ADVANCED SCIENTIFIC LIBRARY
ASL
User's Guide
<Basic Functions Vol.4>

PROPRIETARY NOTICE

PREFACE

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

The manuals corresponding to this product consist of seven volumes, which are divided into the chapters shown below. This manual describes the basic functions, volume 4.

Basic Functions Volume 1

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

Basic Functions Volume 2

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

Basic Functions Volume 3

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

Basic Functions Volume 4

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

Basic Functions Volume 5

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

Basic Functions Volume 6

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

Shared Memory Parallel Functions

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

Document Version 3.0.0-230301 for ASL, March 2023

Remarks

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

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

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

Contents

Chapter 1

INTRODUCTION

1.1 OVERVIEW

1.1.1 Introduction to The Advanced Scientific Library ASL

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

Table 1−1 Classification of functions included in ASL

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

1.1.2 Distinctive Characteristics of ASL

ASL has the following distinctive characteristics.

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

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

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

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

1.2 KINDS OF LIBRARIES

Table 1−2 Kinds of libraries providing ASL

Size of variable(byte)		Declaration of arguments	Kind	Kind of library
integer	real			
4	8	INTEGER(4) REAL(8)	32bit integer Double-precision subroutine	32bit integer library (link option: -lasl_sequential)
4	4	INTEGER(4) REAL(4)	32bit integer Single-precision subroutine	
8	8	INTEGER(8) REAL(8)	64bit integer Double-precision subroutine	64bit integer library (link option: -lasl_sequential_i64)
8	4	INTEGER(8) REAL(4)	64bit integer Single-precision subroutine	

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

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

1.3 ORGANIZATION

This section describes the organization of Chapters 2 and later.

1.3.1 Introduction

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

1.3.2 Organization of Subroutine Description

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

(1) Function

(2) Usage

(3) Arguments

(4) Restrictions

(5) Error indicator

(6) Notes

(7) Example

Each item is described according to the following principles.

1.3.3 Contents of Each Item

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

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

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

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

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

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

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

(a) Arguments

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

(b) Type

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

I : Integer type

D : Double precision real

R : Real

Z : Double precision complex

C : Complex

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

(c) Size

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

1 : Indicates that argument is a variable.

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

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

(d) Input/Output

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

i. When only “Input” appears

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

ii. When only “Output” appears

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

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

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

iv. When “Work” appears

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

(e) Contents

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

- A sample Argument description follows.

Example

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

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

The explanation of the arguments is as follows.

Table 1−3 Sample Arguments

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

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

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

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

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

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

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

(4) **Restrictions**

Restrictions indicate limiting ranges for subroutine arguments.

(5) **Error indicator**

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

Table 1−4 Classification of Error Indicator Output Values

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

(6) **Notes**

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

(7) **Example**

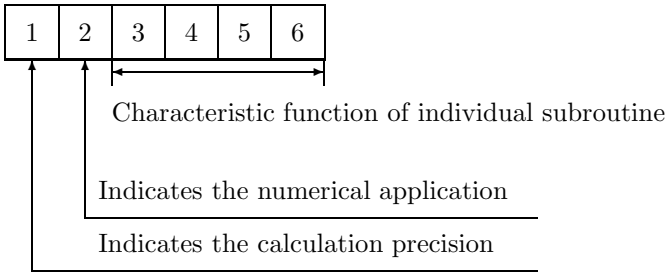
Here gives an example of how to use the subroutine. Note that in some cases, multiple subroutines are combined in a single example. The output results are given in the 32-bit integer version, and may differ within the range of rounding error if the compiler or intrinsic functions are different.

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

1.4 SUBROUTINE NAMES

The subroutines name of ASL 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 letterererererere are used to indicate the application field in the ASL related products.

Letter	Application Field	Volume
A	Storage mode conversion	1
	Basic matrix algebra	1, 7
B	Simultaneous linear equations (direct method)	2, 7
C	Eigenvalues and eigenvectors	1, 7
F	Fourier transforms and their applications	3, 7
	Time series analysis	6
G	Spline function	4
H	Numeric integration	4
I	Special function	5
J	Random number tests	6
K	Ordinary differential equation (initial value problems)	4
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 π or function approximation constants, which are fixed in double-precision operations, for example, are also to be subject to calculations that depend on the length of the multiple length arithmetic operations, the calculation efficiency will be worse than for normal operations.

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

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

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

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

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

Chapter 2

DIFFERENTIAL EQUATIONS AND THEIR APPLICATIONS

2.1 INTRODUCTION

This chapter describes subroutines for ordinary differential equation initial value problems and boundary value problems and integral equations and partial differential equations. Initial value problem subroutines sequentially obtain approximate solutions at discrete points from ordinary differential equations, Ordinary differential equation (boundary value problem) and partial differential equation subroutines obtain approximate solutions at arbitrary points within the boundary, and integral equation subroutines obtain approximate solutions at arbitrary points.

Since this library subroutines described in Sections 2.2.1 through 2.2.4 and Sections 2.3.1 through 2.3.2 correspond to high-order simultaneous ordinary differential equations, they can be used directly, without having to replace the equations by simultaneous ordinary differential equations of the 1st order. Also, simultaneous equations of the 1st order or independent equations of the 1st order can be solved by letting the argument for the order or the argument for the number of simultaneous equations be 1.

If you want to obtain the solution orbit when the step size is automatically controlled in the initial value problem, you also can output the solution for each step size besides outputting the solution at the desired point.

In the boundary value problem, you can use both numerical boundary conditions subroutine, which determine boundary conditions by user specified function values and derivative values at boundary, and function boundary conditions subroutine, which determine boundary conditions by user specified functions and derivatives at boundary.

This library provides subroutines corresponding to equations having the following characteristics.

Ordinary Differential Equations (Initial Value Problems)

(1) High-order simultaneous ordinary differential equations (speed priority), simultaneous ordinary differential equations of the 1st order, and high-order ordinary differential equations
Obtains solutions by using the Runge-Kutta-Verner method with automatic step-size control. Most efficient for non-stiff and weakly stiff problems when the function evaluation cost is low or the precision requirements are not high.

(2) High-order simultaneous ordinary differential equations (precision priority)
Obtains solutions by using a linear multistep method with automatic step-size and automatic degree control. Most efficient for non-stiff and weakly stiff problems when the function evaluation cost is high or the precision requirements are high.

(3) Implicit simultaneous ordinary differential equations
Implicit ordinary differential equations have the form:

$$f_i(x, y_1, y_2, \cdots, y_n, y_1', y_2', \cdots, y_n', \cdots) = 0 \quad \text{for} \quad (i = 1, 2, \cdots, n)$$

where $\dfrac{\partial f_i}{\partial y_j}$ for $(i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$ are non-singular equations. For example, this corresponds to equations such as the following in which y_1', (or y_2') depends on both equation (1) and

equation (2) and cannot be determined from the evaluation of an independent equation.

$$\begin{cases} y_1' y_2' - x^2 = 0 & \text{Initial value } y_1(x_0) = y_{10} \cdots (1) \\ y_1' + y_2' - y_1 - 2x = 0 & \text{Initial value } y_2(x_0) = y_{20} \ldots (2) \end{cases}$$

In addition, this subroutine can solve ordinary differential equations that are taken to be simultaneous with algebraic equations.

An example of ordinary differential equations that are taken to be simultaneous with algebraic equations is as follows. It consists of one group of algebraic equations or nonlinear equations and one group of ordinary differential equations.

$$\begin{cases} y_1' y_2' - x^2 = 0 & \text{Initial value } y_1(x_0) = y_{10} \cdots (1) \\ y_2' + y_3 + 2x = 0 & \text{Initial value } y_2(x_0) = y_{20} \ldots (2) \\ 2y_1 + 3y_2 + y_3 - 1 = 0 & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3) \end{cases}$$

This kind of problem can be solved only by using this subroutine.

Although this subroutine also can solve general high-order simultaneous ordinary differential equations, the calculation efficiency is poor compared with other subroutines since the integration is carried out while solving the nonlinear equations. This subroutine also can solve nonlinear simultaneous equations or nonlinear equations. In this case, partial differential coefficients or differential coefficients need not be input.

(4) Stiff problem high-order simultaneous ordinary differential equations

A stiff problem is a problem for which the solution consists of two or more factors that vary with different scales for variations of the independent variable. An example of a stiff problem is to solve the differential equation $y'' = \lambda y \ \ (\lambda \gg 0)$ with initial conditions $(y = 1, y'' = -\lambda \ \text{ at } \ x = 0)$. The general solution of this differential equation is given by a linear combination of the factors $e^{\lambda x}$ and $e^{-\lambda x}$, which vary with different scales for variations in the independent variable x. Although the correct solution of this problem is $y = e^{-\lambda x}$, if the solution at $x = x_1$ is sought by using:

$$y = e^{-\lambda x_1} + \varepsilon = e^{-\lambda x_1} + \varepsilon' e^{\lambda x_1} \quad (\varepsilon, \varepsilon' : \text{Error in numerical calculations})$$

then as x becomes large:

$$y \to \varepsilon' e^{\lambda x}$$

which has the property that it separates from the correct solution.

This subroutine can be used to efficiently solve such strongly stiff problems.

(5) Ordinary differential equation of the type $M\boldsymbol{y''} + C\boldsymbol{y'} + K\boldsymbol{y} = \boldsymbol{p}(x)$

This subroutine solves the ordinary differential equation $M\boldsymbol{y''} + C\boldsymbol{y'} + K\boldsymbol{y} = \boldsymbol{p}(x)$, which is known as the equation of motion. M, C and K are $n \times n$ matrices called the mass matrix, damping matrix, and stiffness matrix respectively, and $\boldsymbol{p}(x)$ is the external force vector. n represents the number of simultaneous equations. This subroutine carries out processing so that a solution can be obtained even if the mass matrix M is singular such as if it includes zero for a diagonal term (there is a point having mass zero). Also, solutions are obtained for each entered step-size.

Ordinary Differential Equations (Boundary Value Problems)

(1) High-order simultaneous ordinary differential equations, first-order simultaneous ordinary differential equations, and high-order ordinary differential equations

Solves the boundary value problem by finding a suitable initial value according to the multipoint shooting method based on the Runge-Kutta-Verner method. Automatically sets shooting points so

that solutions are obtained accurately and more efficiently than by the general multipoint shooting method and parameterizes the nonlinear calculation portion.

(2) High-order linear ordinary differential equations
Solves the boundary value problem by combining the collocation method within the weighted residual method and the B-spline function. If the ordinary differential equations are linear, solutions can be obtained quickly and precisely.

(3) Second-order linear ordinary differential equations
Generalizes the coefficient determination method for second order ordinary differential equations. Effective for second-order linear ordinary differential equations.

Integral Equations

(1) Fredholm's integral equation of the second kind
Gauss' integration method is used to solve the integral equation to obtain a solution at an arbitrary point by interpolating using a cubic spline function.

(2) Volterra's integral equation of the first kind
Maclaurin's formula is used to solve the integral equation to obtain a solution at an arbitrary point by interpolating using a cubic spline function.

Partial Differential Equations

(1) Two-dimensional inhomogeneous Helmholtz equation
This subroutine uses a two-dimensional five-point difference in a given rectangular area to solve the inhomogeneous Helmholtz equation.

(2) Three-dimensional inhomogeneous Helmholtz equation
This subroutine uses a three-dimensional seven-point difference in a given rectangular area to solve the inhomogeneous Helmholtz equation.

2.1.1 Notes

2.1.1.1 Ordinary Differential Equations (Initial Value Problems)

(1) Since none of these subroutines can obtain the correct solution if a point of discontinuity is included in the calculation range, the calculation range must be divided at any point of discontinuity that exists, and separate calculations must be performed for each subdivided range.

(2) The step size must be extremely small for functions having severe oscillations. However, if the interval for which the problem is to be solved increases, then the cumulative error will increase and precision will decrease.

(3) In many cases, it is unclear whether or not the equations are stiff. In such cases, first try solving the equations by using the subroutine 2.2.2 $\begin{Bmatrix}\text{DKSNCA}\\ \text{RKSNCA}\end{Bmatrix}$. If IERR = 4000 is output by that subroutine, then the equations are considered to be stiff and the subroutine 2.2.4 $\begin{Bmatrix}\text{DKSSCA}\\ \text{RKSSCA}\end{Bmatrix}$ should be used to solve them.

(4) The subroutines 2.2.1 $\begin{Bmatrix}\text{DKSNCS}\\ \text{RKSNCS}\end{Bmatrix}$, 2.2.2 $\begin{Bmatrix}\text{DKSNCA}\\ \text{RKSNCA}\end{Bmatrix}$, and 2.2.4 $\begin{Bmatrix}\text{DKSSCA}\\ \text{RKSSCA}\end{Bmatrix}$ are used for high-order simultaneous ordinary differential equations. However, if the equations are given as conventional simultaneous ordinary differential equations of the 1st order, then do the following.

Conventional	
Function F(X, Y, YP)	$\begin{cases}\text{YP}(1) = f_1(\text{X}, \text{Y}(1), \cdots, \text{Y}(\text{N}))\\ \vdots\\ \text{YP}(\text{N}) = f_n(\text{X}, \text{Y}(1), \cdots, \text{Y}(\text{N}))\end{cases}$
Arguments	
	Interval $x \sim x_f$: X, XF
	Initial value ${y_i}^0$: Y(1), Y(2), $\cdots$, Y(N)
	Number of simultaneous equations n: N

These subroutines	
Function F(X, Y, N)	$\begin{cases}\text{Y}(1,1) = f_1(\text{X}, \text{Y}(1,0), \cdots, \text{Y}(\text{N},0))\\ \vdots\\ \text{Y}(\text{N},1) = f_n(\text{X}, \text{Y}(1,0), \cdots, \text{Y}(\text{N},0))\end{cases}$
Arguments	
	Interval $x \sim x_f$: X, XF
	Initial value ${y_i}^0$: Y(1, 0), Y(2, 0), $\cdots$, Y(N, 0)
	Number of simultaneous equations n: N
Additional arguments	
	Value greater the maximum differential order: MX=1
	Differential order of each equation: M(i) = 1 for ($i = 1, \cdots,$ n)

That is, since Y becomes a two-dimensional array with the simultaneous equation number entered for the left subscript and the differential order entered for the right subscript, the function or initial value must be assigned with two dimensions so that $\mathrm{Y}(i,1)$ represents y_i' and $\mathrm{Y}(i,0)$ represents y_i. Also note that the arguments of the function for which y_i' is to be calculated differ. Moreover, MX and $\mathrm{M}(i)$ are required as additional arguments, and the value 1 must be set for each of them.
If high-order ordinary differential equations are converted to simultaneous type equations, the precision tends to decrease. Therefore, equations should be created directly as high-order equations as much as possible.

(5) Since all subroutines except subroutine 2.2.7 $\begin{Bmatrix}\mathrm{DKMNCN}\\ \mathrm{RKMNCN}\end{Bmatrix}$ output y_i through ${y_i}^{(m)}$ at the XF point or for each step size when automatic step size control is performed, if y_i through ${y_i}^{(m)}$ are required at various points, then either the subroutine must be used continuously while varying XF or output for each automatic step size must be specified and the subroutine must be used continuously with XF fixed at the final point. At this time, the output arguments should be set so they become input arguments for the next execution, and none of the other argument values should be changed, except possibly the XF value.

(6) Subroutine 2.2.7 $\begin{Bmatrix}\mathrm{DKMNCN}\\ \mathrm{RKMNCN}\end{Bmatrix}$ is used differently than the other subroutines since the function is given in the form of coefficient matrices and values are output for each step size.
The solution at the point x_f where the solution is required is obtained as follows. The point where the initial value is given is assumed to be x, $(x_f - x)$ is divided by an integer k and this value is assumed to be the step size Δx. The values ${y_i}^{(j)}$ for $(i = 0, 1, 2)$ are obtained for ${y_i}^{(j)}$ at the point x for $(i = 1, \cdots,$ number of simultaneous equations; $j = 0, 1)$ as initial values, then ${y_i}^{(j)}$ at the point advanced from the previous point by Δx is obtained using the previously obtained ${y_i}^{(j)}$ as initial values, and this task is repeated k times. The larger the value of k that is used, the higher will be the precision.

(7) When you use a subroutine that has a function name as an argument, you must use the EXTERNAL statement to declare the function name that is to be used as the argument.
Example:
Let F has the same name in the main program and the subroutine subprogram

- Main program

```
      ≀
EXTERNAL F
      ≀
CALL DK···(F,···)
CALL RK···(F,···)
      ≀
```

- Subroutine subprogram

```
SUBROUTINE F (···)
      ≀
```

2.1.1.2 Ordinary Differential Equations (Boundary Value Problems)

(1) Since the correct solution is not obtained if there is a point of discontinuity within the interval, the boundary value problem is solved by subdividing the interval at the point of discontinuity.

(2) The subroutine used for nonlinear equations parameterizes the equations by multiplying the nonlinear part by α.
The nonlinear part consists of a multiplication or division of two or more of the ${y_i}^{(j)}$ (i:Array number, j:Differential order) such as $y_1 y_2, \dfrac{y_3'}{y_4''}, y_1 y_1'$ or $(y_1')^2$ or a function other than a sum or scalar multiple of ${y_i}^{(j)}$ such as $\sin(y_1')$ or $\mid y_2 \mid$.
For example, the differential equations:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 y_2 \end{cases}$$

are parameterized as follows:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \alpha y_1 y_2 \end{cases}$$

If there is no nonlinear part, it is unnecessary to parameterize the equations by multiplying by α.
Regardless of whether or not there is a nonlinear part, if parameterizing is not performed, the solution method will be the same as the general multipoint shooting method. This increases the amount of calculations, and in some cases, no solution can be found.
If the linear part is parameterized by mistake, the amount of calculations will be increased, but the solution will not be affected.

(3) When assigning boundary conditions by numerical values, distinguish the starting and ending points by IN, the element number by IB, the differential order by IC, and the value at that boundary by BN. At this time, the element number of IN, IB, IC, and BN must correspond to the respective boundary conditions. The boundary condition input order is arbitrary.

(4) When assigning boundary conditions by a function, if the value of ${y_i}^{(j)}$ at the starting point is ${ya_i}^{(j)}$ and the value at the ending point is ${yb_i}^{(j)}$, determine the boundary conditions by the function $g_k({ya_i}^{(j)}, {yb_i}^{(j)}) = 0$. At this time, the boundary condition input order is arbitrary.

(5) Since no error check of the differential order or number of simultaneous differential equations can be performed within the subroutine, enter these values carefully.

2.1.1.3 Integral Equations

(1) Since neither of these subroutines can obtain the correct solution if there is a point of discontinuity within the interval, the interval must be subdivided at the point of discontinuity and separate calculations must be performed in each subdivision.

(2) In 2.4.2 $\begin{Bmatrix} \text{DOIEV1} \\ \text{ROIEV1} \end{Bmatrix}$, if the number of subdivisions of the integration interval is too large, the cumulative error increases and precision declines.

2.1.2 Algorithms Used

2.1.2.1 Ordinary Differential Equations (Initial Value Problems)

(1) **Runge-Kutta-Verner method**

Since this method enables the truncation error to be estimated, the desired solution is obtained with automatic control of the step size complying with the more lenient value of the required local absolute precision and the required local relative precision.

A single iteration of the Verner method for step size h is expressed as follows for $y' = f(x, y)$.

$$\begin{aligned} y_{n+1} &= y_n + \sum_{i=1}^{8} \gamma_i k_i \\ E &= \sum_{i=1}^{8} \gamma_i^* k_i \\ k_i &= hf\left(x_n + \alpha_i h, y_n + h\sum_{j=1}^{i-1} \beta_{ij} k_j\right) \quad (i = 1, 2, \cdots, 8) \end{aligned}$$

In the expressions shown above, y_{n+1} is the approximate solution having sixth order truncation precision,

Table 2−1 Coefficient Table

i	α_i	β_{i1}	β_{i2}	β_{i3}	β_{i4}	β_{i5}	β_{i6}	β_{i7}	γ_i	γ_i^*
1	0								$\frac{57}{640}$	$\frac{33}{640}$
2	$\frac{1}{18}$	$\frac{1}{18}$							0	0
3	$\frac{1}{6}$	$-\frac{1}{12}$	$\frac{1}{4}$						$-\frac{16}{65}$	$-\frac{132}{325}$
4	$\frac{2}{9}$	$-\frac{2}{81}$	$\frac{4}{27}$	$\frac{8}{81}$					$\frac{1377}{2240}$	$\frac{891}{2240}$
5	$\frac{2}{3}$	$\frac{40}{33}$	$-\frac{4}{11}$	$-\frac{56}{11}$	$\frac{54}{11}$				$\frac{121}{320}$	$-\frac{33}{320}$
6	1	$-\frac{369}{73}$	$\frac{72}{73}$	$\frac{5380}{219}$	$-\frac{12285}{584}$	$\frac{2695}{1752}$			0	$-\frac{73}{700}$
7	$\frac{8}{9}$	$-\frac{8716}{891}$	$\frac{656}{297}$	$\frac{39520}{891}$	$-\frac{416}{11}$	$\frac{52}{27}$	0		$\frac{891}{8320}$	$\frac{891}{8320}$
8	1	$\frac{3015}{256}$	$-\frac{9}{4}$	$-\frac{4219}{78}$	$\frac{5985}{128}$	$-\frac{539}{384}$	0	$\frac{693}{3328}$	$\frac{2}{35}$	$\frac{2}{35}$

and E is the difference between this approximate solution and the approximate solution having fifth order truncation precision. E is assumed to be the truncation error of y_{n+1}.

The method of automatically adjusting the step size h is as follows.

Let ε be max(required absolute precision, required relative precision $\times \left|\frac{y_n + y_{n+1}}{2}\right|$).

Obtain the minimum of $\frac{\varepsilon}{|E|}$ for each of the simultaneous differential equations, and let this value be U.

(a) If $U < 1$ (step size is too big), then update h as follows:
$h = h \max(0.85 \times \sqrt[6]{U}, 0.1)$
and obtain y_{n+1} again.

(b) If $U \geq 1$ (step size is sufficiently small), then:

i. For $1.4 \leq U < 2.4$, leave h unchanged and move on to the calculation for the next step.

ii. For $U < 1.4$ or $U \geq 2.4$, update h as follows:
$h = h \min(0.9 \times \sqrt[6]{U}, 5)$, let y_{n+1} be y_n and move on to the calculation for the next step.

Now, from $y' = f(x, y)$ that is obtained initially, let the error at the final point x_f be $y'(x_f - x)$ and use the following equations to calculate the initial value of h.

$$\begin{aligned} \varepsilon &= \max(\text{required absolute precision}, \text{required relative precision} \times y) \\ h &= \sqrt[6]{\frac{\varepsilon(x_f - x)}{y'(x_f - x)}} \end{aligned}$$

Use the minimum value of h among these for the various simultaneous differential equations.
For high-order differential equation: $y^{(d)} = f(x, y, y', \cdots, y^{(d-1)})$ let:

$$y_1 = y, y_2 = y', y_3 = y'', \cdots, y_d = y^{(d-1)}$$

and consider the following first-order simultaneous differential equations:

$$\begin{cases} y_1' &= y_2 \\ y_2' &= y_3 \\ & \vdots \\ y_d' &= f(x, y_1, \cdots, y_d) \end{cases}$$

The subroutine of this library performs this processing automatically.
Since this method evaluates functions frequently, it is most efficient when function evaluation is not expensive and the precision requirements are not high. (See Reference Bibliography (1).)

(2) **Linear multistep method based on step size or order control quotient difference method**
Let consider differential equation $y^{(d)} = f(x, y, \cdots, y^{(d-1)})$.
Then, let $f(x_n) = f(x_n, y(x_n), \cdots, y^{(d-1)}(x_n))$ and $y_n = y(x_n)$.

(a) If $d = 1$, then the predictor p_{n+1}^0 is given by:

$$p_{n+1}^0 = y_n + \int_{x_n}^{x_{n+1}} f(x)dx$$

(b) If $d > 1$, then for $h_{n+1} = x_{n+1} - x_n$, the predictor $p_{n+1}^{(d-k)}$ is given by:

$$p_{n+1}^{(d-k)} = \sum_{i=0}^{k-1} \frac{h_{n+1}^i}{i!} y_n^{(d-k+i)} + \int_{x_n}^{x_{n+1}} \cdots \int_{x_n}^{s_2} f(s_1)ds_1 \cdots ds_k \quad (k = 1, \cdots, d) \tag{2.1}$$

where $\int ds_i$ is the integral of $y^{(d-i)}$ for the x.

Use previously calculated differential coefficients to calculate a polynomial approximation of this $f(x)$.
If we let the $(q-1)$-th order approximation passing through the q points $(x_i, f(x_i))$ $(i = n, \cdots, n-q+1)$ be $P_{q-1}(x)$, then this approximation is defined as follows by using divided differences.
If we let:

$$\begin{aligned} f[x_n] &= f(x_n) \\ f[x_n, \cdots, x_{n-i}] &= \frac{f[x_n, \cdots, x_{n-i+1}] - f[x_{n-1}, \cdots, x_{n-i}]}{x_n - x_{n-i}} \end{aligned} \tag{2.2}$$

then $P_{q-1}(x)$ is given by:

$$P_{q-1}(x) = f[x_n] + \cdots + (x - x_n) \cdots (x - x_{n-i+1}) f[x_n, \cdots, x_{n-i}] \quad (i = 0, \cdots, q-1) \tag{2.3}$$

Also, the corrector is obtained by integrating the q-th order $P_q^*(x)$ passing through the point $(x_{n+1}, f(x_{n+1}))$ in a similar manner as in expression (2.1), where $P_q^*(x)$ is given by:

$$P_q^*(x) = P_{q-1}(x) + (x - x_n)\cdots(x - x_{n-q+1})f[x_n, \cdots, x_{n-q+1}] \tag{2.4}$$

We define the following symbols for calculating expression (2.1):

$$\begin{aligned}
\tau &= \frac{x - x_n}{h_{n+1}} && (2.5)\\
\xi_i(n) &= h_n + \cdots + h_{n-i+1} = x_n - x_{n-i} && (2.6)\\
\eta_i(n) &= \frac{h_{n+1}}{\xi_i(n)} && (2.7)\\
\beta_0(n) &= 1 && (2.8)\\
\beta_i(n) &= \frac{\xi_1(n+1)\cdots\xi_i(n+1)}{\xi_1(n)\cdots\xi_i(n)} && (2.9)\\
\varphi_0(n) &= f[x_n] && (2.10)\\
\varphi_i(n) &= \xi_1(n)\cdots\xi_i(n)f[x_n, \cdots, x_{n-i}] && (2.11)
\end{aligned}$$

Then, expression (2.3) can be written as follows:

$$P_{q-1}(x) = \sum_{i=0}^{q-1}\left\{\varphi_i(n)\sum_{j=1}^{i}A_{i,j}(n)\tau^j\right\}$$

where $A_{i,j}$ is obtained by using the following recursive relation.

$$\begin{cases}
A_{i,1}(n) = \eta_i(n) & (i = 1, \cdots, q)\\
A_{i+1,j+1}(n) = \eta_{i+1}(n)\sum_{l=j}^{i}A_{l,j}(n) & (j = 1, \cdots, q-2; i = j, \cdots, q-2)
\end{cases} \tag{2.12}$$

Therefore, if we let:

$$\gamma_{ki}(n) = \begin{cases}
\frac{1}{k!} & (i = 0;\ k = 1, \cdots, d)\\
\sum_{j=1}^{i}\frac{j!}{(j+k)!}A_{ij}(n) & (i = 1, \cdots, q-1;\ k = 1, \cdots, d)
\end{cases} \tag{2.13}$$

then, expression (2.1) can be expressed as follows:

$$p_{n+1}^{(d-k)} = \sum_{i=0}^{k-1}\frac{h_{n+1}^i}{i!}y_n^{(d-k+i)} + h_{n+1}^k\sum_{i=0}^{q-1}\gamma_{k,i}(n)\varphi_i(n) \tag{2.14}$$

Similarly, from expression (2.4), the corrector $y_{n+1}^{(d-k)}$ can be expressed as follows:

$$\begin{aligned}
y_{n+1}^{(d-k)} &= p_{n+1}^{(d-k)} + h_{n+1}^k\gamma_{k,q}(n)\frac{\varphi_q(n+1)}{\beta_q(n)} \quad (k = 2, \cdots, d) && (2.15)\\
y_{n+1}^{(d-1)} &= p_{n+1}^{d-1} + h_{n+1}\sum_{i=0}^{q}\gamma_i^*(n)\varphi_i(n+1) && (2.16)
\end{aligned}$$

where $\gamma_i^*(n)$ is obtained from the following expressions.

$$\begin{cases} \gamma_0^*(n) &= 1 \\ \gamma_i^*(n) &= \dfrac{\gamma_{1,i}(n)}{\beta_i(n)} - \dfrac{\gamma_{1,i-1}(n)}{\beta_{i-1}(n)} \end{cases} \tag{2.17}$$

When $q = 1$, the predictor and corrector are given by the following expressions:

$$\begin{aligned} p_{n+1}^{(d-k)} &= \sum_{i=0}^{k-1} \frac{h_{n+1}^i}{i!} y_n^{(d-k+i)} + \frac{h_{n+1}^k}{k!} f(x_n) \\ y_{n+1}^{(d-k)} &= p_{n+1}^{(d-k)} + \frac{h_{n+1}^k}{(k+1)!}(f(x_{n+1}) - f(x_n)) \end{aligned} \tag{2.18}$$

These values are used in the first two steps of the calculation.
Order control is performed as follows.
The local discretization error is estimated according to the following equation from expression (2.16) :

$$E = |h_{n+1} \left\{ \gamma_q^*(n)\varphi_q(n+1) + \gamma_{q+1}^*(n)\varphi_{q+1}(n+1) \right\}| \tag{2.19}$$

Since the required relative precision is converted to absolute precision and the more lenient requirement between that value and the required absolute precision is taken, the required precision is obtained by using the following expression:

$$\begin{aligned} \varepsilon &= \max\left(\zeta_a, \zeta_r |y_{n+1} + h_{n+1}\frac{p'_{n+1}}{2}|\right) \\ &\quad (\zeta_a : \text{required absolute precision}, \quad \zeta_r : \text{required relative precision}) \end{aligned}$$

The convergence rate P_k of the corrector shown in expression (2.16) is defined as follows.

$$P_k = \left| \frac{\gamma_{k+1}^*(n)\varphi_{k+1}(n+1)}{\gamma_{k-1}^*(n)\varphi_{k-1}(n+1)} \right|$$

If the order q is too large, then since the propagation error becomes larger or new error components are included, P_k increases. Also, we defined various variables as follows:

$$\begin{aligned} C_{min} &= \min(p_q, p_{q-1}) \\ C_{max} &= \max(p_q, p_{q-1}) \end{aligned}$$

$$R_c = \begin{cases} 10.0 \times C_{max} & (C_{max} \leq 0.09) \\ 0.9 & (C_{min} < 0.09 < C_{max}) \\ 10.0 \times C_{min} & (0.09 \leq C_{min} \leq 0.105) \\ 1.05 & (C_{min} > 0.105) \end{cases}$$

For each simultaneous component, the order q is increased or decreased by 1 as follows:

(a) If $\frac{E}{\varepsilon} > 0.01$ and $C_{max} < 0.025$ or if $d > 1$ and $C_{max} < 0.0625$ then:
Increase q by 1.
However, for the first 9 iterations, use the following condition:
If $\frac{E}{\varepsilon} > 0.01$ and $C_{max} < 0.09$.

(b) $q > 1$, and if $C_{min} > 0.5$ or $\frac{E}{\varepsilon} < 0.001 \times P_q$ then:
Decrease q by 1.
However, for the first 9 iterations, do not decrease q.

Step size control is performed as follows.
If the number of simultaneous equations is e, then:

$$R_M = \max_j (10.0 \times \frac{E}{\varepsilon}, R_c) \quad (j = 1, \cdots, e)$$

(multiply by 10.0 to consider an extra decimal digit.)

$$\begin{aligned} \gamma &= \begin{cases} 1 + q(j_{max}) & (R_M \geq 1) \\ \max(q(j)) + \max(d(j)) & (R_M < 1) \end{cases} \\ h_{n+2} &= h_{n+1}(R_M)^{-\frac{1}{\gamma}} \end{aligned} \tag{2.20}$$

Start with an arbitrary value h for the initial step size and adjust h according to expression (2.20) until $0.125 \leq R_M \leq 3$.
The entire calculation procedure is as follows.

(a) First, use expression (2.18).

(b) Calculate $\eta_i(n)$ according to expression (2.7).

(c) Calculate $\gamma_{k,i}(n)$ of expression (2.13) according to expressions (2.5) through (2.12).

(d) Calculate $p_{n+1}^{(d-k)}$ $(k = d, d-1, \cdots, 1)$ according to expression (2.14).

(e) Calculate $\beta_i(n)$ $(i = 1, 2, \cdots, q)$ according to the following expressions:

$$\begin{aligned} \xi_i(n+1) &= h_{n+1} + \xi_{i-1}(n) \\ \beta_i(n) &= \beta_{i-1}(n) \frac{\xi_i(n+1)}{\xi_i(n)} \end{aligned}$$

Also, calculate $\gamma_i^*(n)$ according to expression (2.17). Then, approximate $\gamma_{q+1}^*(n)$ according to the following expression:

$$\gamma_{q+1}^*(n) = \frac{\{\gamma_q^*(n)\}^2}{\gamma_{q-1}^*(n)}$$

(f) Let $\varphi_0(n+1)$ be $f[x_{n+1}]$ and calculate $\varphi_{i+1}(n+1)$ according to the following expression:

$$\varphi_{i+1}(n+1) = \varphi_i(n+1) - \beta_i(n)\varphi_i(n) \quad (i = 0, \cdots, q)$$

(g) Calculate the corrector $y_{n+1}^{(d-k)}$ according to expression (2.15) or (2.16).

(h) Estimate the local discretization error according to expression (2.19) and adjust the order q.

(i) Calculate $\varphi_0(n+1)$ and modify $\varphi_i(n+1)$ $(i = 1, \cdots, q)$ according to the following expression:

$$\varphi_i(n+1) = \varphi_i(n+1) + \left\{ f(x_{n+1}, y_{n+1}, \cdots, y_{n+1}^{(d-1)}) - f(x_{n+1}, p_{n+1}, \cdots, p_{n+1}^{(d-1)}) \right\}$$

(j) Adjust the step size according to expression (2.20) and return to step (b).

This method is most efficient when function evaluations are expensive and precision requirements are severe. (See Reference Bibliography (2) and (5).)

(3) **Taylor series method (including processing corresponding to implicit equations)**
Assume that the equation to be solved is expressed as follows:

$$f(x, y, y', \cdots, y^{(d)}) = 0 \tag{2.21}$$

The following Taylor expansions can be used to obtain approximations of $y, y', \cdots, y^{(d-1)}$ at the point $x+h$:

$$\begin{aligned} y(x+h) &= y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \cdots \\ y'(x+h) &= y'(x) + hy''(x) + \frac{h^2}{2!}y'''(x) + \cdots \\ &\vdots \end{aligned}$$

To raise the precision by increasing the number of terms in the Taylor series, we create an expression that includes the higher order derivative $y^{(d+1)}$ by differentiating expression (2.21), then we differentiate again to create an expression that includes $y^{(d+2)}$, and so on. We obtain $y^{(d+1)}, y^{(d+2)}, \cdots$ from these expressions by solving a system of nonlinear simultaneous equations, and then we obtain the approximations at the point $x + h$ by assigning these values in the Taylor expansions.
The above procedure is described in more detail below.
Assume that the group of equations that are entered are as follows:

$$\begin{aligned} f_1(x, y, \cdots, y^{(d)}) &= 0 \\ f_1'(x, y, \cdots, y^{(d)}) &= f_2(x, y, \cdots, y^{(d)}, y^{(d+1)}) = 0 \\ &\vdots \end{aligned} \tag{2.22}$$

If $f_1(x, \cdots)$ or $f_2(x, \cdots)\cdots$ cannot be easily differentiated or even if they can be easily differentiated, to reduce the effort required to differentiate the entered equations, the subroutine has a feature that automatically differentiates functions. For example, automatic difference of $f_1(x, y, \cdots, y^{(d+1)})$ is given as follows performing a central difference calculation:

$$\begin{aligned} f_2(x, y, \cdots, y^{(d)}, y^{(d+1)}) &= f_1'(x, y, \cdots, y^{(d+1)}) \\ &\simeq \frac{1}{2\delta}\{g_1(x+\delta) - g_1(x-\delta)\} \\ &\quad + \frac{y'}{2\delta}\{g_1(y+\delta) - g_1(y-\delta)\} \\ &\quad + \cdots \\ &\quad + \frac{y^{(d+2)}}{2\delta}\left\{g_1(y^{(d+1)}+\delta) - g_1(y^{(d+1)}-\delta)\right\} \\ &= 0 \end{aligned}$$

where the following notations are used.

$$\begin{aligned} g_1(x+\delta) &= f_1(x+\delta, y, \cdots, y^{(i)}, \cdots, y^{(d+1)}) \\ g_1(y^{(i)}+\delta) &= f_1(x, y, \cdots, y^{(i)}+\delta, \cdots, y^{(d+1)}) \quad (i = 0, \cdots, d+1) \end{aligned}$$

For δ, use a small value such as $\sqrt[3]{\text{Units for determining error.}}$. You can also consider higher differentials and determine $g_2(\cdots), g_3(\cdots), \cdots, g_i(\cdots)$ and create the following expression that includes $y^{(d+i)}$:

$$f_{i+1}(x, y, \cdots, y^{(d+i)}) = 0 \quad (i = 1, 2, \cdots)$$

Next, from the group of equations created in this manner, $x, y(x), y'(x), \cdots, y^{(d-1)}(x)$ are assumed to be initial values, and the following system of nonlinear simultaneous equations having $y^{(d)}(x), \cdots, y^{(d+i)}(x)$ and $y(x+h), y'(x+h), \cdots, y^{(d-1)}(x+h)$ as unknowns is created:

$$\begin{cases} y(x+h) - \left\{ y(x) + hy'(x) + \cdots + h^{d+i} \dfrac{y^{(d+i)}(x)}{(d+i)!} \right\} = 0 \\ y'(x+h) - \left\{ y'(x) + hy''(x) + \cdots + h^{d+i-1} \dfrac{y^{(d+i)}(x)}{(d+i-1)!} \right\} = 0 \\ \vdots \\ y^{(d-1)}(x+h) - \left\{ y^{(d-1)}(x) + hy^{(d)}(x) + \cdots + h^{i+1} \dfrac{y^{(d+i)}(x)}{(i+1)!} \right\} = 0 \\ f_1(x, y(x), \cdots, y^{(d)}(x)) = 0 \\ \vdots \\ f_{i+1}(x, y(x), \cdots, y^{(d+i)}(x)) = 0 \end{cases} \tag{2.23}$$

These nonlinear equations are solved and the values $y(x+h), \cdots, y^{(d-1)}(x+h)$ that were obtained are assumed to be the next initial values $y(x), \cdots, y^{(d-1)}(x)$ and $x+h$ is assumed to be the next x.
If the differential equations are simultaneous, then groups of equations are created corresponding to expression (2.23) for each equation, and the simultaneous nonlinear equations are solved by making all of these equations simultaneous. Therefore, as in the following, for example, for an implicit case that includes $y_i^{(d)}$ $(i = 1, \cdots, n)$ in multiple equation calculations:

$$\begin{aligned} &f_1^1(x, y_1, \cdots, y_1^{(d)}, \cdots, y_n, \cdots, y_n^{(d)}) = 0 \\ &\vdots \\ &f_1^n(x, y_1, \cdots, y_1^{(d)}, \cdots, y_n, \cdots, y_n^{(d)}) = 0 \end{aligned} \tag{2.24}$$

(n: Number of simultaneous equations) or even if algebraic equations $f_1^i(x, y_1, y_2, \cdots, y_n) = 0$ are taken as simultaneous equations, a solution is obtained because all of the equations that are taken to be simultaneous will be solved by using the nonlinear simultaneous equations that are processed. Also, as an extreme case, if the given differential equations include no differential terms and they are all nonlinear equations or algebraic equations, then it is assumed that an expression that includes $y(x+h)$ will not be expanded according to a Taylor expansion. The subroutine also has a feature that obtains a solution by creating nonlinear simultaneous equations consisting only of these equations.
If the equations are simultaneous equations that include ordinary differential equations, then integration will continue with a step size of h up to the final point x. Finally, to obtain the solution of $y_i^{(d)}(x+h)$, the value of $y_i^{(d-i)}(x+h)$ that was obtained is entered in the following nonlinear equations:

$$\begin{aligned} &f_1^1(x+h, y_1(x+h), \cdots, \underline{y_1^{(d)}(x+h)}, \cdots, \underline{y_n^{(d)}(x+h)}) = 0 \\ &\vdots \\ &f_1^n(x+h, y_1(x+h), \cdots, \underline{y_1^{(d)}(x+h)}, \cdots, \underline{y_n^{(d)}(x+h)}) = 0 \end{aligned} \tag{2.25}$$

where, terms underlined by ___ are unknowns. The nonlinear simultaneous equations are solved as follows. If given equation is only one equation, which is not simultaneous, then consider the following iteration method for obtaining the value of x where $f(x) = 0$.

$$x_{n+1} = x_n + 2^{\frac{p-3r-1}{3}} S \sinh^{-1}\{f(x_n)\} \quad (S = \pm 1) \tag{2.26}$$

S is assumed to be -1 or 1 as follows.
When $f(x_n) < 0$ (if $f(x)$ increases monotonously, then: $x > x_n$), then $S = -1$.
When $f(x_n) > 0$ (if $f(x)$ decreases monotonously, then: $x < x_n$), then $S = 1$.
The initial values of r and p are assumed to be zero. r performs deceleration control by increasing by 1 each time the sign of the calculated solution of $f(x_n)$ changes, and p performs acceleration control by increasing by 1 when the sign of the calculated solution does not change.
When the following simultaneous equations are solved,

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

then, first, x_1 is assumed to be the unknown in the f_1 expression, and expression correspond to expression (2.26) for f_1 is iterated once with x_2 through x_n taken as initial values. Next, x_2 is assumed to be the unknown in the f_2 expression, and expression correspond to expression (2.26) for f_2 is iterated once with x_1 taken to be the value of the previous calculation and x_3 through x_n taken to be initial values. When the calculations are performed in this way up to f_n, control returns again to f_1 and the procedure described above is repeated. In this way, the calculated values will converge to the correct solution. The initial value $S_i \quad (i = 1, \cdots, n)$ has been determined to be $S_i = -1$ if the slope of f_i relative to x_i is positive and $S_i = 1$ if the slope is negative. However, even if the initial value has been determined in this way, the slope may reverse direction locally and the direction of investigation may be incorrect. Therefore, if $|f_i| > 10,000$ after iterating nine or more times, the direction of investigation is considered to be incorrect, $S_i = -S_i$ is assumed, the initial values are reset, and the simultaneous equations are solved again. Also, if the direction of investigation is considered to be incorrect again even when the direction of investigation was modified during the previous calculations, then the calculation is considered to be impossible, and processing is aborted.
The subroutine of this library assumes that convergence has occurred when

$$x_{n+1} < |\varepsilon x_n| + \varepsilon \text{ and } |f_i(\cdots)| < |10.0 \times \varepsilon x_n| + \varepsilon$$

where the maximum iteration count of these nonlinear simultaneous equations is assumed to be 100 and the convergence decision value ε is given by:

$$\varepsilon = \left\{ \begin{array}{l} \text{Single precision:} 10^{-5} \\ \text{Double precision:} 10^{-12} \end{array} \right\}.$$

Since this method must solve many simultaneous nonlinear equations each time h advances, efficiency is poor. However, it is effective for ordinary differential equations that are simultaneous with algebraic or nonlinear equations or an implicit problem since they cannot be solved by other methods. Also, even when all the equations are nonlinear simultaneous equations, a solution is obtained rather efficiently by only entering the equations to be solved without having to create a subroutine for calculating partial derivatives as required when using Newton's method.
(See Reference Bibliography (3).)

(4) **Gear's method for stiff problems**
This subroutine uses Gear's 1st order through 5th order method. Selection of the order and step size are automatically controlled.

