOP
 6000 FORMAT(/,5X,'*** DMSP1M ***',/,&
             /,5X,' ** INPUT **',/,/,&
             8X,'N    = ',I5,/,&
             8X,'M    = ',I5,/,&
             8X,'INIT = ',I5,/,&
             8X,'ISW  = ',I5,/,&
             8X,'  ITL  IHD       WGHT')
 6010 FORMAT(8X,I5,I5,F10.2)
 6020 FORMAT(/,5X,'**  OUTPUT  **',/,/,&
             8X,'IERR = ',I5,/)
 6030 FORMAT(8X,' D(',I2,') = ',F10.2,8X,'IP(',I2,') =',I10)
      END
```

(d) Output results

```
     *** DMSP1M ***

      ** INPUT **

        N    =     9
        M    =    16
        INIT =     1
        ISW  =     0
          ITL  IHD       WGHT
            1    2       5.00
            2    3       2.00
            1    4       1.00
            1    5       3.00
            2    5       4.00
            5    3       1.00
            3    6       2.00
            4    5       2.00
            5    6       1.00
            4    7       1.00
            7    5       2.00
            5    8       3.00
            5    9       3.00
            6    9       4.00
            7    8       2.00
            8    9       1.00

     **  OUTPUT  **

        IERR =     0

         D( 1) =       0.00        IP( 1) =         1
         D( 2) =       5.00        IP( 2) =         1
         D( 3) =       4.00        IP( 3) =         5
         D( 4) =       1.00        IP( 4) =         1
         D( 5) =       3.00        IP( 5) =         1
         D( 6) =       4.00        IP( 6) =         5
         D( 7) =       2.00        IP( 7) =         4
         D( 8) =       4.00        IP( 8) =         7
         D( 9) =       5.00        IP( 9) =         8
```

## 5.9.2 DMSPMM, RMSPMM Distance Minimization for All Sets of Two Nodes on a Graph

(1) **Function**

On a graph that has $n$ nodes and $m$ branches and that satisfies the condition that it contains no branch with a negative weight when it is a nondirected graph or satisfies the condition that it contains no cycle with a negative length when it is a directed graph, this subroutine obtains the path $P = (v_{i1}, v_{i2}, \cdots, v_{ip})$ for which the sum $W(P)$ of the weights of the branches between two nodes $(v_{i1}, v_{ip})(i = 1, \cdots, n)$ is the minimum and this subroutine also obtains the value of $W(P)$ (distance) at that time.

$$\text{Objective function} \quad : \quad W(P) = \sum_{j=1}^{p-1} w(k_j) \rightarrow Minimum$$

(2) **Usage**

Double precision:
CALL DMSPMM (N, M, ITL, IHD, WGHT, D, IP, ISW, IWK, IERR)
Single precision:
CALL RMSPMM (N, M, ITL, IHD, WGHT, D, IP, ISW, IWK, 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 nodes $n$ |
| 2 | M | I | 1 | Input | Number of branches $m$ |
| 3 | ITL | I | M | Input | Node number of starting point of branch k, tail(k) |
| 4 | IHD | I | M | Input | Node number of ending point of branch k, head(k) |
| 5 | WGHT | {D / R} | M | Input | Weight of branch $k$, $w(k)$ |
| 6 | D | {D / R} | N, N | Output | Shortest distance from the starting point $v_1$ to each node $v_i$. (See Note (a)) |
| 7 | IP | {D / R} | N, N | Output | New node number of each node $v_i$ on the shortest path. (See Note (b)) |
| 8 | ISW | I | 1 | Input | Processing switch. (See Note (c))<br>ISW=0:directed graph<br>ISW=1:undirected graph |
| 9 | IWK | I | N, N | Work | Work area. |
| 10 | IERR | I | 1 | Output | Error indicator |

(4) **Restrictions**

(a) N $\geq$ 2
(b) M $\geq$ 1
(c) WGHT($k$) $\geq$ 0.0, $(k = 1, \cdots,$ M)
(d) 1 $\leq$ ITL($k$) $\leq$ N, $(k = 1, \cdots,$ M)
(e) 1 $\leq$ IHD($k$) $\leq$ N, $(k = 1, \cdots,$ M)
(f) ISW=0 or ISW=1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (c) was not satisfied. | |
| 3200 | Restriction (d) or (e) was not satisfied. | |
| 3300 | Restriction (f) was not satisfied. | |
| 4000 | The given graph had an negative cycle. | |

(6) **Notes**

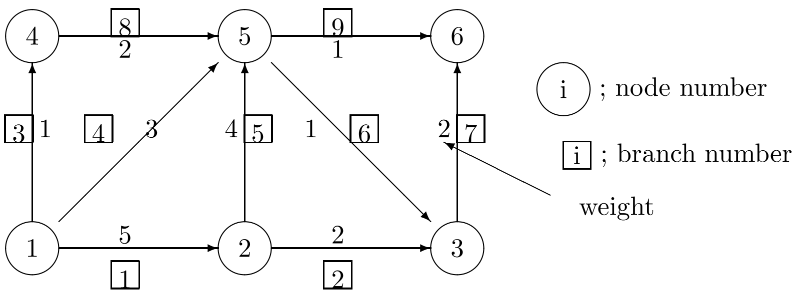
(a) When the value of $D(v_1, v_p)$ is negative, no path exists from node $v_1$ to node $v_p$.

(b) When the series of nodes on the shortest path from node $v_1$ to node $v_p$ is $(v_1, \delta, \cdots, \beta, \alpha, v_p)$, these nodes are obtained so that $\mathrm{IP}(v_1, v_p) = \alpha, \mathrm{IP}(v_1, \alpha) = \beta, \cdots, \mathrm{IP}(v_1, \delta) = v_1$ sequentially from the ending point to the starting point.

(c) For an undirected graph, since each branch is automatically replaced by two directed branches within the subroutine, the input data need not be duplicated in advance.

(7) **Example**

(a) Problem
Obtain the path for which the sum of the weights of the branches from starting point $v_1$ to node $v_p$ on the following kind of graph is the minimum. However, assume that the weights of all branches are nonnegative.




(b) Input data
N=6, M=9, array WGHT for storing the weight of each branch, arrays ITL and IHD for storing the node numbers of the branches, starting point INIT and ISW=1.

(c) Main program

```
      PROGRAM BMSPMM
!
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER ( NMAX=6 )
      PARAMETER ( MMAX=18 )
      DIMENSION ITL(MMAX),IHD(MMAX),IP(NMAX,NMAX)
      DIMENSION WGHT(MMAX),D(NMAX,NMAX)
      DIMENSION IWK(NMAX*NMAX)
!
      READ(5,*) N,M
      DO 100 I=1,M
         READ(5,*)ITL(I),IHD(I),WGHT(I)
 100  CONTINUE
!
      ISW=0
      WRITE(6,6000) N,M,ISW
      DO 110 I=1,M
         WRITE(6,6010)ITL(I),IHD(I),WGHT(I)
 110  CONTINUE
      CALL DMSPMM(N,M,ITL,IHD,WGHT,D,IP,ISW,IWK,IERR)
!
      WRITE(6,6020)IERR
      DO 120 I=1,N
         DO 130 J=1,N
            IF(D(I,J).NE.0) WRITE(6,6030)I,J,D(I,J),I,J,IP(I,J)
 130     CONTINUE
 120  CONTINUE
      STOP
 6000 FORMAT(/,5X,'*** DMSPMM ***',/,&
             /,5X,' ** INPUT **',/,/,&
             8X,'N    = ',I5,/,&
             8X,'M    = ',I5,/,&
             8X,'ISW  = ',I5,/,&
             8X,'  ITL  IHD      WGHT')
 6010 FORMAT(8X,I5,I5,F10.2)
 6020 FORMAT(/,5X,' **  OUTPUT  **',/,/,&
             8X,'IERR = ',I5,/)
 6030 FORMAT(8X,' D(',I2,',',I2,') = ',F10.2,&
             '     IP(',I2,',',I2,') = ',I5)
      END
```

(d) Output results

```
     *** DMSPMM ***

      ** INPUT **

        N    =     6
        M    =     9
        ISW  =     0
          ITL  IHD      WGHT
            1    2      5.00
            2    3      2.00
            1    4      1.00
            1    5      3.00
            2    5      4.00
            5    3      1.00
            3    6      2.00
            4    5      2.00
            5    6      1.00