(a) Predictor and corrector calculations

We assume that the differential equation is given in the form shown below, where $\boldsymbol{y}$ is assumed to be an N-dimensional vector:

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$$

Now, we assume that calculated solutions $\boldsymbol{y}_i \quad (i = 0, \cdots, n-1)$ up to $x = x_{n-1}$ have been obtained and that we are to obtain the calculated solution at $x = x_n$. At this time, a q-th order interpolation polynomial vector $\boldsymbol{p}_{n-1}(x)$ that satisfies the following conditions can be created:

$$\begin{aligned} \boldsymbol{p}_{n-1}(x_{n-i}) &= \boldsymbol{y}_{n-i} \quad (i = 1, \cdots, q) \\ \boldsymbol{p}'_{n-1}(x_{n-1}) &= \boldsymbol{f}(x_{n-1}, \boldsymbol{y}_{n-1}) \end{aligned}$$

The basic concept of this method is to try to determine $\boldsymbol{y}_n$ so that when the q-th order interpolation polynomial vector $\boldsymbol{p}_n(x)$ is created by using the calculated solution $\boldsymbol{y}_n$ at $x = x_n$, it satisfies the following conditions:

$$\begin{aligned} \boldsymbol{p}_n(x_{n-i}) &= \boldsymbol{y}_{n-i} \quad (i = 0, \cdots, q) \\ \boldsymbol{p}'_n(x_n) &= \boldsymbol{f}(x_n, \boldsymbol{y}_n) \end{aligned}$$

Information related to $\boldsymbol{p}_{n-1}(x)$ is retained in the form of the following matrix, which was conceived by **Nordsieck**:

$$Z_{n-1} = \left[\boldsymbol{y}_{n-1}, h\boldsymbol{y}'_{n-1}, h^2 \frac{\boldsymbol{y}''_{n-1}}{2!}, \cdots, h^q \frac{\boldsymbol{y}^{(q)}_{n-1}}{q!} \right]$$

where:

$$\begin{aligned} \boldsymbol{y}^{(q)}_{n-1} &= \boldsymbol{p}^{(q)}_{n-1}(x_{n-1}) \\ h &= x_n - x_{n-1} \end{aligned}$$

The predictor $Z_{n(0)}$ of Z_n is defined as follows:

$$\begin{aligned} Z_{n(0)} &= \left[\boldsymbol{y}_{n(0)}, h\boldsymbol{y}'_{n(0)}, h^2 \frac{\boldsymbol{y}''_{n(0)}}{2!}, \cdots, h^q \frac{\boldsymbol{y}^{(q)}_{n(0)}}{q!} \right] \\ \boldsymbol{y}^{(q)}_{n(0)} &= \boldsymbol{p}^{(q)}_{n-1}(x_n) \end{aligned}$$

$Z_{n(0)}$ can be calculated by using the following expression:

$$Z_{n(0)} = Z_{n-1} A \tag{2.27}$$

where, A is a Pascal triangle matrix whose i, j-th component $a_{i,j}$ is defined as follows:

$$a_{i,j} = \begin{cases} 0 & (i < j) \\ \dfrac{i!}{j!(i-j)!} & (i \geq j) \end{cases} \qquad (i, j = 0, \cdots, q)$$

Now, the polynomial $L_n(s)$ for s is defined as follows:

$$\begin{aligned} L_n(s) &= \prod_{i=1}^{q} (1 + \frac{s}{d_i}) \\ d_i &= \frac{x_n - x_{n-i}}{h} \end{aligned}$$

and the coefficient vector $\boldsymbol{l}$ of $L_n(s)$ is defined as follows:

$$\begin{aligned} \boldsymbol{l} &= [l_0, l_1, \cdots, l_q] \\ L_n(s) &= \sum_{i=0}^{q} l_i s^i \end{aligned}$$

At this time, the following relationship can be shown to occur:

$$\begin{aligned} Z_n &= Z_{n(0)} + \boldsymbol{e}_n \boldsymbol{l} \\ \boldsymbol{e}_n &= \boldsymbol{y}_n - \boldsymbol{y}_{n(0)} \end{aligned} \tag{2.28}$$

Since the following relationship is obtained by writing the first column of this:

$$h\boldsymbol{y}'_n = h\boldsymbol{y}'_{n(0)} + (\boldsymbol{y}_n - \boldsymbol{y}_{n(0)})l_1$$

$\boldsymbol{y}_n$ is calculated as the root of $\boldsymbol{g}(\boldsymbol{y}) = 0$ if $\boldsymbol{g}(\boldsymbol{y})$ is defined as follows:

$$\boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{y} - \boldsymbol{y}_{n(0)} - (\frac{h}{l_1})(\boldsymbol{f}(x_n, \boldsymbol{y}) - \boldsymbol{y}_{n(0)'})$$

To solve the equation $\boldsymbol{g}(\boldsymbol{y}) = 0$, Newton's method is used with $\boldsymbol{y} = \boldsymbol{y}_{n(0)}$ as the starting value. That is, the calculation is performed using the following recursive relation:

$$\begin{aligned} \boldsymbol{y}_{n(m+1)} &= \boldsymbol{y}_{n(m)} - P_m^{-1} \boldsymbol{g}(\boldsymbol{y}_{n(m)}) \\ P_m &= 1 - \frac{h}{l_1} J(x_n, \boldsymbol{y}_{n(m)}) \end{aligned}$$

where, $J(x, \boldsymbol{y})$, which is the Jacobian matrix of $\boldsymbol{f}(x, \boldsymbol{y})$, is defined as follows:

$$J(x, \boldsymbol{y}) = \begin{bmatrix} \dfrac{\partial f_1(x, \boldsymbol{y})}{\partial y_1} & \cdots & \dfrac{\partial f_1(x, \boldsymbol{y})}{\partial y_N} \\ \vdots & & \\ \dfrac{\partial f_N(x, \boldsymbol{y})}{\partial y_1} & \cdots & \dfrac{\partial f_N(x, \boldsymbol{y})}{\partial y_N} \end{bmatrix}$$

To save calculation time, the program directly uses the Jacobian matrix calculated for the previous iteration as much as possible. Also, corrector iterations are performed at most three times.

(b) Order and step size determination

This section describes how error is evaluated and how step size and order are controlled.

If we let the local discretization absolute error be $E_n(q)$ and the local discretization relative error be $R_n(q)$, then the subroutine decides that the calculated solution is to be accepted as follows by using the two user-assigned parameters, "Required local relative precision" E_a and "Required local absolute precision" E_r :

$$\|E_n(q)\| \leq E_a \quad or \quad \|R_n(q)\| \leq E_r \tag{2.29}$$

where, $\|\ \|$ indicates the Max norm. Also, the relative error is the ratio of the absolute error to the maximum value of the calculated solution up to the current time.

If the calculated solution satisfied the condition shown in (2.29), then the step size and order to be used in the next step are selected as follows. In addition to the error values described above, the errors $E_n(q-1)$ and $R_n(q-1)$ at order $q-1$ and the errors $E_n(q+1)$ and $R_n(q+1)$ at order $q+1$ are calculated. These are used to calculate the maximum step size that is permitted. That is, the maximum value among the $\eta_i \quad (i = 1, \cdots, 6)$ shown below is assumed to be the rate of increase of h,

and the order used for the error calculation at that time is assumed to be the order for the next step.

$$\eta_1 = \frac{\sqrt[q]{\frac{E_a}{\|E_n(q-1)\|}}}{1.3}$$

$$\eta_2 = \frac{\sqrt[q]{\frac{E_r}{\|R_n(q-1)\|}}}{1.3}$$

$$\eta_3 = \frac{\sqrt[(q+1)]{\frac{E_a}{\|E_n(q)\|}}}{1.2}$$

$$\eta_4 = \frac{\sqrt[(q+1)]{\frac{E_r}{\|R_n(q)\|}}}{1.2}$$

$$\eta_5 = \frac{\sqrt[(q+2)]{\frac{E_a}{\|E_n(q+1)\|}}}{1.4}$$

$$\eta_6 = \frac{\sqrt[(q+2)]{\frac{E_r}{\|R_n(q+1)\|}}}{1.4}$$

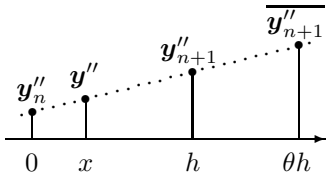
However, if the calculated solution did not satisfy the condition shown in (2.29), then the step size is decreased and the current step is performed again.

The integration starting process, which is self starting, is calculated according to a linear formula.

(5) **Wilson's θ method**

We assume that $\boldsymbol{y}''$ varies rectilinearly at $x = 0$ and $x = h$ as follows:

$$\boldsymbol{y}'' = \boldsymbol{y}''_n \frac{h-x}{h} + \boldsymbol{y}''_{n+1}$$



We integrate this expression with respect to x to obtain the following:

$$\boldsymbol{y}' = \boldsymbol{y}'_n + \frac{\boldsymbol{y}'_n}{h}\left(hx - \frac{x^2}{2}\right) + \frac{\boldsymbol{y}''_{n+1}}{h}\frac{x^2}{2}$$

$$\boldsymbol{y} = \boldsymbol{y}_n + \boldsymbol{y}'_n x + \frac{\boldsymbol{y}''_n}{h}\left(\frac{hx^2}{2} - \frac{x^3}{6}\right) + \frac{\boldsymbol{y}''_{n+1}}{h}\frac{x^3}{6}$$

If we let $x = h$ in the above expressions, then $\boldsymbol{y}'$ and $\boldsymbol{y}$ become $\boldsymbol{y}'_{n+1}$ and $\boldsymbol{y}_{n+1}$. That is, we obtain the following expressions:

$$\boldsymbol{y}'_{n+1} = \boldsymbol{y}'_n + \boldsymbol{y}''_n \frac{h}{2} + \boldsymbol{y}''_{n+1}\frac{h}{2} \tag{2.30}$$

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \boldsymbol{y}'_n h + \boldsymbol{y}''_n \frac{h^2}{3} + \boldsymbol{y}''_{n+1}\frac{h^2}{6} \tag{2.31}$$

If we substitute (2.30) and (2.31) into the original equation, then we obtain:

$$M\boldsymbol{y}''_{n+1} + C\left\{\boldsymbol{y}'_n + (\boldsymbol{y}''_n + \boldsymbol{y}''_{n+1})\frac{h}{2}\right\} + K\left\{\boldsymbol{y}_n + \boldsymbol{y}'h + (2\boldsymbol{y}''_n + \boldsymbol{y}''_{n+1})\frac{h^2}{6}\right\} = \boldsymbol{p}(h)$$

If we solve this for $\boldsymbol{y}''_{n+1}$, we obtain the following simultaneous linear equations:

$$\begin{aligned}&\left\{M+\frac{h}{2}C+\frac{h^2}{6}K\right\}\boldsymbol{y}''_{n+1}\\&=\left[\boldsymbol{p}(h)-C\left\{\boldsymbol{y}'_n+\boldsymbol{y}''_n\frac{h}{2}\right\}-K\left\{\boldsymbol{y}_n+\boldsymbol{y}'_n h+\boldsymbol{y}''_n\frac{h^2}{3}\right\}\right]\end{aligned}\tag{2.32}$$

However, since the solution often is unstable if we solve this directly, we solve for $\overline{\boldsymbol{y}''_{n+1}}$ at the point θh, which is obtained by multiplying the step size h by θ, and then obtain $\boldsymbol{y}''_{n+1}$ according to the following expression:

$$\boldsymbol{y}''_{n+1}=\boldsymbol{y}''_n+\frac{\overline{\boldsymbol{y}''_{n+1}}-\boldsymbol{y}''_n}{\theta}\tag{2.33}$$

It is known that θ should be at least 1.37. However, if this value is too large, the truncation error will increase and precision will worsen. This increase in error appears rather dramatically, for example, even for $\theta=2$. Wilson recommended that 1.4 be used as a practical value for θ.
The discussion above is summarized in the following procedure.

(a) Obtain the first initial value $\boldsymbol{y}''_n$ by solving the following simultaneous linear equations:

$$M\boldsymbol{y}''_n=\{\boldsymbol{p}(x)-C\boldsymbol{y}'_n-K\boldsymbol{y}\}\tag{2.34}$$

However, if the simultaneous linear equations cannot be solved because a zero is included in the diagonal components of the matrix M, then let $\boldsymbol{y}''_n$ be zero, divide the step size h by 8, use Wilson's θ method shown in steps (b) through (d) to obtain $\boldsymbol{y}''_{n+1}$, $\boldsymbol{y}'_{n+1}$ and $\boldsymbol{y}_{n+1}$ at the point that is offset ahead by h, and jump to step (d).

(b) Obtain $\overline{\boldsymbol{y}''_{n+1}}$ at the point that is offset ahead by θh by solving the following simultaneous linear equations:

$$\begin{aligned}\left\{M+\frac{\theta h}{2}C+\frac{(\theta h)^2}{6}K\right\}\overline{\boldsymbol{y}''_{n+1}}\quad&=\boldsymbol{p}(x)+(\boldsymbol{p}(x+h)-\boldsymbol{p}(x))\theta-C\left\{\boldsymbol{y}'_n+\boldsymbol{y}''_n\frac{\theta h}{2}\right\}\\&\quad-K\left\{\boldsymbol{y}_n+\boldsymbol{y}'_n\theta h+\boldsymbol{y}''_n\frac{(\theta h)^2}{3}\right\}\end{aligned}\tag{2.35}$$

Now, $\left\{M+\frac{\theta h}{2}C+\frac{(\theta h)^2}{6}K\right\}$ is fixed as long as the step size h does not change. Therefore, first, an LU decomposition is performed once and then, if forward and back substitution are performed while recreating the right-hand side term for each x, $\overline{\boldsymbol{y}''_{n+1}}$ is obtained at each x.
Since the initial value of $\boldsymbol{y}''_n$ is considered to be inappropriately set if the difference between $\overline{\boldsymbol{y}''_{n+1}}$ and $\boldsymbol{y}''_n$ is considerably large such as on the order of 1/(Unit for determining error) even at a single component, then let $\boldsymbol{y}''_n$ be zero, divide the step size h by 8, use Wilson's θ method shown in steps (b) through (d) to obtain $\overline{\boldsymbol{y}''_{n+1}}$, $\boldsymbol{y}'_n$ and $\boldsymbol{y}_n$ at the point that is offset ahead by h, and jump to step (d).

(c) Obtain $\boldsymbol{y}''_{n+1}$ according to expression (2.33).

(d) Obtain $\boldsymbol{y}_{n+1}$ according to expression (2.31).

(e) Obtain $\boldsymbol{y}'_{n+1}$ according to expression (2.30).

(f) Let $\boldsymbol{y}''_n=\boldsymbol{y}''_{n+1}$, $\boldsymbol{y}'_n=\boldsymbol{y}'_{n+1}$ and $\boldsymbol{y}_n=\boldsymbol{y}_{n+1}$ and execute the procedure again starting from (b) for the next step.

This method can be used, for example, for equations of motion with M corresponding to the mass matrix, C to the damping matrix, K to the stiffness matrix, $\boldsymbol{p}(x)$ to the external force at time x, x to time, $\boldsymbol{y}''$ to acceleration, $\boldsymbol{y}'$ to velocity, and $\boldsymbol{y}$ to position. For an earthquake response analysis, $\boldsymbol{p}(x)$ can be considered to be $\boldsymbol{p}(x)=-M\boldsymbol{y}''_e$, where $\boldsymbol{y}''_e$ is the acceleration of the ground.

2.1.2.2 Ordinary Differential Equations (Boundary Value Problems)

(1) **Multipoint shooting method**

This method solves the boundary value problem by selecting two or more shooting points within the interval and searching for the initial value at the previous shooting point that makes the residual at that point zero.
Let the differential equations be expressed as:

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \cdots, y_n) \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \cdots, y_n) \end{aligned} \tag{2.36}$$

and the boundary conditions as:

$$\begin{aligned} g_1(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \\ g_2(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \\ &\vdots \\ g_n(y_1(a), y_2(a), \cdots, y_n(a), y_1(b), y_2(b), \cdots, y_n(b)) &= 0 \end{aligned} \tag{2.37}$$

(Where, $x = a$:left-hand side boundary; $x = b$:right-hand side boundary)

Let the number of shooting points be n_x, and let the vector that approximates the strict solutions $y_j(x_i)$ $(j = 1, 2, \cdots, n)$ at shooting point x_i $(i = 1, 2, \cdots, n_x - 1)$ be expressed as:

$$\boldsymbol{u}_i = (u_{(i)1}, u_{(i)2}, \cdots, u_{(i)n})$$

Obtain the following equation from (2.36) by letting $\hat{y}_j(x_i) = u_{(i)j}$.

$$\hat{y}_j' = f_j(x, \hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n) \tag{2.38}$$

From (2.38), let the vector obtained by integrating up to x_{i+1} with the vector $\boldsymbol{u}_i$ as the initial value be $\hat{y}_j(x_{i+1})$. Use the Runge-Kutta-Verner method for the initial value problem calculation at this time. (See Section 2.1.2)
The residuals are expressed as follows:

$$\begin{aligned} r_{(i)j} &= \hat{y}_j(x_{i+1}) - u_{(i+1)j} \\ r_{(n_x)j} &= g_j(u_{(1)1}, u_{(1)2}, \cdots, u_{(1)n}, u_{(n_x)1}, u_{(n_x)2}, \cdots, u_{(n_x)n}) \end{aligned} \tag{2.39}$$

Finding the vector $\boldsymbol{u}_i$ that sets the residuals of (2.39) to zero is basic here.
Use Newton's method to solve these $n \times n_x$ nonlinear simultaneous equations.
Let $\Delta\boldsymbol{u}_i$ be the modified vector of vector $\boldsymbol{u}_i$. This modified vector $\Delta\boldsymbol{u}_i$ is obtained by solving the first order simultaneous equations:

$$\begin{bmatrix} A_1 & -I & 0 & & 0 & 0 \\ 0 & A_2 & -I & & 0 & 0 \\ 0 & 0 & A_3 & \cdots & 0 & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & & A_{n_x-1} & -I \\ G_1 & 0 & 0 & & 0 & G_{n_x} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{u}_1 \\ \Delta\boldsymbol{u}_2 \\ \Delta\boldsymbol{u}_3 \\ \vdots \\ \Delta\boldsymbol{u}_{n_x-1} \\ \Delta\boldsymbol{u}_{n_x} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{r}_1 \\ -\boldsymbol{r}_2 \\ -\boldsymbol{r}_3 \\ \vdots \\ -\boldsymbol{r}_{n_x-1} \\ -\boldsymbol{r}_{n_x} \end{bmatrix} \tag{2.40}$$

Where, elements of matrix A_i, G_1 and G_{n_x} is given as follows:

$$\begin{aligned}(A_i)_{jk} &= \frac{\partial \hat{y}_j(x_{i+1})}{\partial u_{(i)k}} \\ (G_1)_{jk} &= \frac{\partial g_j}{\partial u_{(1)k}} \\ (G_{n_x})_{jk} &= \frac{\partial g_j}{\partial u_{(n_x)k}}\end{aligned} \tag{2.41}$$

Since it is difficult to directly solve the first order simultaneous equations shown in (2.40), the calculation is performed as follows:

$A_{n_x} \leftarrow G_1$
$\Delta \boldsymbol{u}_{n_x} \leftarrow -\boldsymbol{r}_{n_x}$
for $i = 1, \cdots, n_x - 1$
  $A_{n_x} \leftarrow A_{n_x} A_i^{-1}$
  $\Delta \boldsymbol{u}_{n_x} \leftarrow \Delta \boldsymbol{u}_{n_x} + A_{n_x} \boldsymbol{r}_i$

$A_{n_x} \leftarrow A_{n_x} + G_{n_x}$
$\Delta \boldsymbol{u}_{n_x} \leftarrow A_{n_x}^{-1} \Delta \boldsymbol{u}_{n_x}$
for $i = n_x - 1, \cdots, 1$
  $\Delta \boldsymbol{u}_i \leftarrow A_i^{-1}(\Delta \boldsymbol{u}_{i+1} - \boldsymbol{r}_i)$

where, A_i^{-1} $(i = 1, 2, \cdots, n_x - 1)$ is obtained as follows, without directly calculating the inverse matrix. If equation (2.38) is differentiated with respect to $u_{(i)k}$, the following is obtained:

$$\frac{d}{dx}\left(\frac{\partial \hat{y}_j}{\partial u_{(i)k}}\right) = \sum_{p=1}^{n} \frac{\partial f_j}{\partial \hat{y}_p} \frac{\partial \hat{y}_p}{\partial u_{(i)k}} \tag{2.42}$$

In this equation, let:

$$(\hat{A}_i(x))_{jk} = \frac{\partial \hat{y}_j(x)}{\partial u_{(i)k}} \quad (\hat{A}_i(x_i) = I, A_i(x_{i+1}) = A_i)$$

to express equation (2.42) as follows:

$$\frac{d}{dx}\hat{A}_i = J\hat{A}_i \quad \left((J)_{jp} = \frac{\partial f_j(x, y_1, y_2, \cdots, y_n)}{\partial y_p}\right)$$

The following equation can be derived from this:

$$\frac{d}{dx}\hat{A}_i^{-1} = -J\hat{A}_i^{-1} \quad \left(\hat{A}_i^{-1}(x_i) = I\right) \tag{2.43}$$

Obtain $A_i^{-1} = \hat{A}_i^{-1}(x_{i+1})$ by integrating equation (2.43) up to x_{i+1}. That is:

$$\begin{aligned}A_i^{-1} &= \exp((x_i - x_{i+1})J) \\ &\simeq I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \frac{B^4}{4!} + \frac{B^5}{5!}\end{aligned}$$

(where, $B = (x_i - x_{i+1})J$)
From the above, the procedure for calculating vector $\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$ is as follows.

(a) As the initial value of $\boldsymbol{u}_i$, set all components to 0.1.

(b) Determine the shooting points.
Determine the smallest integer i for which $i \geq b-a$. The number of shooting points n_x is given by the following formula:

$$n_x = \min(2 \times i + 4, 50)$$

Assign the shooting points equally spaced within the interval according to this value of n_x.

(c) Calculate A_i^{-1} $(i = 1, 2, \cdots, n_x - 1)$.
For the condition number $\|A_i\|\|A_i^{-1}\|$ of matrix A_i, if the relationship:

$$\|A_i\|\|A_i^{-1}\| \geq 2.5$$

is satisfied, add the midpoint of x_i and x_{i+1} as a new shooting point to the previously used shooting points, and then calculate A_i^{-1} again.

(d) Calculate G_1 and G_{n_x}.

(e) Calculate residual $\boldsymbol{r}_i$ $(i = 1, 2, \cdots, n_x)$.

(f) Calculate modified vector $\Delta\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$.

(g) Update vector $\boldsymbol{u}_i$ $(i = 1, 2, \cdots, n_x)$ by using the following equation:

$$\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i + \Delta\boldsymbol{u}_i$$

(h) Determine convergence
Let ε_r be the required relative precision and ε_a be the required absolute precision. (For single precision, let ε_r and ε_a be 5 times the required local precisions used to solve the initial value problem, and for double precision, let them be 10 times the required local precisions.) Consider that the sequence has converged if:

$$\max(|r_{(i)j}|, |\Delta u_{(i)j}|) < \max(\varepsilon_a, \varepsilon_r |u_{(i)j}|)$$

are satisfied for all i $(i = 1, 2, \cdots, n_x)$ and j $(j = 1, 2, \cdots, n)$. If these conditions are not satisfied, return to step (e) and perform the calculations again.
For a nonlinear problem, perform the following calculations for convergence stability.
Multiply the nonlinear calculation part of the given problem by α. At first, linearize the problem by setting the parameter α to zero. When the number of iterations is 10 or the solution has converged, return the parameter α to 1, and solve the original nonlinear problem using the solution at that time as the initial value. When solving the nonlinear problem, recalculate A_i^{-1} for each iteration.

(2) **Collocation method**

This method selects a point x within an interval and makes the residual at that point be zero. At this time, it uses a spline function based on a B-spline to represent the approximate solution of the ordinary differential equations.
Let the differential equation to be solved be expressed as:

$$a_1(x)y^{(m)} + a_2(x)y^{(m-1)} + \cdots + a_{m+1}(x)y + a_{m+2}(x) = 0 \tag{2.44}$$

the boundary conditions at the left-hand side boundary $x = a$ be expressed as:

$$y^{(d_i)}(a) = c_1, y^{(d_2)}(a) = c_2, \cdots, y^{(d_h)} = c_h \tag{2.45}$$

and the boundary conditions at the right-hand side boundary $x = b$ be expressed as:

$$y^{(d_{h+1})}(b) = c_{h+1}, y^{(d_{h+2})}(b) = c_{h+2}, \cdots, y^{(d_m)} = c_m \tag{2.46}$$

where, d_1 through d_m are related as follows:

$$0 \leq d_1 < d_2 < \cdots < d_h < m$$
$$0 \leq d_{h+1} < d_{h+2} < \cdots < d_m < m$$

Now, take n_x selected points x_i $(i = 1, 2, \cdots, n_x)$ within the interval for which the solution is to be obtained, obtain the B-spline function $u(x)$ that satisfies $y(x_i) = u(x_i)$, and consider this to be the approximate solution. If the $(k-1)$st order B-spline basis is represented by $B_{i,k}(x)$, then $u(x)$ can be represented as follows:

$$u(x) = \sum_{i=1}^{n_x} e_i B_{i,k}(x) \tag{2.47}$$

These e_i are unknown coefficients that must be obtained. Use the following equations to obtain them. From the boundary conditions at the left-hand side boundary $x = a$ shown in (2.45),

$$\sum_{i=1}^{n_x} e_i B_{i,k}^{(d_p)}(a) = c_p \quad (p = 1, 2, \cdots, h) \tag{2.48}$$

Next, substitute equation (2.47) into equation (2.44) to obtain:

$$\sum_{l=1}^{m+1} \sum_{i=1}^{n_x} e_i (a_l(x) B_{i,k}^{(m+1-l)}(x)) + a_{m+2}(x) = 0 \tag{2.49}$$

From the boundary conditions at the right-hand side boundary $x = b$ shown in (2.46),

$$\sum_{i=1}^{n_x} e_i B_{i,k}^{(d_p)}(b) = c_q \quad (q = h+1, \cdots, m) \tag{2.50}$$

The above equations can be expressed as simultaneous linear equations:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.51}$$

where, the matrix A, the right-hand side vector $\boldsymbol{b}$ and the vector of unknowns $\boldsymbol{x}$ are given as follows:

$$A = \begin{bmatrix} B_{1,k}^{(d_1)}(a) & \cdots & B_{n_x,k}^{(d_1)}(a) \\ \vdots & & \vdots \\ B_{1,k}^{(d_h)}(a) & \cdots & B_{n_x,k}^{(d_h)}(a) \\ \sum_{l=1}^{m+1} a_l(x_{h+1}) B_{1,k}^{(m+1-l)}(x_{h+1}) & \cdots & \sum_{l=1}^{m+1} a_l(x_{h+1}) B_{n_x,k}^{(m+1-l)}(x_{h+1}) \\ \vdots & & \vdots \\ \sum_{l=1}^{m+1} a_l(x_{n_x-m+h}) B_{1,k}^{(m+1-l)}(x_{n_x-m+h}) & \cdots & \sum_{l=1}^{m+1} a_l(x_{n_x-m+h}) B_{n_x,k}^{(m+1-l)}(x_{n_x-m+h}) \\ B_{1,k}^{(d_{h+1})}(b) & \cdots & B_{n_x,k}^{(d_{h+1})}(b) \\ \vdots & & \vdots \\ B_{1,k}^{(d_m)}(b) & \cdots & B_{n_x,k}^{(d_m)}(b) \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} e_1 \\ \vdots \\ e_h \\ e_{h+1} \\ \vdots \\ e_{n_x-m+h} \\ e_{n_x-m+h+1} \\ \vdots \\ e_{n_x} \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} c_1 \\ \vdots \\ c_h \\ -a_{m+2}(x_{h+1}) \\ \vdots \\ -a_{m+2}(x_{n_x-m+h}) \\ c_{h+1} \\ \vdots \\ c_m \end{bmatrix}$$

From the above, the procedure for calculating the approximate solution $u(x)$ is as follows.

(a) Set the selected points
Determine the smallest integer i for which $i \geq b - a$. The number of selected points n_x is given by the following formula:

$$n_x = \max(\min(5 \times i + 1, 101), m + 2)$$

Assign the selected points equally spaced within the interval according to this value of n_x.

(b) Set the degree of the spline
Set the degree of the spline proportionate to the number of selected points. Initially, determine the smallest integer i for which $i \geq \dfrac{n_x}{10}$. At this time, let the order of the spline j_s be given by:

$$j_s = \max(i + 3, m)$$

(c) Set nodes
Let $k = j_s + 1$. Then the required number of nodes is $n_x + k$. Set these nodes as follows:

$$\begin{aligned} &q_1 = q_2 = \cdots = q_k = x_1 \\ &q_{i+k} = \frac{x_i + x_{i+k}}{2} \quad (i = 1, 2, \cdots, n_x - k) \\ &q_{n_x+1} = q_{n_x+2} = \cdots = q_{n_x+k} = x_{n_x} \end{aligned}$$

(d) Set the B-spline at the selected points
The $(k-1)$st order B-spline $B_{j,k}(x_i) \quad (j = 1, 2, \cdots, n_x)$ at selected points $x_i \quad (i = 1, 2, \cdots, n_x)$ is represented by the recursive relation:

$$B_{j,k}(x) = (\frac{x_i - q_i}{q_{j+k-1} - q_j})B_{j,k-1}(x_i) + (\frac{q_{j+k} - x_i}{q_{j+k} - q_{j+1}})B_{j+1,k-1}(x_i) \tag{2.52}$$

Set the following 0th order B-spline as the starting value:

$$B_{j,1}(x_i) = \begin{cases} 1 & (q_j \leq x_i < q_{j+1}) \\ 0 & (x_i < q_j, x_i \geq q_{j+1}) \end{cases}$$

(e) Differential the B-spline at the selected points
If (2.52) is differentiated in terms of selected point $x_i \quad (i = 1, 2, \cdots, n_x)$, the following recursive relation is obtained:

$$B'_{j,k}(x_i) = (k-1)(\frac{B_{j,k-1}(x_i)}{q_{j+k-1} - q_j} - \frac{B_{j+1,k-1}(x_i)}{q_{j+k} - q_{j+1}}) \tag{2.53}$$

Sequentially differentiate both sides in a similar manner and obtain the higher order differentials of the B-spline by sequentially eliminating the first order differentials according to (2.53).

(f) Obtain the coefficient matrix of the simultaneous equations shown in (2.51) and determine e_i
Substitute the values:

$$B_{j,k}(x_i), B'_{j,k}(x_i), \cdots, B^{(m)}_{j,k}(x_i) \quad (i = 1, 2, \cdots, n_x; j = 1, 2, \cdots, n_x)$$

that were calculated in steps (d) and (e) into the coefficient matrix on the left side of the simultaneous equations shown in (2.51).
The linear combination coefficients e_i $(i = 1, 2, \cdots, n_x)$ are obtained by creating the simultaneous equations shown in (2.51) according to the above and solving them.

(g) Calculate the approximate solution
From (2.47), obtain the approximate solution that passes through the strict solution at the n_x selected points.

(h) Determine convergence
Determine an index γ_j $(j = 1, 2, \cdots, n_x - 1)$ that can take the proportion that the approximate solution curve is separated from the strict solution curve

$$\gamma_j = \left| \frac{\Delta\delta_j}{\delta Y_j} \right|$$

where:
$|\delta Y_j|$ is Maximum absolute value among the terms of the ordinary differential equation calculated at the midpoint of selected points x_j and x_{j+1}

$$|\delta Y_j| = \max_{l=1,2,\cdots,m+1} \left(|\sum_{i=1}^{n_x} e_i a_l(\frac{x_j + x_{j+1}}{2}) B^{(m+1-l)}_{i,k}(\frac{x_j + x_{j+1}}{2})|, |a_{m+2}(\frac{x_j + x_{j+1}}{2})| \right),$$

$|\Delta\delta_j|$ is Absolute value of the residual of the ordinary differential equation at the midpoint

$$|\Delta\delta_j| = |\sum_{l=1}^{m+1} \sum_{i=1}^{n_x} e_i \left(a_l(\frac{x_j + x_{j+1}}{2}) B^{(m+1-l)}_{i,k}(\frac{x_j + x_{j+1}}{2}) \right) + a_{m+2}(\frac{x_j + x_{j+1}}{2})|.$$

Let ε_r be the required relative precision and ε_a be the required absolute precision. Consider that the sequence has converged the following condition (2.54) was satisfied.

$$\gamma_j \leq \varepsilon_r \text{ or } |\Delta\delta_j| \leq \varepsilon_a \quad (j = 1, 2, \cdots, n_x - 1) \tag{2.54}$$

If this condition is not satisfied, add the midpoint for which condition (2.54) was not satisfied as a new selected point to the previously used selected points, return to step (b), and perform the calculations again.

(3) **Coefficient determination method**
Let the differential equation to be solved be expressed as:

$$y'' + a_1(x)y' + a_2(x)y + a_3(x) = 0 \tag{2.55}$$

Use an unknown coefficient c to let the solution of (2.55) be expressed as:

$$y(x) = y_1(x) + cy_2(x) \tag{2.56}$$

Differentiate both sides to obtain:

$$y' = y'_1(x) + cy'_2(x) \tag{2.57}$$

$$y'' = y''_1(x) + cy''_2(x) \tag{2.58}$$

By substituting these in (2.55) and rearranging terms, we see that y_1 and y_2 that satisfy the following equations (2.59) and (2.60) will satisfy equations (2.55) and (2.56).

$$y_1'' + a_1(x)y_1' + a_2(x)y_1 + a_3(x) = 0 \tag{2.59}$$
$$y_2'' + a_1(x)y_2' + a_2(x)y_2 = 0 \tag{2.60}$$

Therefore, we will consider solving equations (2.59) and (2.60) here instead of equation (2.55). Boundary conditions can be divided into the following four cases for the left-hand side boundary $x = a$ and the right-hand side boundary $x = b$:

$$y(a) = ya_0 \quad , \quad y(b) = yb_0 \tag{2.61}$$
$$y(a) = ya_0 \quad , \quad y'(b) = yb_1 \tag{2.62}$$
$$y'(a) = ya_1 \quad , \quad y(b) = yb_0 \tag{2.63}$$
$$y'(a) = ya_1 \quad , \quad y'(b) = yb_1 \tag{2.64}$$

where, ya_0, ya_1, yb_0, yb_1 are constants. The calculation procedure for obtaining the unknown coefficient c is shown below.

(a) Initial value problem calculation of (2.59)
For boundary conditions (2.61) and (2.62), if we solve equation (2.59) with initial values:

$$y_2(a) = 0 \quad , \quad y_2'(a) = 1$$

$y_2(b)$ and $y_2'(b)$ are obtained.
For boundary conditions (2.63) and (2.64), if we solve equation (2.59) with initial values:

$$y_2(a) = 1 \quad , \quad y_2'(a) = 0$$

$y_2(b)$ and $y_2'(b)$ are obtained.

(b) Initial value problem calculation of (2.60)
For boundary conditions (2.61) and (2.62), from equation (2.56):

$$\begin{aligned} y_1(a) &= y(a) - cy_2(a) \\ &= ya_0 \end{aligned}$$

Therefore, if we solve equation (2.60) with initial values:

$$y_1(a) = ya_0 \quad , \quad y_1'(a) = 0$$

$y_1(b)$ and $y_1'(b)$ are obtained.
For boundary conditions (2.63) and (2.64), from equation (2.57):

$$\begin{aligned} y_1'(a) &= y'(a) - cy_2'(a) \\ &= ya_1 \end{aligned}$$

Therefore, if we solve equation (2.60) with initial values:

$$y_1(a) = 0 \quad , \quad y_1'(a) = ya_1$$

$y_1(b)$ and $y_1'(b)$ are obtained.

(c) Unknown coefficient c calculation

For boundary conditions (2.61) and (2.63), from equation (2.56):

$$c = \frac{yb_0 - y_1(b)}{y_2(b)}$$

For boundary conditions (2.62) and (2.64), from equation (2.57):

$$c = \frac{yb_1 - y_1'(b)}{y_2'(b)}$$

Using the value of c obtained from the above, obtain $y(x), y'(x)$ and $y''(x)$ from equations (2.56), (2.57) and (2.58).

The Runge-Kutta-Verner method is used for the initial value problem calculation. (See Section 2.1.2) If $y_2(b)$ or $y_2'(b)$ is zero, create an initial value problem facing towards the left-hand side boundary from the the right-hand side boundary and obtain the unknown coefficient c from it. If the denominator is still zero, an error occurs.

2.1.2.3 Integral Equations

(1) **Fredholm's integral equation of the second kind**

Fredholm's integral equation of the second kind is represented by the following expression.

$$y(t) - \int_a^b K(t,x)y(x)dx = f(t)$$

where,

$f(t)$	:	A continuous known function on domain$[a,b]$ of t
$K(t,x)$	:	Kernel (a continuous known function on domain $[a,b]$ of t and x)
$y(t)$	:	Unknown function to be obtained

The solution method when the derivatives of kernel K relative to t and x are continuous (kernel K is a regular kernel) is explained below. Using numerical integration to approximate the second term on the left side of the equation results in

$$\int_a^b K(t,x)y(x)dx \simeq \sum_{i=1}^{n} W_i K(t,x_i)y(x_i)$$

where, W_i, x_i and n are constants determined by the numerical integration formula. Since Gauss' integral formula (30-point formula) is used here, we have

i		W_i	X_i
1	30	0.00796819249616661	0.996893484074650
2	29	0.0184664683110910	0.983668123279747
3	28	0.0287847078833234	0.960021864968308
4	27	0.0387991925696270	0.926200047429274
5	26	0.0484026728305941	0.882560535792053
6	25	0.0574931562176191	0.829565762382768
7	24	0.0659742298821805	0.767777432104826
8	23	0.0737559747377052	0.697850494793316
9	22	0.0807558952294202	0.620526182989243
10	21	0.0868997872010830	0.536624148142020
11	20	0.0921225222377861	0.447033769538089
12	19	0.0963687371746443	0.352704725530878
13	18	0.0995934205867953	0.254636926167890
14	17	0.101762389748406	0.153869913608584
15	16	0.102852652893559	0.051471842555318

(where, a minus sign is attached to X_i for $i = 1, \cdots, 15$)

$$\begin{aligned} x_i &= \frac{|\, a-b \,|}{2} X_i + \frac{a+b}{2} \\ n &= 30 \end{aligned}$$

Using this approximation, Fredholm's integral equation of the second kind is represented as

$$\begin{bmatrix} 1 - W_1K(x_1, x_1) & -W_2K(x_1, x_2) & \cdots & -W_{30}K(x_1, x_{30}) \\ W_1K(x_2, x_1) & 1 - W_2K(x_2, x_2) & \cdots & -W_{30}K(x_2, x_{30}) \\ \vdots & \vdots & \ddots & \vdots \\ W_1K(x_{30}, x_1) & 1 - W_2K(x_{30}, x_2) & \cdots & 1 - W_{30}K(x_{30}, x_{30}) \end{bmatrix} \begin{bmatrix} y(x_1) \\ y(x_2) \\ \vdots \\ y(x_{30}) \end{bmatrix}$$
$$= \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{30}) \end{bmatrix}$$

These simultaneous linear equations of degree 30 are solved using Gauss' elimination method. By using the values $y(x_1), \cdots, y(x_{30})$ that are obtained, $y(t)$ can be obtained at an arbitrary value of t by interpolation using a cubic spline function. (See Section 6.1.2)

(2) **Volterra's integral equation of the first kind**

Volterra's integral equation of the first kind is represented by the following expression.

$$f(t) = \int_a^t K(t, x) y(x) dx$$

where,

$f(t)$: A finite continuous known function on domain $[a, \infty)$ of t

$K(t, x)$: Kernel (a finite continuous known function or a known function that becomes infinitely large at $x = \alpha$ on domain $[a, \infty]$ of t and x)

$$\int_a^x K(x,y)dy < \infty$$

$y(t)$: Unknown function to be obtained

The solution method when the derivatives of kernel K relative to t and x are continuous (kernel K is a regular kernel) is explained below. Applying Maclaurin's formula to the integration interval $[a, a+h]$, we get

$$f(a+h) = \int_a^{a+h} K(a+h,x)y(x)dx = K(a+h, a+\frac{h}{2})y(a+\frac{h}{2})h$$

and $y(a+\frac{h}{2})$ is given by

$$y(a+\frac{h}{2}) \simeq \frac{f(a+h)}{K(a+h, a+\frac{h}{2})h}$$

Applying Maclaurin's formula to the integration intervals $[a, a+2h], [a, a+3h], \cdots, [a, a+6h]$, we obtain the following simultaneous linear equations.

$$\begin{aligned}
f(a+2h) &\simeq 2h\left\{K(a+2h, a+\frac{h}{2})y(a+\frac{h}{2})\right. \\
&\quad \left.+K(a+2h, a+\frac{3}{2}h)y(a+\frac{3}{2}h)\right\} \\
f(a+3h) &\simeq 3h\left\{\frac{3}{8}K(a+3h, a+\frac{h}{2})y(a+\frac{h}{2})\right. \\
&\quad +\frac{2}{8}K(a+3h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\
&\quad \left.+\frac{3}{8}K(a+3h, a+\frac{5}{2}h)y(a+\frac{5}{2}h)\right\} \\
f(a+4h) &\simeq 4h\left\{\frac{13}{48}K(a+4h, a+\frac{h}{2})y(a+\frac{h}{2})\right. \\
&\quad +\frac{11}{48}K(a+4h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\
&\quad +\frac{11}{48}K(a+4h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) \\
&\quad \left.+\frac{13}{48}K(a+5h, a+\frac{7}{2}h)y(a+\frac{7}{2}h)\right\} \\
f(a+5h) &\simeq 5h\left\{\frac{275}{1152}K(a+5h, a+\frac{h}{2})y(a+\frac{h}{2})\right. \\
&\quad +\frac{100}{1152}K(a+5h, a+\frac{3}{2}h)y(a+\frac{3}{2}h) \\
&\quad +\frac{402}{1152}K(a+5h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) \\
&\quad +\frac{100}{1152}K(a+5h, a+\frac{7}{2}h)y(a+\frac{7}{2}h) \\
&\quad \left.+\frac{275}{1152}K(a+5h, a+\frac{9}{2}h)y(a+\frac{9}{2}h)\right\} \\
f(a+6h) &\simeq 6h\left\{\frac{247}{1280}K(a+6h, a+\frac{h}{2})y(a+\frac{h}{2})\right. \\
&\quad +\frac{139}{1280}K(a+6h, a+\frac{3}{2}h)y(a+\frac{3}{2}h)
\end{aligned}$$

$$
\begin{aligned}
&+\frac{254}{1280}K(a+6h, a+\frac{5}{2}h)y(a+\frac{5}{2}h) \\
&+\frac{254}{1280}K(a+6h, a+\frac{7}{2}h)y(a+\frac{7}{2}h) \\
&+\frac{139}{1280}K(a+6h, a+\frac{9}{2}h)y(a+\frac{9}{2}h) \\
&+\frac{247}{1280}K(a+6h, a+\frac{11}{2}h)y(a+\frac{11}{2}h)\Big\}
\end{aligned}
$$

Solving this, we obtain, $y(a+\frac{3}{2}h), y(a+\frac{5}{2}h), y(a+\frac{7}{2}h), y(a+\frac{9}{2}h)$ and $y(a+\frac{11}{2}h)$.
Next, by solving similar simultaneous linear equations for the integration intervals $[a+5h, a+7h]$, $[a+5h, a+8h], \cdots, [a+5h, a+11h]$, we can obtain $y(a+\frac{13}{2}h), y(a+\frac{15}{2}h), \cdots, y(a+\frac{21}{2}h)$.
In a similar manner, we can obtain the value at each point $y(a+\frac{2j-1}{2}h)$ for $(j=1, \cdots, n)$. By using the values of y that are obtained, $y(x)$ can be obtained at an arbitrary value of x by interpolation using a cubic spline function. (See Section 6.1.2)

2.1.2.4 Partial Differential Equations

(1) **Finite-Difference Approximation**
This library uses the finite difference to solve the following partial differential equations.

$$\nabla^2 + \lambda u = f \tag{2.65}$$

The partial derivatives of the function of two variables on a two-dimensional space $u(x, y)$ and the function of three variables on a three-dimensional space $u(x, y, z)$ are assumed to be sufficiently differentiable and continuous. Also, u_{xx} and u_{yy} are second order partial derivatives with respect to x and to y, respectively, of the function of two variables $u(x, y)$.

$$
\begin{aligned}
u(x+h, y) &= u(x, y) + u_x(x, y)h + u_{xx}(x, y)\frac{h^2}{2!} + \cdots \\
u(x-h, y) &= u(x, y) - u_x(x, y)h + u_{xx}(x, y)\frac{h^2}{2!} - \cdots
\end{aligned}
$$

Similarly, if the functions of two variables $u(x, y+k)$ and $u(x, y-k)$ are each expanded as Taylor series for y, the following equations are obtained.

$$
\begin{aligned}
u(x, y+k) &= u(x, y) + u_y(x, y)h + u_{yy}(x, y)\frac{k^2}{2!} + \cdots \\
u(x, y-k) &= u(x, y) - u_y(x, y)h + u_{yy}(x, y)\frac{k^2}{2!} - \cdots
\end{aligned}
$$

From the above equations, the central differences for $u_{xx}(x, y)$ and $u_{yy}(x, y)$ are obtained.

$$u_{xx}(x, y) \cong \frac{1}{h^2}[u(x+h, y) - 2u(x, y) + u(x-h, y)]$$

$$u_{yy}(x, y) \cong \frac{1}{k^2}[u(x, y+k) - 2u(x, y) + u(x, y+k)]$$

Using these central differences, equation (2.65) will be as follows.

$$\frac{1}{h^2}[\ u(x+h, y) - 2u(x, y) + u(x-h, y)\ ] +$$

$$\frac{1}{k^2}[\ u(x,y+k) - 2u(x,y) + u(x,y+k)\ ] + \lambda\ u(x,y) = \ f(x,y) \tag{2.66}$$

Similarly, for three dimensions, equation (2.67) is derived by expanding $u(x+h,y,z)$, $u(x-h,y,z)$, $u(x,y+k,z)$, $u(x,y-k,z)$, $u(x,y,z-l)$, and $u(x,y,z+l)$ as Taylor series for x, y, and z, respectively, obtaining the central differences, and substituting them in equation (2.65).

$$\frac{1}{h^2}[\ u(x+h,y,z) - 2u(x,y,z) + u(x-h,y,z)\ ]+$$

$$\frac{1}{k^2}[\ u(x,y+k,z) - 2u(x,y,z) + u(x,y+k,z)\ ]+$$

$$\frac{1}{l^2}[\ u(x,y,z+l) - 2u(x,y,z) + u(x,y,z+l)\ ] + \lambda\ u(x,y,z) = \ f(x,y,z) \tag{2.67}$$

Calculating equation (2.66) or (2.67) at each grid point within a discretized region produces simultaneous linear equations with the value of u at each grid point as the variable, and the value of u is obtained by solving those simultaneous linear equations.