      **  OUTPUT  **

        IERR =     0

         D( 1, 2) =       5.00     IP( 1, 2) =     1
         D( 1, 3) =       4.00     IP( 1, 3) =     5
         D( 1, 4) =       1.00     IP( 1, 4) =     1
         D( 1, 5) =       3.00     IP( 1, 5) =     1
         D( 1, 6) =       4.00     IP( 1, 6) =     5
         D( 2, 3) =       2.00     IP( 2, 3) =     2
         D( 2, 5) =       4.00     IP( 2, 5) =     2
         D( 2, 6) =       4.00     IP( 2, 6) =     3
         D( 3, 6) =       2.00     IP( 3, 6) =     3
         D( 4, 3) =       3.00     IP( 4, 3) =     5
         D( 4, 5) =       2.00     IP( 4, 5) =     4
         D( 4, 6) =       3.00     IP( 4, 6) =     5
         D( 5, 3) =       1.00     IP( 5, 3) =     5
         D( 5, 6) =       1.00     IP( 5, 6) =     5
```

### 5.9.3 DMSP11, RMSP11
### Distance Minimization for Two Nodes on a Graph

(1) **Function**

On a graph that has $n$ nodes and $m$ branches and for which all branches have nonnegative weights, this subroutine obtains the path $P = (v_1, v_2, \cdots, v_i)$ for which the sum $W(P)$ of the weights $w(k_j)$ $(v_j, v_{j+1})$ of the branches from a given node $v_1$ to the other nodes $v_p$ is the minimum and this subroutine also obtains the value of $W(P)$ (shortest distance) at that time.

$$\text{Objective function} \quad : \quad W(P) = \sum_{j=1}^{p-1} w(k_j) \to min$$

(2) **Usage**

Double precision:

CALL DMSP11 (N, M, ITL, IHD, WGHT, INIT, IEND, D, IP, ISW, IWK, WK, IERR)

Single precision:

CALL RMSP11 (N, M, ITL, IHD, WGHT, INIT, IEND, D, IP, ISW, IWK, WK, 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 nodes $n$ |
| 2 | M | I | 1 | Input | Number of branches $m$ |
| 3 | ITL | I | M | Input | Node number of starting point of branch k, tail(k) |
| 4 | IHD | I | M | Input | Node number of ending point of branch k, head(k) |
| 5 | WGHT | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | M | Input | Weight of branch $k$, $w(k)$ |
| 6 | INIT | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Node number of starting point $v_1$ |
| 7 | IEND | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Input | Node number of ending point $v_p$ |
| 8 | D | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | 1 | Output | Shortest distance from the starting point $v_1$ to each node $v_i$. (See Note (a)) |
| 9 | IP | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | N | Output | New node number of each node $v_i$ on the shortest path. (See Note (b)) |
| 10 | ISW | I | 1 | Input | Processing switch. (See Note (c))<br>ISW=0:directed graph<br>ISW=1:undirected graph |
| 11 | IWK | I | See Contents | Work | Work area.<br>**Size**: $4 \times \text{N} + 2 \times \text{M}$ |
| 12 | WK | $\begin{Bmatrix} \text{D} \\ \text{R} \end{Bmatrix}$ | $2 \times \text{M}$ | Work | Work area. |
| 13 | IERR | I | 1 | Output | Error indicator |

## (4) Restrictions

(a) $\text{N} \geq 2$

(b) $\text{M} \geq 1$

(c) $\text{WGHT}(k) \geq 0.0, \quad (k = 1, \cdots, \text{M})$

(d) $1 \leq \text{ITL}(k) \leq \text{N}, \quad (k = 1, \cdots, \text{M})$

(e) $1 \leq \text{IHD}(k) \leq \text{N}, \quad (k = 1, \cdots, \text{M})$

(f) $1 \leq \text{INIT} \leq \text{N}$

(g) $1 \leq \text{IEND} \leq \text{N}$

(h) ISW=0 or ISW=1

(5) **Error indicator**

| IERR value | Meaning | Processing |
|---|---|---|
| 0 | Normal termination. | |
| 3000 | Restriction (a) or (b) was not satisfied. | Processing is aborted. |
| 3100 | Restriction (c) was not satisfied. | |
| 3200 | Restriction (d) or (e) was not satisfied. | |
| 3300 | Restriction (f) or (g) was not satisfied. | |
| 3400 | Restriction (h) was not satisfied. | |

(6) **Notes**

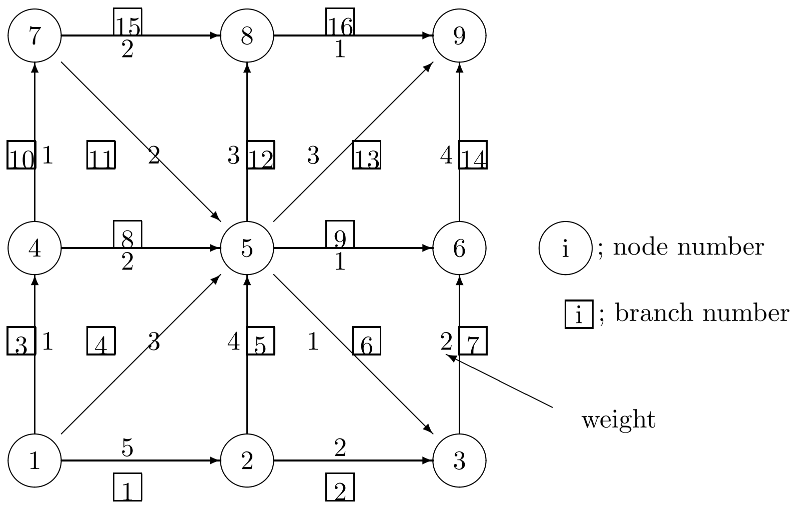
(a) When the value of $D(v_1, v_p)$ is negative, no path exists from node $v_1$ to node $v_p$.

(b) When the series of nodes on the shortest path from node $v_1$ to node $v_p$ is $(v_1, \delta, \cdots, \beta, \alpha, v_p)$, these nodes are obtained so that $\mathrm{IP}(v_p) = \alpha, \mathrm{IP}(\alpha) = \beta, \cdots, \mathrm{IP}(\delta) = v_1$ sequentially from the ending point to the starting point.

(c) For an undirected graph, since each branch is automatically replaced by two directed branches within the subroutine, the input data need not be duplicated in advance.

(7) **Example**

(a) Problem
Obtain the path for which the sum of the weights of the branches from starting point $v_1$ to ending point $v_p$ on the following kind of graph is the minimum. However, assume that the weights of all branches are nonnegative.




(b) Input data
N=9, M=16, array WGHT for storing the weight of each branch, arrays ITL and IHD for storing the node numbers of the branches, starting point INIT =1, ending point IEND=8 and ISW=0.

(c) Main program