(2) **Finite-Difference Approximation of Boundary Condition**

This section explains the handling of boundary conditions for Dirichlet problems and Neumann problems in this library. In both cases, a virtual grid is established outside of the given area, and the boundary conditions are applied on that virtual grid. The Dirichlet condition assigns the u value on the virtual grid, and the Neumann condition assigns the differential coefficient $\frac{\partial u}{\partial n}$ in the direction of the outer-facing normal of u. Within this library, at the first grid point $(i,j) = (1,1)$ of a discretized rectangular area, for example, the boundary conditions of the bottom edge of the area according to equation (2.66) are as follows.

$$\begin{cases} \text{Dirichlet condition} & u(i,0) = S \quad (S : \text{Value of boundary condition u}) \\ \text{Neumann condition} & u(i,1) = u(i,0) \end{cases}$$

2.1.3 Reference Bibliography

(1) Verner, J. H. , "Explicit Runge-Kutta methods with estimate of the local truncation error", SIAM J. Numer. Anal. Vol.15, No.4, pp.618-641, (1978).

(2) Krogh, F. T. , "A Variable Step Variable Order Multistep Method for the Numerical Solution of Ordinary Differential Equations", Information Processing 68, North-Holand Pub. Co. , pp.194-199, (1968).

(3) Kantaris, N. and Howden, P. F. , "The Universal Equation Solver", SIGMA PRESS, (1983).

(4) Gupta, G. K. , Sacks-Davis, R. and Tischer, P. E. , "A Review of Recent Development in Solving ODEs", ACM Comp. Surveys, Vol.17, No.1, pp.5-47, (1985).

(5) Shampine, L. F. and Gordon, M. K. , "Computer Solution of Ordinary Differential Equations", Freeman, (1975).

(6) Shampine, L. F. , Watts, H. A. and Davenport, S. M. , "Solving Nonstiff Ordinary Differential Equations-the State of the Art", SIAM Review, Vol.18, No.3, pp.376-411, (1976).

(7) Byrne, G. D. and Hindmarsh, A. C. , "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations", ACM Trans. Math. Softw. , Vol.1, pp.71-96, (1975).

(8) Hindmarsh, A. C. and Byrne, G. D. , "EPISODE:An Effective Package for the Integration of Systems of Ordinary Differential Equations", UCID-30112, Rev.1, Lawrence Livermore Laboratory, (1977).

(9) R. F. Churchhouse, ed. , "Handbook of Applicable Mathematics, vol. III", John Wiley & Sons Inc. , (1981)

2.2 ORDINARY DIFFERENTIAL EQUATIONS (INITIAL VALUE PROBLEMS)

2.2.1 DKSNCS, RKSNCS
High-Order Simultaneous Ordinary Differential Equations (Speed Priority)

(1) **Function**

DKSNCS or RKSNCS solves an ordinary differential equation initial value problem based on automatic step size control when function evaluation is inexpensive and precision requirements are not high.

(2) **Usage**

Double precision:

CALL DKSNCS (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, WK, IERR)

Single precision:

CALL RKSNCS (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, 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	F	–	–	Input	Name of subroutine F(X, Y, N) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	See Contents	Input	Initial values of $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)} - 1)$ at $x = x_0$. $\mathrm{Y(i, j)} = \mathrm{y_i^{(j)}}$ **Size**: $\mathrm{N}, 0 : \mathrm{MX}$
				Output	Calculated solution $y_i^{(j)}$ $(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)})$ at $x = x_e$.
4	N	I	1	Input	Number of simultaneous equations
5	MX	I	1	Input	Maximum differential order (MAX(M(i)))
6	M	I	N	Input	Differential order M(i) of the left-hand side of each of the simultaneous equations

No.	Argument	Type	Size	Input/ Output	Contents
7	XF	{D R}	1	Input	Final point x_f where solution is to be obtained
8	ER	{D R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
9	EA	{D R}	1	Input	Required local absolute precision (Default value: Expressible positive minimum value $\times 2^{24}$)
10	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is called for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
11	ISR	I	1	Input	0: Integration up to x_f is performed and output. If no error occurs, $x_e = x_f$. Non-zero: x_e and $y_i^{(j)}$ are output each time the step size automatically advances between x_0 and x_f. The final output is at the point x_f, with $x_e = x_f$.
12	WK	{D R}	See Contents	Work	Work area The step size that was used last is entered in WK(1). **Size**: $8 \times \mathrm{N} \times (\mathrm{MX}+1)+1$
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq 1, \mathrm{NST} \geq 0$

(b) $\mathrm{M(i)} \geq 1, \mathrm{MX} \geq \mathrm{MAX(M(i))}$ $(\mathrm{i} = 1, \cdots, \mathrm{N})$

(c) ER$\geq e_r$ where e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(d) EA $\geq$ (Expressible positive minimum value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA 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 ER or EA set to the default value.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000	The step size became too small during the calculation.	The value of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the high-order simultaneous ordinary differential equations:

$$\begin{cases} y_1^{(m_1)} &= f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \quad\vdots & \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ y_i^{(m_i)} &= f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \quad\vdots & \quad\vdots \qquad\qquad\qquad\qquad \vdots \\ y_n^{(m_n)} &= f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{cases}$$

(Where, if left-hand side is $y_i^{(m_i)}$, then differential order j of corresponding right-hand side $y_i^{(j)}$ must satisfy $j \leq m_i - 1$.) this subroutine F(X, Y, N) (in double-precision) should be created as follows:

SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:∗)
$\mathrm{Y}(1, \mathrm{M}(1)) = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
$\quad\vdots$
$\mathrm{Y}(\mathrm{i}, \mathrm{M}(\mathrm{i})) = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
$\quad\vdots$
$\mathrm{Y}(\mathrm{N}, \mathrm{M}(\mathrm{N})) = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
RETURN
END

where the following correspondences are assumed.
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y(i, 0)}$, $y_i^{(j)} \leftrightarrow \mathrm{Y(i, j)}$

Example:

$$\begin{cases} 8(y_1'')^2 - 2 - y_2 y_1' = 0 \quad (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 \quad (y_2' > 0) \ \cdots\cdots \text{②} \end{cases}$$

According to equation ①, y_1'' is: $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$
According to equation ②, y_2' is: $y_2' = \sqrt{1 + (y_1')^2}$
Therefore, define the subroutine as follows:

SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:∗)
Y(1, 2) = SQRT(0.25D0 + 0.125D0∗Y(2, 0)∗Y(1, 1))
Y(2, 1) = SQRT(1.0D0 + Y(1, 1)∗∗2)
RETURN
END

The Input arguments: N=2, MX=2, M(1)=2, M(2)=1. Assign initial values for Y(1, 0), Y(1, 1) and Y(2, 0).

(b) Set NST=0 when integrating for the first time.

(c) When integrating continuously, use the output value of X, Y, and NST directly as the next input values.

(d) If IERR=4000, you can either use subroutine 2.2.4 $\left\{\begin{matrix}\text{DKSSCA}\\\text{RKSSCA}\end{matrix}\right\}$ beginning from point x_e, or you can continue to solve the problem by making the required precision more lenient.

(e) This subroutine uses the Runge-Kutta-Verner method.

(7) **Example**

(a) Problem

Solve the following simultaneous quadratic ordinary differential equations:

$$\begin{cases} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

based on the following initial conditions at $x = 0.0$.

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

(b) Input data

Name of subroutine F(X, Y, N):FKSNCS, X=0.0, N=2, Y(1, 0)=1.0, Y(1, 1)=0.0, Y(2, 0)=0.0, Y(2, 1)=1.0, MX=2, M(1)=2, M(2)=2, XF, ER, EA, NST=0 and ISR=0.

(c) Main program

```
      PROGRAM BKSNCS
! *** EXAMPLE OF DKSNCS ***
      REAL(8) Y(2,0:2),WK(49),X,XF,ER,EA
      INTEGER  M(2)
      PARAMETER (N = 2,MX = 2)
      EXTERNAL FKSNCS
!
      M(1) = 2
      M(2) = 2
      NST = 0
      ISR = 0
      READ(*,*) X
      READ(*,*) ((Y(I,J),J=0,M(I)-1),I=1,2)
      READ(*,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,((I,J,Y(I,J),J=0,M(I)-1),I=1,N)
      WRITE(6,1300) N,MX,(I,M(I),I=1,N)
      WRITE(6,1500) ER,EA
      WRITE(6,1600) NST,ISR
      DO 10 J=3,6,3
         IF(J.EQ.6) WRITE(6,1100)
         XF = J
         WRITE(6,1400) XF
         CALL DKSNCS(FKSNCS,X,Y,N,MX,M,XF,ER,EA,NST,ISR,WK,IERR)
         WRITE(6,1700)
         WRITE(6,2000) IERR
         WRITE(6,2100) X,((I,K,Y(I,K),K=0,M(I)),I=1,N)
         WRITE(6,2200) NST
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKSNCS ***',/,/,8X,'Y1'''' = -Y1/R',/,8X,&
                   'Y2'''' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X        =',F9.5,/,/,(8X,'Y(',I2,',',I2,') =',F9.5,/))
 1300 FORMAT(8X,'N        =',I4,/,/,8X,'MX       =',I4,/,/,&
             (8X,'M(',I2,')    =',I4,/))
 1400 FORMAT(8X,'XF       =',F9.5,/)
 1500 FORMAT(8X,'ER       =',G12.5,/,/,8X,'EA       =',G12.5,/)
 1600 FORMAT(8X,'NST      =',I4,/,/,8X,'ISR      =',I4,/)
 1700 FORMAT(6X,'** OUTPUT **',/)
 2000 FORMAT(8X,'IERR     =',I5,/)
 2100 FORMAT(8X,'X        =',F9.5,/,/,&
             7X,'(SOLUTION)',/,/,(8X,'Y(',I2,',',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKSNCS(X,Y,N)
      REAL(8) Y(N,0:*),X,R
!
      R = (SQRT(Y(1,0)**2+Y(2,0)**2))**3
      Y(1,2) = X*0-Y(1,0)/R
```

```
      Y(2,2) = -Y(2,0)/R
      RETURN
      END
```

(d) Output results

```
*** DKSNCS ***
   Y1'' = -Y1/R
   Y2'' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3
 **INPUT **
   X        =  0.00000
   Y( 1, 0) =  1.00000
   Y( 1, 1) =  0.00000
   Y( 2, 0) =  0.00000
   Y( 2, 1) =  1.00000
   N        =   2
   MX       =   2
   M( 1)    =   2
   M( 2)    =   2
   ER       =  0.0000
   EA       = 0.10000E-09
   NST      =   0
   ISR      =   0
   XF       =  3.00000
 ** OUTPUT **
   IERR     =    0
   X        =  3.00000
  (SOLUTION)
   Y( 1, 0) =  -0.9899924966D+00
   Y( 1, 1) =  -0.1411200081D+00
   Y( 1, 2) =   0.9899924965D+00
   Y( 2, 0) =   0.1411200081D+00
   Y( 2, 1) =  -0.9899924966D+00
   Y( 2, 2) =  -0.1411200081D+00
  (STEP NUMBER OF CALCULATION)
   NST      =   56
 **INPUT **
   XF       =  6.00000
 ** OUTPUT **
   IERR     =    0
   X        =  6.00000
  (SOLUTION)
   Y( 1, 0) =   0.9601702866D+00
   Y( 1, 1) =   0.2794154983D+00
   Y( 1, 2) =  -0.9601702866D+00
   Y( 2, 0) =  -0.2794154983D+00
   Y( 2, 1) =   0.9601702866D+00
   Y( 2, 2) =   0.2794154983D+00
  (STEP NUMBER OF CALCULATION)
   NST      =  112
```

2.2.2 DKSNCA, RKSNCA
High-Order Simultaneous Ordinary Differential Equations (Precision Priority)

(1) **Function**

DKSNCA or RKSNCA solves ordinary differential equations based on automatic step size and automatic order control when function evaluation is expensive and precision requirements are high.

(2) **Usage**

Double precision:

CALL DKSNCA (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, IWK, WK, IERR)

Single precision:

CALL RKSNCA (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, 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	F	–	–	Input	Name of subroutine F(X, Y, N) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	See Contents	Input	Initial values of $y_i^{(j)}(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)} - 1)$ at $x = x_0$. $\mathrm{Y(i, j)} = \mathrm{y_i^{(j)}}$ **Size**: $\mathrm{N}, 0 : \mathrm{MX}$
				Output	Calculated solution $y_i^{(j)}(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)})$ at $x = x_e$.
4	N	I	1	Input	Number of simultaneous equations
5	MX	I	1	Input	Maximum differential order (MAX(M(i)))
6	M	I	N	Input	Differential order M(i) of the left-hand side of each of the simultaneous equations
7	XF	{D / R}	1	Input	Final point x_f where solution is to be obtained

No.	Argument	Type	Size	Input/ Output	Contents
8	ER	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	1	Input	Required local relative precision **Default value**: Double precision : 10^{-14} Single precision : 10^{-5}
9	EA	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Expressible positive minimum value $\times 2^{24}$)
10	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is called for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
11	ISR	I	1	Input	0: Integration up to x_f is performed and output. If no error occurs, $x_e = x_f$. Non-zero: x_e and $y_i^{(j)}$ are output each time the step size automatically advances between x_0 and x_f. The final output is at the point x_f, with $x_e = x_f$.
12	IWK	I	$2 \times \text{N} + 3$	Work	Work area The order q that was used in equation i is entered in IWK(i).
13	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area The step size that was used last is entered in WK(1). **Size**: $\text{MX} \times (\text{N} + 19) + 20 \times \text{N} + 40$
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1, NST≥0

(b) $\text{M(i)} \geq 1, \text{MX} \geq \text{MAX(M(i))}$ $(\text{i} = 1, \cdots, \text{N})$

(c) ER≥ e_r. Where, e_r= double precision:10^{-14}, single precision:10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(d) EA ≥ (Expressible positive minimum value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA 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 ER or EA set to the default value.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000	The step size became too small during the calculation.	The value of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the high-order simultaneous ordinary differential equations:

$$\begin{cases} y_1^{(m_1)} &=& f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \quad\vdots & \vdots & \qquad\qquad\qquad\qquad \vdots \\ y_i^{(m_i)} &=& f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \quad\vdots & \vdots & \qquad\qquad\qquad\qquad \vdots \\ y_n^{(m_n)} &=& f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{cases}$$

(Where, if left-hand side is $y_i^{(m_i)}$, then differential order j of corresponding right-hand side $y_i^{(j)}$ must satisfy $j \leq m_i - 1$.) this subroutine F(X, Y, N) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:*)
```

$\mathrm{Y}(1, \mathrm{M}(1)) = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$

$\vdots$

$\mathrm{Y}(\mathrm{i}, \mathrm{M}(\mathrm{i})) = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$

$\vdots$

$\mathrm{Y}(\mathrm{N}, \mathrm{M}(\mathrm{N})) = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$

```
RETURN
END
```

where the following correspondences are assumed.

$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y}(\mathrm{i}, 0)$, $y_i^{(j)} \leftrightarrow \mathrm{Y}(\mathrm{i}, \mathrm{j})$

Example:

$$\begin{cases} 8(y_1'')^2 - 2 - y_2 y_1' = 0 \quad (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 \quad (y_2' > 0) \ \cdots\cdots \text{②} \end{cases}$$

According to equation ①, y_1'' is: $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$

According to equation ②, y_2' is: $y_2' = \sqrt{1 + (y_1')^2}$

Therefore, define the subroutine as follows:

```
SUBROUTINE F(X, Y, N)
REAL(8) X, Y
```

DIMENSION Y(N, 0:∗)
Y(1, 2) = SQRT(0.25D0 + 0.125D0 ∗ Y(2, 0) ∗ Y(1, 1))
Y(2, 1) = SQRT(1.0D0 + Y(1, 1) ∗ ∗2)
RETURN
END

The Input arguments: N=2, MX=2, M(1)=2, M(2)=1. Assign initial values for Y(1, 0), Y(1, 1), Y(2, 0).

(b) Set NST=0 when integrating for the first time.

(c) When integrating continuously, use the output value of X, Y, and NST directly as the next input values. Also, work area IWK and WK must never be overwritten.

(d) For single-precision calculations, since IERR=4000 is likely to be output, you should use double precision as much as possible. If IERR=4000, you can either use subroutine 2.2.4 $\left\{\begin{matrix}\text{DKSSCA}\\ \text{RKSSCA}\end{matrix}\right\}$ beginning from point x_e, or you can continue to solve the problem by making the required precision more lenient.

(e) This subroutine uses a linear multistep method that uses a quotient difference method.

(7) **Example**

(a) Problem
Solve the following simultaneous quadratic ordinary differential equations:

$$\begin{cases} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

based on the following initial conditions at $x = 0.0$.

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

(b) Input data
Name of subroutine F(X, Y, N):FKSNCA, X=0.0, N=2, Y(1, 0)=1.0, Y(1, 1)=0.0, Y(2, 0)=0.0, Y(2, 1)=1.0, MX=2, M(1)=2, M(2)=2, XF, ER, EA, NST=0 and ISR=0.

(c) Main program

```
      PROGRAM BKSNCA
! *** EXAMPLE OF DKSNCA ***
      REAL(8) Y(2,0:2),WK(122),X,XF,ER,EA
      INTEGER  M(2),IWK(2*2+3)
      PARAMETER (N = 2,MX = 2)
      EXTERNAL FKSNCA
!
      M(1) = 2
      M(2) = 2
      NST = 0
      ISR = 0
      READ(*,*) X
      READ(*,*) ((Y(I,J),J=0,M(I)-1),I=1,2)
      READ(*,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,((I,J,Y(I,J),J=0,M(I)-1),I=1,N)
      WRITE(6,1300) N,MX,(I,M(I),I=1,N)
      WRITE(6,1500) ER,EA
      WRITE(6,1600) NST,ISR
      DO 10 J=3,6,3
         IF(J.EQ.6) WRITE(6,1100)
         XF = J
         WRITE(6,1400) XF
         CALL DKSNCA(FKSNCA,X,Y,N,MX,M,XF,ER,EA,NST,ISR,IWK,WK,IERR)
         WRITE(6,1700)
         WRITE(6,2000) IERR
         WRITE(6,2100) X,((I,K,Y(I,K),K=0,M(I)),I=1,N)
         WRITE(6,2200) NST
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKSNCA ***',/,/,&
```

```
                8X,'Y1'''' = -Y1/R',/,&
                8X,'Y2'''' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X        =',F9.5,/,/,(8X,'Y(',I2,',',I2,') =',F9.5,/))
 1300 FORMAT(8X,'N        =',I4,/,/,8X,'MX       =',I4,/,/,&
             (8X,'M(',I2,')    =',I4,/))
 1400 FORMAT(8X,'XF       =',F9.5,/)
 1500 FORMAT(8X,'ER       =',G12.5,/,/,8X,'EA       =',G12.5,/)
 1600 FORMAT(8X,'NST      =',I4,/,/,8X,'ISR      =',I4,/)
 1700 FORMAT(6X,'** OUTPUT **',/)
 2000 FORMAT(8X,'IERR     =',I5,/)
 2100 FORMAT(8X,'X        =',F9.5,/,/,&
            7X,'(SOLUTION)',/,/,(8X,'Y(',I2,',',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKSNCA(X,Y,N)
      REAL(8) Y(N,0:*),X,R
!
      R = (SQRT(Y(1,0)**2+Y(2,0)**2))**3
      Y(1,2) = X*0-Y(1,0)/R
      Y(2,2) = -Y(2,0)/R
      RETURN
      END
```

(d) Output results

```
*** DKSNCA ***

   Y1'' = -Y1/R
   Y2'' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3

 **INPUT **

   X        =  0.00000

   Y( 1, 0) =  1.00000

   Y( 1, 1) =  0.00000

   Y( 2, 0) =  0.00000

   Y( 2, 1) =  1.00000

   N        =   2

   MX       =   2

   M( 1)    =   2

   M( 2)    =   2

   ER       =  0.0000

   EA       = 0.10000E-09

   NST      =   0

   ISR      =   0

   XF       =  3.00000

 ** OUTPUT **

   IERR     =    0

   X        =  3.00000

  (SOLUTION)

   Y( 1, 0) =  -0.9899924966D+00

   Y( 1, 1) =  -0.1411200084D+00

   Y( 1, 2) =   0.9899924965D+00

   Y( 2, 0) =   0.1411200086D+00

   Y( 2, 1) =  -0.9899924965D+00

   Y( 2, 2) =  -0.1411200085D+00

  (STEP NUMBER OF CALCULATION)

   NST      =   35

 **INPUT **

   XF       =  6.00000

 ** OUTPUT **

   IERR     =    0
```

```
 X        =  6.00000
(SOLUTION)
 Y( 1, 0) =   0.9601702866D+00
 Y( 1, 1) =   0.2794154984D+00
 Y( 1, 2) =  -0.9601702867D+00
 Y( 2, 0) =  -0.2794154983D+00
 Y( 2, 1) =   0.9601702866D+00
 Y( 2, 2) =   0.2794154983D+00
(STEP NUMBER OF CALCULATION)
 NST      =   48
```

2.2.3 DKINCT, RKINCT
Implicit Simultaneous Ordinary Differential Equations

(1) **Function**

DKINCT or RKINCT solves an initial value problem when ordinary differential equations are taken as simultaneous equations with implicit ordinary differential equations or with algebraic or nonlinear equations. It also can solve general ordinary differential equations or nonlinear simultaneous equations.

(2) **Usage**

Double precision:

CALL DKINCT (F, X, Y, N, M, K, XF, IDV, NST, ISD, IWK, WK, IERR)

Single precision:

CALL RKINCT (F, X, Y, N, M, K, XF, IDV, NST, ISD, 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	F	–	–	Input	Name of subroutineF(X, Y, N, JI, FN) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	See Contents	Input	Initial values of $y_i^{(j)}$ at $x = x_0$ $(i = 1, \cdots, \mathrm{N}\ ;\ j = 0, \cdots, \mathrm{M(i)} - 1)$. $\mathrm{Y(i,j)} = \mathrm{y_i^{(j)}}$. **Size**: $\mathrm{N}, 0 : \mathrm{MX}$ Here, $\mathrm{MX} = \mathrm{MAX(M(i) + K(i) + ISD(i) - 1)}$
				Output	Calculated solution $y_i^{(j)}$ at $x = x_e$. $(i = 1, \cdots, \mathrm{N}\ ;\ j = 0, \cdots, \mathrm{M(i)})$
4	N	I	1	Input	Number of simultaneous equations
5	M	I	N	Input	Differential order M(i) of each of the simultaneous equations
6	K	I	N	Input	Number of equations in the subset consisting of one of the simultaneous equations and the equations obtained by differentiating that equation; K(i)

No.	Argument	Type	Size	Input/ Output	Contents
7	XF	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Location x_f where solution is to be obtained. (If no error occurs, $x_e = x_f$
8	IDV	I	1	Input	Number of integration subdivisions of the interval from x_0 to x_f. (The integration step size becomes $\lvert x_f - x_0\rvert$/IDV.
9	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is used for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
10	ISD	I	N	Input	Automatic function differentiation switch ISD(i) = 0: Automatic differentiation of the i-th equation group is not performed. (K(i) > 1 and the differentiated equations are defined in subroutineF.) ISD(i) = 1: Precision is raised by performing further automatic differentiation of the i-th equation group. (If the i-th equation is difficult to differentiate, then K(i) = 1 and ISD(i) = 1 should be set without creating the differentiated equations.)
11	IWK	I	$12 \times \mathrm{N} + 1$	Work	Work area
12	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	$24 \times \mathrm{N}$	Work	Work area
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N≥1, NST≥0, IDV≥1

(b) $0 \leq \mathrm{M(i)} \leq 4$, $\mathrm{K(i)} \geq 1$ $(\mathrm{i} = 1, \cdots, \mathrm{N})$

(c) ISD(i)=0 or 1, $\mathrm{M(i)} + \mathrm{K(i)} + \mathrm{ISD(i)} \leq 6$ $(\mathrm{i} = 1, \cdots, \mathrm{N})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
4000	Nonlinear simultaneous equations could not be solved.	

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N, JI, FN) that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the implicit simultaneous ordinary differential equations:

$$\begin{cases} f_1(x, \cdots, y_1^{(m_1)}, \cdots) = 0 \\ f_2(x, \cdots, y_2^{(m_2)}, \cdots) = 0 \\ \vdots \\ f_i(x, \cdots, y_i^{(m_i)}, \cdots) = 0 \\ \vdots \\ f_n(x, \cdots, y_n^{(m_n)}, \cdots) = 0 \end{cases}$$

(where $y_i^{(m_i)}$ is the term which has maximum differential order and m_i is corresponding differential order), this subroutine F(X, Y, N, JI, FN) (in double-precision) should be created as follows:

SUBROUTINE F(X, Y, N, JI, FN)
REAL(8) X, Y, FN
DIMENSION Y(N, 0:∗)
IF (JI .EQ. 1) THEN
FN=$f_1(x, \cdots, y_1^{(m_1)}, \cdots)$
ELSE IF (JI .EQ. 2) THEN
FN=$f_1'(x, \cdots, y_1^{(m_1+1)}, \cdots)$
ELSE IF (JI .EQ. 3) THEN
FN=$f_1''(x, \cdots, y_1^{(m_1+2)}, \cdots)$
⋮
ELSE IF (JI .EQ. k_1) THEN
FN=$f_1^{(k_1-1)}(x, \cdots, y_1^{(m_1+k_1-1)}, \cdots)$
ELSE IF (JI .EQ. (k_1+1)) THEN
FN=$f_2(x, \cdots, y_2^{(m_2)}, \cdots)$
⋮
ELSE IF (JI .EQ. (k_1+k_2)) THEN
FN=$f_2^{(k_2-1)}(x, \cdots, y_2^{(m_2+k_2-1)}, \cdots)$
⋮
ELSE IF (JI .EQ. $(\sum_{i=1}^{n} k_i)$) THEN
FN=$f_n^{(k_n-1)}(x, \cdots, y_n^{(m_n+k_n-1)}, \cdots)$
ENDIF
RETURN
END

where the following correspondences are assumed.
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow$ Y(i, 0), $y_i^{(j)} \leftrightarrow$ Y(i, j)
In this case, you must carefully determine the number of i-th equation group $k_i \leftrightarrow$ K(i) such that restriction (c) is satisfied. And also you must determine the size of array Y from them.

i. If the i-th expression is difficult to differentiate, then either use automatic differentiation by assuming K(i) = 1 and ISD(i) = 1 without creating the differentiated equations directly, or use a

difference equation. If $f(y'', y', y, x) = 0$ is given, then for $h = \sqrt[3]{\text{Unit for determining error}}$, let difference equation be as follows:

$$\begin{aligned} f'(y'', y', y, x) = \; & y''' \frac{\partial f(y'', y', y, x)}{\partial y''} + y'' \frac{\partial f(y'', y', y, x)}{\partial y'} \\ & + y' \frac{\partial f(y'', y', y, x)}{\partial y} + \frac{\partial f(y'', y', y, x)}{\partial x} \\ \simeq \; & y''' \frac{f(y''+h, y', y, x) - f(y''-h, y', y, x)}{2h} \\ & + y'' \frac{f(y'', y'+h, y, x) - f(y'', y'-h, y, x)}{2h} \\ & + y' \frac{f(y'', y', y+h, x) - f(y'', y', y-h, x)}{2h} \\ & + \frac{f(y'', y', y, x+h) - f(y'', y', y, x-h)}{2h} \end{aligned}$$

For example, when

$$f(y_1'', y_1', y_1, x) = y_1^2 + x y_1' + \log(\cos(y_1'' x^2)) = 0$$

hold and if part of expression $g(y_1'', x) = \log(\cos(y_1'' x^2))$ is difficult to differentiate, use the difference equation as follows:

$$\begin{aligned} f'(y_1''', y_1'', y_1', y_1, x) = \; & 2 y_1 y_1' + y_1' + x y_1'' \\ & + y_1''' \frac{\log(\cos((y_1''+h)x^2)) - \log(\cos((y_1''-h)x^2))}{2h} \\ & + \frac{\log(\cos(y_1''(x+h)^2)) - \log(\cos(y_1''(x-h)^2))}{2h} \end{aligned}$$

Note that $g(y_1'', x)$ is not a function of both y_1' and y_1.

ii. The equations that are taken to be simultaneous should be arranged as follows. A equation, which contains terms of the highest order differential $y_i^{(m_i)} \leftrightarrow \mathrm{Y(i, M(i))}$ of y_i, is a first equation in the i-th equation group. And if plural equations contain terms of the highest order differential $y_i^{(m_i)}$ then it is better to choose a equation that contains the most dominant one for the first equation in the i-th equation group. Further, the first equation in the i-th equation group must contain a term of $y_i^{(m_i)}$.

Example

$$\begin{cases} 3y_1 + 3y_2 + y_3 - 1 = 0 & \cdots\cdots\cdots ① \\ 2y_1' + y_2 + 2y_3 - 6 = 0 & \cdots\cdots\cdots ② \\ y_1 + 2y_2 + 3y_3 - 5 = 0 & \cdots\cdots\cdots ③ \end{cases}$$

These equations should be rearranged so that (②) is a first equation in the first equation group, (①) is a first equation in the second equation group, and (③) is a first equation in the third equation group.

iii. This subroutine also can solve simultaneous nonlinear equations. For this case, the subroutine F(X, Y, N, JI, FN) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, Y, N, JI, FN)
REAL(8) X, Y, FN
DIMENSION Y(N, 0:*)
IF (JI .EQ. 1) THEN
FN=f_1(x, y_1, ···)
ELSE IF (JI .EQ. 2) THEN
FN=f_2(x, y_2, ···)
    ⋮
ELSE IF (JI .EQ. N) THEN
FN=f_n(x, y_n, ···)
```

ENDIF
RETURN
END

where the following correspondences are assumed.
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow$ Y(i, 0)
In this usage, let the parameters be as follows:
N = Number of simultaneous equations
M(i) = ISD(i) = 0 (i = $1, \cdots,$ N) K(i) = 1 (i = $1, \cdots,$ N)
X = XF = 0
Y(i, j) (i = $1, \cdots,$ N, j = 0) :not necessary (arbitrary).
IDV = 1, NST = 0
$y_i = 0 (i = 1, \cdots,$ N) will be assumed as initial values for the calculations.
To solve a nonlinear equation, let N = 1 in the above parameters.

Subroutine creation examples (in double-precision):

i. For an ordinary case:

$$\begin{cases} 8(y_1'')^2 - 2 - y_2 y_1' = 0 \\ (y_2')^2 - 1 - (y_1')^2 = 0 \end{cases}$$

SUBROUTINE F(X, Y, N, JI, FN)
REAL(8) X, Y, FN
DIMENSION Y(N, 0:∗)
IF (JI .EQ. 1) THEN
FN = 8.0D0 ∗ Y(1, 2) ∗ ∗2 − 2.0D0 − Y(2, 0) ∗ Y(1, 1)
$\cdots\cdots\cdots (8(y_1'')^2 - 2 - y_2 y_1' = 0)$
ELSE IF (JI .EQ. 2) THEN
FN = 16.0D0 ∗ Y(1, 2) ∗ Y(1, 3) − Y(2, 0) ∗ Y(1, 2) − Y(2, 1) ∗ Y(1, 1)
$\cdots\cdots\cdots (16 y_1'' y_1''' - y_2 y_1'' - y_2' y_1' = 0)$
ELSE IF (JI .EQ. 3) THEN
FN = Y(2, 1) ∗ ∗2 − 1.0D0 − Y(1, 1) ∗ ∗2
$\cdots\cdots\cdots ((y_2')^2 - 1 - (y_1')^2 = 0)$
ELSE IF (JI .EQ. 4) THEN
FN = 2.0D0 ∗ Y(2, 1) ∗ Y(2, 2) − 2.0D0 ∗ Y(1, 1) ∗ Y(1, 2)
$\cdots\cdots\cdots (2 y_2' y_2'' - 2 y_1' y_1'' = 0)$
ENDIF
RETURN
END

Assume the following arguments:
N=2, M(1)=2, M(2)=1, K(1)=2, and K(2)=2 and assign the initial values for Y(1, 0), Y(1, 1), Y(2, 0).

ii. When differential equations are implicitly taken to be simultaneous with an algebraic equation:

$$\begin{cases} y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0 \\ y_2'' - y_3' - \dfrac{x^2}{2} = 0 \\ y_3' - \dfrac{x^2}{2} = 0 \\ y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0 \end{cases}$$

SUBROUTINE F(X, Y, N, JI, FN)
REAL(8) X, Y, FN
DIMENSION Y(N, 0:∗)
IF (JI .EQ. 1) THEN
FN = Y(1, 3) − Y(2, 2) − 2.0D0 ∗ Y(3, 1) − X ∗ X
$\cdots\cdots\cdots (y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0)$
ELSE IF (JI .EQ. 2) THEN
FN = Y(1, 4) − Y(2, 3) − 2.0D0 ∗ Y(3, 2) − 2.0D0 ∗ X
$\cdots\cdots\cdots (y_1^{(4)} - y_2^{(3)} - 2y_3'' - 2x = 0)$
ELSE IF (JI .EQ. 3) THEN
FN = Y(1, 5) − Y(2, 4) − 2.0D0 ∗ Y(3, 3) − 2.0D0
$\cdots\cdots\cdots (y_1^{(5)} - y_2^{(4)} - 2y_3^{(3)} - 2 = 0)$
ELSE IF (JI .EQ. 4) THEN
FN = Y(2, 2) − Y(3, 1) − X ∗ X/2.0D0
$\cdots\cdots\cdots (y_2'' - y_3' - \frac{x^2}{2} = 0)$
ELSE IF (JI .EQ. 5) THEN
FN = Y(2, 3) − Y(3, 2) − X
$\cdots\cdots\cdots (y_2^{(3)} - y_3'' - x = 0)$
ELSE IF (JI .EQ. 6) THEN
FN = Y(2, 4) − Y(3, 3) − 1.0D0
$\cdots\cdots\cdots (y_2^{(4)} - y_3^{(3)} - 1 = 0)$
ELSE IF (JI .EQ. 7) THEN
FN = Y(3, 1) − X ∗ X/2.D0
$\cdots\cdots\cdots (y_3' - \frac{x^2}{2} = 0))$
ELSE IF (JI .EQ. 8) THEN
FN = Y(3, 2) − X
$\cdots\cdots\cdots (y_3'' - x = 0)$
ELSE IF (JI .EQ. 9) THEN
FN = Y(3, 3) − 1.0D0
$\cdots\cdots\cdots (y_3^{(3)} - 1 = 0)$
ELSE IF (JI .EQ. 10) THEN
FN = Y(4, 0) − 2.0D0 ∗ Y(1, 0) + Y(2, 0) − 2.0D0 ∗ Y(3, 0) − 1.0D0
$\cdots\cdots\cdots (y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0)$
ENDIF
RETURN
END

Assume the following arguments:
N=4, M(1)=3, M(2)=2, M(3)=1, M(4)=0, K(1)=3, K(2)=3, K(3)=3, K(4)=1, ISD(i) = 0 (i = $1, \cdots 4$) and assign initial values for
Y(1, 0), Y(1, 1), Y(1, 2), Y(2, 0), Y(2, 1), Y(3, 0).

iii. For nonlinear simultaneous equations:

$$\begin{cases} (y_1 - 5)^2 + (y_2 - 2)^2 = 4 \\ (y_1 - 2)^2 + (y_2 - 2)^2 = 4 \end{cases}$$

SUBROUTINE F(X, Y, N, JI, FN)
REAL(8) X, Y, FN

```
DIMENSION Y(N, 0:*)
IF (JI .EQ. 1) THEN
FN = (Y(1,0) − 5.0D0) * *2 + (Y(2,0) − 2.0D0) * *2 − 4.0D0
ELSE IF (JI .EQ. 2) THEN
FN = (Y(1,0) − 2.0D0) * *2 + (Y(2,0) − 2.0D0) * *2 − 4.0D0
ENDIF
RETURN
END
```

Assume the following arguments: N=2, M(1)=0, M(2)=0, K(1)=1, K(2)=1, ISD(1)=0, ISD(2)=0, IDV=1, X=0, XF=0 and assume that Y(1, 0) and Y(2, 0) need not be input. Although these equations have two solution correspond to intersectional points of two circles, a solution closest to the origin $(y_1, y_2) = (0.0, 0.0)$ is returned.

(b) If ISD(i) = 1, further automatic differentiation will be performed for the i-th equation group and the precision will increase. However, the calculation speed will decrease somewhat. When K(i) = 1, ISD(i) = 1 should be set except when solving simultaneous nonlinear equations.

(c) If you create as many groups of equations as possible and either increase the number of expressions K(i) in the subsets or increase the number of integration subdivisions IDV, then the precision will increase but the calculation speed will decrease.

(d) Set NST = 0 when integrating for the first time.

(e) When integrating continuously, use the output values of X, Y, and NST directly as the next input values. Also, work areas IWK and WK must never be overwritten.

(f) Since IERR = 4000 is likely to be output for single-precision calculations, you should use double precision as much as possible. If IERR = 4000, you often will be able to solve the problem by increasing IDV.

(7) **Example**

(a) Problem

Solve the following simultaneous third order ordinary differential equations and algebraic equation:

$$\begin{cases} y_1^{(3)} - y_2'' - 2y_3' - x^2 = 0 \\ y_2'' - y_3' - \dfrac{x^2}{2} = 0 \\ y_3' - \dfrac{x^2}{2} = 0 \\ y_4 - 2y_1 + y_2 - 2y_3 - 1 = 0 \end{cases}$$

based on the following initial conditions at $x = 1.0$:
$y_1 = 0.05$, $y_1' = 0.25$, $y_1'' = 1.0$
$y_2 = 0.08333$, $y_2' = 0.333$
$y_3 = 0.1666$

(b) Input data

Name of subroutine F(X, Y, N, JI, FN): FKINCT, X=0.0, N=4,
Y(1, 0)=0.05, Y(1, 1)=0.25, Y(1, 2)=1.0, Y(2, 0)=0.08333, Y(2, 1)=0.333, Y(3, 0)=0.1666,
M(1)=3, M(2)=2, M(3)=1, M(4)=0,
K(1)=2, K(2)=2, K(3)=2, K(4)=1,
XF=1.5, IDV=10, NST=0 (for first use),
ISD(1)=1, ISD(2)=1, ISD(3)=1 and ISD(4)=0.

(c) Main program

```
      PROGRAM BKINCT
! *** EXAMPLE OF DKINCT ***
      REAL(8) Y(4,0:5),WK(120),X,XF
      INTEGER  M(4),L(4),ISD(4),IWK(49)
      PARAMETER (N = 4)
      EXTERNAL FKINCT
!
      M(1) = 3
      M(2) = 2
      M(3) = 1
      M(4) = 0
      L(1) = 2
      L(2) = 2
      L(3) = 2
      L(4) = 1
      ISD(1) = 1
      ISD(2) = 1
      ISD(3) = 1
      ISD(4) = 0
      NST = 0
      READ(*,*) X
      READ(*,*) ((Y(I,J),J=0,M(I)-1),I=1,3)
      READ(*,*) XF
      READ(*,*) IDV
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,((I,J,Y(I,J),J=0,M(I)-1),I=1,N)
      WRITE(6,1300) N,(I,M(I),I=1,N)
      WRITE(6,1400) (I,L(I),I=1,N)
      WRITE(6,1500) XF
      WRITE(6,1600) IDV,NST
      WRITE(6,1700) (I,ISD(I),I=1,N)
      CALL DKINCT(FKINCT,X,Y,N,M,L,XF,IDV,NST,ISD,IWK,WK,IERR)
      WRITE(6,1800)
      WRITE(6,2000) IERR
      WRITE(6,2100) X,((I,J,Y(I,J),J=0,M(I)),I=1,N)
      WRITE(6,2200) NST
      STOP
 1000 FORMAT(' ',/,5X,'*** DKINCT ***',/,/,&
                 8X,'Y1''''''-Y2''''-2*Y3''-X**2 = 0',/,&
                 8X,'Y2''''-Y3''-X**2/2       = 0',/,&
                 8X,'Y3''-X**2/2            = 0',/,&
                 8X,'Y4-2*Y1+Y2-2*Y3-1     = 0',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X        =',F9.5,/,/,(8X,'Y(',I2,',',I2,') =',F9.5,/))
 1300 FORMAT(8X,'N        =',I4,/,/,(8X,'M(',I2,')    =',I4,/))
 1400 FORMAT((8X,'L(',I2,')    =',I4,/))
 1500 FORMAT(8X,'XF       =',F9.5,/)
 1600 FORMAT(8X,'IDV      =',I4,/,/,8X,'NST      =',I4,/)
 1700 FORMAT((8X,'ISD(',I2,')  =',I4,/))
 1800 FORMAT(6X,'** OUTPUT **',/)
 2000 FORMAT(8X,'IERR     =',I5,/)
 2100 FORMAT(8X,'X        =',F9.5,/,/,&
             7X,'(SOLUTION)',/,/,(8X,'Y(',I2,',',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKINCT(X,Y,N,JI,FN)
      REAL(8) Y(N,0:*),X,FN
!
      IF(JI.EQ.1) THEN
         FN = Y(1,3)-Y(2,2)-2.D0*Y(3,1)-X*X
      ELSEIF(JI.EQ.2) THEN
         FN = Y(1,4)-Y(2,3)-2.D0*Y(3,2)-(X+X)
      ELSEIF(JI.EQ.3) THEN
         FN = Y(2,2)-Y(3,1)-X*X*0.5D0
      ELSEIF(JI.EQ.4) THEN
         FN = Y(2,3)-Y(3,2)-X
      ELSEIF(JI.EQ.5) THEN
         FN = Y(3,1)-X*X*0.5D0
      ELSEIF(JI.EQ.6) THEN
         FN = Y(3,2)-X
      ELSEIF(JI.EQ.7) THEN
         FN = Y(4,0)-2.D0*Y(1,0)+Y(2,0)-2.D0*Y(3,0)-1.D0
      ENDIF
      RETURN
      END
```

(d) Output results

```
   *** DKINCT ***

      Y1'''-Y2''-2*Y3'-X**2 = 0
      Y2''-Y3'-X**2/2       = 0
      Y3'-X**2/2            = 0
      Y4-2*Y1+Y2-2*Y3-1     = 0

    **INPUT **

      X        =  1.00000
```

```
  Y( 1, 0) =  0.05000
  Y( 1, 1) =  0.25000
  Y( 1, 2) =  1.00000
  Y( 2, 0) =  0.08333
  Y( 2, 1) =  0.33300
  Y( 3, 0) =  0.16660
  N        =   4
  M( 1)    =   3
  M( 2)    =   2
  M( 3)    =   1
  M( 4)    =   0
  L( 1)    =   2
  L( 2)    =   2
  L( 3)    =   2
  L( 4)    =   1
  XF       =  1.50000
  IDV      =  10
  NST      =   0
  ISD( 1)  =   1
  ISD( 2)  =   1
  ISD( 3)  =   1
  ISD( 4)  =   0
** OUTPUT **
  IERR     =    0
  X        =  1.50000
 (SOLUTION)
  Y( 1, 0) =   0.3796875000D+00
  Y( 1, 1) =   0.1265625000D+01
  Y( 1, 2) =   0.3375000000D+01
  Y( 1, 3) =   0.6750000000D+01
  Y( 2, 0) =   0.4217050000D+00
  Y( 2, 1) =   0.1124666667D+01
  Y( 2, 2) =   0.2250000000D+01
  Y( 3, 0) =   0.5624333333D+00
  Y( 3, 1) =   0.1125000000D+01
  Y( 4, 0) =   0.2462536667D+01
 (STEP NUMBER OF CALCULATION)
  NST      =   10
```

2.2.4 DKSSCA, RKSSCA
Stiff Problem High-Order Simultaneous Ordinary Differential Equations

(1) **Function**

DKSSCA or RKSSCA solves a stiff ordinary differential equation initial value problem based on automatic step size and automatic order control.

(2) **Usage**

Double precision:

CALL DKSSCA (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, IWK, WK, IERR)

Single precision:

CALL RKSSCA (F, X, Y, N, MX, M, XF, ER, EA, NST, ISR, 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	F	–	–	Input	Name of subroutine F(X, Y, N) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	See Contents	Input	Initial values of $y_i^{(j)}(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)} - 1)$ at $x = x_0$. $\mathrm{Y(i,j)} = \mathrm{y_i^{(j)}}$. **Size**: $\mathrm{N}, 0 : \mathrm{MX}$
				Output	Calculated solution $y_i^{(j)}(i = 1, \cdots, \mathrm{N};\ j = 0, \cdots, \mathrm{M(i)})$ at $x = x_e$.
4	N	I	1	Input	Number of simultaneous equations
5	MX	I	1	Input	Maximum differential order (MAX(M(i)))
6	M	I	N	Input	Differential order M(i) of the left-hand side of each of the simultaneous equations
7	XF	{D / R}	1	Input	Final point x_f where solution is to be obtained

No.	Argument	Type	Size	Input/ Output	Contents
8	ER	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local relative precision **Default value**: Double precision : 10^{-14} Single precision : 10^{-5}
9	EA	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Expressive positive minimum value $\times 2^{24}$)
10	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is called for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
11	ISR	I	1	Input	Parameter that specifies timing for returning to user program. (See Notes (d))
12	IWK	I	See Contents	Work	Work area **Size**: $\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{M(i)} + 6$
13	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area The step size that was used last is entered in WK(1). **Size**: $28 + (\mathrm{K}+9) \times \mathrm{K} + 2 \times \mathrm{N} \times (\mathrm{MX}+1)$ Where, $\mathrm{K} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{M(i)}$
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} > 0, \mathrm{NST} \geq 0$

(b) $\mathrm{MX} \geq \mathrm{M(i)} \geq 1$ $(i = 1, \cdots, \mathrm{N})$

(c) $\mathrm{ISR} == \{0, 1, 2\}$

(d) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(e) EA $\geq$ (Expressible positive minimum value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (d) or (e) was not satisfied.	Processing is performed with ER or EA set to the default value.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
4000	The step size became too small during the calculation.	The values of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.
4100	The equation for obtaining the corrector could not be solved.	The values of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the high-order simultaneous ordinary differential equations:

$$\left\{\begin{array}{lcl} y_1^{(m_1)} & = & f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_i^{(m_i)} & = & f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots & \vdots \\ y_n^{(m_n)} & = & f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{array}\right.$$

(If the left-hand side is $y_i^{(m_i)}$, then differential order j of corresponding right-hand side $y_i^{(j)}$ must satisfy $j \leq m_i - 1$.), this subroutine F(X, Y, N) (in double-precision) should be created as follows:

SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:∗)
$\mathrm{Y}(1, \mathrm{M}(1)) = f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
$\vdots$
$\mathrm{Y}(\mathrm{i}, \mathrm{M}(\mathrm{i})) = f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
$\vdots$
$\mathrm{Y}(\mathrm{N}, \mathrm{M}(\mathrm{N})) = f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)})$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y}(\mathrm{i}, 0)$, $y_i^{(j)} \leftrightarrow \mathrm{Y}(\mathrm{i}, \mathrm{j})$

Example:

$$\left\{\begin{array}{ll} 8(y_1'')^2 - 2 - y_2 y_1' = 0 & (y_1'' > 0) \cdots\cdots \text{①} \\ (y_2')^2 - 1 - (y_1')^2 = 0 & (y_{2'} > 0) \cdots\cdots \text{②} \end{array}\right.$$

According to equation ①, $y_1'' = \sqrt{\frac{1}{4} + \frac{1}{8} \cdot y_2 \cdot y_1'}$.

According to equation ②, $y_2' = \sqrt{1 + (y_1')^2}$.
Therefore, define the subroutine as follows:

```
SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:*)
Y(1, 2) = SQRT(0.25D0 + 0.125D0 * Y(2, 0) * Y(1, 1))
Y(2, 1) = SQRT(1.0D0 + Y(1, 1) * *2)
RETURN
END
```

The input arguments:
N=2, MX=2, M(1)=2, M(2)=1.

Assign initial values for Y(1, 0), Y(1, 1) and Y(2, 0).

(b) If the value assigned for ER or EA is less than the default value, then the default value will be set.

(c) If LER is assumed to be a vector of the local relative errors and LEA is assumed to be a vector of the local absolute errors, then the solution is accepted if either $\| \text{LER} \| \leq \text{ER}$ or $\| \text{LEA} \| \leq \text{EA}$ is satisfied, where $\| \ \|$ is the Max norm. The relative errors are calculated as the error relative to the maximum value of previous calculations.

(d) Specify the value of ISR as follows.

ISR=0: After integrating past XF, interpolate at XF.

ISR=1: Calculate exactly up to XF, without interpolating.

ISR=2: Calculate only one integration interval towards XF.

If the differential coefficients are not defined at a point past XF or if there is a point of discontinuity just beyond XF, then specify ISR = 1.

(e) When integrating continuously, the output values of X and Y must be directly as the next input values. In addition, IWK and WK must not be overwritten.

(f) If IERR = 4000, you can continue to solve the problem continuously by making the required precision more lenient.

(7) **Example**

(a) Problem
Solve the following simultaneous quadratic ordinary differential equations:

$$\begin{cases} y_1'' = -\frac{y_1}{\gamma} \\ y_2'' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

under the following initial conditions at $x = 0.0$:

$$y_1(0.0) = 1.0, y_1'(0.0) = 0.0, y_2(0.0) = 0.0, y_2'(0.0) = 1.0$$

(b) Input data
Name of subroutine F(X, Y, N):FKSSCA, X=0.0, N=2, Y(1, 0)=1.0, Y(1, 1)=0.0, Y(2, 0)=0.0, Y(2, 1)=1.0, MX=2, M(1)=2, M(2)=2, XF, ER, EA, NST=0 and ISR=0.

(c) Main program

```
      PROGRAM BKSSCA
! *** EXAMPLE OF DKSSCA ***
      REAL(8) Y(2,0:2),WK(92),X,XF,ER,EA
      INTEGER    M(2),IWK(5,2)
```

```
      PARAMETER (N = 2,MX = 2)
      EXTERNAL   FKSSCA
!
      M(1) = 2
      M(2) = 2
      NST  = 0
      READ(*,*) ISR
      READ(*,*) X
      READ(*,*) ((Y(I,J),J=0,M(I)-1),I=1,2)
      READ(*,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,((I,J,Y(I,J),J=0,M(I)-1),I=1,N)
      WRITE(6,1300) N,MX,(I,M(I),I=1,N)
      WRITE(6,1500) ER,EA
      WRITE(6,1600) NST,ISR
      DO 10 J=3,6,3
         IF(J.EQ.6) WRITE(6,1100)
         XF = J
         WRITE(6,1400) XF
         CALL DKSSCA(FKSSCA,X,Y,N,MX,M,XF,ER,EA,NST,ISR,IWK,WK,IERR)
         WRITE(6,1700)
         WRITE(6,2000) IERR
         WRITE(6,2100) X,((I,K,Y(I,K),K=0,M(I)),I=1,N)
         WRITE(6,2200) NST
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKSSCA ***',/,/,8X,'Y1'''' = -Y1/R',/,8X,&
                   'Y2'''' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X         =',F9.5,/,/,(8X,'Y(',I2,',',I2,') =',F9.5,/))
 1300 FORMAT(8X,'N         =',I4,/,/,8X,'MX       =',I4,/,/,&
             (8X,'M(',I2,')    =',I4,/))
 1400 FORMAT(8X,'XF        =',F9.5,/)
 1500 FORMAT(8X,'ER        =',G12.5,/,/,8X,'EA       =',G12.5,/)
 1600 FORMAT(8X,'NST       =',I4,/,/,8X,'ISR      =',I4,/)
 1700 FORMAT(6X,'** OUTPUT **',/)
 2000 FORMAT(8X,'IERR      =',I5,/)
 2100 FORMAT(8X,'X         =',F9.5,/,/,&
             7X,'(SOLUTION)',/,/,(8X,'Y(',I2,',',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKSSCA(X,Y,N)
      REAL(8) Y(N,0:*),X,R
!
      R = (SQRT(Y(1,0)**2+Y(2,0)**2))**3
      Y(1,2) = X*0-Y(1,0)/R
      Y(2,2) = -Y(2,0)/R
      RETURN
      END
```

(d) Output results

```
   *** DKSSCA ***

      Y1'' = -Y1/R
      Y2'' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3

    **INPUT **

      X        =  0.00000

      Y( 1, 0) =  1.00000

      Y( 1, 1) =  0.00000

      Y( 2, 0) =  0.00000

      Y( 2, 1) =  1.00000

      N        =   2

      MX       =   2

      M( 1)    =   2

      M( 2)    =   2

      ER       =  0.0000

      EA       =  0.0000

      NST      =   0

      ISR      =   0

      XF       =  3.00000

    ** OUTPUT **

      IERR     =    0
```

```
  X        =  3.00000
 (SOLUTION)
  Y( 1, 0) =  -0.9899924966D+00
  Y( 1, 1) =  -0.1411200080D+00
  Y( 1, 2) =   0.9899924966D+00
  Y( 2, 0) =   0.1411200080D+00
  Y( 2, 1) =  -0.9899924966D+00
  Y( 2, 2) =  -0.1411200080D+00
 (STEP NUMBER OF CALCULATION)
  NST      =  551
**INPUT **
  XF       =  6.00000
** OUTPUT **
  IERR     =    0
  X        =  6.00000
 (SOLUTION)
  Y( 1, 0) =   0.9601702867D+00
  Y( 1, 1) =   0.2794154981D+00
  Y( 1, 2) =  -0.9601702867D+00
  Y( 2, 0) =  -0.2794154981D+00
  Y( 2, 1) =   0.9601702867D+00
  Y( 2, 2) =   0.2794154981D+00
 (STEP NUMBER OF CALCULATION)
  NST      = 1053
```

2.2.5 DKFNCS, RKFNCS
Simultaneous Ordinary Differential Equations of the 1st Order

(1) **Function**

DKFNCS or RKFNCS solves a simultaneous ordinary differential equation of the 1st order initial value problem that satisfies required local precision based on automatic step size control.

(2) **Usage**

Double precision:

```
CALL DKFNCS  (F, X, Y, N, XF, ER, EA, NST, WK, IERR)
```

Single precision:

```
CALL RKFNCS  (F, X, Y, N, XF, ER, EA, NST, 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	F	–	–	Input	Name of subroutine F(X, Y, N) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x.
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	N, 0 : 1	Input	Initial values of $y_i(i = 1, \cdots, \mathrm{N})$ at $x = x_0$. $\mathrm{Y(i, 0) = y_i}$.
				Output	Calculated solution $y_i^{(j)}(i = 1, \cdots, \mathrm{N}; j = 0, 1)$ at $x = x_e$
4	N	I	1	Input	Number of simultaneous equations
5	XF	{D / R}	1	Input	Final point x_f where solution is to be obtained
6	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
7	EA	{D / R}	1	Input	Required local absolute precision (Default value: Expressible positive minimum value $\times 2^{24}$)
8	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is called for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
9	WK	{D / R}	$16 \times \mathrm{N} + 1$	Work	Work area The step size that was used last is entered in WK(1).
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq 1$, $\mathrm{NST} \geq 0$

(b) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Expressible positive minimum value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA 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 ER or EA set to the default value.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	The step size became too small during the calculation.	The value of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the simultaneous ordinary differential equations:

$$\begin{cases} y_1' &=& f_1(x, y_1, \cdots, y_i, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_i' &=& f_i(x, y_1, \cdots, y_i, \cdots, y_n) \\ \vdots & \vdots & \vdots \\ y_n' &=& f_n(x, y_1, \cdots, y_i, \cdots, y_n) \end{cases}$$

this subroutine F(X, Y, N) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:*)
Y(1,1) = f1(x, y1, ..., yi, ..., yn)
  :
Y(i,1) = fi(x, y1, ..., yi, ..., yn)
  :
Y(N,1) = fn(x, y1, ..., yi, ..., yn)
RETURN
END
```

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow$ Y(i, 0)

Example:

$$\begin{cases} y_1' &=& y_2 & \cdots\cdots ① \\ y_2' &=& \dfrac{4y_1}{x^2} + \dfrac{2y_2}{x} & \cdots\cdots ② \end{cases}$$

Define the subroutine as follows:

```
SUBROUTINE F(X, Y, N)
REAL(8) X, Y
DIMENSION Y(N, 0:1)
Y(1, 1)=Y(2, 0)
Y(2,1) = 4.0D0 * Y(1,0)/(X * X) + 2.0D0 * Y(2,0)/X
```

RETURN
END

and assume that N=2.

(b) Set NST=0 when integrating for the first time.

(c) When integrating continuously, use the output values of X, Y, and NST directly as the next input values to solve the problem while successively assigning values for XF.

(d) If IERR= 4000, you can either use subroutine 2.2.4 $\left\{\begin{matrix}\text{DKSSCA}\\\text{RKSSCA}\end{matrix}\right\}$ beginning from point x_e or you can continue to solve the problem by making the required precision more lenient.

(e) This subroutine uses the Runge-Kutta-Verner method.

(7) **Example**

(a) Problem

Solve that following simultaneous ordinary differential equations of the 1st order for $x = 3.0$ and $x = 6.0$.

$$\begin{cases} y_1' = y_3 \\ y_2' = y_4 \\ y_3' = -\frac{y_1}{\gamma} \\ y_4' = -\frac{y_2}{\gamma} \quad \gamma = (y_1^2 + y_2^2)^{\frac{3}{2}} \end{cases}$$

under the following initial conditions at $x = 0.0$:

$$y_1 = 1.0, y_2 = 0.0, y_3 = 0.0, y_4 = 1.0$$

(b) Input data

Name of subroutine F(X, Y, N):FKFNCS, X=0.0, N=2, Y(1, 0)=1.0, Y(2, 0)=0.0, Y(3, 0)=0.0, Y(4, 0)=1.0, XF, ER, EA and NST=0 (for the first use).

(c) Main program

```
      PROGRAM BKFNCS
! *** EXAMPLE OF DKFNCS ***
      REAL(8) Y(4,0:1),WK(65),X,XF,ER,EA
      PARAMETER (N = 4)
      EXTERNAL FKFNCS
!
      NST = 0
      READ(*,*) X
      READ(*,*) (Y(I,0),I=1,N)
      READ(*,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,(I,Y(I,0),I=1,N)
      WRITE(6,1300) N
      WRITE(6,1500) ER,EA
      WRITE(6,1600) NST
      DO 10 J=3,6,3
         IF(J.EQ.6) WRITE(6,1100)
         XF = J
         WRITE(6,1400) XF
         CALL DKFNCS(FKFNCS,X,Y,N,XF,ER,EA,NST,WK,IERR)
         WRITE(6,1700)
         WRITE(6,2000) IERR
         WRITE(6,2100) X,((I,K,Y(I,K),K=0,1),I=1,N)
         WRITE(6,2200) NST
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKFNCS ***',/,/,8X,'Y1'' = Y3',&
       /,8X,'Y2'' = Y4',/,8X,&
       'Y3'' = -Y1/R',/,8X,'Y4'' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X         =',F9.5,/,/,(8X,'Y(',I2,', 0) =',F9.5,/))
 1300 FORMAT(8X,'N         =',I4,/)
 1400 FORMAT(8X,'XF        =',F9.5,/)
 1500 FORMAT(8X,'ER        =',G12.5,/,/,8X,'EA       =',G12.5,/)
 1600 FORMAT(8X,'NST       =',I4,/)
 1700 FORMAT(6X,'** OUTPUT **',/)
```

```
 2000 FORMAT(8X,'IERR     =',I5,/)
 2100 FORMAT(8X,'X        =',F9.5,/,/,&
             7X,'(SOLUTION)',/,/,(8X,'Y(',I2,',',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKFNCS(X,Y,N)
      REAL(8) Y(N,0:*),X,R
!
      R = (SQRT(Y(1,0)**2+Y(2,0)**2))**3
      Y(1,1) = Y(3,0)
      Y(2,1) = Y(4,0)
      Y(3,1) = -Y(1,0)/R
      Y(4,1) = -Y(2,0)/R
      RETURN
      END
```

(d) Output results

```
   *** DKFNCS ***

     Y1' = Y3
     Y2' = Y4
     Y3' = -Y1/R
     Y4' = -Y2/R    R = (SQRT(Y1**2+Y2**2))**3

    **INPUT **

     X        =  0.00000

     Y( 1, 0) =  1.00000

     Y( 2, 0) =  0.00000

     Y( 3, 0) =  0.00000

     Y( 4, 0) =  1.00000

     N        =   4

     ER       =  0.0000

     EA       = 0.10000E-09

     NST      =   0

     XF       =  3.00000

    ** OUTPUT **

     IERR     =    0

     X        =  3.00000

    (SOLUTION)

     Y( 1, 0) =  -0.9899924966D+00

     Y( 1, 1) =  -0.1411200081D+00

     Y( 2, 0) =   0.1411200081D+00

     Y( 2, 1) =  -0.9899924966D+00

     Y( 3, 0) =  -0.1411200081D+00

     Y( 3, 1) =   0.9899924965D+00

     Y( 4, 0) =  -0.9899924966D+00

     Y( 4, 1) =  -0.1411200081D+00

    (STEP NUMBER OF CALCULATION)

     NST      =   56

    **INPUT **

     XF       =  6.00000

    ** OUTPUT **

     IERR     =    0

     X        =  6.00000

    (SOLUTION)

     Y( 1, 0) =   0.9601702866D+00

     Y( 1, 1) =   0.2794154983D+00

     Y( 2, 0) =  -0.2794154983D+00
```

```
 Y( 2, 1) =   0.9601702866D+00
 Y( 3, 0) =   0.2794154983D+00
 Y( 3, 1) =  -0.9601702866D+00
 Y( 4, 0) =   0.9601702866D+00
 Y( 4, 1) =   0.2794154983D+00
(STEP NUMBER OF CALCULATION)
 NST      =  112
```

2.2.6 DKHNCS, RKHNCS
High-Order Ordinary Differential Equation

(1) **Function**

DKHNCS or RKHNCS solves a single high-order ordinary differential equation initial value problem that satisfies required local precision based on automatic step size control.

(2) **Usage**

Double precision:

```
CALL DKHNCS  (F, X, Y, M, XF, ER, EA, NST, WK, IERR)
```

Single precision:

```
CALL RKHNCS  (F, X, Y, M, XF, ER, EA, NST, 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	F	–	–	Input	Name of subroutine F(X, Y) that defines the differential equations as functions of x and y.
2	X	{D / R}	1	Input	Initial value x_0 of independent variable x.
				Output	Final destination point x_e of independent variable x
3	Y	{D / R}	0 : M	Input	Initial values of $y^{(j)}(j = 0, \cdots, \mathrm{M}-1)$ at $x = x_0$. Y(j) = $\mathrm{y}^{(\mathrm{j})}$.
				Output	Calculated solution $y^{(j)}(j = 0, \cdots, \mathrm{M})$ at $x = x_e$
4	M	I	1	Input	Differential order of the left-hand side of the equation (maximum differential order)
5	XF	{D / R}	1	Input	Final point x_f where solution is to be obtained
6	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
7	EA	{D / R}	1	Input	Required local absolute precision (Default value: Expressible positive minimum value $\times 2^{24}$)
8	NST	I	1	Input	Step count initial value (Enter 0 when the subroutine is called for the first time)
				Output	Total calculated step count (Input value when integrating consecutively)
9	WK	{D / R}	$8 \times \mathrm{M} + 9$	Work	Work area The step size that was used last is entered in WK(1).
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) M $\geq$ 1, NST $\geq$ 0

(b) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Expressible positive minimum value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA 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 ER or EA set to the default value.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	The step size became too small during the calculation.	The value of x_e and $y_i^{(j)}$ at that time are output, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, Y, N), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the high-order ordinary differential equation:

$$y^{(m)} = f(x, y, \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(If the left-hand side is $y^{(m)}$, then differential order j of right-hand side $y^{(j)}$ must satisfy $j \leq m-1$, where m is constant.), this subroutine F(X, Y) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, Y)
REAL(8) X, Y
DIMENSION Y(0:*)
Y(M) = f(x, y, ···, y^(j), ···, y^(m-1))
RETURN
END
```

where the following correspondences are assumed.
$x \leftrightarrow$X, $y \leftrightarrow$ Y(0), $y^{(j)} \leftrightarrow$ Y(j)

Example:

$$y'' = \frac{4y}{x^2} + \frac{2y'}{x}$$

Define the subroutine as follows:

```
SUBROUTINE F(X, Y)
REAL(8) X, Y
DIMENSION Y(0:*)
Y(2)=4.0D0*Y(0)/(X*X)+2.0D0*Y(1)/X
RETURN
END
```

and assume that M=2.

(b) Set NST=0 when integrating for the first time.

(c) When integrating continuously, use the output values of X, Y, and NST directly as the next input values to solve the problem while successively assigning values for XF.

(d) If IERR= 4000, you can either use subroutine 2.2.4 $\left\{\begin{matrix}\text{DKSSCA}\\ \text{RKSSCA}\end{matrix}\right\}$ beginning from point x_e or you can continue to solve the problem by making the required precision more lenient.

(e) This subroutine uses the Runge-Kutta-Verner method.

(7) **Example**

(a) Problem

Solve the following ordinary differential equation for $x = 5.0$ and $x = 10.0$:

$$y'' = \frac{4y}{x^2} + \frac{2y'}{x}$$

under the following initial conditions at $x = 1.0$:

$y = 5.0$, $y' = 3.0$.

(b) Input data

Name of subroutine F(X, Y): FHHNCS, X=1.0, Y(0)=5.0, Y(1)=3.0, M=2, XF, ER, EA and NST=0 (for the first use).

(c) Main program

```
      PROGRAM BKHNCS
! *** EXAMPLE OF DKHNCS ***
      REAL(8) Y(0:2),WK(25),X,XF,ER,EA
      PARAMETER(M = 2)
      EXTERNAL FKHNCS
!
      NST = 0
      READ(*,*) X
      READ(*,*) (Y(I),I=0,M-1)
      READ(*,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) X,(I,Y(I),I=0,M-1)
      WRITE(6,1300) M
      WRITE(6,1500) ER,EA
      WRITE(6,1600) NST
      DO 10 J=5,10,5
         IF(J.EQ.10) WRITE(6,1100)
         XF = J
         WRITE(6,1400) XF
         CALL DKHNCS(FKHNCS,X,Y,M,XF,ER,EA,NST,WK,IERR)
         WRITE(6,1700)
         WRITE(6,2000) IERR
         WRITE(6,2100) X,(I,Y(I),I=0,M)
         WRITE(6,2200) NST
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKHNCS ***',/,/,8X,'Y'''' = 4*Y/X**2 + 2*Y''/X',/)
 1100 FORMAT(6X,'**INPUT **',/)
 1200 FORMAT(8X,'X         =',F9.5,/,/,(8X,'Y(',I2,') =',F9.5,/))
 1300 FORMAT(8X,'M         =',I4,/)
 1400 FORMAT(8X,'XF        =',F9.5,/)
 1500 FORMAT(8X,'ER        =',G12.5,/,/,8X,'EA       =',G12.5,/)
 1600 FORMAT(8X,'NST       =',I4,/)
 1700 FORMAT(6X,'** OUTPUT **',/)
 2000 FORMAT(8X,'IERR      =',I5,/)
 2100 FORMAT(8X,'X         =',F9.5,/,/,&
             7X,'(SOLUTION)',/,/,(8X,'Y(',I2,') =',D19.10,/))
 2200 FORMAT(7X,'(STEP NUMBER OF CALCULATION)',/,/,8X,'NST      =',I5,/)
      END


      SUBROUTINE FKHNCS(X,Y)
      REAL(8) Y(0:*),X
!
      Y(2) = 4.0D0*Y(0)/(X*X)+2.0D0*Y(1)/X
      RETURN
      END
```

(d) Output results

```
 *** DKHNCS ***

    Y'' = 4*Y/X**2 + 2*Y'/X

  **INPUT **

    X         =  1.00000

    Y( 0) =  5.00000

    Y( 1) =  3.00000

    M         =   2

    ER        =  0.0000
```

```
  EA       = 0.10000E-09
  NST      =   0
  XF       =  5.00000
** OUTPUT **
  IERR     =    0
  X        =  5.00000
 (SOLUTION)
  Y( 0) =   0.1000680000D+04
  Y( 1) =   0.7998640000D+03
  Y( 2) =   0.4800544000D+03
 (STEP NUMBER OF CALCULATION)
  NST      =  119
**INPUT **
  XF       = 10.00000
** OUTPUT **
  IERR     =    0
  X        = 10.00000
 (SOLUTION)
  Y( 0) =   0.1600034000D+05
  Y( 1) =   0.6399966000D+04
  Y( 2) =   0.1920006800D+04
 (STEP NUMBER OF CALCULATION)
  NST      =  176
```

2.2.7 DKMNCN, RKMNCN
Ordinary Differential Equation of the Type $M\mathbf{y}'' + C\mathbf{y}' + K\mathbf{y} = \mathbf{p}(x)$

(1) **Function**

DKMNCN or RKMNCN solves a matrix form simultaneous ordinary differential equation of 2nd order initial value problem which is known as the equation of motion:

$$M\boldsymbol{y}'' + C\boldsymbol{y}' + K\boldsymbol{y} = \boldsymbol{p}(x)$$

where M is mass matrix, C is damping matrix, K is stiffness matrix.

(2) **Usage**

Double precision:

```
CALL DKMNCN  (M, C, K, N, PO, P, Y, H, STA, ISW, IWK, WK, IERR)
```

Single precision:

```
CALL RKMNCN  (M, C, K, N, PO, P, Y, H, STA, ISW, 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	M	{D / R}	N,N	Input	Mass matrix M
2	C	{D / R}	N,N	Input	Damping matrix C
3	K	{D / R}	N,N	Input	Stiffness matrix K
4	N	I	1	Input	Number of simultaneous equations n
5	PO	{D / R}	N	Input	External force $\boldsymbol{p}(x_0)$ at $x = x_0$ (A value must be input when the subroutine is called for the first time)
				Output	Updated by value of $\boldsymbol{p}(x_0 + h)$ (Input value when integrating consecutively)
6	P	{D / R}	N	Input	External force $\boldsymbol{p}(x_0 + h)$ at $x = x_0 + h$
7	Y	{D / R}	$\mathrm{N}, 0 : 2$	Input	Initial values of $y_i^{(j)}(i = 1, \cdots, \mathrm{N}; j = 0, 1)$ at $x = x_0$. (Input a value when the subroutine is used for the first time.) When integrating consecutively, input the value of Y(i, j) that was output during the previous iteration.
				Output	Calculated solution $y_i^{(j)}(i = 1, \cdots, \mathrm{N}; j = 0, 1, 2)$ at $x = x_0 + h$ (Input value when integrating successively)
8	H	{D / R}	1	Input	Step size h of x
9	STA	{D / R}	1	Input	Constant θ for obtaining a solution with stability (Default value is 1.4)

No.	Argument	Type	Size	Input/ Output	Contents
10	ISW	I	1	Input	Step size modification switch. Let ISW be zero when the subroutine is used for the first time. When integrating consecutively without changing H and STA, let ISW be the output value when the subroutine was used during the previous iteration. When H or STA is changed, let ISW be 2.
				Output	Next input value for integrating consecutively Without changing H and STA
11	IWK	I	N	Work	Work area
12	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	N × N + N	Work	Work area
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 1

(b) STA ≥ 1.0
(except when 0.0 is entered in order to set STA to the default value)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (b) was not satisfied.	Processing is performed with STA set to the default value.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	The simultaneous linear equations could not be solved. Although it was the first integration, ISW ≠ 0.	

(6) **Notes**

(a) This subroutine calculate solution at $x = x_0 + h$ when the condition at $x = x_0$ for small enough step size h is given. So, if $x_f - x_0$, which is difference between start point x_0 and end point x_f, is large, set step sizes $h_i (i = 1, \cdots, v)$ for which $x_f = x_0 + \sum_{i=1}^{v} h_i$ met, and you should calculate calling v times this subroutine. And if the external force $\boldsymbol{p}(x)$ varies with respect to x, prepare the values of the external force at each point $\boldsymbol{p}(x_0 + \sum_{i=1}^{j} h_i)$ $(j = 1, \cdots, v)$ and they must input for P sequentially.

(b) When integrating for the first time, set ISW=0 and input the external force values $\boldsymbol{p}(x_0)$ and $\boldsymbol{p}(x_0 + h)$ for PO and P, respectively. Where, h is step size.

(c) When integrating consecutively, use the output values of PO and Y directly as the next input values. Also, for ISW, use the value output during the previous iteration directly as then next input value if

the step size H and constant STA do not change, and set ISW=2 if H or STA does change. Also, never overwrite WK.

(d) Although a stable solution is obtained if the value of STA is greater than or equal to 1.37, the error will become quite large if STA is greater than or equal to 2.0. According to Wilson, STA should be around 1.4.

(e) This subroutine uses the Wilson's θ method.

(7) **Example**

(a) Problem

Solve the following a matrix form simultaneous ordinary differential equation of 2nd order:

$$M\boldsymbol{y}'' + C\boldsymbol{y}' + K\boldsymbol{y} = \boldsymbol{p}(x)$$

based on the following initial conditions at $x = x_0$

$$\boldsymbol{y}|_{x=x_0} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ -10.0 \end{bmatrix},\ \boldsymbol{y}'|_{x=x_0} = \begin{bmatrix} 10.0 \\ 0.5 \\ 0.0 \\ 0.0 \end{bmatrix}.$$

Mass matrix M, damping matrix C and stiffness matrix K are given as follows.

$$M = \begin{bmatrix} 20.0 & -1.5 & 0.0 & 0.0 \\ -10.0 & 3.0 & -4.0 & 0.0 \\ 0.0 & -1.5 & 8.0 & 0.0 \\ 0.0 & 0.0 & -4.0 & 0.0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.0 & -15.0 & 0.0 & 0.0 \\ 0.0 & 30.0 & -24.0 & 0.0 \\ 0.0 & -15.0 & 48.0 & -2.0 \\ 0.0 & 0.0 & -24.0 & 4.0 \end{bmatrix}$$

$$K = \begin{bmatrix} 2000.0 & 0.0 & 0.0 & 0.0 \\ -1000.0 & 0.0 & -100.0 & 0.0 \\ 0.0 & 0.0 & 200.0 & -20.0 \\ 0.0 & 0.0 & -100.0 & -40.0 \end{bmatrix}$$

The external force vector $\boldsymbol{p}(x)$ (constant) is given as follows:

$$\boldsymbol{p}(x) = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}.$$

(b) Input data

Mass matrix M, damping matrix C, stiffness matrix K, N=4,
the external force $\boldsymbol{p}(x)$ (set for PO and P),
initial values $\boldsymbol{y}|_{x=x_0}$ and $\boldsymbol{y}'|_{x=x_0}$ at $x = x_0$ (set for array Y),
step size h=0.005 (set for H, constant for each step),
STA=0.0 and ISW=0 (for the first time).
For the input values shown above, output the values at $x = x_0 + 50h$, and $x = x_0 + 100h$, where, $x_0 = 0.0$.

(c) Main program

```
      PROGRAM BKMNCN
! *** EXAMPLE OF BKMNCN ***
      REAL(8) M,C,K,PO,P,Y,WK,H,STA,HTMP
      DIMENSION M(4,4),C(4,4),K(4,4),PO(4),P(4),Y(4,0:2),WK(20)
      INTEGER IWK(4)
      PARAMETER(N = 4)
!
      ISW = 0
      READ(*,*) ((M(I,J),J=1,N),I=1,N)
      READ(*,*) ((C(I,J),J=1,N),I=1,N)
      READ(*,*) ((K(I,J),J=1,N),I=1,N)
      READ(*,*) (PO(I),I=1,N)
      READ(*,*) (P(I),I=1,N)
      READ(*,*) ((Y(I,J),J=0,1),I=1,N)
      READ(*,*) H,STA
      HTMP = H
      WRITE(6,1000) ((M(I,J),J=1,N),I=1,N)
      WRITE(6,1001) ((C(I,J),J=1,N),I=1,N)
      WRITE(6,1002) ((K(I,J),J=1,N),I=1,N)
      WRITE(6,1003) N,(PO(I),I=1,N)
      WRITE(6,1004) (P(I),I=1,N)
      WRITE(6,1005) ((Y(I,J),J=0,1),I=1,N)
      WRITE(6,1006) H,STA,ISW
      WRITE(6,1007)
      DO 10 I=1,100
         CALL DKMNCN(M,C,K,N,PO,P,Y,H,STA,ISW,IWK,WK,IERR)
         IF(MOD(I,50).EQ.0)THEN
            WRITE(6,2000) IERR
            WRITE(6,2100) HTMP,((Y(J,L),L=0,2),J=1,N)
         ENDIF
         HTMP = HTMP+H
   10 CONTINUE
      STOP
 1000 FORMAT(' ',/,5X,'*** DKMNCN ***',/,&
            /,8X,'M*Y'''+C*Y''+K*Y = P(X)',/,&
            /,6X,'** INPUT **',/,&
            /,8X,'M         = I',4F8.1,' I',/,&
           (18X,' I',4F8.1,' I'))
 1001 FORMAT(/,8X,'C         = I',4F8.1,' I',/,(18X,' I',4F8.1,' I'))
 1002 FORMAT(/,8X,'K         = I',4F8.1,' I',/,(18X,' I',4F8.1,' I'))
 1003 FORMAT(/,8X,'N         =',I4,/,/,8X,'PO(T)    = I',4F8.1,' I',/)
 1004 FORMAT(8X,'P (T)    = I',4F8.1,' I',/)
 1005 FORMAT(8X,'Y(I,J)   = I',2F8.1,' I',/,(18X,' I',2F8.1,' I'))
 1006 FORMAT(/,8X,'H         =',F7.3,/,&
            /,8X,'STA       =',F7.3,/,&
            /,8X,'ISW       =',I4,/)
 1007 FORMAT(6X,'** OUTPUT **')
 2100 FORMAT(/,8X,'X         =',F7.3,/,&
            /,8X,'Y(I,J)    = I',3D18.10,' I',/,&
           (18X,' I',3D18.10,' I'))
 2000 FORMAT(/,8X,'IERR      =',I5)
      END
```

(d) Output results

```
 *** DKMNCN ***

    M*Y''+C*Y'+K*Y = P(X)

  ** INPUT **

    M         = I    20.0    -1.5     0.0     0.0 I
                I   -10.0     3.0    -4.0     0.0 I
                I     0.0    -1.5     8.0     0.0 I
                I     0.0     0.0    -4.0     0.0 I

    C         = I     0.0   -15.0     0.0     0.0 I
                I     0.0    30.0   -24.0     0.0 I
                I     0.0   -15.0    48.0    -2.0 I
                I     0.0     0.0   -24.0     4.0 I

    K         = I  2000.0     0.0     0.0     0.0 I
                I -1000.0     0.0  -100.0     0.0 I
                I     0.0     0.0   200.0   -20.0 I
                I     0.0     0.0  -100.0    40.0 I

    N         =   4

    PO(T)     = I     0.0     0.0     0.0     0.0 I

    P (T)     = I     0.0     0.0     0.0     0.0 I

    Y(I,J)    = I     0.0    10.0 I
                I     0.0     0.5 I
                I     0.0     0.0 I
                I   -10.0     0.0 I

    H         =  0.005

    STA       =  0.000

    ISW       =   0
```

```
** OUTPUT **

  IERR     =    0

  X        =  0.250

  Y(I,J)   = I  0.5996355514D+00 -0.8004496070D+01 -0.5991999087D+02 I
             I  0.4612429593D-01  0.4118136346D-01 -0.4121159414D+00 I
             I  0.4058631976D-17  0.2007807635D-15  0.5098358437D-14 I
             I -0.8255154355D+00  0.8255064959D+01 -0.8253256530D+02 I

  IERR     =    0

  X        =  0.500

  Y(I,J)   = I -0.9599492257D+00  0.2814828914D+01  0.9592727423D+02 I
             I  0.4990162928D-01  0.3372412684D-02 -0.3374888325D-01 I
             I  0.1902745623D-16 -0.2099899040D-15  0.1267992670D-13 I
             I -0.6776426206D-01  0.6776352823D+00 -0.6774868334D+01 I
```

2.3 ORDINARY DIFFERENTIAL EQUATIONS (BOUNDARY VALUE PROBLEMS)

2.3.1 DOSNNV, ROSNNV
High-Order Simultaneous Ordinary Differential Equations (Numerical Boundary Conditions)

(1) **Function**

DOSNNV or ROSNNV uses the multipoint shooting method to solve a boundary value problem for high-order simultaneous ordinary differential equations for which boundary conditions are given as input values.

(2) **Usage**

Double precision:

CALL DOSNNV (F, XA, XB, IN, IB, IC, BN, M, N, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROSNNV (F, XA, XB, IN, IB, IC, BN, M, N, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, N, ALF) that defines the differential equations as functions of x and y.
2	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
3	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
4	IN	I	See Contents	Input	Positions where boundary conditions are set (XA side:0, XB side: Nonzero) **Size:** $\sum_{i=1}^{N} M(i)$
5	IB	I	See Contents	Input	Element numbers i of variable $y_i^{(j)}$ which gives boundary conditions value **Size:** $\sum_{i=1}^{N} M(i)$

No.	Argument	Type	Size	Input/ Output	Contents
6	IC	I	See Contents	Input	Differential order j of variable $y_i^{(j)}$ which gives boundary conditions value **Size:** $\sum_{i=1}^{N} M(i)$
7	BN	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	See Contents	Inputs	Boundary condition values given for variable $y_i^{(j)}$ **Size:** $\sum_{i=1}^{N} M(i)$
8	M	I	N	Input	Maximum differential order of variable y_i in differential equations
9	N	I	1	Input	Number of simultaneous equations.
10	X	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	K	Input	x coordinate x_i where approximate solutions are calculated
11	K	I	1	Input	Number of calculation points
12	ER	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
13	EA	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Unit for determining error)
14	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of shooting points
15	NEV	I	1	Input	Maximum number of iterations (Default value :100)
				Output	Actual number of iterations
16	Y	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	See Contents	Output	Solutions $y_j^{(v)}(x_i)$ $Y(i,j,v) = y_j^{(v)}(x_i)$ **Size:** $K, N, 0 : MAX(M(i))$
17	ISW	I	1	Input	Parameterization processing switch 0: Parameterization not performed Nonzero: Parameterization performed
18	IWK	I	See Contents	Work	Work area **Size**: $\sum_{i=1}^{N} M(i)$
19	WK	$\left\{\begin{matrix}D\\R\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(N \times KM + 1)^2 \times (NX + 1) + N \times KM \times MAX(3 \times N \times KM, KM + 15)$ Where, $KM = MAX(M(i))$
20	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) XA < XB

(b) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) NEV > 0 (Except when 0 is entered in order to set NEV to the default value)

(e) N $\geq$ 1

(f) M(i) $\geq$ 1 (i $= 1, 2, \cdots,$ N)

(g) 1 $\leq$ IB(i) $\leq$ N and 0 $\leq$ IC(i) < M(IB(i)) (i $= 1, 2, \cdots, \sum_{j=1}^{N}$ M(j))

(h) K $\geq$ 1

(i) XA $\leq$ X(i) $\leq$ XB (i $= 1, 2, \cdots,$ K)

(j) NX $\geq \min(5i + 1, 51)$
where i is the minimum integer for which XB $-$ XA $\leq$ i

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA $\neq$ XB).	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with the corresponding default value set.
3000	XA=XB	Processing is aborted.
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
3050	Restriction (i) was not satisfied.	
3060	Restriction (j) was not satisfied.	
4000	The shooting point could not be added due to server oscillation (large derivative), etc.	
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc).	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for high-order simultaneous ordinary differential equations expressed as follows:

$$\begin{cases} y_1^{(m_1)} &= f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots \qquad\qquad\qquad\qquad \vdots \\ y_i^{(m_i)} &= f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots \qquad\qquad\qquad\qquad \vdots \\ y_n^{(m_n)} &= f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{cases}$$

(Where, if left-hand side is $y_i^{(m_i)}$, then differential order j of corresponding right-hand side $y_i^{(j)}$ must satisfy $j \leq m_i - 1$.) if there is a multiplication or division of two or more of the variables $y_i^{(j)}$ in the terms on the right-hand side of any of these equations such as $y_1 \cdot y_1'$, $\frac{y_2}{y_3'}$, y_4^2 or a function other than a sum or scalar multiple of $y_i^{(j)}$ such as $\sin(y_1)$ or $\mathrm{abs}(y_2')$, parameterize this nonlinear problem by multiplying this portion by ALF.

Example :

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

Parameterize this nonlinear problem as follows:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

When the problem has been parameterized, set ISW = 1. A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, N, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, N, ALF) (in double-precision) for the high-order simultaneous ordinary differential equations that were parameterized as described in Note (a) should be created as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:∗)
$\mathrm{Y}(1, \mathrm{M}(1)) = f_1(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
⋮
$\mathrm{Y}(\mathrm{i}, \mathrm{M}(\mathrm{i})) = f_i(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
⋮
$\mathrm{Y}(\mathrm{N}, \mathrm{M}(\mathrm{N})) = f_n(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y}(\mathrm{i}, 0)$, $y_i^{(j)} \leftrightarrow \mathrm{Y}(\mathrm{i}, \mathrm{j})$
$f_i(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f_i(x, \cdots)$.
Example:

When parameterized ordinary differential equations are as follows:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\text{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

define the subroutine as follows:

```
SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:*)
Y(1, 2)=Y(2, 0)
Y(2,1) = Y(1,1) − ALF ∗ (Y(1,0) ∗ Y(2,0))
RETURN
END
```

The Input arguments: N=2, M(1)=2, M(2)=1, ISW=1.

(c) If the boundary conditions of the high-order simultaneous ordinary differential equations are given by:

$$\text{left-hand side boundary} \quad \begin{cases} y_{a_1}^{(b_1)} = c_1 \\ y_{a_2}^{(b_2)} = c_2 \\ \quad \vdots \\ y_{a_p}^{(b_p)} = c_p \end{cases}$$

$$\text{right-hand side boundary} \quad \begin{cases} y_{a_{p+1}}^{(b_{p+1})} = c_{p+1} \\ y_{a_{p+2}}^{(b_{p+2})} = c_{p+2} \\ \quad \vdots \\ y_{a_q}^{(b_q)} = c_q \end{cases} \qquad q = \sum_{i=1}^{n} m_i$$

set array IN, IB, IC, and BN as follows: IN(i) = 0 (i = 1, 2, ⋯, p)
IN(i) = 1 (i = p + 1, p + 2, ⋯, q)
IB(i) = a_i(i = 1, 2, ⋯, q)
IC(i) = b_i(i = 1, 2, ⋯, q)
BN(i) = c_i(i = 1, 2, ⋯, q)

Example: If the boundary conditions are given by:

$$\text{left-hand side boundary} \quad \begin{cases} y_1' = 0.0 \\ y_2 = 1.0 \end{cases} \quad \text{right-hand side boundary} \quad y_1' = 2.0$$

the input values of IN, IB, IC, and BN are as follows:
IN(1)=0, IB(1)=1, IC(1)=1, BN(1)=0.0
IN(2)=0, IB(2)=2, IC(2)=0, BN(2)=1.0
IN(3)=1, IB(3)=1, IC(3)=1, BN(3)=2.0

(7) **Example**

(a) Solve the following simultaneous ordinary differential equations:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

under the following boundary conditions:

$$y_1'|_{x=0.0} = 0.0, y_1|_{x=0.0} = 0.486, y_1'|_{x=3.0} = 2.0$$

(b) Input data

Name of subroutine F(X, Y, N, ALF): FOSNNV, N=2, XA=0.0, XB=3.0,
IN(1)=0, IB(1)=1, IC(1)=1, BN(1)=0.0,
IN(2)=0, IB(2)=1, IC(2)=0, BN(2)=0.486,
IN(3)=1, IB(3)=1, IC(3)=1, BN(3)=2.0,
M(1)=2, M(2)=1, X, K=3, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOSNNV
! *** EXAMPLE OF DOSNNV
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOSNNV
      PARAMETER (N=2,L=3)
      DIMENSION IN(3),IB(3),IC(3),BN(3),M(N),X(L),Y(L,N,0:2),IWK(10),&
                WK(1000)
!
      READ(5,*) XA,XB
      READ(5,*) (IN(I),I=1,3,1)
      READ(5,*) (IB(I),I=1,3,1)
      READ(5,*) (IC(I),I=1,3,1)
      READ(5,*) (BN(I),I=1,3,1)
      READ(5,*) (M(I),I=1,N,1)
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100) XA,XB
      WRITE(6,1200)
      WRITE(6,1300) (IN(I),IB(I),IC(I),BN(I),I=1,3)
      WRITE(6,1400)
      WRITE(6,1500) (I,M(I),I=1,N)
      WRITE(6,1600) N
      WRITE(6,1700)
      WRITE(6,1800) (I,X(I),I=1,L)
      WRITE(6,1900) L
      WRITE(6,2000) ER
      WRITE(6,2100) EA
      WRITE(6,2200) NX
      WRITE(6,2300) NEV
      WRITE(6,2400) ISW
      CALL DOSNNV(FOSNNV,XA,XB,IN,IB,IC,BN,M,N,X,L,ER,EA,NX,NEV,Y,&
                  ISW,IWK,WK,IERR)
      WRITE(6,2500)
      WRITE(6,2600) IERR
      WRITE(6,2700) NX
      WRITE(6,2800) NEV
      WRITE(6,2900)
      WRITE(6,3000) (((I,J,K,Y(I,J,K),K=0,M(J)),J=1,N),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOSNNV ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *',/,&
      6X,'Y1'''' = Y2',/,&
      6X,'Y2''  = Y1''-Y1*Y2',/,&
      3X,'** INPUT **')
 1100 FORMAT(6X,'INTERVAL = ( ',F4.1,', ',F4.1,'  )')
 1200 FORMAT(6X,'(BOUNDARY CONDITION)',/,&
      6X,'POSITION',5X,'INDEX OF Y',5X,'ORDER OF Y',8X,'VALUE')
 1300 FORMAT(9X,I2,12X,I2,13X,I2,9X,F11.8)
 1400 FORMAT(6X,'(ORDER OF EACH DIFFERENTIAL EQUATIONS)')
 1500 FORMAT(6X,'M(',I2,') = ',I2)
 1600 FORMAT(6X,'SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS = ',&
      I2)
 1700 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1800 FORMAT(6X,'X(',I2,')= ',F4.1)
 1900 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 2000 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 2100 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 2200 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
```

```
 2300 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2400 FORMAT(6X,'ISW = ',I2)
 2500 FORMAT(3X,'** OUTPUT **')
 2600 FORMAT(6X,'IERR = ',I4)
 2700 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2800 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2900 FORMAT(6X,'(APPROXIMATE VALUES)')
 3000 FORMAT(6X,'Y(',I2,',',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOSNNV(X,Y,N,ALF)
      REAL(8) X,Y,ALF
      DIMENSION Y(N,0:*)
!
      Y(1,2)=Y(2,0)
      Y(2,1)=Y(1,1)-ALF*(Y(1,0)*Y(2,0))
      RETURN
      END
```

(d) Output results

```
*** DOSNNV ***
  * SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *
    Y1'' = Y2
    Y2'  = Y1'-Y1*Y2
 ** INPUT **
    INTERVAL = (  0.0,  3.0  )
    (BOUNDARY CONDITION)
    POSITION     INDEX OF Y     ORDER OF Y        VALUE
        0             1              1          1.00000000
        0             1              0          0.48600000
        1             1              1          2.00000000
    (ORDER OF EACH DIFFERENTIAL EQUATIONS)
    M( 1) =  2
    M( 2) =  1
    SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS =  2
    (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
    X( 1)=  0.0
    X( 2)=  1.5
    X( 3)=  3.0
    NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   3
    REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
    REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
    MAXIMUM NUMBER OF SHOOTING POINTS =  50
    MAXIMUM ITERATIVE NUMBER =   0
    ISW =  1
 ** OUTPUT **
    IERR =    0
    PRACTICAL NUMBER OF SHOOTING POINTS =  19
    PRACTICAL ITERATIVE NUMBER =  19
    (APPROXIMATE VALUES)
    Y( 1, 1, 0) =  0.4860000000D+00
    Y( 1, 1, 1) =  0.1000000000D+01
    Y( 1, 1, 2) = -0.5284077691D+00
    Y( 1, 2, 0) = -0.5284077691D+00
    Y( 1, 2, 1) =  0.1256806176D+01
    Y( 2, 1, 0) =  0.1938971346D+01
    Y( 2, 1, 1) =  0.1194711339D+01
    Y( 2, 1, 2) =  0.5134697904D+00
    Y( 2, 2, 0) =  0.5134697904D+00
    Y( 2, 2, 1) =  0.1991081286D+00
    Y( 3, 1, 0) =  0.4343763169D+01
    Y( 3, 1, 1) =  0.2000000000D+01
    Y( 3, 1, 2) =  0.4858967029D+00
    Y( 3, 2, 0) =  0.4858967029D+00
    Y( 3, 2, 1) = -0.1106202018D+00
```

2.3.2 DOSNNF, ROSNNF
High-Order Simultaneous Ordinary Differential Equations (Function Boundary Conditions)

(1) **Function**

DOSNNF or ROSNNF uses the multipoint shooting method to solve a boundary value problem for high-order simultaneous ordinary differential equations for which boundary conditions are given as functions.