```
      PROGRAM BMSP11
!
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER ( NMAX=10 )
      PARAMETER ( MMAX=40 )
```

```
      DIMENSION ITL(MMAX),IHD(MMAX),IP(NMAX)
      DIMENSION WGHT(MMAX)
      DIMENSION IWK(4*NMAX+2*MMAX),WK(2*MMAX+NMAX)
!
      READ(5,*) N,M
      DO 100 I=1,M
         READ(5,*)ITL(I),IHD(I),WGHT(I)
 100  CONTINUE
!
      INIT=1
      IEND=8
      ISW=0
      WRITE(6,6000) N,M,INIT,IEND,ISW
      DO 110 I=1,M
         WRITE(6,6010)ITL(I),IHD(I),WGHT(I)
 110  CONTINUE
      CALL DMSP11(N,M,ITL,IHD,WGHT,INIT,IEND,D,IP,ISW,IWK,WK,IERR)
!
      WRITE(6,6020)IERR
      WRITE(6,6030)D
      DO 120 I=1,N
         WRITE(6,6040)I,IP(I)
 120  CONTINUE
      STOP
 6000 FORMAT(/,5X,'*** DMSP11 ***',/,&
             /,5X,' ** INPUT **',/,/,&
             8X,'N    = ',I5,/,&
             8X,'M    = ',I5,/,&
             8X,'INIT = ',I5,/,&
             8X,'IEND = ',I5,/,&
             8X,'ISW  = ',I5,/,&
             8X,'  ITL  IHD      WGHT')
 6010 FORMAT(8X,I5,I5,F10.2)
 6020 FORMAT(/,5X,' **  OUTPUT  **',/,/,&
             8X,'IERR = ',I5,/)
 6030 FORMAT(8X,'D    = ',F10.2,/)
 6040 FORMAT(8X,'IP(',I2,') =',I5)
      END
```

(d) Output results