(2) **Usage**

Double precision:

CALL DOSNNF (F, FB, XA, XB, M, N, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROSNNF (F, FB, XA, XB, M, N, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, N, ALF) that defines the differential equations as functions of x and y.
2	FB	–	–	Input	Name of subroutine FB(YA, YB, N, G) that defines boundary conditions.
3	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
4	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
5	M	I	N	Input	Differential order of the left-hand side of each of the simultaneous equations
6	N	I	1	Input	Number of simultaneous equations
7	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
8	K	I	1	Input	Number of calculation points
9	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}

No.	Argument	Type	Size	Input/ Output	Contents
10	EA	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Unit for determining error)
11	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of iterations
12	NEV	I	1	Input	Maximum number of iterations (Default value:100)
				Output	Actual number of iterations
13	Y	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Output	Solutions $y_j^{(v)}(x_i)$ $\text{Y(i, j, v)} = y_j^{(v)}(x_i)$ **Size:** $\text{K, N, 0 : MAX(M(i))}$
14	ISW	I	1	Input	Parameterization processing switch 0: Parameterization not performed Nonzero: Parameterization performed
15	IWK	I	See Contents	Work	Work area **Size**: $\sum_{\text{i=1}}^{\text{N}} \text{M(i)}$
16	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**:$(\text{N} \times \text{KM} + 1)^2 \times (\text{NX} + 1) + \text{N} \times \text{KM} \times \text{MAX}(3 \times \text{N} \times \text{KM}, \text{KM} + 15)$ where $\text{KM} = \text{MAX(M(i))}$
17	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\text{XA} < \text{XB}$

(b) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) $\text{NEV} > 0$ (Except when 0.0 is entered in order to set NEV to the default value)

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

(f) $\text{M(i)} \geq 1$ $(\text{i} = 1, 2, \cdots, \text{N})$

(g) $\text{K} \geq 1$

(h) $\text{XA} \leq \text{X(i)} \leq \text{XB}$ $(\text{i} = 1, 2, \cdots, \text{K})$

(i) $\text{NX} \geq \min(5\text{i} + 1, 51)$
Where, i is minimum integer for which $\text{XB} - \text{XA} \leq \text{i}$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA ≠ XB)	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with the corresponding default value set.
3000	XA = XB	Processing is aborted.
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
3050	Restriction (i) was not satisfied.	
4000	The shooting point could not be added due to severe oscillation (large derivative).	
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc).	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for high-order simultaneous ordinary differential equations expressed as follows:

$$\begin{cases} y_1^{(m_1)} &= f_1(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots \qquad\qquad \vdots \\ y_i^{(m_i)} &= f_i(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \\ \vdots & \vdots \qquad\qquad \vdots \\ y_n^{(m_n)} &= f_n(x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots, y_n^{(m_n-1)}) \end{cases}$$

(Where, if left-hand side is $y_i^{(m_i)}$, then differential order j of corresponding right-hand side $y_i^{(j)}$ must satisfy $j \leq m_i - 1$.) if there is a multiplication or division of two or more of the variables $y_i^{(j)}$ in the terms on the right-hand side of any of these equations such as $y_1 \cdot y_1'$, $\frac{y_2}{y_3'}$, y_4^2 or a function other than a sum or scalar multiple of $y_i^{(j)}$ such as $\sin(y_1)$ or $\mathrm{abs}(y_2')$, parameterize this nonlinear problem by multiplying this portion by ALF.

Example :

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

Parameterize this nonlinear problem as follows:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

When the problem has been parameterized, set ISW = 1. A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, N, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, N, ALF) (in double-precision) for the high-order simultaneous ordinary differential equations that were parameterized as described in Note (a) should be created as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:∗)
$\mathrm{Y}(1, \mathrm{M}(1)) = f_1(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
$\vdots$
$\mathrm{Y}(\mathrm{i}, \mathrm{M}(\mathrm{i})) = f_i(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
$\vdots$
$\mathrm{Y}(\mathrm{N}, \mathrm{M}(\mathrm{N})) = f_n(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_i^{(j)}, \cdots, y_i^{(m_i-1)}, \cdots)$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y}(\mathrm{i}, 0)$, $y_i^{(j)} \leftrightarrow \mathrm{Y}(\mathrm{i}, \mathrm{j})$
$f_i(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f_i(x, \cdots)$.
Example:
When parameterized ordinary differential equations are as follows:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

define the subroutine as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:∗)
Y(1, 2)=Y(2, 0)
$\mathrm{Y}(2, 1) = \mathrm{Y}(1, 1) - \mathrm{ALF} * (\mathrm{Y}(1, 0) * \mathrm{Y}(2, 0))$
RETURN
END

The Input arguments: N=2, M(1)=2, M(2)=1, ISW=1.

(c) The actual name of subroutine FB(YA, YB, N, G), that defines the boundary conditions, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

When q boundary conditions are given as follows using values $y_i^{(j)}(a)$ of $y_i^{(j)}$ at left-hand side boundary

$x = a$ and values $y_i^{(j)}(b)$ of $y_i^{(j)}$ at right–hand side boundary $x = b$:

$$\begin{cases} g_1(y_1(a), \cdots, y_i^{(j)}(a), \cdots, y_n^{(m_n-1)}(a), y_1(b), \cdots, y_i^{(j)}(b), \cdots, y_n^{(m_n-1)}(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_v(y_1(a), \cdots, y_i^{(j)}(a), \cdots, y_n^{(m_n-1)}(a), y_1(b), \cdots, y_i^{(j)}(b), \cdots, y_n^{(m_n-1)}(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_q(y_1(a), \cdots, y_i^{(j)}(a), \cdots, y_n^{(m_n-1)}(a), y_1(b), \cdots, y_i^{(j)}(b), \cdots, y_n^{(m_n-1)}(b)) = 0 \end{cases}$$

the subroutine FB(YA, YB, N, G) (in double-precision) should be created as follows:

```
SUBROUTINE FB(YA, YB, N, G)
REAL(8) YA, YB, G
DIMENSION YA(N, 0:*), YB(N, 0:*), G(*)
G(1) = g1(y1(a), ···, yi^(j)(a), ···, yn^(mn−1)(a), y1(b), ···, yi^(j)(b), ···, yn^(mn−1)(b))
G(2) = g2(y1(a), ···, yi^(j)(a), ···, yn^(mn−1)(a), y1(b), ···, yi^(j)(b), ···, yn^(mn−1)(b))
  ⋮
G(q) = gq(y1(a), ···, yi^(j)(a), ···, yn^(mn−1)(a), y1(b), ···, yi^(j)(b), ···, yn^(mn−1)(b))
RETURN
END
```

where the following correspondences are assumed:
$n \leftrightarrow$N, $y_i(a) \leftrightarrow$ YA(i, 0), $y_i^{(j)}(a) \leftrightarrow$ YA(i, j)
$y_i(b) \leftrightarrow$ YB(i, 0), $y_i^{(j)}(b) \leftrightarrow$ YB(i, j)

Example:

When boundary conditions are given as follows:

$$\begin{cases} y_1(a) - y_2(b) = 0.0 \\ y_1'(a) = 1.0 \\ y_1'(b) = 2.0 \end{cases}$$

define the subroutine as follows:

```
SUBROUTINE FB(YA, YB, N, G)
REAL(8) YA, YB, G
DIMENSION YA(N, 0:*), YB(N, 0:*), G(*)
G(1) = YA(1, 0) − YB(2, 0)
G(2) = YA(1, 1) − 1.0D0
G(3) = YB(1, 1) − 2.0D0
RETURN
END
```

(7) **Example**

(a) Problem

Solve the following simultaneous ordinary differential equations:

$$\begin{cases} y_1'' = y_2 \\ y_2' = y_1' - y_1 \cdot y_2 \end{cases}$$

under the following boundary conditions:

$$\begin{cases} y_1(0.0) - y_2(3.0) = 0.0 \\ y_1'(0.0) = 1.0 \\ y_1'(3.0) = 2.0 \end{cases}$$

(b) Input data

Name of subroutine F(X, Y, N, ALF): FOSNNF, name of subroutine FB(YA, YB, N, G): GOSNNF, N=2, XA=0.0, XB=3.0,

M(1)=2, M(2)=1, X, K=3, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOSNNF
! *** EXAMPLE OF DOSNNF
      IMPLICIT INTEGER (I-N)
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOSNNF,FOSNN2
      PARAMETER (N=2,L=3)
      DIMENSION M(N),X(L),Y(L,N,0:2),IWK(10),WK(1400)
!
      READ(5,*) XA,XB
      READ(5,*) (M(I),I=1,N,1)
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) XA,XB
      WRITE(6,1300)
      WRITE(6,1400) (I,M(I),I=1,N)
      WRITE(6,1500) N
      WRITE(6,1600)
      WRITE(6,1700) (I,X(I),I=1,L)
      WRITE(6,1800) L
      WRITE(6,1900) ER
      WRITE(6,2000) EA
      WRITE(6,2100) NX
      WRITE(6,2200) NEV
      WRITE(6,2300) ISW
      CALL DOSNNF(FOSNNF,FOSNN2,XA,XB,M,N,X,L,ER,EA,NX,NEV,Y,ISW,IWK,&
                  WK,IERR)
      WRITE(6,2400)
      WRITE(6,2500) IERR
      WRITE(6,2600) NX
      WRITE(6,2700) NEV
      WRITE(6,2800)
      WRITE(6,2900) (((I,J,K,Y(I,J,K),K=0,M(J)),J=1,N),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOSNNF ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *',/,&
      6X,'Y1'''' = Y2',/,&
      6X,'Y2''  = Y1''-Y1*Y2')
 1100 FORMAT(4X,'* BOUNDARY CONDITIONS *',/,&
      6X,'YA1-YB2 = 0.0',/,&
      6X,'YA1''    = 1.0    (YA=Y(X=0.0),YB=Y(X=3.0))',/,&
      6X,'YB1''    = 2.0',/,&
      3X,'** INPUT **')
 1200 FORMAT(6X,'INTERVAL = ( ',F11.8,', ',F11.8,' )')
 1300 FORMAT(6X,'(ORDER OF EACH DIFFERENTIAL EQUATIONS)')
 1400 FORMAT(6X,'M(',I2,') = ',I2)
 1500 FORMAT(6X,'SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS = ',&
      I2)
 1600 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1700 FORMAT(6X,'X(',I2,')= ',F11.8)
 1800 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1900 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 2000 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 2100 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
 2200 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2300 FORMAT(6X,'ISW = ',I2)
```

```
 2400 FORMAT(3X,'** OUTPUT **')
 2500 FORMAT(6X,'IERR = ',I4)
 2600 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2700 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2800 FORMAT(6X,'(APPROXIMATE VALUES)')
 2900 FORMAT(6X,'Y(',I2,',',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOSNNF(X,Y,N,ALF)
      REAL(8) X,Y,ALF
      DIMENSION Y(N,0:*)
!
      Y(1,2)=Y(2,0)
      Y(2,1)=Y(1,1)-ALF*(Y(1,0)*Y(2,0))
      RETURN
      END


      SUBROUTINE FOSNN2(YA,YB,N,G)
      REAL(8) YA,YB,G
      DIMENSION YA(N,0:*),YB(N,0:*),G(*)
!
      G(1)=YA(1,0)-YB(2,0)
      G(2)=YA(1,1)-1.0D00
      G(3)=YB(1,1)-2.0D00
      RETURN
      END
```

(d) Output results

```
 *** DOSNNF ***
   * SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *
     Y1'' = Y2
     Y2'  = Y1'-Y1*Y2
   * BOUNDARY CONDITIONS *
     YA1-YB2 = 0.0
     YA1'    = 1.0    (YA=Y(X=0.0),YB=Y(X=3.0))
     YB1'    = 2.0
  ** INPUT **
     INTERVAL = (  0.00000000,  3.00000000 )
     (ORDER OF EACH DIFFERENTIAL EQUATIONS)
     M( 1) =  2
     M( 2) =  1
     SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS =  2
     (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
     X( 1)=  0.00000000
     X( 2)=  1.50000000
     X( 3)=  3.00000000
     NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   3
     REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
     REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
     MAXIMUM NUMBER OF SHOOTING POINTS =  50
     MAXIMUM ITERATIVE NUMBER =   0
     ISW =  1
  ** OUTPUT **
     IERR =    0
     PRACTICAL NUMBER OF SHOOTING POINTS =  19
     PRACTICAL ITERATIVE NUMBER =  19
     (APPROXIMATE VALUES)
     Y( 1, 1, 0) =  0.4859121785D+00
     Y( 1, 1, 1) =  0.1000000000D+01
     Y( 1, 1, 2) = -0.5284245597D+00
     Y( 1, 2, 0) = -0.5284245597D+00
     Y( 1, 2, 1) =  0.1256767929D+01
     Y( 2, 1, 0) =  0.1938862632D+01
     Y( 2, 1, 1) =  0.1194690129D+01
     Y( 2, 1, 2) =  0.5134736993D+00
     Y( 2, 2, 0) =  0.5134736993D+00
     Y( 2, 2, 1) =  0.1991351608D+00
     Y( 3, 1, 0) =  0.4343636569D+01
     Y( 3, 1, 1) =  0.2000000000D+01
     Y( 3, 1, 2) =  0.4859121785D+00
     Y( 3, 2, 0) =  0.4859121785D+00
     Y( 3, 2, 1) = -0.1106259080D+00
```

2.3.3 DOFNNV, ROFNNV
First-Order Simultaneous Ordinary Differential Equations (Numerical Boundary Conditions)

(1) **Function**

DOFNNV or ROFNNV uses the multipoint shooting method to solve a boundary value problem for first-order simultaneous ordinary differential equations for which boundary conditions are given as input values.

(2) **Usage**

Double precision:

CALL DOFNNV (F, XA, XB, IN, IB, BN, N, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROFNNV (F, XA, XB, IN, IB, BN, N, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, N, ALF) that defines the differential equations as a function of x and y.
2	XA	{ D / R }	1	Input	x coordinate of left-hand side boundary
3	XB	{ D / R }	1	Input	x coordinate of right-hand side boundary
4	IN	I	N	Input	Positions where boundary conditions are set (XA side:0, XB side: Nonzero)
5	IB	I	N	Input	Element numbers i of y_i which gives boundary conditions value
6	BN	{ D / R }	N	Input	Boundary condition values given for variable y_i.

No.	Argument	Type	Size	Input/ Output	Contents
7	N	I	1	Input	Number of simultaneous equations
8	X	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	K	Input	x coordinate x_i where approximate solutions are calculated
9	K	I	1	Input	Number of calculation points
10	ER	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
11	EA	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Unit for determining error)
12	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of shooting points
13	NEV	I	1	Input	Maximum number of iterations (Default value: 100)
				Output	Actual number of iterations
14	Y	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Solutions $y_j^{(v)}(x_i)$ $\mathrm{Y(i,j,v)} = y_j^{(v)}(x_i)$ **Size:** $\mathrm{K,N,0:1}$
15	ISW	I	1	Input	Parameterization processing switch 0: Parameterization not performed Nonzero: Parameterization performed
16	IWK	I	N	Work	Work area
17	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(\mathrm{N}+1)^2\times(\mathrm{NX}+1)+\mathrm{N}\times\mathrm{MAX}(2\times\mathrm{N},17)$
18	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{XA} < \mathrm{XB}$

(b) ER$\geq e_r$. where e_r= double precision:10^{-14}, single precision: 10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) NEV > 0 (Except when 0 is entered in order to set NEV to the default value)

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

(f) $1 \leq \mathrm{IB(i)} \leq \mathrm{N}$ $(\mathrm{i} = 1, 2, \cdots, \sum_{\mathrm{j}=1}^{\mathrm{N}} \mathrm{M(j)})$

(g) $\mathrm{K} \geq 1$

(h) $\mathrm{XA} \leq \mathrm{X(i)} \leq \mathrm{XB}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{K})$

(i) $\mathrm{NX} \geq \min(5\mathrm{i}+1, 51)$
where i is the minimum integer for which $\mathrm{XB} - \mathrm{XA} \leq \mathrm{i}$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA $\neq$ XB)	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is aborted.
3000	XA=XB	
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
3050	Restriction (i) was not satisfied.	
4000	The shooting point could not be added due to severe oscillation (large derivative), etc.	Processing is aborted.
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc).	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for first-order simultaneous ordinary differential equations expressed as follows:

$$\begin{cases} y_1' &= f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \cdots, y_n) \\ \vdots & \vdots \qquad\qquad \vdots \\ y_n' &= f_n(x, y_1, y_2, \cdots, y_n) \end{cases}$$

if there is a multiplication or division of two or more of the variables y_i in the terms on the right-hand side of any of these equations such as $y_1 \cdot y_2$, $\frac{y_2}{y_3}$ or y_4^2 or a function other than a sum or scalar multiple of y_i such as $\sin(y_1)$ or $\mathrm{abs}(y_2)$, parameterize this nonlinear problem by multiplying this portion by ALF.

Example :

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - y_1 \cdot y_2 \end{cases}$$

Parameterize this nonlinear problem as follows:

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

When the problem has been parameterized, set ISW = 1.

A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, N, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, N, ALF) (in double-precision) for the first-order simultaneous ordinary differential equations that were parameterized as described in Note (a) should be created as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:1)
$\mathrm{Y}(1,1) = f_1(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
$\vdots$
$\mathrm{Y}(\mathrm{i},1) = f_i(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
$\vdots$
$\mathrm{Y}(\mathrm{N},1) = f_n(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y(i,0)}$
$f_i(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f_i(x, \cdots)$.
Example:
When parameterized ordinary differential equations are as follows:

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

define the subroutine as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:1)
Y(1, 1)=Y(2, 0)
$\mathrm{Y}(2,1) = \mathrm{Y}(1,0) - \mathrm{ALF} * (\mathrm{Y}(1,0) * \mathrm{Y}(2,0))$
RETURN
END

The Input arguments: N=2, ISW=1.

(c) If the boundary conditions of the first-order simultaneous ordinary differential equations are given by:

$$\text{left-hand side boundary} \quad \begin{cases} y_{a_1} = c_1 \\ y_{a_2} = c_2 \\ \quad \vdots \\ y_{a_p} = c_p \end{cases} \qquad \text{right-hand side boundary} \quad \begin{cases} y_{a_{p+1}} = c_{p+1} \\ y_{a_{p+2}} = c_{p+2} \\ \quad \vdots \\ y_{a_n} = c_n \end{cases}$$

set array IN, IB and BN as follows: $\mathrm{IN(i)} = 0 \ (\mathrm{i} = 1, 2, \cdots, \mathrm{p})$
$\mathrm{IN(i)} = 1 \ (\mathrm{i} = \mathrm{p}+1, \mathrm{p}+2, \cdots, \mathrm{n})$
$\mathrm{IB(i)} = \mathrm{a_i} (\mathrm{i} = 1, 2, \cdots, \mathrm{n})$
$\mathrm{BN(i)} = \mathrm{c_i} (\mathrm{i} = 1, 2, \cdots, \mathrm{n})$
Example: If the boundary conditions are given by:

left-hand side boundary $y_1 = 1.0$ right-hand side boundary $y_2 = 2.0$

the input values of IN, IB and BN are as follows:
IN(1)=0, IB(1)=1, BN(1)=1.0
IN(2)=1, IB(2)=2, BN(2)=2.0

(7) **Example**

(a) Problem
Solve the following simultaneous ordinary differential equations:

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1' - y_1 \cdot y_2 \end{cases}$$

under the following boundary conditions:

$$y_2|_{x=0.0} = 1.0, y_2|_{x=1.0} = 2.0$$

(b) Input data
Name of subroutine F(X, Y, N, ALF): FOFNNV, N=2, XA=0.0, XB=1.0,
IN(1)=0, IB(1)=2, BN(1)=1.0,
IN(2)=1, IB(2)=2, BN(2)=2.0,
X, K=3, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOFNNV
! *** EXAMPLE OF DOFNNV
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOFNNV
      PARAMETER (N=2,L=3)
      DIMENSION IN(N),IB(N),BN(N),X(L),Y(L,N,0:1),IWK(2),WK(493)
!
      READ(5,*) XA,XB
      READ(5,*) (IN(I),I=1,N,1)
      READ(5,*) (IB(I),I=1,N,1)
      READ(5,*) (BN(I),I=1,N,1)
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100) XA,XB
      WRITE(6,1200)
      WRITE(6,1300) (IN(I),IB(I),BN(I),I=1,N)
      WRITE(6,1400) N
      WRITE(6,1500)
      WRITE(6,1600) (I,X(I),I=1,L)
      WRITE(6,1700) L
      WRITE(6,1800) ER
      WRITE(6,1900) EA
      WRITE(6,2000) NX
      WRITE(6,2100) NEV
      WRITE(6,2200) ISW
      CALL DOFNNV(FOFNNV,XA,XB,IN,IB,BN,N,X,L,ER,EA,NX,NEV,Y,&
                  ISW,IWK,WK,IERR)
      WRITE(6,2300)
      WRITE(6,2400) IERR
      WRITE(6,2500) NX
      WRITE(6,2600) NEV
      WRITE(6,2700)
      WRITE(6,2800) (((I,J,K,Y(I,J,K),K=0,1),J=1,N),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOFNNV ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *',/,&
      6X,'Y1'' = Y2',/,&
      6X,'Y2'' = -Y1-Y1*Y2',/,&
      3X,'** INPUT **')
 1100 FORMAT(6X,'INTERVAL = ( ',F4.1,', ',F4.1,'  )')
 1200 FORMAT(6X,'(BOUNDARY CONDITION)',/,&
      6X,'POSITION',5X,'INDEX OF Y',5X,'VALUE')
 1300 FORMAT(9X,I2,12X,I2,8X,F6.1)
 1400 FORMAT(6X,'SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS = ',&
      I2)
 1500 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1600 FORMAT(6X,'X(',I2,')= ',F4.1)
 1700 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1800 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 1900 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 2000 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
```

```
 2100 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2200 FORMAT(6X,'ISW = ',I2)
 2300 FORMAT(3X,'** OUTPUT **')
 2400 FORMAT(6X,'IERR = ',I4)
 2500 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2600 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2700 FORMAT(6X,'(APPROXIMATE VALUES)')
 2800 FORMAT(6X,'Y(',I2,',',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOFNNV(X,Y,N,ALF)
      REAL(8) X,Y,ALF
      DIMENSION Y(N,0:1)
!
      Y(1,1)=Y(2,0)
      Y(2,1)=-Y(1,0)-ALF*(Y(1,0)*Y(2,0))
      RETURN
      END
```

(d) Output results

```
*** DOFNNV ***
  * SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *
    Y1' = Y2
    Y2' = -Y1-Y1*Y2
 ** INPUT **
    INTERVAL = (  0.0,  1.0  )
    (BOUNDARY CONDITION)
    POSITION     INDEX OF Y     VALUE
        0             2           1.0
        1             2           2.0
    SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS =  2
    (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
    X( 1)=  0.0
    X( 2)=  0.5
    X( 3)=  1.0
    NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   3
    REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
    REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
    MAXIMUM NUMBER OF SHOOTING POINTS =  50
    MAXIMUM ITERATIVE NUMBER =   0
    ISW =  1
 ** OUTPUT **
    IERR =    0
    PRACTICAL NUMBER OF SHOOTING POINTS =   6
    PRACTICAL ITERATIVE NUMBER =  20
    (APPROXIMATE VALUES)
    Y( 1, 1, 0) = -0.1271982157D+01
    Y( 1, 1, 1) =  0.1000000000D+01
    Y( 1, 2, 0) =  0.1000000000D+01
    Y( 1, 2, 1) =  0.2543964314D+01
    Y( 2, 1, 0) = -0.4596234105D+00
    Y( 2, 1, 1) =  0.2161122455D+01
    Y( 2, 2, 0) =  0.2161122455D+01
    Y( 2, 2, 1) =  0.1452925884D+01
    Y( 3, 1, 0) =  0.6548807705D+00
    Y( 3, 1, 1) =  0.2000000000D+01
    Y( 3, 2, 0) =  0.2000000000D+01
    Y( 3, 2, 1) = -0.1964642311D+01
```

2.3.4 DOFNNF, ROFNNF
First-Order Simultaneous Ordinary Differential Equations (Function Boundary Conditions)

(1) **Function**

DOFNNF or ROFNNF uses the multipoint shooting method to solve a boundary value problem for first-order simultaneous ordinary differential equations for which boundary conditions are given as functions.

(2) **Usage**

Double precision:

CALL DOFNNF (F, FB, XA, XB, N, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROFNNF (F, FB, XA, XB, N, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, N, ALF) that defines the differential equations as a function of x and y.
2	FB	–	–	Input	Name of subroutine FB(YA, YB, N, G) that defines boundary conditions.
3	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
4	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
5	N	I	1	Input	Number of simultaneous equations
6	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
7	K	I	1	Input	Number of calculation points
8	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
9	EA	{D / R}	1	Input	Required local absolute precision (Default value: Unit for determining error)

No.	Argument	Type	Size	Input/ Output	Contents
10	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of shooting points
11	NEV	I	1	Input	Maximum number of iterations (Default value: 100)
				Output	Actual number of iterations
12	Y	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Solutions $y_j^{(v)}(x_i)$ $\mathrm{Y(i,j,v)} = y_j^{(v)}(x_i)$ **Size:** $\mathrm{K,N,0:1}$
13	ISW	I	1	Input	Parametrization processing switch 0: Parametrization not performed Nonzero: Parametrization performed
14	IWK	I	N	Work	Work area
15	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(\mathrm{N}+1)^2\times(\mathrm{NX}+1)+\mathrm{N}\times\mathrm{MAX}\ (2\times\mathrm{N},17)$
16	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{XA} < \mathrm{XB}$

(b) ER$\geq e_r$. where e_r= double precision:10^{-14}, single precision: 10^{-5}
(Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) NEV > 0 (Except when 0 is entered in order to set NEV to the default value)

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

(f) $\mathrm{K} \geq 1$

(g) $\mathrm{XA} \leq \mathrm{X(i)} \leq \mathrm{XB}\ (\mathrm{i} = 1, 2, \cdots, \mathrm{K})$

(h) $\mathrm{NX} \geq \min(5\mathrm{i}+1, 51)$
where i is the minimum integer for which $\mathrm{XB} - \mathrm{XA} \leq \mathrm{i}$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA $\neq$ XB.)	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is aborted.

IERR value	Meaning	Processing
3000	XA=XB	Processing is aborted.
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
4000	The shooting point could not be added due to severe oscillation (large derivative), etc.	
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc).	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for first-order simultaneous ordinary differential equations expressed as follows:

$$\begin{cases} y_1' &= f_1(x, y_1, y_2, \cdots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \cdots, y_n) \\ \vdots & \vdots \qquad\qquad \vdots \\ y_n' &= f_n(x, y_1, y_2, \cdots, y_n) \end{cases}$$

if there is a multiplication or division of two or more of the variables y_i in the terms on the right-hand side of any of these equations such as $y_1 \cdot y_2$, $\frac{y_2}{y_3}$ or y_4^2 or a function other than a sum or scalar multiple of y_i such as $\sin(y_1)$ or $\mathrm{abs}(y_2)$, parameterize this nonlinear problem by multiplying this portion by ALF.

Example :

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - y_1 \cdot y_2 \end{cases}$$

Parameterize this nonlinear problem as follows:

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

When the problem has been parameterized, set ISW = 1. A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, N, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, N, ALF) (in double-precision) for the high-order simultaneous ordinary differential equations that were parameterized as described in Note (a) should be created as follows:

```
SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
```

DIMENSION Y(N, 0:1)
$\mathrm{Y}(1,1) = f_1(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
$\vdots$
$\mathrm{Y}(\mathrm{i},1) = f_i(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
$\vdots$
$\mathrm{Y}(\mathrm{N},1) = f_n(\mathrm{ALF}, x, y_1, \cdots, y_i, \cdots, y_n)$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $n \leftrightarrow$N, $y_i \leftrightarrow \mathrm{Y}(\mathrm{i},0)$
$f_i(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f_i(x, \cdots)$.

Example:

When parameterized ordinary differential equations are as follows:

$$\begin{cases} y_1' = y_2 \\ y_2' = y_1 - \underline{\mathrm{ALF} \cdot y_1 \cdot y_2} \end{cases}$$

define the subroutine as follows:

SUBROUTINE F(X, Y, N, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(N, 0:1)
Y(1, 1)=Y(2, 0)
$\mathrm{Y}(2,1) = \mathrm{Y}(1,0) - \mathrm{ALF} * (\mathrm{Y}(1,0) * \mathrm{Y}(2,0))$
RETURN
END

The Input arguments: N=2, ISW=1.

(c) The actual name of subroutine FB(YA, YB, N, G), that defines the boundary conditions, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

When n boundary conditions are given as follows using values $y_i(a)$ of y_i at left-hand side boundary $x = a$ and values $y_i(b)$ of y_i at right–hand side boundary $x = b$:

$$\begin{cases} g_1(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \\ \qquad\qquad \vdots \\ g_v(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \\ \qquad\qquad \vdots \\ g_n(y_1(a), \cdots, y_i, \cdots, y_n(a), y_1(b), \cdots, y_i(b), \cdots, y_n(b)) = 0 \end{cases}$$

the subroutine FB(YA, YB, N, G) (in double-precision) should be created as follows:

SUBROUTINE FB(YA, YB, N, G)
REAL(8) YA, YB, G
DIMENSION YA(N), YB(N), G(N)
$\mathrm{G}(1) = \mathrm{g}_1(\mathrm{y}_1(\mathrm{a}), \cdots, \mathrm{y}_\mathrm{i}, \cdots, \mathrm{y}_\mathrm{n}(\mathrm{a}), \mathrm{y}_1(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{i}(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{n}(\mathrm{b}))$
$\mathrm{G}(2) = \mathrm{g}_2(\mathrm{y}_1(\mathrm{a}), \cdots, \mathrm{y}_\mathrm{i}, \cdots, \mathrm{y}_\mathrm{n}(\mathrm{a}), \mathrm{y}_1(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{i}(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{n}(\mathrm{b}))$
$\vdots$
$\mathrm{G}(\mathrm{n}) = \mathrm{g}_\mathrm{n}(\mathrm{y}_1(\mathrm{a}), \cdots, \mathrm{y}_\mathrm{i}, \cdots, \mathrm{y}_\mathrm{n}(\mathrm{a}), \mathrm{y}_1(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{i}(\mathrm{b}), \cdots, \mathrm{y}_\mathrm{n}(\mathrm{b}))$

RETURN
END

where the following correspondences are assumed.
$n \leftrightarrow$N, $y_i(a) \leftrightarrow$ YA(i, 0), $y_i(b) \leftrightarrow$ YB(i, 0)

Example:

When boundary conditions are given as follows:

$$\begin{cases} y_1(a) - y_2(b) = 0.0 \\ y_2(a) = 1.0 \end{cases}$$

define the subroutine as follows:

SUBROUTINE FB(YA, YB, N, G)
REAL(8) YA, YB, G
DIMENSION YA(N), YB(N), G(N)
G(1) = YA(1) − YB(2)
G(2) = YA(2) − 1.0D0
RETURN
END

(7) **Example**

(a) Problem

Solve the following simultaneous ordinary differential equations:

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1 - y_1 \cdot y_2 \end{cases}$$

under the following boundary conditions:

$$\begin{cases} y_1(0.0) - y_2(1.0) = 0.0 \\ y_2(0.0) = 1.0 \end{cases}$$

(b) Input data

Name of subroutine F(X, Y, N, ALF): FOFNNF and name of subroutine FB(YA, YB, N, G): GOFNNF,
N=2, XA=0.0, XB=1.0,
X, K=3, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOFNNF
! *** EXAMPLE OF DOFNNF
      IMPLICIT INTEGER (I-N)
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOFNNF,FOFNN2
      PARAMETER (N=2,L=3)
      DIMENSION X(L),Y(L,N,0:1),IWK(10),WK(1000)
!
      READ(5,*) XA,XB
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) XA,XB
      WRITE(6,1300) N
      WRITE(6,1400)
      WRITE(6,1500) (I,X(I),I=1,L)
      WRITE(6,1600) L
      WRITE(6,1700) ER
      WRITE(6,1800) EA
      WRITE(6,1900) NX
      WRITE(6,2000) NEV
      WRITE(6,2100) ISW
      CALL DOFNNF(FOFNNF,FOFNN2,XA,XB,N,X,L,ER,EA,NX,NEV,Y,ISW,IWK,&
```

```
                WK,IERR)
      WRITE(6,2200)
      WRITE(6,2300) IERR
      WRITE(6,2400) NX
      WRITE(6,2500) NEV
      WRITE(6,2600)
      WRITE(6,2700) (((I,J,K,Y(I,J,K),K=0,1),J=1,N),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOFNNF ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *',/,&
      6X,'Y1'' = Y2',/,&
      6X,'Y2'' = -Y1-Y1*Y2')
 1100 FORMAT(4X,'* BOUNDARY CONDITIONS *',/,&
      6X,'YA1-YB2 = 0.0',/,&
      6X,'YA2     = 1.0    (YA=Y(X=0.0),YB=Y(X=1.0))',/,&
      3X,'** INPUT **')
 1200 FORMAT(6X,'INTERVAL = ( ',F11.8,', ',F11.8,' )')
 1300 FORMAT(6X,'SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS = ',&
      I2)
 1400 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1500 FORMAT(6X,'X(',I2,')= ',F11.8)
 1600 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1700 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 1800 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 1900 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
 2000 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2100 FORMAT(6X,'ISW = ',I2)
 2200 FORMAT(3X,'** OUTPUT **')
 2300 FORMAT(6X,'IERR = ',I4)
 2400 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2500 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2600 FORMAT(6X,'(APPROXIMATE VALUES)')
 2700 FORMAT(6X,'Y(',I2,',',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOFNNF(X,Y,N,ALF)
!
      IMPLICIT REAL(8) (A-H,O-Z)
      DIMENSION Y(N,0:1)
!
      Y(1,1)=Y(2,0)
      Y(2,1)=-Y(1,0)-ALF*(Y(1,0)*Y(2,0))
      RETURN
      END


      SUBROUTINE FOFNN2(YA,YB,N,G)
      REAL(8) YA,YB,G
      DIMENSION YA(N),YB(N),G(N)
!
      G(1)=YA(1)-YB(2)
      G(2)=YA(2)-1.0D00
      RETURN
      END
```

(d) Output results

```
 *** DOFNNF ***
   * SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS *
     Y1' = Y2
     Y2' = -Y1-Y1*Y2
   * BOUNDARY CONDITIONS *
     YA1-YB2 = 0.0
     YA2     = 1.0    (YA=Y(X=0.0),YB=Y(X=1.0))
  ** INPUT **
     INTERVAL = (  0.00000000,  1.00000000 )
     SIMULTANEOUS NUMBER OF DIFFERENTIAL EQUATIONS =  2
     (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
     X( 1)=  0.00000000
     X( 2)=  0.50000000
     X( 3)=  1.00000000
     NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   3
     REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
     REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
     MAXIMUM NUMBER OF SHOOTING POINTS =  50
     MAXIMUM ITERATIVE NUMBER =   0
     ISW =  1
  ** OUTPUT **
     IERR =    0
     PRACTICAL NUMBER OF SHOOTING POINTS =   6
     PRACTICAL ITERATIVE NUMBER =  18
     (APPROXIMATE VALUES)
     Y( 1, 1, 0) =  0.1586848073D+00
     Y( 1, 1, 1) =  0.1000000000D+01
     Y( 1, 2, 0) =  0.1000000000D+01
     Y( 1, 2, 1) = -0.3173696147D+00
     Y( 2, 1, 0) =  0.5844246865D+00
     Y( 2, 1, 1) =  0.6485837053D+00
     Y( 2, 2, 0) =  0.6485837053D+00
     Y( 2, 2, 1) = -0.9634730151D+00
```

```
Y( 3, 1, 0) =  0.7849127611D+00
Y( 3, 1, 1) =  0.1586848073D+00
Y( 3, 2, 0) =  0.1586848073D+00
Y( 3, 2, 1) = -0.9094664914D+00
```

2.3.5 DOHNNV, ROHNNV
High-Order Ordinary Differential Equation (Numerical Boundary Conditions)

(1) **Function**

DOHNNV or ROHNNV uses the multipoint shooting method to solve a boundary value problem for a high-order ordinary differential equation for which boundary conditions are given as input values.

(2) **Usage**

Double precision:

CALL DOHNNV (F, XA, XB, IN, IC, BN, M, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROHNNV (F, XA, XB, IN, IC, BN, M, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, M, ALF) that defines the differential equations as a function of x and y.
2	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
3	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
4	IN	I	M	Input	Positions where boundary conditions are set (XAside :0, XBside: Nonzero)
5	IC	I	M	Input	Differential order j of variable $y^{(j)}$ which gives boundary conditions value
6	BN	{D / R}	M	Input	Boundary condition values given for variable $y^{(j)}$
7	M	I	1	Input	Maximum differential order of variable y in differential equations
8	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
9	K	I	1	Input	Number of calculation points

No.	Argument	Type	Size	Input/ Output	Contents
10	ER	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
11	EA	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Required local absolute precision (Default value: Unit for determining error)
12	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of shooting points
13	NEV	I	1	Input	Maximum number of iterations (Default value:100)
				Output	Actual number of iterations
14	Y	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	$\mathrm{K}, 0 : \mathrm{M}$	Output	Solutions $y^{(v)}(x_i)$ $\mathrm{Y(i, v)} = y^{(v)}(x_i)$
15	ISW	I	1	Input	Parameterization processing switch 0: Parameterization not performed Nonzero: Parameterization performed
16	IWK	I	M	Work	Work area
17	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(\mathrm{M}+1)^2 \times (\mathrm{NX}+1) + \mathrm{M} \times \mathrm{MAX}(2 \times \mathrm{M} + 1, 16)$
18	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{XA} < \mathrm{XB}$

(b) ER$\geq e_r$. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(c) EA $\geq$ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) $\mathrm{NEV} > 0$ (Except when 0 is entered in order to set NEV to the default value)

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

(f) $0 \leq \mathrm{IC(i)} < \mathrm{M}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{M})$

(g) $\mathrm{K} \geq 1$

(h) $\mathrm{XA} \leq \mathrm{X(i)} \leq \mathrm{XB}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{K})$

(i) $\mathrm{NX} \geq \min(5\mathrm{i} + 1, 51)$
Where, i is minimum integer for which $\mathrm{XB} - \mathrm{XA} \leq \mathrm{i}$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA $\neq$ XB)	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with the corresponding default value set.
3000	XA=XB	Processing is aborted.
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
3050	Restriction (i) was not satisfied.	
4000	The shooting point could not be added due to severe oscillation (large derivative), etc.	
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc.)	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for a high-order ordinary differential equation expressed as follows:

$$y^{(m)} = f(x, y, y', \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(Where, differential order j of right-hand side $y^{(j)}$ must satisfy $j \leq m-1$.) if there is a multiplication or division of two or more of the variables $y^{(j)}$ in the terms on the right-hand side of the equation such as $y \cdot y'$, $\frac{y}{y'}$, y^2, or a function other than a sum or scalar multiple of $y^{(j)}$ such as sin(y) or abs(y'), parameterize this nonlinear problem by multiplying this portion by ALF.
Example :

$$y'' = y' + y^2$$

Parameterize this nonlinear problem as follows:

$$y'' = y' + \underline{\mathrm{ALF} \cdot y^2}$$

When the problem has been parameterized, set ISW = 1.
A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, M, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, M, ALF) (in double-precision) for the high-order ordinary differential equation that was parameterized as described in Note (a) should be created as follows:

```
SUBROUTINE F(X, Y, M, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(0:M)
Y(M) = f(ALF, x, y, ···, y^(j), ···, y^(m−1))
RETURN
END
```

where the following correspondences are assumed.
$x \leftrightarrow$X, $m \leftrightarrow$M, $y \leftrightarrow$ Y(0), $y^{(j)} \leftrightarrow$ Y(j)
$f(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f(x, \cdots)$.
Example:
When parameterized ordinary differential equation is as follows:

$$y'' = y' + \underline{\mathrm{ALF} \cdot y^2}$$

define the subroutine as follows:

```
SUBROUTINE F(X, Y, M, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(0:M)
Y(2) = Y(1) + ALF * Y(0) * *2
RETURN
END
```

The Input arguments: M=2, ISW=1.

(c) If the boundary conditions of the high-order ordinary differential equation are given by:

$$\text{left-hand side boundary} \begin{cases} y^{(b_1)} = c_1 \\ y^{(b_2)} = c_2 \\ \quad \vdots \\ y^{(b_p)} = c_p \end{cases} \qquad \text{right-hand side boundary} \begin{cases} y^{(b_{p+1})} = c_{p+1} \\ y^{(b_{p+2})} = c_{p+2} \\ \quad \vdots \\ y^{(b_m)} = c_m \end{cases}$$

set array IN, IC and BN as follows:
IN(i) = 0 (i = 1, 2, ···, p)
IN(i) = 1 (i = p + 1, p + 2, ···, m)
IC(i) = b_i(i = 1, 2, ···, m)
BN(i) = c_i(i = 1, 2, ···, m)
Example:
If the boundary conditions are given by:

$$\text{left-hand side boundary} \quad y = 1.0 \quad \text{right-hand side boundary} \quad y' = 2.0$$

the input values of IN, IC and BN are as follows:
IN(1)=0, IC(1)=0, BN(1)=1.0
IN(2)=1, IC(2)=1, BN(2)=2.0

(7) **Example**

(a) Problem

Solve the following ordinary differential equation:

$$y'' = y' + y^2$$

under the following boundary conditions:

$$y|_{x=1.0} = 1.0, y'|_{x=2.0} = 2.0$$

(b) Input data

Name of subroutine F(X, Y, M, ALF): FOHNNV, XA=1.0, XB=2.0,
IN(1)=0, IC(1)=0, BN(1)=1.0,
IN(2)=1, IC(2)=1, BN(2)=2.0,
M=2, X, K=6, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOHNNV
! *** EXAMPLE OF DOHNNV
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOHNNV
      PARAMETER (M=2,L=6)
      DIMENSION IN(M),IC(M),BN(M),X(L),Y(L,0:M),IWK(10),WK(1000)
!
      READ(5,*) XA,XB
      READ(5,*) (IN(I),I=1,M,1)
      READ(5,*) (IC(I),I=1,M,1)
      READ(5,*) (BN(I),I=1,M,1)
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100) XA,XB
      WRITE(6,1200)
      WRITE(6,1300) (IN(I),IC(I),BN(I),I=1,M)
      WRITE(6,1400) M
      WRITE(6,1500)
      WRITE(6,1600) (I,X(I),I=1,L)
      WRITE(6,1700) L
      WRITE(6,1800) ER
      WRITE(6,1900) EA
      WRITE(6,2000) NX
      WRITE(6,2100) NEV
      WRITE(6,2200) ISW
      CALL DOHNNV(FOHNNV,XA,XB,IN,IC,BN,M,X,L,ER,EA,NX,NEV,Y,&
                  ISW,IWK,WK,IERR)
      WRITE(6,2300)
      WRITE(6,2400) IERR
      WRITE(6,2500) NX
      WRITE(6,2600) NEV
      WRITE(6,2700)
      WRITE(6,2800) ((I,J,Y(I,J),J=0,M),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOHNNV ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *',/,&
      6X,'Y'''' = Y''+Y**2',/,&
      3X,'** INPUT **')
 1100 FORMAT(6X,'INTERVAL = ( ',F4.1,', ',F4.1,'  )')
 1200 FORMAT(6X,'(BOUNDARY CONDITION)',/,&
      6X,'POSITION',5X,'ORDER OF Y',5X,'VALUE')
 1300 FORMAT(9X,I2,12X,I2,8X,F6.1)
 1400 FORMAT(6X,'ORDER OF DIFFERENTIAL EQUATION    = ',I2)
 1500 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1600 FORMAT(6X,'X(',I2,')= ',F4.1)
 1700 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1800 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 1900 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 2000 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
 2100 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2200 FORMAT(6X,'ISW = ',I2)
 2300 FORMAT(3X,'** OUTPUT **')
 2400 FORMAT(6X,'IERR = ',I4)
 2500 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2600 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2700 FORMAT(6X,'(APPROXIMATE VALUES)')
 2800 FORMAT(6X,'Y(',I2,',',I2,') = ',D17.10)
!
      END
```

```
      SUBROUTINE FOHNNV(X,Y,M,ALF)
      REAL(8) X,Y,ALF
      DIMENSION Y(0:M)
!
      Y(2)=Y(1)+ALF*Y(0)**2
      RETURN
      END
```

(d) Output results

```
 *** DOHNNV ***
   * SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *
     Y'' = Y'+Y**2
  ** INPUT **
     INTERVAL = (  1.0,  2.0  )
     (BOUNDARY CONDITION)
     POSITION     ORDER OF Y     VALUE
         0             0           1.0
         1             1           2.0
     ORDER OF DIFFERENTIAL EQUATION    =  2
     (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
     X( 1)=  1.0
     X( 2)=  1.2
     X( 3)=  1.4
     X( 4)=  1.6
     X( 5)=  1.8
     X( 6)=  2.0
     NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   6
     REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
     REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
     MAXIMUM NUMBER OF SHOOTING POINTS =  10
     MAXIMUM ITERATIVE NUMBER =   0
     ISW =  1
  ** OUTPUT **
     IERR =    0
     PRACTICAL NUMBER OF SHOOTING POINTS =   6
     PRACTICAL ITERATIVE NUMBER =  12
     (APPROXIMATE VALUES)
     Y( 1, 0) =  0.1000000000D+01
     Y( 1, 1) = -0.8307119791D-01
     Y( 1, 2) =  0.9169288021D+00
     Y( 2, 0) =  0.1002909611D+01
     Y( 2, 1) =  0.1190766991D+00
     Y( 2, 2) =  0.1124904386D+01
     Y( 3, 0) =  0.1051306055D+01
     Y( 3, 1) =  0.3767460702D+00
     Y( 3, 2) =  0.1481990492D+01
     Y( 4, 0) =  0.1159746886D+01
     Y( 4, 1) =  0.7272513256D+00
     Y( 4, 2) =  0.2072264164D+01
     Y( 5, 0) =  0.1352378945D+01
     Y( 5, 1) =  0.1231842529D+01
     Y( 5, 2) =  0.3060771341D+01
     Y( 6, 0) =  0.1669843256D+01
     Y( 6, 1) =  0.2000000000D+01
     Y( 6, 2) =  0.4788376498D+01
```

2.3.6 DOHNNF, ROHNNF
High-Order Ordinary Differential Equation (Function Boundary Conditions)

(1) **Function**

DOHNNF or ROHNNF uses the multipoint shooting method to solve a boundary value problem for a high-order ordinary differential equation for which boundary conditions are given as functions.

(2) **Usage**

Double precision:

CALL DOHNNF (F, FB, XA, XB, M, X, K, ER, EA, NX, NEV, Y, ISW, IWK, WK, IERR)

Single precision:

CALL ROHNNF (F, FB, XA, XB, M, X, K, ER, EA, NX, NEV, Y, 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	F	–	–	Input	Name of subroutine F(X, Y, M, ALF) that defines the differential equation as a function of x and y.
2	FB	–	–	Input	Name of subroutine FB(YA, YB, M, G) that defines boundary conditions.
3	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
4	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
5	M	I	1	Input	Maximum differential order of variable y in differential equations
6	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
7	K	I	1	Input	Number of calculation points
8	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
9	EA	{D / R}	1	Input	Required local absolute precision (Default value: Unit for determining error)

No.	Argument	Type	Size	Input/ Output	Contents
10	NX	I	1	Input	Maximum number of shooting points
				Output	Actual number of shooting points
11	NEV	I	1	Input	Maximum number of iterations (Default value: 100)
				Output	Actual number of iterations
12	Y	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	K, 0 : M	Output	Solutions $y^{(v)}(x_i)$ Y(i, v) = $y^{(v)}(x_i)$
13	ISW	I	1	Input	Parameterization processing switch 0: Parameterization not performed Nonzero: Parameterization performed
14	IWK	I	M	Work	Work area
15	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(\mathrm{M}+1)^2 \times (\mathrm{NX}+1) + \mathrm{M} \times \mathrm{MAX}(2 \times \mathrm{M} + 1, 16)$
16	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) XA < XB

(b) ER≥ e_r. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(c) EA ≥ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) NEV > 0 (Except when 0 is entered in order to set NEV to the default value)

(e) M ≥ 1

(f) K ≥ 1

(g) XA ≤ X(i) ≤ XB (i = 1, 2, ⋯, K)

(h) NX ≥ min(5i + 1, 51)
Where, i is minimum integer for which XB − XA ≤ i

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where XA ≠ XB)	Processing is performed assuming XA and XB are replaced each other.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with the corresponding default value set.

IERR value	Meaning	Processing
3000	XA=XB	Processing is aborted.
3010	Restriction (e) was not satisfied.	
3020	Restriction (f) was not satisfied.	
3030	Restriction (g) was not satisfied.	
3040	Restriction (h) was not satisfied.	
4000	The shooting point could not be added due to severe oscillation (large derivative), etc.	
4500	The step size became too small during the initial value problem calculation (existence of singularity, etc.)	
5000	The maximum number of shooting points was reached.	
5500	The maximum number of iterations was reached (including cases when solution does not exist or is indefinite).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) In a nonlinear problem for a high-order ordinary differential equation expressed as follows:

$$y^{(m)} = f(x, y, y', \cdots, y^{(j)}, \cdots, y^{(m-1)})$$

(Where, differential order j of right-hand side $y^{(j)}$ must satisfy $j \leq m-1$.) if there is a multiplication or division of two or more of the variables $y^{(j)}$ in the terms on the right-hand side of the equation such as $y \cdot y'$, $\frac{y}{y'}$, y^2, or a function other than a sum or scalar multiple of $y^{(j)}$ such as sin(y) or abs(y'), parameterize this nonlinear problem by multiplying this portion by ALF.

Example :

$$y'' = y' + y^2$$

Parameterize this nonlinear problem as follows:

$$y'' = y' + \underline{\text{ALF} \cdot y^2}$$

When the problem has been parameterized, set ISW = 1.
A linear problem need not be parameterized. In this case, set ISW = 0.

(b) The actual name of subroutine F(X, Y, M, ALF), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

The subroutine F(X, Y, M, ALF) (in double-precision) for the high-order ordinary differential equation that was parameterized as described in Note (a) should be created as follows:

```
SUBROUTINE F(X, Y, M, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(0:M)
Y(M) = f(ALF, x, y, ···, y^(j), ···, y^(m-1))
RETURN
END
```

where the following correspondences are assumed:
$x \leftrightarrow$X, $m \leftrightarrow$M, $y \leftrightarrow$ Y(0), $y^{(j)} \leftrightarrow$ Y(j)
$f(\mathrm{ALF}, x, \cdots)$ is parameterized one for $f(x, \cdots)$.
Example:
When parameterized ordinary differential equation is as follows:

$$y'' = y' + \underline{\mathrm{ALF} \cdot y^2}$$

define the subroutine as follows:

```
SUBROUTINE F(X, Y, M, ALF)
REAL(8) X, Y, ALF
DIMENSION Y(0:M)
Y(2) = Y(1) + ALF * Y(0) * *2
RETURN
END
```

The Input arguments: M=2, ISW=1.

(c) The actual name of subroutine FB(YA, YB, M, G), that defines the boundary conditions, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

When m boundary conditions are given as follows using values $y^{(j)}(a)$ of $y^{(j)}$ at left-hand side boundary $x = a$ and values $y^{(j)}(b)$ of $y^{(j)}$ at right-hand side boundary $x = b$:

$$\begin{cases} g_1(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \\ g_2(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ g_m(y(a), \cdots, y^{(j)}(a), \cdots, y^{(m-1)}(a), y(b), \cdots, y^{(j)}(b), \cdots, y^{(m-1)}(b)) = 0 \end{cases}$$

the subroutine FB(YA, YB, M, G) (in double-precision) should be created as follows:

```
SUBROUTINE FB(YA, YB, M, G)
REAL(8) YA, YB, G
DIMENSION YA(0:M-1), YB(0:M-1), G(M)
G(1) = g1(y(a), ···, y(j)(a), ···, y(m-1)(a), y(b), ···, y(j)(b), ···, y(m-1)(b))
G(2) = g2(y(a), ···, y(j)(a), ···, y(m-1)(a), y(b), ···, y(j)(b), ···, y(m-1)(b))
  ⋮
G(m) = gm(y(a), ···, y(j)(a), ···, y(m-1)(a), y(b), ···, y(j)(b), ···, y(m-1)(b))
RETURN
END
```

where the following correspondences are assumed.
$m \leftrightarrow$M, $y(a) \leftrightarrow$ YA(0), $y^{(j)}(a) \leftrightarrow$ YA(j)
$y(b) \leftrightarrow$ YB(0), $y^{(j)}(b) \leftrightarrow$ YB(j)
Example:
When boundary conditions are given as follows:

$$\begin{cases} y(a) - y(b) = 0.0 \\ y'(b) = 1.0 \end{cases}$$

define the subroutine as follows:

```
SUBROUTINE FB(YA, YB, M, G)
REAL(8) YA, YB, G
DIMENSION YA(0:M-1), YB(0:M-1), G(M)
G(1) = YA(0) − YB(0)
G(2) = YB(1) − 1.0D0
RETURN
END
```

(7) **Example**

(a) Problem

Solve the following ordinary differential equation:

$$y'' = y' + y^2$$

under the following boundary conditions:

$$\begin{cases} y(1.0) - y(2.0) = 0.0 \\ y(2.0) = 2.0 \end{cases}$$

(b) Input data

Name of subroutine F(X, Y, M, ALF): FOHNNF and name of subroutine FB(YA, YB, M, G): GOHNNF, XA=1.0, XB=2.0, M=2, X, K=3, ER, EA, NX, NEV=0 and ISW=1.

(c) Main program

```
      PROGRAM BOHNNF
! *** EXAMPLE OF DOHNNF
      IMPLICIT INTEGER (I-N)
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOHNNF,FOHNN2
      PARAMETER (M=2,L=3)
      DIMENSION X(L),Y(L,0:M),IWK(10),WK(1000)
!
      READ(5,*) XA,XB
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      READ(5,*) NX,NEV
      READ(5,*) ISW
      WRITE(6,1000)
      WRITE(6,1100)
      WRITE(6,1200) XA,XB
      WRITE(6,1300) M
      WRITE(6,1400)
      WRITE(6,1500) (I,X(I),I=1,L)
      WRITE(6,1600) L
      WRITE(6,1700) ER
      WRITE(6,1800) EA
      WRITE(6,1900) NX
      WRITE(6,2000) NEV
      WRITE(6,2100) ISW
      CALL DOHNNF(FOHNNF,FOHNN2,XA,XB,M,X,L,ER,EA,NX,NEV,Y,ISW,IWK,&
                  WK,IERR)
      WRITE(6,2200)
      WRITE(6,2300) IERR
      WRITE(6,2400) NX
      WRITE(6,2500) NEV
      WRITE(6,2600)
      WRITE(6,2700) ((I,J,Y(I,J),J=0,M),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOHNNF ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *',/,&
      6X,'Y'''= Y''+Y**2')
 1100 FORMAT(4X,'* BOUNDARY CONDITIONS *',/,&
      6X,'YA-YB = 0.0',/,&
      6X,'YB    = 2.0    (YA=Y(X=1.0),YB=Y(X=2.0))',/,&
      3X,'** INPUT **')
 1200 FORMAT(6X,'INTERVAL = ( ',F11.8,', ',F11.8,' )')
 1300 FORMAT(6X,'ORDER OF DIFFERENTIAL EQUATION    = ',I2)
 1400 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1500 FORMAT(6X,'X(',I2,')= ',F11.8)
 1600 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1700 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 1800 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 1900 FORMAT(6X,'MAXIMUM NUMBER OF SHOOTING POINTS = ',I3)
```

```
 2000 FORMAT(6X,'MAXIMUM ITERATIVE NUMBER = ',I3)
 2100 FORMAT(6X,'ISW = ',I2)
 2200 FORMAT(3X,'** OUTPUT **')
 2300 FORMAT(6X,'IERR = ',I4)
 2400 FORMAT(6X,'PRACTICAL NUMBER OF SHOOTING POINTS = ',I3)
 2500 FORMAT(6X,'PRACTICAL ITERATIVE NUMBER = ',I3)
 2600 FORMAT(6X,'(APPROXIMATE VALUES)')
 2700 FORMAT(6X,'Y(',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOHNNF(X,Y,M,ALF)
      REAL(8) X,Y,ALF
      DIMENSION Y(0:M)
!
      Y(2)=Y(1)+ALF*Y(0)**2
      RETURN
      END


      SUBROUTINE FOHNN2(YA,YB,M,G)
      REAL(8) YA,YB,G
      DIMENSION YA(0:M-1),YB(0:M-1),G(M)
!
      G(1)=YA(0)-YB(0)
      G(2)=YB(0)-2.0D0
      RETURN
      END
```

(d) Output results

```
*** DOHNNF ***
  * SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *
    Y''= Y'+Y**2
  * BOUNDARY CONDITIONS *
    YA-YB = 0.0
    YB    = 2.0    (YA=Y(X=1.0),YB=Y(X=2.0))
 ** INPUT **
    INTERVAL = (  1.00000000,  2.00000000 )
    ORDER OF DIFFERENTIAL EQUATION    =  2
    (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
    X( 1)=  1.00000000
    X( 2)=  1.50000000
    X( 3)=  2.00000000
    NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   3
    REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
    REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
    MAXIMUM NUMBER OF SHOOTING POINTS =  50
    MAXIMUM ITERATIVE NUMBER =   0
    ISW =  1
 ** OUTPUT **
    IERR =    0
    PRACTICAL NUMBER OF SHOOTING POINTS =   6
    PRACTICAL ITERATIVE NUMBER =  11
    (APPROXIMATE VALUES)
    Y( 1, 0) =  0.2000000000D+01
    Y( 1, 1) = -0.1324134770D+01
    Y( 1, 2) =  0.2675865230D+01
    Y( 2, 0) =  0.1644546644D+01
    Y( 2, 1) = -0.1036799250D+00
    Y( 2, 2) =  0.2600853739D+01
    Y( 3, 0) =  0.2000000000D+01
    Y( 3, 1) =  0.1780609498D+01
    Y( 3, 2) =  0.5780609498D+01
```

2.3.7 DOHNLV, ROHNLV
High-Order Linear Ordinary Differential Equation

(1) **Function**

DOHNLV or ROHNLV uses the collocation method to solve a boundary value problem for a high-order linear ordinary differential equation for which boundary conditions are given as input values.

(2) **Usage**

Double precision:

```
CALL DOHNLV  (F, XA, XB, IN, IC, BN, M, X, K, ER, EA, NX, Y, IWK, WK, IERR)
```

Single precision:

```
CALL ROHNLV  (F, XA, XB, IN, IC, BN, M, X, K, ER, EA, NX, Y, 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	F	–	–	Input	Name of subroutine F(X, AL, M) that defines the coefficients and constant term of the high-order linear ordinary differential equation.
2	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
3	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
4	IN	I	M	Input	Positions where boundary conditions are set (XA side: 0, XB side: Nonzero)
5	IC	I	M	Input	Differential order j of variable $y^{(j)}$ which gives boundary conditions value
6	BN	{D / R}	M	Input	Boundary condition values given for variable $y^{(j)}$
7	M	I	1	Input	Maximum differential order of variable y in differential equations
8	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
9	K	I	1	Input	Number of calculation points
10	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
11	EA	{D / R}	1	Input	Required local absolute precision (Default value: Unit for determining error)
12	NX	I	1	Input	Maximum number of selected points
				Output	Actual number of selected points
13	Y	{D / R}	K, 0 : M	Output	Solutions $y^{(v)}(x_i)$ Y(i, v) = $y^{(v)}(x_i)$
14	IWK	I	3 × M + NX	Work	Work area
15	WK	{D / R}	See Contents	Work	Work area **Size**: 2 × NX × NX + 5 × NX + 2 × M + 2
16	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) XA < XB

(b) ER≥ e_r. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(c) EA ≥ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) M ≥ 0

(e) 0 ≤ IC(i) < M (i = $1, 2, \cdots,$ M)

(f) K ≥ 1

(g) XA ≤ X(i) ≤ XB (i = $1, 2, \cdots,$ K)

(h) NX ≥ MAX(MIN(5 × i + 1, 101), M + 2)
Where, i is minimum integer for which XB − XA ≤ i

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where, XA ≠ XB).	Processing is performed assuming that XA and XB are replaced each other.
1500	Restriction (b) or (c) was not satisfied.	Processing is performed with the corresponding default value set.
3000	XA=XB	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	
3030	Restriction (f) was not satisfied.	
3040	Restriction (g) was not satisfied.	
3050	Restriction (h) was not satisfied.	
4000	The simultaneous first-order equations could not be solved.	
5000	The maximum number of selected points was reached (including the case when there is no solution).	The solution at that time is returned, and processing is aborted.

(6) **Notes**

(a) The actual name of subroutine F(X, AL, M), that defines the differential equation, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the following high-order linear ordinary differential equation:

$$f_1(x)y^{(m)} + f_2(x)y^{(m-1)} + \cdots + f_{m+1}(x)y + f_{m+2}(x) = 0$$

the subroutine F(X, AL, M) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, AL, M)
REAL(8) X, AL
```

DIMENSION AL(M+2)
AL(1) = $f_1(x)$
AL(2) = $f_2(x)$
$\vdots$
AL(M + 2) = $f_{m+2}(x)$
RETURN
END

where the following correspondences are assumed:
$x \leftrightarrow$X, $m \leftrightarrow$M
Example:

$$y'' + 2xy + x = 0$$

SUBROUTINE F(X, AL, M)
REAL(8) X, AL
DIMENSION AL(M+2)
AL(1)=1.0D0
AL(2)=0.0D0
AL(3) = 2 ∗ X
AL(4)=X
RETURN
END

The Input argument:M=2.

(b) If the boundary conditions of the high-order linear ordinary differential equation are given by:

$$\text{left-hand side boundary} \begin{cases} y^{(b_1)} = c_1 \\ y^{(b_2)} = c_2 \\ \quad \vdots \\ y^{(b_p)} = c_p \end{cases} \quad \text{right-hand side boundary} \begin{cases} y^{(b_{p+1})} = c_{p+1} \\ y^{(b_{p+2})} = c_{p+2} \\ \quad \vdots \\ y^{(b_m)} = c_m \end{cases}$$

set array IN, IC and BN as follows:
IN(i) = 0 (i = 1, 2, $\cdots$, p)
IN(i) = 1 (i = p + 1, p + 2, $\cdots$, m)
IC(i) = b_i(i = 1, 2, $\cdots$, m)
BN(i) = c_i(i = 1, 2, $\cdots$, m)
Example: If the boundary conditions are given by:

$$\text{left-hand side boundary} \quad y = 1.0 \quad \text{right-hand side boundary} \quad y' = 2.0$$

the input values of IN, IC and BN are as follows:
IN(1)=0, IC(1)=0, BN(1)=1.0
IN(2)=1, IC(2)=1, BN(2)=2.0

(7) **Example**

(a) Problem
Solve the following ordinary differential equation:

$$y'' + 2xy + x = 0$$

under the following boundary conditions:

$$y|_{x=0.0} = 1.0, y'|_{x=1.0} = 2.0$$

(b) Input data

Name of subroutine F(X, AL, M):FOHNLV, XA=0.0, XB=1.0,
IN(1)=0, IC(1)=0, BN(1)=1.0,
IN(2)=1, IC(2)=1, BN(2)=2.0,
M=2, X, K=6, ER, EA and NX.

(c) Main program

```
      PROGRAM BOHNLV
! *** EXAMPLE OF DOHNLV
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOHNLV
      PARAMETER (M=2,K=6)
      PARAMETER (NX0=100)
      DIMENSION IN(M),IC(M),BN(M),X(K),Y(K,0:M),IWK(3*M+NX0)
      DIMENSION WK(2*NX0*NX0+5*NX0+2*M+2)
!
      READ(5,*) XA,XB
      READ(5,*) (IN(I),I=1,M,1)
      READ(5,*) (IC(I),I=1,M,1)
      READ(5,*) (BN(I),I=1,M,1)
      READ(5,*) (X(I),I=1,K,1)
      READ(5,*) ER,EA
      READ(5,*) NX
      WRITE(6,1000)
      WRITE(6,1100) XA,XB
      WRITE(6,1200)
      WRITE(6,1300) (IN(I),IC(I),BN(I),I=1,M)
      WRITE(6,1400) M
      WRITE(6,1500)
      WRITE(6,1600) (I,X(I),I=1,K)
      WRITE(6,1700) K
      WRITE(6,1800) ER
      WRITE(6,1900) EA
      WRITE(6,2000) NX
      CALL DOHNLV(FOHNLV,XA,XB,IN,IC,BN,M,X,K,ER,EA,NX,Y,IWK,WK,IERR)
      WRITE(6,2100)
      WRITE(6,2200) IERR
      WRITE(6,2300) NX
      WRITE(6,2400)
      WRITE(6,2500) ((I,J,Y(I,J),J=0,M),I=1,K)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOHNLV ***',/,&
      4X,'* SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *',/,&
      6X,'Y'''+2*X*Y+X = 0.0',/,&
      3X,'** INPUT **')
 1100 FORMAT(6X,'INTERVAL = ( ',F4.1,', ',F4.1,'  )')
 1200 FORMAT(6X,'(BOUNDARY CONDITION)',/,&
      6X,'POSITION',5X,'ORDER OF Y',5X,'VALUE')
 1300 FORMAT(9X,I2,12X,I2,8X,F6.1)
 1400 FORMAT(6X,'ORDER OF DIFFERENTIAL EQUATION = ',I2)
 1500 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1600 FORMAT(6X,'X(',I2,')= ',F4.1)
 1700 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1800 FORMAT(6X,'REQUIRED RELATIVE PRECISION = ',D8.1)
 1900 FORMAT(6X,'REQUIRED ABSOLUTE PRECISION = ',D8.1)
 2000 FORMAT(6X,'MAXIMUM NUMBER OF SELECTED POINTS = ',I3)
 2100 FORMAT(3X,'** OUTPUT **')
 2200 FORMAT(6X,'IERR = ',I4)
 2300 FORMAT(6X,'PRACTICAL NUMBER OF SELECTED POINTS = ',I3)
 2400 FORMAT(6X,'(APPROXIMATE VALUES)')
 2500 FORMAT(6X,'Y(',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOHNLV(X,AL,M)
      REAL(8) X,AL
      DIMENSION AL(M+2)
!
      AL(1)=1.0D0
      AL(2)=0.0D0
      AL(3)=2.0D0*X
      AL(4)=X
      RETURN
      END
```

(d) Output results

```
 *** DOHNLV ***
   * SYSTEM OF ORDINARY DIFFERENTIAL EQUATION *
     Y''+2*X*Y+X = 0.0
```

```
** INPUT **
   INTERVAL = (  0.0,  1.0  )
   (BOUNDARY CONDITION)
   POSITION     ORDER OF Y     VALUE
       0             0           1.0
       1             1           2.0
   ORDER OF DIFFERENTIAL EQUATION =  2
   (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
   X( 1)=  0.0
   X( 2)=  0.2
   X( 3)=  0.4
   X( 4)=  0.6
   X( 5)=  0.8
   X( 6)=  1.0
   NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   6
   REQUIRED RELATIVE PRECISION =  0.1D-09
   REQUIRED ABSOLUTE PRECISION =  0.1D-09
   MAXIMUM NUMBER OF SELECTED POINTS = 100
** OUTPUT **
   IERR =    0
   PRACTICAL NUMBER OF SELECTED POINTS =  42
   (APPROXIMATE VALUES)
   Y( 1, 0) =  0.1000000000D+01
   Y( 1, 1) =  0.8544827835D+01
   Y( 1, 2) = -0.2246451913D-09
   Y( 2, 0) =  0.2702689947D+01
   Y( 2, 1) =  0.8439349772D+01
   Y( 2, 2) = -0.1281075979D+01
   Y( 3, 0) =  0.4349720445D+01
   Y( 3, 1) =  0.7944231536D+01
   Y( 3, 2) = -0.3879776356D+01
   Y( 4, 0) =  0.5837763648D+01
   Y( 4, 1) =  0.6811782134D+01
   Y( 4, 2) = -0.7605316378D+01
   Y( 5, 0) =  0.7019211479D+01
   Y( 5, 1) =  0.4854783090D+01
   Y( 5, 2) = -0.1203073837D+02
   Y( 6, 0) =  0.7719439396D+01
   Y( 6, 1) =  0.2000000000D+01
   Y( 6, 2) = -0.1643887879D+02
```

2.3.8 DOLNLV, ROLNLV
Second-Order Linear Ordinary Differential Equation

(1) **Function**

DOLNLV or ROLNLV uses the coefficient determination method to solve a boundary value problem for a second-order linear ordinary differential equation for which boundary conditions are given as input values.

(2) **Usage**

Double precision:

```
CALL DOLNLV  (F, XA, XB, IN, IC, BN, X, K, ER, EA, Y, WK, IERR)
```

Single precision:

```
CALL ROLNLV  (F, XA, XB, IN, IC, BN, X, K, ER, EA, Y, 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	F	–	–	Input	Name of subroutine F(X, AL) that defines the coefficients and constant term of the second-order linear ordinary differential equation.
2	XA	{D / R}	1	Input	x coordinate of left-hand side boundary
3	XB	{D / R}	1	Input	x coordinate of right-hand side boundary
4	IN	I	2	Input	Positions where boundary conditions are set (XA side: 0, XB side: Nonzero)
5	IC	I	2	Input	Differential order j of variable $y^{(j)}$ which gives boundary conditions value
6	BN	{D / R}	2	Input	Boundary condition values given for variable $y^{(j)}$
7	X	{D / R}	K	Input	x coordinate x_i where approximate solutions are calculated
8	K	I	1	Input	Number of calculation points
9	ER	{D / R}	1	Input	Required local relative precision **Default value**: Double precision : 10^{-12} Single precision : 10^{-5}
10	EA	{D / R}	1	Input	Required local absolute precision (Default value: Unit for determining error)
11	Y	{D / R}	K , 0:2	Output	Solutions $y^{(v)}(x_i)$ $\mathrm{Y(i, v)} = y^{(v)}(x_i)$
12	WK	{D / R}	$6 \times \mathrm{K} + 35$	Work	Work area
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) XA < XB

(b) ER≥ e_r. Where, e_r= double precision:10^{-14}, single precision: 10^{-5} (Except when 0.0 is entered in order to set ER to the default value)

(c) EA ≥ (Minimum expressible absolute value) $\times 2^{24}$
(Except when 0.0 is entered in order to set EA to the default value)

(d) $0 \leq \mathrm{IC(i)} < 2$ $(\mathrm{i} = 1, 2)$

(e) $K \geq 1$

(f) $XA \leq X(i) \leq XB$ $(i = 1, 2, \cdots, K)$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied. (where, XA $\neq$ XB).	Processing is performed assuming that XA and XB are replaced each other.
1500	Restriction (b) or (c) was not satisfied.	Processing is performed with the corresponding default value set.
3000	XA=XB	Processing is aborted.
3010	Restriction (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	
3030	Restriction (f) was not satisfied.	
4000	The step size became too small during the initial value problem calculation.	
4500	There is no solution or the solution is indefinite.	

(6) **Notes**

(a) The actual name of subroutine F(X, AL), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created (For details, See Section 2.1.1.1 (4), (7)).

For the following second-order linear ordinary differential equation:

$$f_1(x)y'' + f_2(x)y' + f_3(x)y + f_4(x) = 0$$

the subroutine F(X, AL) (in double-precision) should be created as follows:

```
SUBROUTINE F(X, AL)
REAL(8) X, AL
DIMENSION AL(4)
AL(1) = f1(x)
AL(2) = f2(x)
AL(3) = f3(x)
AL(4) = f4(x)
RETURN
END
```

where the following correspondence is assumed.

$x \leftrightarrow$X

Example:

$$y'' + 2xy + x = 0$$

```
SUBROUTINE F(X, AL)
REAL(8) X, AL
DIMENSION AL(4)
AL(1)=1.0D0
AL(2)=0.0D0
AL(3) = 2 * X
AL(4)=X
RETURN
END
```

(b) If the boundary conditions of the second-order ordinary differential equation are given by:

$$\text{left-hand side boundary} \quad y^{(b_1)} = c_1 \quad \text{right-hand side boundary} \quad y^{(b_2)} = c_2$$

set array IN, IC and BN as follows:
IN(1) = 0
IN(2) = 1
IC(i) = b_i (i = 1, 2)
BN(i) = c_i (i = 1, 2)
Example: If the boundary conditions are given by:

$$\text{left-hand side boundary} \quad y = 1.0 \quad \text{right-hand side boundary} \quad y' = 2.0$$

the input values of IN, IC and BN are as follows:
IN(1)=0, IC(1)=0, BN(1)=1.0
IN(2)=1, IC(2)=1, BN(2)=2.0

(7) **Example**

(a) Problem
Solve the following ordinary differential equation:

$$y'' + 2xy + x = 0$$

under the following boundary conditions:

$$y|_{x=0.0} = 1.0, y'|_{x=1.0} = 2.0$$

(b) Input data
Name of subroutine F(X, AL):FOLNLV, XA=0.0, XB=1.0,
IN(1)=0, IC(1)=0, BN(1)=1.0,
IN(2)=1, IC(2)=1, BN(2)=2.0,
X, K=6, ER and EA.

(c) Main program

```
      PROGRAM BOLNLV
! *** EXAMPLE OF DOLNLV ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FOLNLV
      PARAMETER (L=6)
      DIMENSION IN(2),IC(2),BN(2),X(L),Y(L,0:2),WK(100)
!
      READ(5,*) XA,XB
      READ(5,*) (IN(I),I=1,2,1)
      READ(5,*) (IC(I),I=1,2,1)
```

```
      READ(5,*) (BN(I),I=1,2,1)
      READ(5,*) (X(I),I=1,L,1)
      READ(5,*) ER,EA
      WRITE(6,1000)
      WRITE(6,1100) XA,XB
      WRITE(6,1200)
      WRITE(6,1300) (IN(I),IC(I),BN(I),I=1,2)
      WRITE(6,1400)
      WRITE(6,1500) (I,X(I),I=1,L)
      WRITE(6,1600) L
      WRITE(6,1700) ER
      WRITE(6,1800) EA
      CALL DOLNLV(FOLNLV,XA,XB,IN,IC,BN,X,L,ER,EA,Y,WK,IERR)
      WRITE(6,1900)
      WRITE(6,2000) IERR
      WRITE(6,2100)
      WRITE(6,2200) ((I,J,Y(I,J),J=0,2),I=1,L)
      STOP
!
 1000 FORMAT(' ',/,2X,'*** DOLNLV ***',/,&
      4X,'* LINEAR ORDINARY DIFFERENTIAL EQUATION *',/,&
      6X,'Y''''+2*X*Y+X = 0.0',/,&
      3X,'** INPUT **')
 1100 FORMAT(6X,'INTERVAL = (',F4.1,',',F4.1,' )')
 1200 FORMAT(6X,'(BOUNDARY CONDITIONS)',/,&
      6X,'POSITION',5X,'ORDER OF Y',5X,'VALUE')
 1300 FORMAT(9X,I2,12X,I2,8X,F6.1)
 1400 FORMAT(6X,'(POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)')
 1500 FORMAT(6X,'X(',I2,')= ',F4.1)
 1600 FORMAT(6X,'NUMBER OF POINTS WHERE APPROXIMATE VALUES ',&
      'ARE COMPUTED =  ',I2)
 1700 FORMAT(6X,'REQUIRED LOCAL RELATIVE PRECISION = ',D8.1)
 1800 FORMAT(6X,'REQUIRED LOCAL ABSOLUTE PRECISION = ',D8.1)
 1900 FORMAT(3X,'** OUTPUT **')
 2000 FORMAT(5X,'IERR = ',I4)
 2100 FORMAT(5X,'(APPROXIMATE VALUES)')
 2200 FORMAT(5X,'Y(',I2,',',I2,') = ',D17.10)
!
      END


      SUBROUTINE FOLNLV(X,AL)
      REAL(8) X,AL
      DIMENSION AL(4)
!
      AL(1)=1.0D0
      AL(2)=0.0D0
      AL(3)=2.0D0*X
      AL(4)=X
      RETURN
      END
```

(d) Output results

```
 *** DOLNLV ***
   * LINEAR ORDINARY DIFFERENTIAL EQUATION *
     Y''+2*X*Y+X = 0.0
  ** INPUT **
     INTERVAL = ( 0.0, 1.0 )
     (BOUNDARY CONDITIONS)
     POSITION     ORDER OF Y     VALUE
         0            0           1.0
         1            1           2.0
     (POINTS WHERE APPROXIMATE VALUES ARE COMPUTED)
     X( 1)=  0.0
     X( 2)=  0.2
     X( 3)=  0.4
     X( 4)=  0.6
     X( 5)=  0.8
     X( 6)=  1.0
     NUMBER OF POINTS WHERE APPROXIMATE VALUES ARE COMPUTED =   6
     REQUIRED LOCAL RELATIVE PRECISION =  0.0D+00
     REQUIRED LOCAL ABSOLUTE PRECISION =  0.0D+00
  ** OUTPUT **
    IERR =    0
    (APPROXIMATE VALUES)
    Y( 1, 0) =  0.1000000000D+01
    Y( 1, 1) =  0.8544827835D+01
    Y( 1, 2) =  0.0000000000D+00
    Y( 2, 0) =  0.2702689947D+01
    Y( 2, 1) =  0.8439349772D+01
    Y( 2, 2) = -0.1281075979D+01
    Y( 3, 0) =  0.4349720445D+01
    Y( 3, 1) =  0.7944231536D+01
    Y( 3, 2) = -0.3879776356D+01
    Y( 4, 0) =  0.5837763648D+01
    Y( 4, 1) =  0.6811782134D+01
    Y( 4, 2) = -0.7605316378D+01
    Y( 5, 0) =  0.7019211479D+01
    Y( 5, 1) =  0.4854783090D+01
    Y( 5, 2) = -0.1203073837D+02
    Y( 6, 0) =  0.7719439396D+01
    Y( 6, 1) =  0.2000000000D+01
    Y( 6, 2) = -0.1643887879D+02
```

2.4 INTEGRAL EQUATIONS

2.4.1 DOIEF2, ROIEF2
Fredholm's Integral Equation of the Second Kind

(1) **Function**

DOIEF2 or ROIEF2 uses Gauss' integral formula to solve Fredholm's integral equation of the second kind

$$y(t) - \int_a^b K(t,x)y(x)dx = f(t)$$

to obtain a value $y(t)$ at an arbitrary point $t = t_i$ in the interval

$$-0.4984|a-b| + \frac{a+b}{2} \leq t_i \leq 0.4984|a-b| + \frac{a+b}{2} \quad (i = 1, 2, \cdots, n)$$

according to interpolation using a cubic spline function.

(2) **Usage**

Double precision:

```
CALL DOIEF2  (FF, FK, XA, XB, TI, N, Y, IERR)
```

Single precision:

```
CALL ROIEF2  (FF, FK, XA, XB, TI, N, Y, 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	FF	{ D / R }	−	Input	Name of function subprogram FF(T) that defines the known function $f(t)$
2	FK	{ D / R }	−	Input	Name of function subprogram FK(T, X) that defines the regular kernel $K(t,x)$
3	XA	{ D / R }	1	Input	Lower bound a of the integration interval
4	XB	{ D / R }	1	Input	Upper bound b of the integration interval
5	TI	{ D / R }	N	Input	Points t_i where approximate solutions are obtained
6	N	I	1	Input	Number of calculation points n

No.	Argument	Type	Size	Input/ Output	Contents
7	Y	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	N	Output	Approximate solutions $y(t_i)$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) XA≠XB

(b) N>1

(c) $-0.4984|\text{XA}-\text{XB}|+\dfrac{\text{XA}+\text{XB}}{2}\leq \text{TI(i)}\leq 0.4984|\text{XA}-\text{XB}|+\dfrac{\text{XA}+\text{XB}}{2}\quad (\text{i}=1,\cdots,\text{N})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied.	$\text{Y(i)}=\text{FF(TI(i))}\ (\text{i}=1,\cdots,\text{N})$
1100	Restriction (c) was not satisfied.	An extrapolated value is output using spline coefficients at the endpoint.
3000	Restriction (b) was not satisfied.	Processing is aborted.
4000	An error occurred when solving the simultaneous linear equations.	

(6) **Notes**

(a) The actual name of function subprogram FF(T) and FK(T, X) must be declared using an EXTERNAL statement in the user program, and the actual function subprogram must be created. This function subprogram FF(T) and FK(T, X) (in double-precision) should be created as follows:

```
REAL(8) FUNCTION FF(T)
REAL(8) T
FF=f(t)
RETURN
END
REAL(8) FUNCTION FK(T, X)
REAL(8) T, X
FK=K(t,x)
RETURN
END
```

where the following correspondences are assumed.
$t \leftrightarrow$T, $x \leftrightarrow$X

(7) **Example**

(a) Problem
Solve the following integral equation

$$y(t)-\int_1^2 (x+t)y(x)dx=t$$

for x =1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8 and 1.9.

(b) Input data

Name of function subprogram FF(T): FOIEF2, name of function subprogram FK(T, X): GOIEF2, XA=1.0, XB=2.0, TI and N=9.

(c) Main program

```
      PROGRAM BOIEF2
! *** EXAMPLE OF DOIEF2 ***
      INTEGER NA
      PARAMETER ( NA=10 )
      REAL(8) XA,XB,FOIEF2,GOIEF2,Y(NA),T(NA)
      INTEGER N,IERR,I
      EXTERNAL FOIEF2,GOIEF2
!
      READ(5,*)XA
      READ(5,*)XB
      READ(5,*)N
      DO 10 I=1,N
        READ(5,*)T(I)
   10 CONTINUE
      WRITE(6,1000)
      WRITE(6,1100)XA,XB
      CALL DOIEF2(FOIEF2,GOIEF2,XA,XB,T,N,Y,IERR)
      WRITE(6,2000)IERR
      DO 20 I=1,N
        WRITE(6,3000)I,T(I),Y(I)
   20 CONTINUE
      STOP
!
 1000 FORMAT(' ',/,/,' *** DOIEF2 ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'LIMITS OF INTEGRATION',/,&
             8X,'XA = ',F8.4,/,&
             8X,'XB = ',F8.4)
 2000 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5,/,&
             6X,'SOLUTION',/,&
             8X,'I',9X,'T(I)',11X,'Y(I)')
 3000 FORMAT(6X,I3,2F15.8)
      END

      REAL(8) FUNCTION FOIEF2(T)
      REAL(8) T
      FOIEF2=T
      RETURN
      END

      REAL(8) FUNCTION GOIEF2(T,X)
      REAL(8) X,T
      GOIEF2=X+T
      RETURN
      END
```

(d) Output results

```
*** DOIEF2 ***
 ** INPUT **
     LIMITS OF INTEGRATION
       XA =   1.0000
       XB =   2.0000
 ** OUTPUT **
     IERR =     0
     SOLUTION
       I          T(I)           Y(I)
       1     1.10000000    -0.85600000
       2     1.20000000    -0.83200000
       3     1.30000000    -0.80800000
       4     1.40000000    -0.78400000
       5     1.50000000    -0.76000000
       6     1.60000000    -0.73600000
       7     1.70000000    -0.71200000
       8     1.80000000    -0.68800000
       9     1.90000000    -0.66400000
```

2.4.2 DOIEV1, ROIEV1
Volterra's Integral Equation of the First Kind

(1) **Function**

DOIEV1 or ROIEV1 uses Maclaurin's formula to solve Volterra's integral equation of the first kind

$$\int_a^t K(t,x)y(x)dx = f(t)$$

to obtain a value $y(x)$ at an arbitrary point $x = x_i \ \ (i = 1, 2, \cdots, n)$ according to interpolation using a cubic spline function.

(2) **Usage**

Double precision:

CALL DOIEV1 (FF, FK, XA, XB, M, XI, N, Y, W, IERR)

Single precision:

CALL ROIEV1 (FF, FK, XA, XB, M, XI, N, Y, 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	FF	{D / R}	−	Input	Name of function subprogram FF(T) that defines the known function $f(t)$
2	FK	{D / R}	−	Input	Name of function subprogram FK(T, X) that defines the regular kernel $K(t,x)$
3	XA	{D / R}	1	Input	Lower bound a of the integration interval (See Notes (b))
4	XB	{D / R}	1	Input	Upper bound of points where approximate solutions are calculated (See Notes (b))
5	M	I	1	Input	Number of subdivisions of the integration interval (See Notes (b))
6	XI	{D / R}	N	Input	Points x_i where approximate solutions are obtained
7	N	I	1	Input	Number of calculation points n
8	Y	{D / R}	N	Output	Approximate solutions $y(x_i)$
9	W	{D / R}	$5 \times M + 2$	Work	Work area
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N>1

(b) M>1

(c) XA<XB

(d) $\mathrm{XA} + \dfrac{\mathrm{XB} - \mathrm{XA}}{2 \times \mathrm{M}} < \mathrm{XI(i)} < \mathrm{XB} \quad (\mathrm{i} = 1, \cdots, \mathrm{N})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (d) was not satisfied.	An extrapolated value is output using spline coefficients at the endpoint.
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 actual name of function subprogram FF(T) and FK(T, X) must be declared using an EXTERNAL statement in the user program, and the actual function subprogram must be created. This function subprogram FF(T) and FK(T, X) (in double-precision) should be created as follows:

```
REAL(8) FUNCTION FF(T)
REAL(8) T
FF=f(t)
RETURN
END

REAL(8) FUNCTION FK(T, X)
REAL(8) T, X
FK=K(t,x)
RETURN
END
```

where the following correspondences are assumed.
$t \leftrightarrow$T, $x \leftrightarrow$X

(b) Use $\dfrac{\mathrm{XB} - \mathrm{XA}}{\mathrm{M}}$ as the step size h for determining the integral calculation interval. (See Section 2.1.2)

(7) **Example**

(a) Problem
Solve the following integral equation

$$t = \int_0^t (1 + t - x) y(x) dx$$

for x =0.025, 0.075, 0.125, 0.175, 0.225, 0.275, 0.325, 0.375, 0.425, 0.475 and 0.500.