```
     *** DMSP11 ***

      ** INPUT **

        N    =     9
        M    =    16
        INIT =     1
        IEND =     8
        ISW  =     0
          ITL  IHD      WGHT
            1    2      5.00
            2    3      2.00
            1    4      1.00
            1    5      3.00
            2    5      4.00
            5    3      1.00
            3    6      2.00
            4    5      2.00
            5    6      1.00
            4    7      1.00
            7    5      2.00
            5    8      3.00
            5    9      3.00
            6    9      4.00
            7    8      2.00
            8    9      1.00

      **  OUTPUT  **

        IERR =     0

        D    =       4.00

        IP( 1) =    0
        IP( 2) =    1
        IP( 3) =    5
        IP( 4) =    1
        IP( 5) =    1
        IP( 6) =    5
        IP( 7) =    4
        IP( 8) =    7
        IP( 9) =    5
```

# Appendix A

# GLOSSARY

(1) **Graph**
For the finite set of points denoted by $V = \{v_1, v_2, \cdots, v_n\}$ and the set of pairs of points belonging to $V$ denoted by $V \times V = \{(v_i, v_j)|v_i \in V, v_j \in V\}$, the combination of $E \subseteq V \times V$ and $V$ is called a graph, which is denoted by $G = (V, E)$.

(2) **Vertex and Edge**
For a given graph $G = (V, E)$, an element of $V$ is called a vertex of graph $G$ and an element of $E$ is called an edge of graph $G$.
**Note:** The following is a list of synonyms which have been used in the literature, not always with the indicated pairs:

| | | | | | |
|---|---|---|---|---|---|
| vertex | point | node | junction | 0-simplex | element |
| edge | line | arc | branch | 1-simplex | element |

(3) **Directed edge**
For graph $G = (V, E)$, if the elements of $E$ are handled so that $(v_i, v_j)$ and $(v_j, v_i)$ $(v_i \in V, v_j \in V)$ are differentiated, each element of $E$ is called a directed edge. For the directed edge denoted by $e = (v_i, v_j)$, $tail(e) = v_i$ and $head(e) = v_j$.

(4) **Circuit**
For graph $G = (V, E)$, if the collection of elements of $V$ denoted by $p = (v_{j_1}, v_{j_2}, \cdots, v_{j_m})$ satisfies $(v_{j_k}, v_{j_{k+1}}) \in E$ for an arbitrary $1 \leq k \leq m-1$, $p$ is called a path on graph $G$. In particular, the path for which $v_{j_1} = v_{j_m}$ is called a circuit.

(5) **Tree**
A graph containing no circuits is called a tree.

(6) **Network**
If two types of real numbers called the cost coefficient $c(e)$ and capacity $u(e)$ and two elements of $V$ called the tail of the edge $s$ and head of the edge $t$ are given for each edge $e \in E$ of graph $G = (V, E)$, the combination of $G$ and $c$, $u$, $s$, and $t$ is called a network, which is denoted by $N = (G, c, u, s, t)$. Even when dealing with only some of $c$, $u$, $s$, and $t$, it is still called a network. For example, when dealing with a problem in which the tail of each edge $s$ and head of each edge $t$ have not specifically been fixed, the network is represented by $N = (G, c, u)$.

(7) **Flow**
The vector $x(e)$ satisfying $0 \leq x(e) \leq u(e)$ for each edge $e \in E$ of a network is called the flow on network $N$.

(8) **Minimal-cost flow problem**
When real numbers $b(v)$ are given for each vertex $v \in V$ of a given network $N$ and the following relationships hold:

$$\sum_{tail(e)=v} x(e) - \sum_{head(e)=v} x(e) = b(v)$$

$$\sum_{v \in V} b(v) = 0$$

the problem of obtaining flows $x(e)$ that minimize:

$$\sum_{e \in E} c(e)x(e)$$

is called the minimal-cost flow problem.

## 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 $\varepsilon$, which is also called the machine $\varepsilon$, 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