(b) Input data
Name of function subprogram FF(T): FOIEV1, name of function subprogram FK(T, X): GOIEV1, XA=0.0, XB=0.5, XI, N=11 and M=11.

(c) Main program

```
      PROGRAM BOIEV1
! *** EXAMPLE OF DOIEV1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      INTEGER NA,MA
      PARAMETER ( NA=11,MA=11 )
      REAL(8) XA,T(NA),Y(NA),W(5*MA+2),FOIEV1,GOIEV1
      INTEGER N,M,IERR
      EXTERNAL FOIEV1,GOIEV1
!
      READ(5,*)XA
      READ(5,*)XB
      READ(5,*)M
      READ(5,*)N
      DO 10 I=1,N
        READ(5,*)T(I)
   10 CONTINUE
      WRITE(6,1000)
      WRITE(6,1100)XA,XB
      CALL DOIEV1(FOIEV1,GOIEV1,XA,XB,M,T,N,Y,W,IERR)
      WRITE(6,2000)IERR
      DO 20 I=1,N
         WRITE(6,3000)I,T(I),Y(I)
   20 CONTINUE
      STOP
!
 1000 FORMAT(' ',/,/,' *** DOIEV1 ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'LIMITS OF INTEGRATION',/,&
             8X,'XA = ',F8.4,/,&
             8X,'XB = ',F8.4)
 2000 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5,/,&
             6X,'SOLUTION',/,&
             8X,'I',9X,'T(I)',11X,'Y(I)')
 3000 FORMAT(6X,I3,2F15.8)
      END

      REAL(8) FUNCTION FOIEV1(T)
      REAL(8) T
      FOIEV1=T
      RETURN
      END

      REAL(8) FUNCTION GOIEV1(T,X)
      REAL(8) T,X
      GOIEV1=1+T-X
      RETURN
      END
```

(d) Output results

```
*** DOIEV1 ***
 ** INPUT **
     LIMITS OF INTEGRATION
       XA =   0.0000
       XB =   0.5000
 ** OUTPUT **
     IERR =     0
     SOLUTION
       I         T(I)           Y(I)
       1     0.02500000     0.97560699
       2     0.07500000     0.92786186
       3     0.12500000     0.88204941
       4     0.17500000     0.83958255
       5     0.22500000     0.79866884
       6     0.27500000     0.75982328
       7     0.32500000     0.72253125
       8     0.37500000     0.68704092
       9     0.42500000     0.65407133
      10     0.47500000     0.62191160
      11     0.50000000     0.60648528
```

2.5 PARTIAL DIFFERENTIAL EQUATIONS

2.5.1 DOPDH2, ROPDH2
Two-Dimensional Inhomogeneous Helmholtz Equation

(1) **Function**

This subroutine uses a two-dimensional five-point difference in a given rectangular area to solve the following inhomogeneous Helmholtz equation.

$$u_{xx} + u_{yy} + \lambda u = f(x, y)$$

(2) **Usage**

Double precision:

CALL DOPDH2 (DLMD, FUNC, NX, NY, XL, XU, YL, YU, S1, S2, S3, S4, IMAX, EPS, U, IMX, ISW, IW, W, IERR)

Single precision:

CALL ROPDH2 (DLMD, FUNC, NX, NY, XL, XU, YL, YU, S1, S2, S3, S4, IMAX, EPS, U, IMX, ISW, IW, 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	DLMD	{D / R}	1	Input	Value of the coefficient of u in the difference equation λ.
2	FUNC	–	–	Input	Name of the subroutine subprogram F(X,Y) that defines the differential equation as a function of x and y
3	NX	I	1	Input	Number of division in X-direction n_x
4	NY	I	1	Input	Number of division in Y-direction n_y
5	XL	{D / R}	1	Input	The value of X along the lower point of the domain.
6	XU	{D / R}	1	Input	The value of X along the upper point of the domain.
7	YL	{D / R}	1	Input	The value of Y along the upper point of the domain.
8	YU	{D / R}	1	Input	The value of Y along the upper point of the domain.
9	S1	{D / R}	NX+1	Input	Value of Dirichlet boundary condition of bottom edge (when ISW(1) $\neq$ 1)When ISW(1) = 1, the area is only allocated.
10	S2	{D / R}	NY+1	Input	Value of Dirichlet boundary condition of right edge (when ISW(2) $\neq$ 1)When ISW(2) = 1, the area is only allocated.
11	S3	{D / R}	NX+1	Input	Value of Dirichlet boundary condition of top edge (when ISW(3) $\neq$ 1)When ISW(3) = 1, the area is only allocated.
12	S4	{D / R}	NY+1	Input	Value of Dirichlet boundary condition of left edge (when ISW(4) $\neq$ 1)When ISW(4) = 1, the area is only allocated.

No.	Argument	Type	Size	Input/ Output	Contents
13	IMAX	I	1	Input	Maximum number of iterations when using an iterative method to solve simultaneous linear equations.
14	EPS	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Input	Truncation residual norm $\parallel \boldsymbol{b} - A\boldsymbol{u} \parallel / \parallel \boldsymbol{b} \parallel$ **Specifiable range:** ≥Underflow decision value (See Note (e))
15	U	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Solution matrix$u(x_i, y_j)$. $\mathrm{U}(i,j) = u(x_L + i(x_U - x_L)/n_x, y_L + j(y_U - y_L)/n_y)$ **Size**: $0:\mathrm{IMX}, 0:\mathrm{NY}$
16	IMX	I	1	Input	Adjustable dimension of array U
17	ISW	I	4	Input	Boundary condition selection switch ISW(1) =0: Add Dirichlet condition to bottom edge =1: Add Neumann condition to bottom edge ISW(2) =0: Add Dirichlet condition to right edge =1: Add Neumann condition to right edge ISW(3) =0: Add Dirichlet condition to top edge =1: Add Neumann condition to top edge ISW(4) =0: Add Dirichlet condition to left edge =1: Add Neumann condition to left edge
18	IW	I	7	Work	Work area
19	W	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $13 \times (\mathrm{NX}+1) \times (\mathrm{NY}+1) + 5$
20	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX $\geq$ 2, NY $\geq$ 2

(b) For the boundary conditions to be added to the edges, the Dirichlet condition must be added to at least one edge.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1400	A diagonal element produced by ILU decomposition became less than ϵ times the absolute value of the original diagonal element. (See Note (c))	Replaces diagonal element produced by ILU decomposition with (original diagonal element) , $\times\epsilon$ and processing continues.
2000	Maximum number of iterations has been reached.	Processing continues without performing acceleration.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	The norm of the right-hand side vector is greater than $\sqrt{\text{overflow decision value}}$. (See Notes (d), (e))	
4100	The norm of the residual is greater than $\sqrt{\text{overflow decision value}}$. (See Notes (d), (e))	
4200	$\parallel (\boldsymbol{r}, \boldsymbol{r}^*) \parallel$ is smaller than underflow decision value.	
4210	$\parallel (\boldsymbol{r}, \boldsymbol{r}^*) \parallel$ is greater than overflow decision value.	
4300	$\parallel (A\boldsymbol{p}, \boldsymbol{p}^*) \parallel$ is smaller than underflow decision value.	
4310	$\parallel (A\boldsymbol{p}, \boldsymbol{p}^*) \parallel$ is greater than overflow decision value.	
5000	Any of diagonal element has absolute value smaller than underflow decision value.	

(6) **Notes**

(a) The actual name of subroutine F(X, Y), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created. For the high-order simultaneous ordinary differential equations:

$$u_{xx} + u_{yy} + \lambda u = f(x, y)$$

this subroutine F(X, Y) (in double-precision) should be created as follows:

```
SUBROUTINE FUNC(X, Y)
REAL(8) X, Y
FUNC=f(x,y)
```

```
RETURN
END
```

where the following correspondences are assumed.
$x \leftrightarrow$X,$y \leftrightarrow$Y
Example:

$$f(x, y) = 6x + 6y$$

Therefore, define the subroutine as follows:

```
SUBROUTINE FUNC(X, Y)
REAL(8) X, Y
FUNC=6.0D0*X + 6.0D0*Y
RETURN
END
```

(b) For the Dirichlet conditions S1, S2, S3, and S4, NX + 1 (S1, S3) values and NY + 1 (S1, S3) values, excluding values at the vertices of the virtual grid, are stored in arrays sequentially in ascending order of coordinate values.

(c) The singularity prevention constant ϵ is set to 0.1. (See the high-speed function version (4.1.2.2).)

(d) The L^2 norm shown below is used for the norm.

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{\mathrm{N}} {r_i}^2 \right)^{1/2} \quad \text{where, } \boldsymbol{r} = (r_1, r_2, r_3, \cdots, r_{\mathrm{N}})^T$$

(e) (Maximum value$\times 10^{-3}$) is set for the overflow decision value and (Minimum positive value$\times 10^{3}$) is set for the underflow decision value.

(7) **Example**

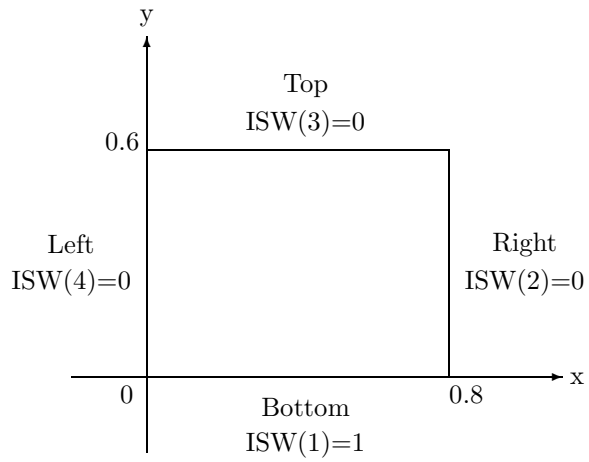
(a) Problem (two-dimensional Poisson equation)
Solve the following two-dimensional Poisson equation :

$$u_{xx} + u_{yy} = f(x, y)$$

under the boundary conditions below.

$$\begin{cases} \frac{\partial u}{\partial y} = 0 & \text{(on the bottom edge)} \\ u = 1 & \text{(on the other edges)} \end{cases}$$

The given two-dimensional area is discretized by an NX $\times$ NY orthogonal grid as shown in the following figure.



(b) Input data

DLMD=0.0,

NX=40, NY=50,

XL=0.0, XU=0.6,

YL=0.0, YU=0.8,

IMAX=200, EPS=1.D−10 and

ISW=(1, 0, 0, 0).

(c) Main program

```
      PROGRAM BOPDH2
! *** BOPDH2 - FINITE DIFFERENCE METHOD  ( POISSON EQUATION ) ***
      PARAMETER ( IMX=70 , JMX=70                                    )
      PARAMETER ( INMX=(IMX+1)*(JMX+1)                               )
!
      INTEGER          IMAX, IERR, NX, NY, ISW(4), I, J, IWK(7)
      REAL(8) S1(IMX+1), S2(JMX+1), S3(IMX+1), S4(JMX+1)
      REAL(8) X0, X1, Y0, Y1, EPS, DLMD, FUNC
      REAL(8) U(0:IMX,0:JMX), WK(INMX*13+5)
      DATA ONE/1.0D0/
!
      EXTERNAL         FUNC
!
      READ (5,   *) ( ISW(I) ,I=1,4 )
      READ (5,   *) DLMD
      READ (5,   *) NX,NY
      READ (5,   *) X0,X1
      READ (5,   *) Y0,Y1
      READ (5,   *) IMAX
      READ (5,   *) EPS
!
      WRITE(6,1000)
      WRITE(6,1400) 'ISW  ',( ISW(I) ,I=1,4 )
      WRITE(6,1200) 'DLMD ',DLMD
      WRITE(6,1100) 'NX   ',NX
      WRITE(6,1100) 'NY   ',NY
      WRITE(6,1300) 'X    ',X0,X1
      WRITE(6,1300) 'Y    ',Y0,Y1
      WRITE(6,1400) 'IMAX ',IMAX
      WRITE(6,1300) 'EPS  ',EPS
!
!CCCC DIRICRET CONDITION
!
      DO 110 I=1,NX+1
         S1(I) = ONE
         S3(I) = ONE
 110  CONTINUE
      DO 120 J=1,NY+1
         S2(J) = ONE
         S4(J) = ONE
 120  CONTINUE
!
      CALL DOPDH2&
         ( DLMD,FUNC,NX,NY,X0,X1,Y0,Y1,S1,S2,S3,S4,IMAX ,EPS ,&
           U(0,0),IMX,ISW, IWK, WK ,IERR                               )
!
      WRITE(6,1700)
      WRITE(6,1100) 'IERR',IERR
      WRITE(6,1500)
!
      JSP = NY/5
```

```
      ISP = NX/4
      IF( NX/ISP .GT. 4 ) THEN
         ISP = ISP + 1
      ENDIF
      DO 130 J=NY,0,-JSP
         WRITE(6,1600) J,( U(I,J),I=0,NX,ISP )
 130  CONTINUE
      WRITE(6,2000) ( I,I=0,NX,ISP )
      WRITE(6,2100)
!
 1000 FORMAT(/, 1X, '   ', '*** DOPDH2 ***',/,/,&
                1X, '   ', '** INPUT **',/)
 1100 FORMAT(  1X,'     ', A, '  ', '=', '  ', I5                    )
 1200 FORMAT(  1X,'     ', A, '  ', '=', '  ', F5.2                  )
 1300 FORMAT(  1X,'     ', A, '  ', '=', 100(2X,F5.2)                )
 1400 FORMAT(  1X,'     ', A, '  ', '=', 7(2X,I5)                    )
 1500 FORMAT(/,/,1X, '   ', '** SOLUTION U(I,J) **',/)
 1600 FORMAT(  1X,'   ',I3,'  ',10(1X,1PD12.5)                       )
 1700 FORMAT(/,/,1X,'   ', '** OUTPUT **',/)
 1900 FORMAT(  1X,'          ',100('  ',F7.2)                        )
 2000 FORMAT(  1X,'Y-DIRECTION','   ',5(I3,10X)                      )
 2100 FORMAT(  1X,'             X-DIRECTION'                         )
!
      STOP
      END
!
      REAL(8) FUNCTION FUNC( X, Y )
      REAL(8) X,Y
      FUNC = 6.0D0*X + 6.0D0*Y
      RETURN
      END
```

(d) Output results

```
   *** DOPDH2 ***

   ** INPUT **

     ISW    =      1      0      0      0
     DLMD   =   0.00
     NX     =     40
     NY     =     50
     X      =   0.00   0.60
     Y      =   0.00   0.80
     IMAX   =    200
     EPS    =   0.00


   ** OUTPUT **

     IERR  =      0


   ** SOLUTION U(I,J) **

    50    9.97411D-01  9.84991D-01  9.80409D-01  9.81787D-01  9.96118D-01
    40    9.87165D-01  8.96801D-01  8.60071D-01  8.77950D-01  9.82286D-01
    30    9.85582D-01  8.75213D-01  8.27226D-01  8.53046D-01  9.80175D-01
    20    9.86949D-01  8.81985D-01  8.33543D-01  8.59141D-01  9.81440D-01
    10    9.89186D-01  8.97396D-01  8.51976D-01  8.74415D-01  9.83657D-01
     0    9.90686D-01  9.06994D-01  8.63419D-01  8.83990D-01  9.85153D-01
Y-DIRECTION     0           10           20           30           40
           X-DIRECTION
```

2.5.2 DOPDH3, ROPDH3
Three-Dimensional Inhomogeneous Helmholtz Equation

(1) **Function**

This subroutine uses a three-dimensional seven-point difference in a given rectangular area to solve the following inhomogeneous Helmholtz equation.

$$u_{xx} + u_{yy} + u_{zz} + \lambda u = f(x, y, z)$$

(2) **Usage**

Double precision:

CALL DOPDH3 (DLMD, FUNC, NX, NY, NZ, XL, XU, YL, YU, ZL, ZU, S1, S2, S3, S4, S5, S6, IMX, JMX, IMAX, EPS, U, ISW, IW, W, IERR)

Single precision:

CALL ROPDH3 (DLMD, FUNC, NX, NY, NZ, XL, XU, YL, YU, ZL, ZU, S1, S2, S3, S4, S5, S6, IMX, JMX, IMAX, EPS, U, ISW, IW, 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	DLMD	{ D / R }	1	Input	Value of the coefficient of u in the difference equation λ.
2	FUNC	–	–	Input	Name of the subroutine subprogram F(X,Y,Z) that defines the differential equations as a function of x, y and z.
3	NX	I	1	Input	Number of division in X-direction n_x
4	NY	I	1	Input	Number of division in Y-direction n_y
5	NZ	I	1	Input	Number of division in Z-direction n_z
6	XL	I	1	Input	The value of X along the lower point of the domain.
7	XU	I	1	Input	The value of X along the upper upper of the domain.
8	YL	I	1	Input	The value of Y along the lower point of the domain.
9	YU	I	1	Input	The value of Y along the upper point of the domain.
10	ZL	I	1	Input	The value of z along the lower point of the domain.
11	ZU	I	1	Input	The value of Y along the upper point of the domain.
12	S1	I	See Contents	Input	Value of Dirichlet boundary condition of bottom surface (when ISW(1) $\neq$ 1)When ISW(1) = 1, the area is only allocated. **Size**: IMX + 1, NZ + 1

No.	Argument	Type	Size	Input/ Output	Contents
13	S2	I	See Contents	Input	Value of Dirichlet boundary condition of right surface (when ISW(2) $\neq 1$)When ISW(2) $= 1$, the area is only allocated. **Size**: JMX $+ 1$, NZ $+ 1$
14	S3	I	See Contents	Input	Value of Dirichlet boundary condition of top surface (when ISW(3) $\neq 1$)When ISW(3) $= 1$, the area is only allocated. **Size**: IMX $+ 1$, NZ $+ 1$
15	S4	I	See Contents	Input	Value of Dirichlet boundary condition of left surface (when ISW(4) $\neq 1$)When ISW(4) $= 1$, the area is only allocated. **Size**: JMX $+ 1$, NZ $+ 1$
16	S5	I	See Contents	Input	Value of Dirichlet boundary condition of front surface (when ISW(5) $\neq 1$)When ISW(5) $= 1$, the area is only allocated. **Size**: IMX $+ 1$, NY $+ 1$
17	S6	I	See Contents	Input	Value of Dirichlet boundary condition of back surface (when ISW(6) $\neq 1$)When ISW(6) $= 1$, the area is only allocated. **Size**: IMX $+ 1$, NY $+ 1$
18	IMX	I	1	Input	Adjustable dimension of array S1, S3, S5, S6
19	JMX	I	1	Input	Adjustable dimension of array S2, S4
20	IMAX	I	1	Input	Maximum number of iterations when using an iterative method to solve simultaneous linear equations
21	EPS	I	1	Input	Truncation residual norm $\parallel \boldsymbol{b} - A\boldsymbol{u} \parallel / \parallel \boldsymbol{b} \parallel$ **Specifiable range:** $\geq$Underflow decision value (See Note (e))
22	U	$\left\{\begin{matrix}\text{D}\\ \text{R}\end{matrix}\right\}$	See Contents	Output	Solution matrix$u(x_i, y_j, z_k)$ U$(i, j, k) = u\ (x_L + i(x_U - x_L)/n_x\ ,\ y_L + j(y_U - y_L)/n_y\ ,\ z_L + k(z_U - z_L)/n_z)$ **Size**: 0 : IMX, 0 : JMX, 0 : NZ

No.	Argument	Type	Size	Input/ Output	Contents
23	ISW	I	6	Input	Boundary condition selection switch ISW(1) =0 : Add Dirichlet condition to bottom surface. =1 : Add Neumann condition to bottom surface. ISW(2) =0 : Add Dirichlet condition to right surface. =1 : Add Neumann condition to right surface. ISW(3) =0 : Add Dirichlet condition to top surface. =1 : Add Neumann condition to top surface. ISW(4) =0 : Add Dirichlet condition to left surface. =1 : Add Neumann condition to left surface. ISW(5) =0 : Add Dirichlet condition to front surface. =1 : Add Neumann condition to front surface. ISW(6) =0 : Add Dirichlet condition to back surface. =1 : Add Neumann condition to back surface.
24	IW	I	7	Work	Work area
25	W	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $15 \times (\mathrm{NX}+1) \times (\mathrm{NY}+1) \times (\mathrm{NZ}+1)+5$
26	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NX} \geq 2$, $\mathrm{NY} \geq 2$, $\mathrm{NZ} \geq 2$

(b) For the boundary conditions to be added to the edges, the Dirichlet condition must be added to at least one edge.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1400	A diagonal element produced by ILU decomposition became less than ϵ times the absolute value of the original diagonal element. (See Note (c))	Replaces diagonal element produced by ILU decomposition with (original diagonal element), $\times\epsilon$ and processing continues.
2000	Maximum number of iterations has been reached.	Processing continues without performing acceleration.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	The norm of right-hand side vector is greater than $\sqrt{\text{overflow decision value}}$. (See Notes (d), (e))	
4100	The norm of the residual is greater than $\sqrt{\text{overflow decision value}}$. (See Notes (d), (e))	
4200	$\parallel (\boldsymbol{r}, \boldsymbol{r^*}) \parallel$ is smaller than underflow decision value.	
4210	$\parallel (\boldsymbol{r}, \boldsymbol{r^*}) \parallel$ is greater than overflow decision value.	
4300	$\parallel (A\boldsymbol{p}, \boldsymbol{p^*}) \parallel$ is smaller than underflow decision value.	
4310	$\parallel (A\boldsymbol{p}, \boldsymbol{p^*}) \parallel$ is greater than overflow decision value.	
5000	Any of diagonal element has absolute value smaller than underflow decision value.	

(6) **Notes**

(a) The actual name of subroutine F(X, Y), that defines the differential equations, must be declared using an EXTERNAL statement in the user program, and the actual subroutine must be created. For the high-order simultaneous ordinary differential equations:

$$u_{xx} + u_{yy} + u_{zz} + \lambda u = f(x, y, z)$$

this subroutine F(X, Y, Z) (in double-precision) should be created as follows:

```
SUBROUTINE FUNC(X, Y, Z)
REAL(8) X, Y, Z
FUNC=f(x,y,z)
RETURN
END
```

where the following correspondences are assumed.
$x \leftrightarrow$X,$y \leftrightarrow$Y, $z \leftrightarrow$Z

Example:

$$f(x, y) = 6x + 6y + 6z$$

Therefore, define the subroutine as follows:

```
SUBROUTINE FUNC(X, Y, Z)
REAL(8) X, Y, Z
FUNC=6.0D0*X + 6.0D0*Y + 6.0D0*Z
RETURN
END
```

(b) For the Dirichlet conditions S1, S2, S3, S4, S5, and S6, $(NX + 1) \times (NZ + 1)$ (S1, S3) values and $(NY + 1) \times (NZ + 1)$(S2, S4) values and $(NX + 1) \times (NY + 1)$(S5, S6) value, excluding values at the vertices of the virtual grid, are stored in arrays sequentially in ascending order of coordinate values.

(c) The singularity prevention constant ϵ is set to 0.1. (See the high-speed function version (4.1.2.2).)

(d) The L^2 norm shown below is used for the norm.

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{N} {r_i}^2 \right)^{1/2} \quad where, \boldsymbol{r} = (r_1, r_2, r_3, \cdots, r_N)^T$$

(e) (Maximum value$\times 10^{-3}$) is set for the overflow decision value and (Minimum positive value$\times 10^{3}$) is set for the underflow decision value.

(7) **Example**

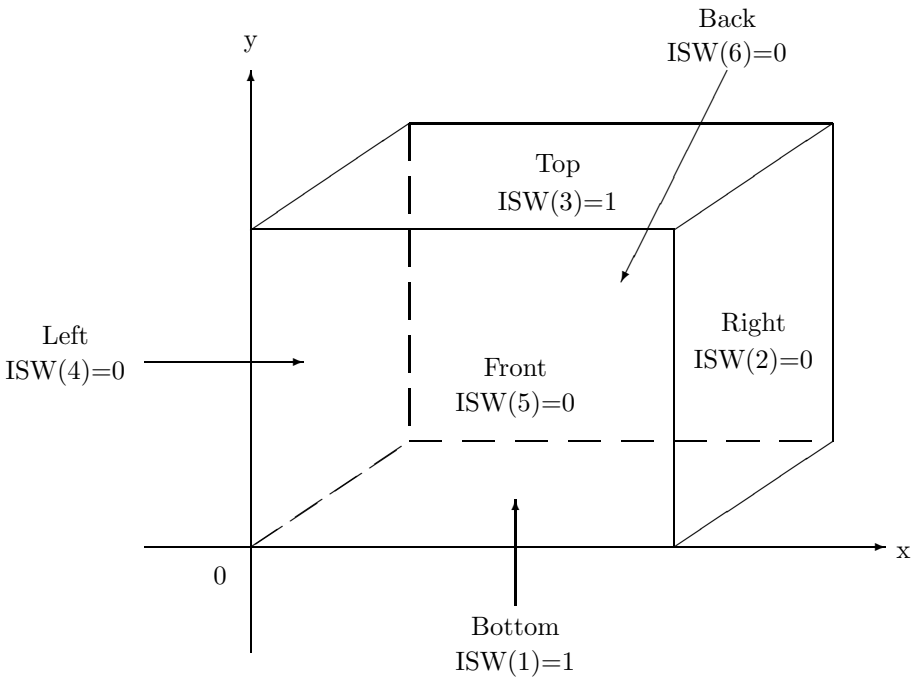
(a) Problem (three-dimensional Poisson equation)
Solve the following three-dimensional Poisson equation :

$$u_{xx} + u_{yy} + u_{zz} = f(x, y, z)$$

under the boundary conditions below.

$$\begin{cases} \frac{\partial u}{\partial y} = 0 & \text{(on the bottom and top surface)} \\ u = 1 & \text{(on the other surfaces)} \end{cases}$$

The given three-dimensional area is discretized by an $NX \times NY \times NZ$ orthogonal grid as shown in the following figure.



(b) Input data

DLMD=0.0,

NX=20, NY=30, NZ=40,

XL=0.0, XU=0.8,

YL=0.0, YU=0.6,

ZL=0.0, ZU=1.2,

IMAX=200, EPS=1.D−10 and

ISW=(1, 0, 1, 0, 0, 0).

(c) Main program

```
      PROGRAM BOPDH3
! *** BOPDH2 - FINITE DIFFERENCE METHOD ***
      PARAMETER ( IMX=90 , JMX=90 , KMX=90                    )
      PARAMETER ( INMX=(IMX+1)*(JMX+1)*(KMX+1)                )
!
      INTEGER          IMAX, IERR, NX, NY, NZ, ISW(6), I, J, K, IWK(7)
      REAL(8) S1(IMX+1, KMX+1), S2(JMX+1, KMX+1)
      REAL(8) S3(IMX+1, KMX+1), S4(JMX+1, KMX+1)
      REAL(8) S5(IMX+1, JMX+1), S6(IMX+1, JMX+1)
      REAL(8) X0, X1, Y0, Y1, Z0, Z1, EPS, DLMD, FUNC
      REAL(8) U(0:IMX, 0:JMX, 0:KMX), WK(INMX*15+5)
      DATA ONE/1.0D0/
!
      EXTERNAL         FUNC
!
      READ (5,   *) ( ISW(I) ,I=1,6 )
      READ (5,   *) DLMD
      READ (5,   *) NX, NY, NZ
      READ (5,   *) X0, X1
      READ (5,   *) Y0, Y1
      READ (5,   *) Z0, Z1
      READ (5,   *) IMAX
      READ (5,   *) EPS
!
      DO 130 J=1,NY+1
      DO 140 K=1,NZ+1
         S2(J,K) = ONE
         S4(J,K) = ONE
 140  CONTINUE
 130  CONTINUE
!
      DO 150 I=1,NX+1
      DO 160 J=1,NY+1
         S5(I,J) = ONE
         S6(I,J) = ONE
 160  CONTINUE
```

```
 150  CONTINUE
!
      WRITE(6,1000)
      WRITE(6,1400) 'ISW  ',( ISW(I) ,I=1,6 )
      WRITE(6,1200) 'DLMD ',DLMD
      WRITE(6,1100) 'NX   ',NX
      WRITE(6,1100) 'NY   ',NY
      WRITE(6,1100) 'NZ   ',NZ
      WRITE(6,1300) 'X    ',X0,X1
      WRITE(6,1300) 'Y    ',Y0,Y1
      WRITE(6,1300) 'Z    ',Z0,Z1
      WRITE(6,1400) 'IMAX ',IMAX
      WRITE(6,1300) 'EPS  ',EPS
!
      CALL DOPDH3&
         ( DLMD, FUNC, NX, NY, NZ, X0, X1, Y0, Y1, Z0, Z1, S1, S2, S3,&
           S4, S5, S6, IMX, JMX, IMAX, EPS, U(0,0,0), ISW,&
           IWK, WK ,IERR                                              )
!
      WRITE(6,1700)
      WRITE(6,1100) 'IERR',IERR
      WRITE(6,1500)
!
      ISP = NX/4
      JSP = NY/5
      KSP = NZ/4
      IF( NX/ISP .GT. 4 ) THEN
         ISP = ISP + 1
      ENDIF
      DO 170 K=0,NZ,KSP
         WRITE(6,2000) K
         DO 180 J=NY,0,-JSP
            WRITE(6,2300) J,( U(I,J,K),I=0,NX,ISP )
 180     CONTINUE
         WRITE(6,2100) ( I,I=0,NX,ISP )
         WRITE(6,2200)
 170  CONTINUE
!
 1000 FORMAT( /,1X, '   ', '*** DOPDH2 ***',/,/,&
                '   ', '** INPUT **',/)
 1100 FORMAT(  1X,'     ', A, '  ', '=', '  ', I5                    )
 1200 FORMAT(  1X,'     ', A, '  ', '=', '  ', F5.2                  )
 1300 FORMAT(  1X,'     ', A, '  ', '=', 100('  ',F5.2)              )
 1400 FORMAT(  1X,'     ', A, '  ', '=', 7('  ',I5)                  )
 1500 FORMAT(/,/,1X,'   ', '** SOLUTION U(I,J) **',/)
 1600 FORMAT(  1X,'     ', 1PD12.5                                   )
 1700 FORMAT(/,/,1X,'   ', '** OUTPUT **',/)
 2000 FORMAT(  1X,'  ', 'Z-DIRECTION = ',I2,/)
 2100 FORMAT(  1X,'Y-DIRECTION','   ',5(I3,'          ')            )
 2200 FORMAT(  1X,'             X-DIRECTION',/,/)
 2300 FORMAT(  1X,'   ',I3,'  ',10(' ',1PD12.5)                     )
!
      STOP
      END
!
      REAL(8) FUNCTION FUNC( X, Y, Z )
      REAL(8) X,Y,Z
      FUNC = 6.0D0*X + 6.0D0*Y + 6.0D0*Z
      RETURN
      END
```

(d) Output results

```
 *** DOPDH2 ***

 ** INPUT **

   ISW    =      1      0      1      0      0      0
   DLMD   =   0.00
   NX     =     20
   NY     =     30
   NZ     =     40
   X      =   0.00   0.80
   Y      =   0.00   1.20
   Z      =   0.00   0.90
   IMAX   =    200
   EPS    =   0.00


 ** OUTPUT **

   IERR  =      0


 ** SOLUTION U(I,J) **

 Z-DIRECTION =  0

  30    9.85345D-01  9.47888D-01  9.34063D-01  9.39395D-01  9.79702D-01
  24    9.87096D-01  9.52456D-01  9.39143D-01  9.43963D-01  9.81454D-01
  18    9.89436D-01  9.59508D-01  9.47229D-01  9.51015D-01  9.83794D-01
  12    9.91867D-01  9.67023D-01  9.55912D-01  9.58531D-01  9.86225D-01
   6    9.94206D-01  9.74074D-01  9.63996D-01  9.65581D-01  9.88564D-01
   0    9.95957D-01  9.78641D-01  9.69075D-01  9.70148D-01  9.90315D-01
```

```
Y-DIRECTION     0           5           10          15          20
            X-DIRECTION


   Z-DIRECTION = 10

    30    9.07943D-01  6.01485D-01  4.86170D-01  5.51643D-01  8.84539D-01
    24    9.15727D-01  6.29091D-01  5.18367D-01  5.79249D-01  8.92324D-01
    18    9.27990D-01  6.77228D-01  5.75911D-01  6.27387D-01  9.04587D-01
    12    9.41101D-01  7.29723D-01  6.39085D-01  6.79881D-01  9.17698D-01
     6    9.53362D-01  7.77849D-01  6.96611D-01  7.28007D-01  9.29959D-01
     0    9.61140D-01  8.05453D-01  7.28790D-01  7.55607D-01  9.37737D-01
Y-DIRECTION     0           5           10          15          20
            X-DIRECTION


   Z-DIRECTION = 20

    30    8.80482D-01  4.73611D-01  3.21678D-01  4.16701D-01  8.54904D-01
    24    8.89236D-01  5.06032D-01  3.59977D-01  4.49122D-01  8.63659D-01
    18    9.03483D-01  5.64140D-01  4.30233D-01  5.07229D-01  8.77907D-01
    12    9.18847D-01  6.27998D-01  5.07936D-01  5.71088D-01  8.93270D-01
     6    9.33091D-01  6.86087D-01  5.78167D-01  6.29177D-01  9.07514D-01
     0    9.41836D-01  7.18504D-01  6.16441D-01  6.61590D-01  9.16260D-01
Y-DIRECTION     0           5           10          15          20
            X-DIRECTION


   Z-DIRECTION = 30

    30    8.90854D-01  5.41448D-01  4.17289D-01  4.91603D-01  8.67448D-01
    24    8.98638D-01  5.69056D-01  4.49489D-01  5.19210D-01  8.75234D-01
    18    9.10902D-01  6.17194D-01  5.07034D-01  5.67349D-01  8.87497D-01
    12    9.24013D-01  6.69688D-01  5.70208D-01  6.19843D-01  9.00608D-01
     6    9.36273D-01  7.17813D-01  6.27734D-01  6.67969D-01  9.12869D-01
     0    9.44050D-01  7.45414D-01  6.59913D-01  6.95570D-01  9.20648D-01
Y-DIRECTION     0           5           10          15          20
            X-DIRECTION


   Z-DIRECTION = 40

    30    9.78985D-01  9.32004D-01  9.16719D-01  9.23510D-01  9.73343D-01
    24    9.80737D-01  9.36573D-01  9.21799D-01  9.28079D-01  9.75095D-01
    18    9.83077D-01  9.43625D-01  9.29886D-01  9.35131D-01  9.77435D-01
    12    9.85507D-01  9.51140D-01  9.38569D-01  9.42646D-01  9.79865D-01
     6    9.87847D-01  9.58191D-01  9.46653D-01  9.49697D-01  9.82205D-01
     0    9.89597D-01  9.62758D-01  9.51731D-01  9.54264D-01  9.83956D-01
Y-DIRECTION     0           5           10          15          20
            X-DIRECTION
```

Chapter 3

NUMERICAL DIFFERENTIALS

3.1 INTRODUCTION

This chapter describes subroutines that obtain numerical differentials of functions at given points.
This library provides the following subroutine for obtaining numerical differentials of a function of one variable.

(1) Numerical differentials of a function
This subroutine obtains arbitrary order differential values of a function of one variable at a given point.
This library provides the following subroutines for obtaining numerical differentials of a single function of many variables.

(2) Gradient vector of a function of many variables

(3) Hessian matrix of a function of many variables
The subroutine in (2) obtains the first order partial differential values (gradient vector) of a function of many variables at given points.
The subroutine in (3) obtains the second order partial differential values (Hessian matrix) of a function of many variables at given points.
This library provides the following subroutine for obtaining numerical differentials of multiple functions of many variables.

(4) Jacobian matrix of multiple functions of many variables
This subroutine obtains the first order partial differential values (Jacobian matrix) of multiple functions of many variables at given points.

3.1.1 Notes

(1) A value that is larger than the default value should be entered as the required relative precision.

(2) The functions or subroutines, whose name passed to subroutine, are created as follows. If a subroutine that has a function name for an argument is used, the function name to be used must be declared as an actual argument in the main program by using the EXTERNAL statement.
Example:

Numerical differentials of a function (Let F has the same name in the main program and the function subprogram)

- Main program

```
    ≀
EXTERNAL F
    ≀
CALL {DQFODX / RQFODX} (F, ···)
    ≀
```

- Function subprogram

```
FUNCTION F (X)
    ≀
F = f(X)
    ≀
RETURN
END
```

Example:

Gradient vector of a function of many variables and Hessian matrix of a function of many variables (Let F has the same name in the main program and the function subprogram)

- Main program

```
    ≀
EXTERNAL F
    ≀
CALL {DQMO** / RQMO**} (F, ···)
    ≀
```

- Function subprogram

```
FUNCTION F (X)
DIMENSION X(*)
    ≀
```

$\boxed{\text{F}} = f(\text{X}(1), \cdots, \text{X}(\text{NX}))$

≀

RETURN

END

Example:

Jacobian matrix of multiple functions of many functions of many variables (Let $\boxed{\text{SUB}}$ has the same name in the main program and the subroutine)

- Main program

  ≀

  EXTERNAL $\boxed{\text{SUB}}$

  ≀

  CALL $\begin{Bmatrix}\text{DQMOJX}\\ \text{RQMOJX}\end{Bmatrix}(\boxed{\text{SUB}}, \cdots)$

  ≀

- Subroutine

  SUBROUTINE $\boxed{\text{SUB}}(\text{X}, \text{NX}, \text{F}, \text{NF})$

  DIMENSION $\text{X}(\text{NX}), \text{F}(\text{NF})$

  ≀

  $\text{F}(1) = f_1(\text{X}(1), \cdots, \text{X}(\text{NX}))$

  ≀

  $\text{F}(\text{NF}) = f_{\text{NF}}(\text{X}(1), \cdots, \text{X}(\text{NX}))$

  ≀

  RETURN

  END

3.1.2 Algorithms Used

3.1.2.1 Richardson's extrapolation

Obtain the differential value $f^{(1)}(x)$ of a function $f(x)$ at x by using differences.
From the Taylor expansion of $f(x+h)$ and $f(x-h)$, $f^{(1)}(x)$ can be written as:

$$f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{2h} + c_1 h^2 + c_2 h^4 + \cdots + c_i h^{2i} + \cdots \tag{3.1}$$

where:

$$c_i = -\frac{f^{(2i+1)}(x)}{(2i+1)!} \ (i = 1, 2, \cdots)$$

Let the central difference of the step size h be expressed as:

$$D_0^{(0)} = \frac{f(x+h) - f(x-h)}{2h}$$

Then, the error of $D_0^{(0)}$ and $f^{(1)}(x)$ is on the order of h^2 from equation (3.1).
Now, divide the step size in half by setting it to $\frac{h}{2}$ and let the central difference be expressed as:

$$D_0^{(1)} = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{2(\frac{h}{2})}$$

If $D_0^{(0)}$ and $D_0^{(1)}$ are used to define the quantity:

$$D_1^{(0)} = \frac{1}{3}(4D_0^{(1)} - D_0^{(0)})$$

then, from equation (3.1) and the equation obtained by replacing h by $\frac{h}{2}$ in equation (3.1), express the error of this $D_1^{(0)}$ and $f^{(1)}(x)$ as:

$$f^{(1)}(x) - D_1^{(0)} = -\frac{1}{4}c_2 h^4 + \cdots$$

which is on the order of h^4. $D_1^{(0)}$ gives a higher order approximation to $f^{(1)}(x)$ than $D_0^{(0)}$.
Generalizing this procedure, by sequentially eliminating the high order terms of h in equation (3.1), a high order approximation of $f^{(1)}(x)$ is sequentially obtained. First, start with a suitable h and calculate:

$$D_0^{(k)} = \frac{f(x+\frac{h}{2^k}) - f(x-\frac{h}{2^k})}{2(\frac{h}{2^k})} \ (k = 0, 1, 2, \cdots)$$

Next, for $m = 1, 2, \cdots$ calculate the following (Richardson extrapolation):

$$\begin{aligned} D_m^{(k)} &= \frac{4^m D_{m-1}^{(k+1)} - D_{m-1}^{(k)}}{4^m - 1} \\ &= D_{m-1}^{(k+1)} + \frac{1}{4^m - 1}(D_{m-1}^{(k+1)} - D_{m-1}^{(k)}) \end{aligned}$$

(See Figure 3−1) The error of $D_m^{(0)}$ is on the order of $h^{2(m+1)}$.

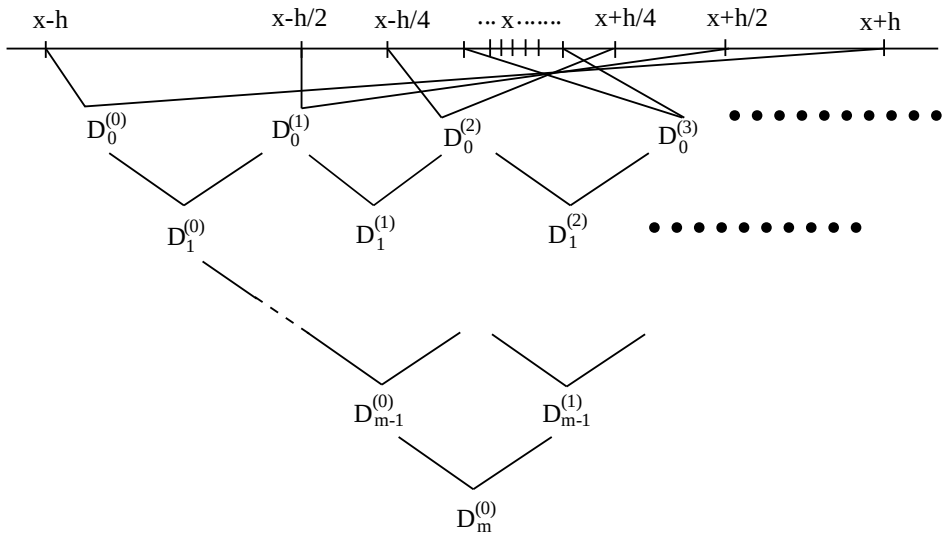


Figure 3−1 Richardson's Extrapolation

Use the following condition for determining convergence:

$$\frac{|D_m^{(0)} - D_{m-1}^{(0)}|}{|D_m^{(0)}|} \leq 3 \times \text{EPSN} \quad \text{(where EPSN is the required relative precision)}$$

Assume that the sequence has converged when this condition holds and use $D_m^{(0)}$ as the approximation of $f^{(1)}(x)$.

3.1.2.2 Numerical differentials of a function

Consider a high order differential formula for obtaining a differential of an arbitrary order.
If n is an even number, let the n-th order differential formula be as follows:

$$f^{(n)}(x) \sim \frac{\sum_k b_k(f(x+kh) + f(x-kh))}{h^n}$$

where, k satisfies the relationship $1 \leq k \leq \frac{n}{2}$.
From the binomial theorem, the coefficients b_k are considered to be:

$$\begin{aligned} b_k &= {}_{n/2}C_k \quad (k \text{ is an odd number}) \\ b_k &= -{}_{n/2}C_k \quad (k \text{ is an even number}) \end{aligned}$$

If n is an odd number, let the n-th order differential formula be as follows:

$$f^{(n)}(x) \sim \frac{\sum_k b_k(f(x+kh) - f(x-kh))}{2h^n}$$

where, k satisfies the relationship $1 \leq k \leq \lfloor \frac{n}{2} \rfloor + 1$ (where the notation $\lfloor x \rfloor$ represents the maximum integer that does not exceed x).
From the binomial theorem, the coefficients b_k are considered to be:

$$\begin{aligned} b_k &= {}_{\lfloor \frac{n}{2} \rfloor + 1}C_k \quad (k \text{ is an odd number}) \\ b_k &= -{}_{\lfloor \frac{n}{2} \rfloor + 1}C_k \quad (k \text{ is an even number}) \end{aligned}$$

and then when $n \geq 5$, the following values are taken by using the b_k values that had been obtained:

$$b_k \leftarrow b_k - b_{k-2} \ (k = \lfloor \frac{n}{2} \rfloor + 1, \cdots, 3)$$

Richardson's extrapolation is performed for the n-th order differential formula obtained in this way.
In addition, processing is aborted if the step size becomes smaller than the following value:

$\max(x\varepsilon, \varepsilon^{\frac{1}{n+1}})$ (ε is the unit for determining error)

3.1.2.3 Gradient vector of a function of many variables

For multiple variables $x_i, \cdots, x_{nx}$, fix the variables other than x_i $(i = 1, \cdots, nx)$. Then, obtain the first order differential for x_i by using Richardson's extrapolation. That is, the gradient vector for function f in terms of x_i is given by:

$$\frac{\partial f}{\partial x_i} \ (i = 1, \cdots, nx)$$

Processing is aborted if the step size becomes smaller than the following value when Richardson's extrapolation is performed:

$$\max_{i=1,\cdots,nx}(x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon \text{ is the unit for determining error})$$

3.1.2.4 Hessian matrix of a function of multiple variables

Obtain the first order differential values for each of the variables $x_1, \cdots, x_{nx}$. Perform Richardson's extrapolation for that first order differential value.
For multiple variables $x_1, \cdots, x_{nx}$, fix the variables other than x_i $(i = 1, \cdots, nx)$ and obtain the first order differential in terms of x_i $(i = 1, \cdots, nx)$ for the first order differential values of each of the variables.
That is, the (i, j)-th element of the Hessian matrix for the function is given by:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \ (i = 1, \cdots, nx; \ j = 1, \cdots, nx)$$

Processing is aborted if the step size becomes smaller than the following value when Richardson's extrapolation is performed:

$$\max_{i=1,\cdots,nx}(x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon \text{ is the unit for determining error})$$

3.1.2.5 Jacobian matrix of a function of multiple variables

Using an algorithm similar to the one described for obtaining the gradient vector, obtain the first order differential values of each of the multiple functions $f_1, \cdots, f_{nf}$ in terms of each of the variables $x_1, \cdots, x_{nx}$.
That is, the (i, j)-th element of the Jacobian matrix is given by:

$$\frac{\partial f_i}{\partial x_j} \ (i = 1, \cdots, nf; \ j = 1, \cdots, nx)$$

Processing is aborted if the step size becomes smaller than the following value when Richardson's extrapolation is performed:

$$\max_{i=1,\cdots,nx}(x_i\varepsilon, \varepsilon^{0.25}) \ (\varepsilon \text{ is the unit for determining error})$$

3.1.3 Reference Bibliography

(1) Hitotsumatsu Shin and Togawa Hayato editors, “Error in Numerical Calculations”, Kyoritsu Shuppan, (1975).

(2) Mori Masatake, “Numerical Calculation Programming”, Iwanami Shoten, (1986).

3.2 NUMERICAL DIFFERENTIALS

3.2.1 DQFODX, RQFODX
Numerical Differentials of a Function

(1) **Function**

DQFODX or RQFODX uses differences and Richardson's extrapolation to obtain the n-th order differential value $f^{(n)}(x)$ of the function $f(x)$.

(2) **Usage**

Double precision:

CALL DQFODX (F, X, N, EPS, MR, H, DEL, WK, IERR)

Single precision:

CALL RQFODX (F, X, N, EPS, MR, H, DEL, 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	F	{D / R}	−	Input	Name of function subprogram F(X) that defines the function $f(x)$ of x. (See Notes (a))
2	X	{D / R}	1	Input	Differentiation point x.
3	N	I	1	Input	Differential order n
4	EPS	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64)
5	MR	I	1	Input	Maximum number of iterations (Default value: 100)
				Output	Actual number of iterations
6	H	{D / R}	1	Input	Initial step size
7	DEL	{D / R}	1	Output	The n-th order differential value $f^{(n)}(x)$ of the function at the differentiation point
8	WK	{D / R}	See Contents	Work	Work area (See Notes (b)) **Size**: $\lfloor \mathrm{N}/2 \rfloor + \mathrm{MR} + 2$ ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x.)
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS $\geq \varepsilon \times 64$
(except when 0.0 is entered in order to set EPS to the default value,
ε is the unit for determining error.)

(b) MR > 0
(except when 0.0 is entered in order to set MR to the default value)

(c) N > 0

(d) H $>$ MAX(X $\times \varepsilon, \varepsilon^{1/(\mathrm{N}+3)}$)
(ε is the unit for determining error.)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with the corresponding default value set.
2000	The error became large before the required precision was achieved or the step size was too small.	The calculation value at that time is output, and processing is aborted.
2500	The maximum number of iterations was reached without the required precision being achieved.	
3000	Restriction (c) or (d) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The actual name in the first argument F must be declared by using an `EXTERNAL` statement in the user program, and a function subprogram having the actual name of F must be created.
The function subprogram (in double-precision) should be created as follows:

```
REAL(8) FUNCTION F(X)
REAL(8) X
F=f(x)
RETURN
END
```

(b) When reserving the area for the eighth argument WK as an array, let the size of the array be $\lfloor \mathrm{N}/2 \rfloor + 102$ if MR ≤ 0.

(c) Enter a value larger than the default value as the required relative precision.

(d) The value of the sixth argument H should be on the order of 0.5 to 1.0.

(e) For the convergence decision, consider that the sequence has converged when the following relationship holds:
$\{|$ Extrapolation value $| - |$ Previous extrapolation value $| \leq |$ EPS $\times$ Extrapolation value $|\}$

(f) If a default value is shown in the Contents column of the table describing the arguments, the default value will be set if 0 is entered for an integer type argument or 0.0 is entered for a real type argument.

(g) Since precision often decreases if N $>$ 5 is set for the differential order when the single precision subroutine is used, the double precision subroutine should be used. However, N $\leq$ 15 should be set even when the double precision subroutine is used.

(7) **Example**

(a) Problem

Obtain an approximate value of the fifth order differential of the function $f(x) = \sin(x)$ at $x = -3.75$.

(b) Input data

Function subprogram name: FQFODX.

X=−3.75, N=5, EPS=0.0, MR=15 and H=0.5.

(c) Main program

```
      PROGRAM  BQFODX
! *** EXAMPLE OF DQFODX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(N=5)
      DIMENSION WK(19)
      EXTERNAL FQFODX
!
      WRITE(6,1000)
      READ(5,*) X
      READ(5,*) EPS
      READ(5,*) MR
      READ(5,*) H
      WRITE(6,1100) N
      WRITE(6,1200) X
      WRITE(6,1300) EPS,MR,H
      CALL DQFODX(FQFODX,X,N,EPS,MR,H,DEL,WK,IERR)
      WRITE(6,1400) IERR
      WRITE(6,1500) MR,DEL
      STOP
!
 1000 FORMAT(' ',/,/,' *** DQFODX ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'ORDER OF DIFFERENTIATION = ',I2)
 1200 FORMAT(6X,'X =',F12.5)
 1300 FORMAT(6X,'EPS = ',1PD18.9,/,&
             6X,'MR  = ',3X,I2,/,&
             6X,'H   = ',1PD18.9)
 1400 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5)
 1500 FORMAT(6X,'ITERATION NUMBER   = ',3X,I2,/,&
             6X,'DIFFERENTIAL VALUE = ',1PD18.9)
      END


      REAL(8) FUNCTION FQFODX(X)
      REAL(8) X
      FQFODX = DSIN(X)
      RETURN
      END
```

(d) Output results

```
*** DQFODX ***
 ** INPUT **
     ORDER OF DIFFERENTIATION =  5
     X =    -3.75000
     EPS =    0.000000000D+00
     MR  =    15
     H   =    5.000000000D-01
 ** OUTPUT **
     IERR =     0
     ITERATION NUMBER   =     3
     DIFFERENTIAL VALUE =   -8.205593577D-01
```

3.2.2 DQMOGX, RQMOGX
Gradient Vector of a Function of Many Variables

(1) **Function**

DQMOGX or RQMOGX uses differences and Richardson's extrapolation to obtain the gradient vector $\partial f/\partial x_j$ of a function of many variables, $f(\boldsymbol{x})(\boldsymbol{x} = \{x_j\}; j = 1, \cdots, nx)$.

(2) **Usage**

Double precision:

```
CALL DQMOGX  (F, X, NX, EPS, MR, H, GRAD, WK, IERR)
```

Single precision:

```
CALL RQMOGX  (F, X, NX, EPS, MR, H, GRAD, 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	F	{D / R}	−	Input	Name of function subprogram F(X) that defines the function $f(\boldsymbol{x})$ of $\boldsymbol{x}$. (See Notes (a))
2	X	{D / R}	NX	Input	Differentiation point $x(x_1, \cdots, x_{nx})$.
3	NX	I	1	Input	Number nx of independent variables X
4	EPS	{D / R}	1	Input	Required relative precision (Default value: (Unit for determining error)$^{\frac{2}{3}}$
5	MR	I	1	Input	Maximum number of iterations (Default value: 100)
6	H	{D / R}	1	Input	Initial step size
7	GRAD	{D / R}	NX	Output	Gradient vector $\partial f/\partial \boldsymbol{x}$ of the function at the differentiation point
8	WK	{D / R}	MR + 1	Work	Work area (See Notes (b))
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS $\geq \varepsilon \times 64$
(except when 0.0 is entered in order to set EPS to the default value,
ε is the unit for determining error.)

(b) MR > 0
(except when 0.0 is entered in order to set MR to the default value)

(c) NX > 0

(d) H $\geq$ MAX$(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i = 1, \cdots, \text{NX})$
(ε is the unit for determining error.)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with the corresponding default value set.
2000	The error became large before the required precision was achieved or the step size was too small.	The calculation value at that time is output, and processing is aborted.
2500	The maximum number of iterations was reached without the required precision being achieved.	
3000	Restriction (c) or (d) was not satisfied.	

(6) **Notes**

(a) The actual name in the first argument F must be declared by using an `EXTERNAL` statement in the user program, and a function subprogram having the actual name of F must be created.
The function subprogram (in double-precision) should be created as follows:

```
REAL(8) FUNCTION F(X)
REAL(8) X
F=f(x(1),...,x(nx))
RETURN
END
```

(b) When reserving the area for the eighth argument WK as an array, let the size of the array be 101 if MR ≤ 0.

(c) Enter a value larger than the default value as the required relative precision.

(d) The value of the sixth argument H should be on the order of 0.5 to 1.0.

(e) For the convergence decision, consider that the sequence has converged when the following relationship holds:
{| Extrapolation value | − | Previous extrapolation value |≤| EPS × Extrapolation value |}

(f) If a default value is shown in the Contents column of the table describing the arguments, the default value will be set if 0 is entered for an integer type argument or 0.0 is entered for a real type argument.

(g) If several different errors occur at the same time, the most severe error value will be output for the error indicator, and the other error information may disappear.

(7) **Example**

(a) Problem

Obtain gradient vector of the function $f(x_1, x_2, x_3) = x_1 x_2 x_3$ at $(x_1, x_2, x_3) = (2.1, 8.59, 0.315)$.

(b) Input data

Function subprogram name: FQMOGX.

X(1)=2.1, X(2)=8.59, X(3)=0.315, NX=3, EPS=1.0D−9, MR=15 and H=0.5.

(c) Main program

```
      PROGRAM  BQMOGX
! *** EXAMPLE OF DQMOGX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(NX=3,MR=15)
      DIMENSION X(NX),GRAD(NX),WK(MR+1)
      EXTERNAL FQMOGX
!
      WRITE(6,1000)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) EPS
      READ(5,*) H
      WRITE(6,1100) NX
      WRITE(6,1200) (X(I),I=1,NX)
      WRITE(6,1300) EPS,MR,H
      CALL DQMOGX(FQMOGX,X,NX,EPS,MR,H,GRAD,WK,IERR)
      WRITE(6,1400) IERR
      DO 10 I=1,NX
         WRITE(6,1500) I,GRAD(I)
   10 CONTINUE
      STOP
!
 1000 FORMAT(' ',/,/,' *** DQMOGX ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'NUMBER OF VARIABLE = ',I2)
 1200 FORMAT(6X,'X = ',3F12.5)
 1300 FORMAT(6X,'EPS = ',1PD18.9,/,&
             6X,'MR  = ',5X,I2,/,&
             6X,'H   = ',1PD18.9)
 1400 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5,/)
 1500 FORMAT(6X,'GRAD(',I2,') = ',1PD18.9)
      END


      REAL(8) FUNCTION FQMOGX(X)
      REAL(8) X
      DIMENSION X(*)
      FQMOGX = X(1)*X(2)*X(3)
      RETURN
      END
```

(d) Output results

```
*** DQMOGX ***
 ** INPUT **
     NUMBER OF VARIABLE =  3
     X =      2.10000     8.59000     0.31500
     EPS =    1.000000000D-09
     MR  =      15
     H   =    5.000000000D-01
 ** OUTPUT **
     IERR =     0

     GRAD( 1) =    2.705850000D+00
     GRAD( 2) =    6.615000000D-01
     GRAD( 3) =    1.803900000D+01
```

3.2.3 DQMOHX, RQMOHX
Hessian of a Function of Many Variables

(1) **Function**

DQMOHX or RQMOHX uses Richardson's extrapolation to obtain the Hessian $\partial^2 f/(\partial x_i \partial x_j)$ $(i = 1, \cdots, nx;$ $j = 1, \cdots, nx)$ of a function of many variables, $f(\boldsymbol{x})(\boldsymbol{x} = \{x_j\}; j = 1, \cdots, nx)$.

(2) **Usage**

Double precision:

```
CALL DQMOHX  (F, X, NX, EPS, MR, H, HES, IWK, WK, IERR)
```

Single precision:

```
CALL RQMOHX  (F, X, NX, EPS, MR, H, HES, 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	F	{D / R}	–	Input	Name of function subprogram F(X) that defines the function $f(\boldsymbol{x})$ of $\boldsymbol{x}$. (See Notes (a))
2	X	{D / R}	NX	Input	Differentiation point $(x_1, \cdots, x_{nx})$.
3	NX	I	1	Input	Number nx of independent variables X
4	EPS	{D / R}	1	Input	Required relative precision (Default value: (Unit for determining error)$^{\frac{4}{9}}$
5	MR	I	1	Input	Maximum number of iterations (Default value: 100)
6	H	{D / R}	1	Input	Initial step size
7	HES	{D / R}	NX× NX	Output	Hessian $\partial^2 f/(\partial x_i \partial x_j)$ of the function at the differentiation point
8	IWK	I	NX	Work	Work area
9	WK	{D / R}	See Contents	Work	Work area (See Notes (b)) **Size**: $(\mathrm{NX} + \mathrm{MR} + 1) \times (5 + \mathrm{MR})$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS $\geq \varepsilon \times 64$
(except when 0.0 is entered in order to set EPS to the default value,
ε is the unit for determining error.)

(b) MR > 0
(except when 0.0 is entered in order to set MR to the default value)

(c) NX > 0

(d) H $\geq$ MAX$(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i = 1, \cdots, \text{NX})$
(ε is the unit for determining error.)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with the corresponding default value set.
2000	The error became large before the required precision was achieved or the step size was too small.	The calculation value at that time is output, and processing is aborted.
2500	The maximum number of iterations was reached without the required precision being achieved.	
3000	Restriction (c) or (d) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The actual name in the first argument F must be declared by using an `EXTERNAL` statement in the user program, and a function subprogram having the actual name of F must be created.
The function subprogram (in double-precision) should be created as follows:

```
REAL(8) FUNCTION F(X)
REAL(8) X
F=f(x(1),...,x(nx))
RETURN
END
```

(b) When reserving the area for the eighth argument WK as an array, let the size of the array be (NX + 101) $\times$ 105 if MR ≤ 0.

(c) Enter a value larger than the default value as the required relative precision.

(d) The value of the sixth argument H should be on the order of 0.5 to 1.0.

(e) For the convergence decision, consider that the sequence has converged when the following relationship holds:
{| Extrapolation value | $-$ | Previous extrapolation value |$\leq$| EPS $\times$ Extrapolation value |}

(f) If a default value is shown in the Contents column of the table describing the arguments, the default value will be set if 0 is entered for an integer type argument or 0.0 is entered for a real type argument.

(g) Each elements of Hessian at differentiation point are stored in HES as shown below. HES(i, j) = $\partial^2 \mathrm{f}/\partial \mathrm{x_i} \partial \mathrm{x_j}$ $(i = 1, 2, \cdots, \mathrm{NX};\ j = 1, 2, \cdots, \mathrm{NX})$

(h) If several different errors occur at the same time, the most severe error value will be output for the error indicator, and the other error information may disappear.

(7) **Example**

(a) Problem
Obtain Hessian of the function $f(x_1, x_2, x_3) = x_1 x_2 x_3$ at $(x_1, x_2, x_3) = (5.5, 1.0, 8.0)$.

(b) Input data
Function subprogram name: FQMOHX,
X(1)=5.5, X(2)=1.0, X(3)=8.0, NX=3, EPS=1.0D−10, MR=15 and H=0.5.

(c) Main program

```
      PROGRAM  BQMOHX
! *** EXAMPLE OF DQMOHX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(NX=3,MR=15)
      DIMENSION X(NX),HES(NX*NX),WK((NX+MR+1)*(5+MR)),IWK(NX)
      EXTERNAL FQMOHX
!
      WRITE(6,1000)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) EPS
      READ(5,*) H
      WRITE(6,1100) NX
      WRITE(6,1200) (X(I),I=1,NX)
      WRITE(6,1300) EPS,MR,H
      CALL DQMOHX(FQMOHX,X,NX,EPS,MR,H,HES,IWK,WK,IERR)
      WRITE(6,1400) IERR
      WRITE(6,1500) ((HES(NX*(I-1)+J),I=1,NX),J=1,NX)
      STOP
!
 1000 FORMAT(' ',/,/,' *** DQMOHX ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'NUMBER OF VARIABLE = ',I2)
 1200 FORMAT(6X,'X = ',3F12.5)
 1300 FORMAT(6X,'EPS = ',1PD18.9,/,&
             6X,'MR  = ',3X,I2,/,&
             6X,'H   = ',1PD18.9)
 1400 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5)
 1500 FORMAT(6X,'HESSE(I,J) = I',3(1PD18.9),'   I',/,&
                          (17X,'  I',3(1PD18.9),'   I'))
      END


      REAL(8) FUNCTION FQMOHX(X)
      REAL(8) X
      DIMENSION X(*)
      FQMOHX = X(1)*X(2)*X(3)
      RETURN
      END
```

(d) Output results

```
*** DQMOHX ***
 ** INPUT **
     NUMBER OF VARIABLE =  3
     X =      5.50000     1.00000     8.00000
     EPS =    1.0000000000D-10
     MR  =    15
     H   =    5.000000000D-01
 ** OUTPUT **
     IERR =     0
     HESSE(I,J) = I   0.000000000D+00   8.000000000D+00   1.000000000D+00  I
                  I   8.000000000D+00   1.000000000D+00   8.000000000D+00  I
                  I   1.000000000D+00   8.000000000D+00   0.000000000D+00  I
```

3.2.4 DQMOJX, RQMOJX
Jacobian of Multiple Function of Many Variables

(1) **Function**

DQMOJX or RQMOJX uses Richardson's extrapolation to obtain the Jacobian $J = \{\partial f_i/\partial x_j\}$ of multiple functions of many variables, $\boldsymbol{f}(\boldsymbol{x}) = \{f_i(x_j)\}(i = 1, \cdots, nf;\ j = 1, \cdots, nx)$.

(2) **Usage**

Double precision:

```
CALL DQMOJX  (SUB, X, NX, NF, EPS, MR, H, RJAC, IWK, WK, IERR)
```

Single precision:

```
CALL RQMOJX  (SUB, X, NX, NF, EPS, MR, H, RJAC, 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, NX, F, NF) that defines function $\boldsymbol{f}(\boldsymbol{x})$. (See Notes (a))
2	X	{ D / R }	NX	Input	Differentiation point $(x_1, \cdots, x_{nx})$.
3	NX	I	1	Input	Number nx of independent variables X
4	NF	I	1	Input	Number nf of simultaneous functions
5	EPS	{ D / R }	1	Input	Required relative precision (Default value: (Unit for determining error)$^{\frac{2}{3}}$
6	MR	I	1	Input	Maximum number of iterations (Default value: 100)
7	H	{ D / R }	1	Input	Initial step size
8	RJAC	{ D / R }	NF× NX	Output	The values of Jacobian $J = \{\partial f_i/\partial x_j\}$
9	IWK	I	NF	Work	Work area
10	WK	{ D / R }	See Contents	Work	Work area (See Notes (b)) **Size**: NF × (5 + MR)
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS $\geq \varepsilon \times 64$
(except when 0.0 is entered in order to set EPS to the default value, ε is the unit for determining error.)

(b) MR > 0
(except when 0.0 is entered in order to set MR to the default value)

(c) NX > 0 and NF > 0

(d) H $\geq$ MAX$(x_i \times \varepsilon, \varepsilon^{0.25})$ $(i = 1, \cdots, \mathrm{NX})$
(ε is the unit for determining error.)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with the corresponding default value set.
2000	The error became large before the required precision was achieved or the step size was too small.	The calculation value at that time is output, and processing is aborted.
2500	The maximum number of iterations was reached without the required precision being achieved.	
3000	Restriction (c) or (d) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The actual name in the first argument SUB must be declared by using an `EXTERNAL` statement in the user program, and a subroutine subprogram having the actual name of SUB must be created. The subroutine subprogram (in double-precision) should be created as follows:

```
SUBROUTINE SUB(X, NX, F, NF)
REAL(8) X, F
DIMENSION X(NX), F(NF)
    F(1) = f1(x(1), ···, x(nx))
    F(2) = f2(x(1), ···, x(nx))
    ⋮
    F(NF) = fnf(x(1), ···, x(nx))
RETURN
END
```

(b) Each elements of Jacobian at differentiation point are stored in RJAC as shown below. RJAC(i, j) = $\partial \mathrm{f_i}/\partial \mathrm{x_j}$ $(i = 1, 2, \cdots, \mathrm{NF};\ j = 1, 2, \cdots, \mathrm{NX})$

(c) When reserving the area for the eighth argument WK as an array, let the size of the array be NF $\times$ 105 if MR ≤ 0.

(d) Enter a value larger than the default value as the required relative precision.

(e) The value of the sixth argument H should be on the order of 0.5 to 1.0.

(f) For the convergence decision, consider that the sequence has converged when the following relationship holds:
{| Extrapolation value | − | Previous extrapolation value |≤| EPS × Extrapolation value |}

(g) If a default value is shown in the Contents column of the table describing the arguments, the default value will be set if 0 is entered for an integer type argument or 0.0 is entered for a real type argument.

(h) If several different errors occur at the same time, the most severe error value will be output for the error indicator, and the other error information may disappear.

(7) **Example**

(a) Problem
Obtain Jacobian of the following vector of many variables functions:

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ f_3(\boldsymbol{x}) \\ f_4(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} x_1x_2x_3 \\ x_1 + x_2 + x_3 \\ x_1x_2 + x_2x_3 + x_3x_1 \\ x_1x_2x_3 + x_1x_2 + x_1 \end{bmatrix}$$

at $(x_1, x_2, x_3) = (6.8, 9.1, 3.4)$.

(b) Input data
Subroutine name: FQMOJX.
X(1)=6.8, X(2)=9.1, X(3)=3.4, NX=3, NF=4, EPS=1.0D−8, MR=15 and H=0.5.

(c) Main program

```
      PROGRAM  BQMOJX
! *** EXAMPLE OF DQMOJX ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(NX=3,NF=4,MR=15)
      DIMENSION X(NX),RJAC(NF*NX),WK(NF*(5+MR)),IWK(NF)
      EXTERNAL FQMOJX
!
      WRITE(6,1000)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) EPS
      READ(5,*) H
      WRITE(6,1100) NX,NF
      WRITE(6,1200) (X(I),I=1,NX)
      WRITE(6,1300) EPS,MR,H
      CALL DQMOJX(FQMOJX,X,NX,NF,EPS,MR,H,RJAC,IWK,WK,IERR)
      WRITE(6,1400) IERR
      WRITE(6,1500) ((RJAC(NF*(I-1)+J),I=1,NX),J=1,NF)
      STOP
!
 1000 FORMAT(' ',/,/,' *** DQMOJX ***',/,2X,'** INPUT **')
 1100 FORMAT(6X,'NUMBER OF VARIABLE = ',I2,/,&
             6X,'NUMBER OF FUNCTION = ',I2)
 1200 FORMAT(6X,'X = ',3F12.5)
 1300 FORMAT(6X,'EPS = ',1PD18.9,/,&
             6X,'MR  = ',3X,I2,/,&
             6X,'H   = ',1PD18.9)
 1400 FORMAT(2X,'** OUTPUT **',/,6X,'IERR = ',I5)
 1500 FORMAT(6X,'JACOBI(I,J) = I',3(1PD18.9),'  I',/,&
                          (18X,'  I',3(1PD18.9),'  I'))
      END


      SUBROUTINE FQMOJX(X,NX,F,NF)
      REAL(8) X,F
      DIMENSION X(NX),F(NF)
      F(1) = X(1)*X(2)*X(3)
      F(2) = X(1)+X(2)+X(3)
      F(3) = X(1)*X(2)+X(2)*X(3)+X(3)*X(1)
      F(4) = X(1)*X(2)*X(3)+X(1)*X(2)+X(1)
      RETURN
      END
```

(d) Output results

```
*** DQMOJX ***
 ** INPUT **
     NUMBER OF VARIABLE =  3
     NUMBER OF FUNCTION =  4
     X =      6.80000     9.10000     3.40000
     EPS =    1.000000000D-08
     MR  =    15
     H   =    3.000000000D-01
 ** OUTPUT **
     IERR =     0
     JACOBI(I,J) = I   3.094000000D+01   1.000000000D+00   1.250000000D+01  I
                   I   1.000000000D+00   1.250000000D+01   4.104000000D+01  I
                   I   1.250000000D+01   4.104000000D+01   2.312000000D+01  I
                   I   4.104000000D+01   2.312000000D+01   1.000000000D+00  I
```

Chapter 4

NUMERICAL INTEGRATION

4.1 INTRODUCTION

This chapter describes subroutines that integrate functions using automatic integration methods.
Although numerical integration can be performed using numerical table input or function input, this chapter deals only with subroutines using function input. For numerical table input, you can use subroutines described in Chapter 3, "Spline Functions."
When integrating a function, you can obtain high-precision solutions quickly by selecting the optimum solution method according to the function characteristics. This library provides subroutines corresponding the function characteristics as follows.

(1) Arbitrary functions
Subroutines for arbitrary functions quickly obtain the value of the integral for functions such as oscillatory functions, peak-type functions, functions having end-point singularities, and functions having interior-point singularities when you have no information about the characteristics of the function. These subroutines also output information related to the singular points. If the function has a considerable number of singularities or if it oscillates wildly making it impossible to obtain a high-precision solution, the subroutines calculate an approximate value and output it together with its error. If you want to obtain a high-precision solution, you should use other subroutines that use information about singular points. Although subroutines for arbitrary functions are the most general-purpose subroutines, they are unsuitable for oscillatory functions over an infinite interval.

(2) $f(x) \times w(x)$ type functions
Subroutines for $f(x) \times w(x)$ type functions obtain the value of the integral when either a trigonometric function, algebraic or logarithmic function having singularities at its endpoints or the function $\frac{1}{x-c}$ is used as the weight function $w(x)$.

(3) Oscillatory or peak-type functions
Use subroutines for oscillatory or peak-type functions if the function does not have a considerable number of singularities. You can increase computational efficiency by selecting the value of ISW according to whether the function is an oscillatory or peak-type function.

(4) Singular function
You can use subroutines intended for singular functions to obtain the value of the integral even if the function has no singularities. These subroutines are more efficient than other subroutines, however, if the function has a considerable number of singularities. You should not use these subroutines for an oscillatory function. Subroutines are provided for when singular points are located at the end points and at interior points. You must provide information, however, about singular point positions when they are located at interior points.

(5) Singular functions for which singularity information is unknown
These subroutines, like the ones for arbitrary functions described in (1), obtain the value of the integral for any type of function when you have no information about function characteristics. These subroutines are suitable for functions having a more heavily singular point than the subroutines described in (1). In addition,

if the subroutine is unable to reach the required precision, it may abort processing before completion without switching to processing to obtain an approximate solution like the subroutines described in (1) do. In addition, these subroutines require a great deal of calculation time. Therefore, you should use these subroutines if you want to obtain a very precise integral value when you know the function has singularities but you either do not know the positions of the singular points or you cannot determine them precisely.

4.1.1 Notes

(1) Figures 4−1 and 4−2 are flow diagrams that explain how to use the various subroutines introduced above.

Figure 4−1 Using Subroutines for a Finite Interval

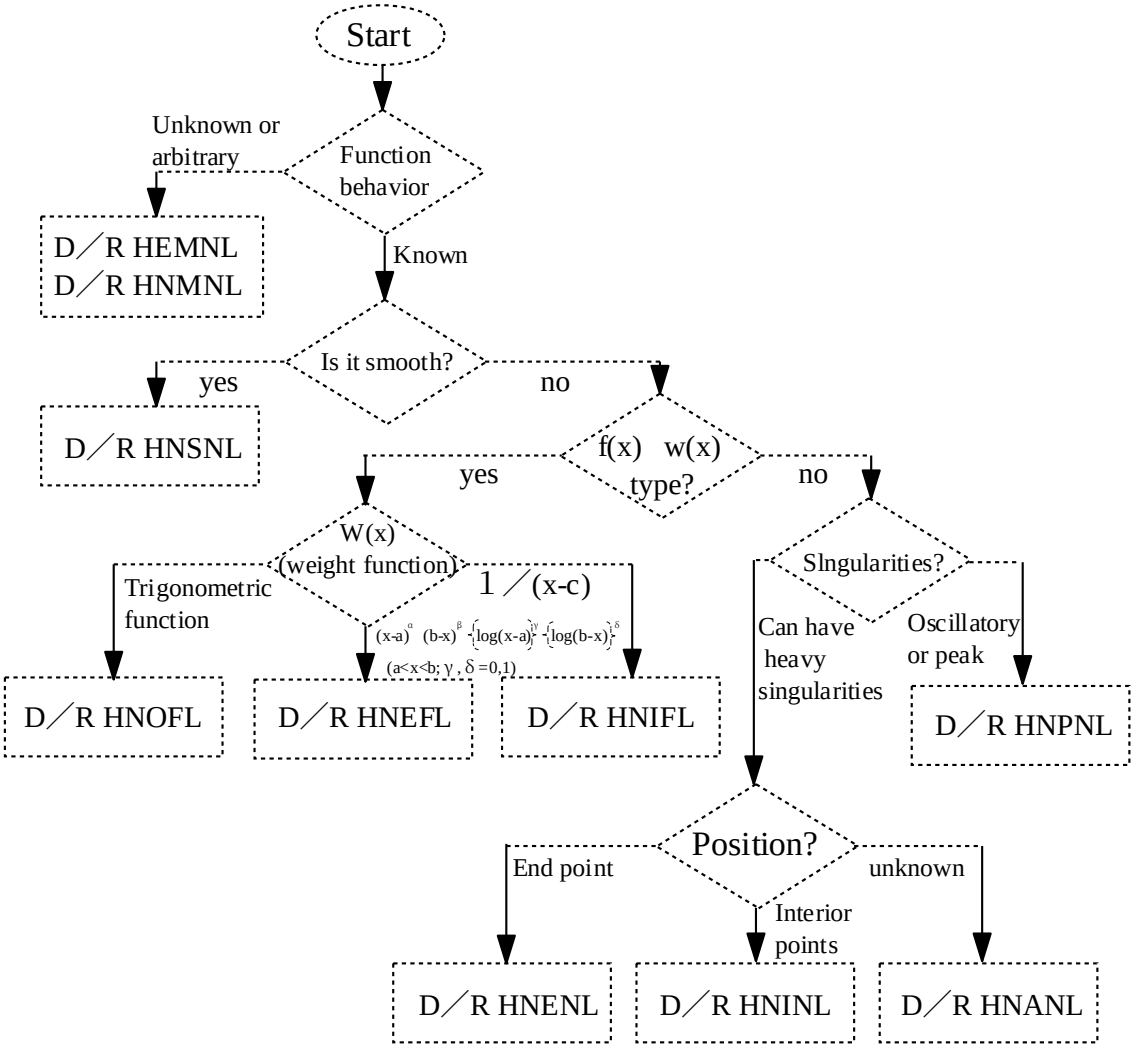
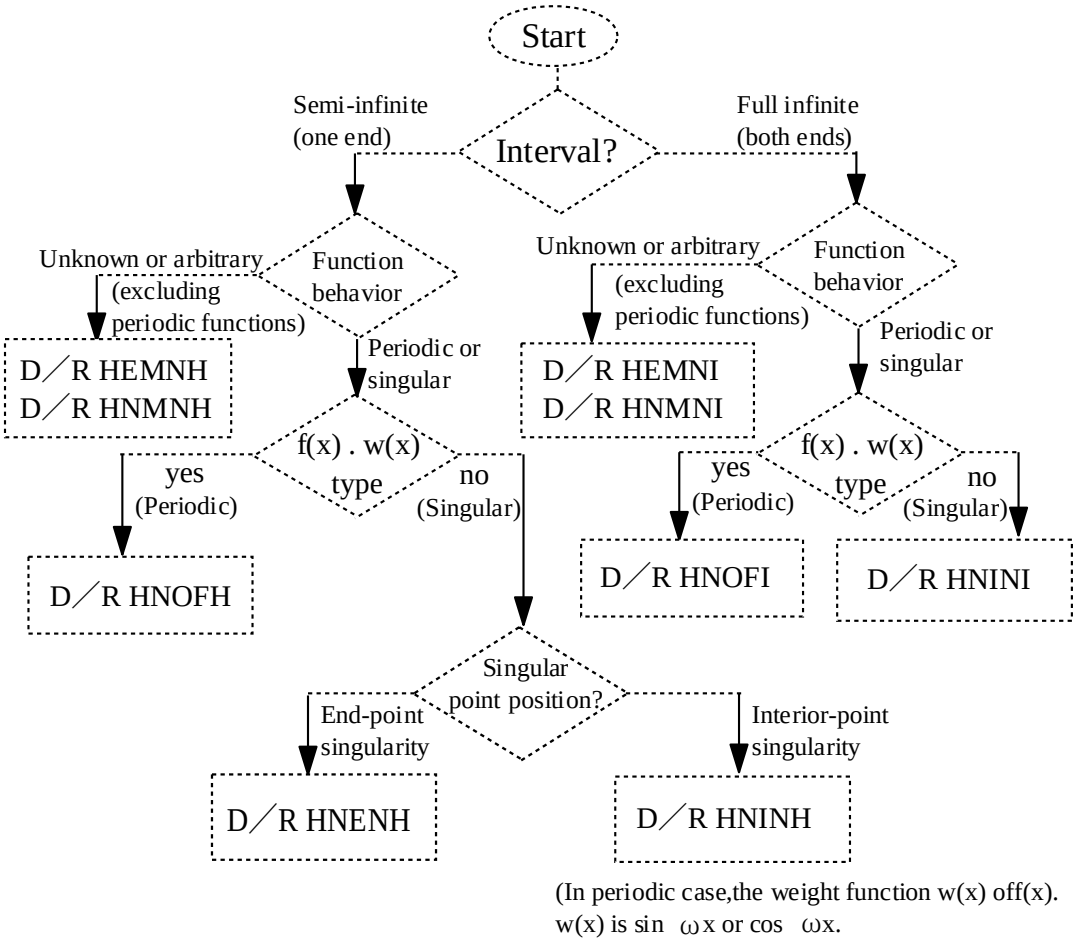


Figure 4−2 Using Subroutines for an Infinite Interval



(2) Note the following points concerning programs using these subroutines.

① You must declare the integrand function name to be used as an argument as EXTERNAL.

② You must take precautionary measures to prevent an overflow from occurring within the integration interval (such as setting the function value to 0.0 at singular points).

Example

One-dimensional integration (Let F has the same name in the main program and the function subprogram)

- Main program

```
      ≀
EXTERNAL F
      ≀
CALL {DHEMNL / RHEMNL} (F, ···)
      ≀
```

- Function subprogram

FUNCTION F (X)

≀

IF(ABS(X − 1.0).LT.UF)THEN
F = 0.0
ELSE
F = 1.0/SQRT(ABS(X − 1.0))
ENDIF

→ { Measure to prevent overflow due to division by zero (unnecessary if overflow will not occur). }

≀

Example

Two-dimensional integration (Let F , A and B have the same names in the main program and the function subprograms)

- Main program

≀

EXTERNAL F , A , B

≀

CALL {DHNFNM / RHNFNM} (F , A , B , ⋯)

≀

- Function subprograms

FUNCTION F (X, Y)

≀

FUNCTION A (Y)

≀

FUNCTION B (Y)

≀

Example

Multi-dimensional integration (Let F and R have the same names in the main program, the function subprogram and subroutine subprogram)

- Main program

≀

EXTERNAL F , R

≀

CALL {DHNFML / RHNFML} (F , R , ⋯)

≀

- Function subprogram

```
FUNCTION F (X, M)
DIMENSION X(M)
    ≀
F = ··· Expression which variable is X(1), X(2), ···, X(M).
    ≀
```

- Subroutine subprogram

```
SUBROUTINE R (I, X, A, B, M)
DIMENSION X(M), A(M), B(M)
    ≀
IF(I.EQ.1)THEN
    A(1) = ···
    B(1) = ···   ··· Expression which variable is X(2), X(3), ···, X(M).
ELSE IF(I.EQ.2)THEN
    A(2) = ···
    B(2) = ···   ··· Expression which variable is X(3), ···, X(M).
ELSE IF(I.EQ.3)THEN
    ≀
ELSE IF(I.EQ.M)THEN
    A(M) = ···
    B(M) = ···   ··· Constant
    ≀
```

(3) If several errors occur, the error number of the most severe error is output as the error indicator; other error information may be lost.

(4) These subroutines in this chapter calculate the value of the integral of functions to reach the required relative and/or absolute precision, which is given as an input argument. These subroutines obtain as output arguments the estimate value of absolute error as well as the integral value.

Notes Required relative precision, required absolute precision, and estimate value of absolute error are defined as follows:

Let us denote an numerical approximate solution of the integral value by Q and the precise solution by Q_0. Define an absolute error and a relative error by the following formulas:

$$\begin{cases} \text{(absolute error)} &= & |\, Q - Q_0 \,| \\ \text{(relative error)} &= & |\, (Q - Q_0)/Q \,| \end{cases}$$

Then, a required relative (absolute) precision ER (EA) means a tolerable upper limit for the relative (absolute) error, respectively. An estimate value of absolute error means a upper bound for an absolute error that is estimated by some numerical error estimation criterion.

When a required relative precision ER and/or required absolute precision EA is given as an input, these subroutines in this chapter will seek for an approximate solution of the integral value Q in such a precision as to satisfy the following condition(s):

$$\begin{cases} \text{AE/Q} & < & \text{ER} \\ \text{AE} & < & \text{EA} \end{cases}$$

(5) With all the subroutines in this chapter excluding the subroutines for multi-dimensional integration over a finite interval, the value used as the default value for required absolute precision is (minimum absolute value) $\times 2^{24}$.

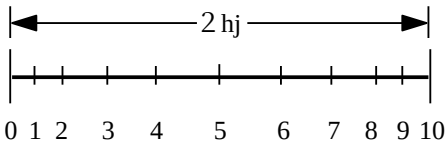
4.1.2 Algorithms Used

4.1.2.1 Adaptive Newton-Cotes rule (Integration of arbitrary functions)

The subroutines use an adaptive automatic integral method based on the Newton-Cotes 9-point rule and having additional functions for strengthening error estimation, easing convergence decisions, and detecting and processing singular points.
If $f(x)$ is assigned as the integrand $[a, b]$ as the integral interval, and ε_0 as the required absolute precision, then the automatic integral method automatically partitions the integral interval and calculates the integral value to a precision within the permitted error. The integral value and its error as well as a convergence decision are required on each subdivision interval in order to control this partitioning.
These values can be obtained as described below.

(1) Integral value and its error
For example, if the interval width, which is assumed to be $2h_j$, is partitioned by the 9-point rule as shown in Figure 4−3, then the integral value Q_j and its revised value ξ_j are expressed by equations (4.2) and (4.4).

Figure 4−3 Taking Interior Division Points (9-point Rule Example)



$$Q_j = \frac{h_j}{14175}\{989(f_0+f_{10})+5888(f_2+f_8)-928(f_3+f_7) \quad (4.1)$$
$$+10496(f_4+f_6)-4540f_5\} \quad (4.2)$$

$$\xi_j = \frac{-4736h_j}{468242775}\{3003(f_0+f_{10})-16384(f_1+f_9)+27720(f_2+f_8) \quad (4.3)$$
$$-38220(f_3+f_7)+56056(f_4+f_6)-64350f_5\} \quad (4.4)$$

If we use the Newton-Cotes n-point rule and assume that the $(n+1)$-th differential coefficient is almost fixed, then the error ε_j for $Q_j+\xi_j$ is $\frac{|\ \xi_j\ |}{(2^{n+1}-1)}$.
However, the $(n+1)$-th and subsequent differential coefficients cannot actually be ignored and often become larger than this.
The following method is used to further subdivide a given interval i into two parts obtaining intervals j and $j+1$ and to estimate ε_j from ξ_i and ξ_j. If we assume that the ratio of $\varepsilon+|\ \xi\ |$ and $|\ \xi\ |$ is fixed, then:

$$C = \frac{|\xi_j+\xi_{j+1}|}{|\xi_i|}$$

(If the $(n+1)$-th differential coefficient is fixed, then $C = \frac{1}{2^{n+1}}$.) If the function characteristics are almost identical in the neighboring interval, we can assume that $\xi_j = \xi_{j+1}$. From this we obtain:

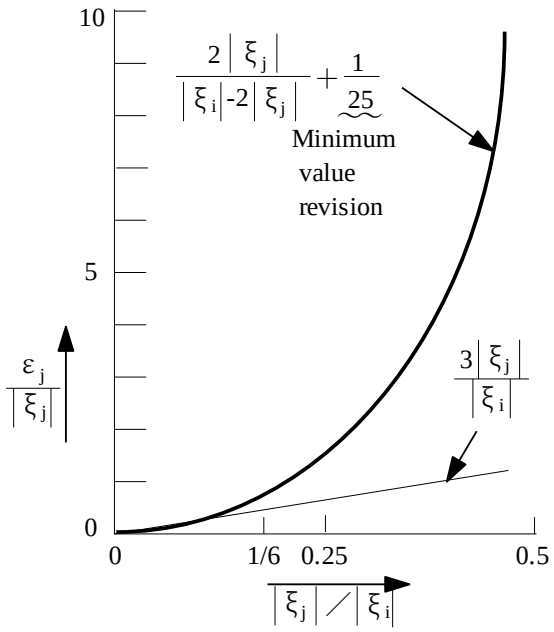
$$\varepsilon_j = \frac{C}{1-C}\ |\ \xi_j\ | = \frac{2{\xi_j}^2}{|\ \xi_i\ |-2\ |\ \xi_j\ |} \quad (4.5)$$

The sum of these ε_j values for all intervals becomes the entire amount of error.
In the program, to keep the error estimate from being too small where the minimum value of ε_j is bounded

by $\frac{|\xi_j|}{25}$ (for single precision, $\frac{|\xi_j|}{8}$) and to prevent the error from being excessively evaluated in an area having $|\xi_i| \leq 6 |\xi_j|$ and a small reduction rate ξ such as in the neighborhood of a singular point, the following relationships are assumed:
$\frac{1}{1-C} = 1.5$ and $\varepsilon_j = 3C\frac{|\xi_j|}{2} = 3\frac{{\xi_j}^2}{|\xi_i|}$. (See Figure 4−4) In addition, for stability, a large value is estimated for ε_j involved in the convergence decision by taking $\varepsilon_j = \frac{|\xi_j|}{2.0}$.

If the value actually taken for $\frac{|\xi_j|}{|\xi_i|}$ exceeds 0.25, then the series generally does not converge and the point

Figure 4−4 Revising ε_j Values



should be processed as a singular point. In its neighborhood, however, the value of $\frac{|\xi_j|}{|\xi_i|}$ may fluctuate dramatically or the $\frac{|\xi_j|}{|\xi_i|}$ convergence-time value may be close to 1.0. In this case, since the estimate of the error ε_j is too large, the error is approximated by the dashed line in Figure 4−4 for $\frac{|\xi_j|}{|\xi_i|} \geq \frac{1}{6}$ to revise the error for $\frac{|\xi_j|}{|\xi_i|} = 0.25$.

(2) Method of determining convergence in each interval
Let $h_0 = \frac{b-a}{2}$, and $h_j = \frac{(\text{Width of subdivision interval } j)}{2}$ for $(j = 1, \cdots, n)$. If we assume that the error for a solution that is considered to have converged has a uniform probability density $g(x)$ on the interval $[-\varepsilon_j, \varepsilon_j]$, then $g(x) = \frac{1}{2\varepsilon_j}$ and the variance in this interval at this time is $\frac{{\varepsilon_j}^2}{3}$. Therefore, the variance for the entire interval is $\sum_{j=1}^{n} \frac{{\varepsilon_j}^2}{3}$. If we assume $\varepsilon_j = \sqrt{\frac{h_j}{h_0}}\varepsilon_0$ (where ε_0 is the required absolute precision of the integral on $[a, b]$), then the variance for the integral value over the entire interval is $\sigma^2 = \frac{{\varepsilon_0}^2}{3}$. Therefore, the error becomes the normal distribution $\sigma = \frac{\varepsilon_0}{\sqrt{3}}$, and the probability that the value is within the range from $-\varepsilon_0$ through ε_0 becomes 91.6 percent. Therefore, if we let ε_j be the value $\frac{|\xi_j|}{2.0}$ obtained in (a), partitioning

ends when the following inequality holds and $Q_j + \xi_j$ is the integral value over the given interval.

$$\frac{|\,\xi_j\,|}{2.0} < \sqrt{\frac{h_j}{h_0}}(\max(\varepsilon_0, \varepsilon_0' I))$$

ε_0': Required relative precision of the integral value over $[a, b]$

I: Estimate of total integral value

[If Q_1 is the integral value obtained by the 9-point Newton-Cotes rule, the $I = Q_1 + \xi_1 + \xi_2 + \cdots + \xi_n + q$ (Revision for singular point processing.)]

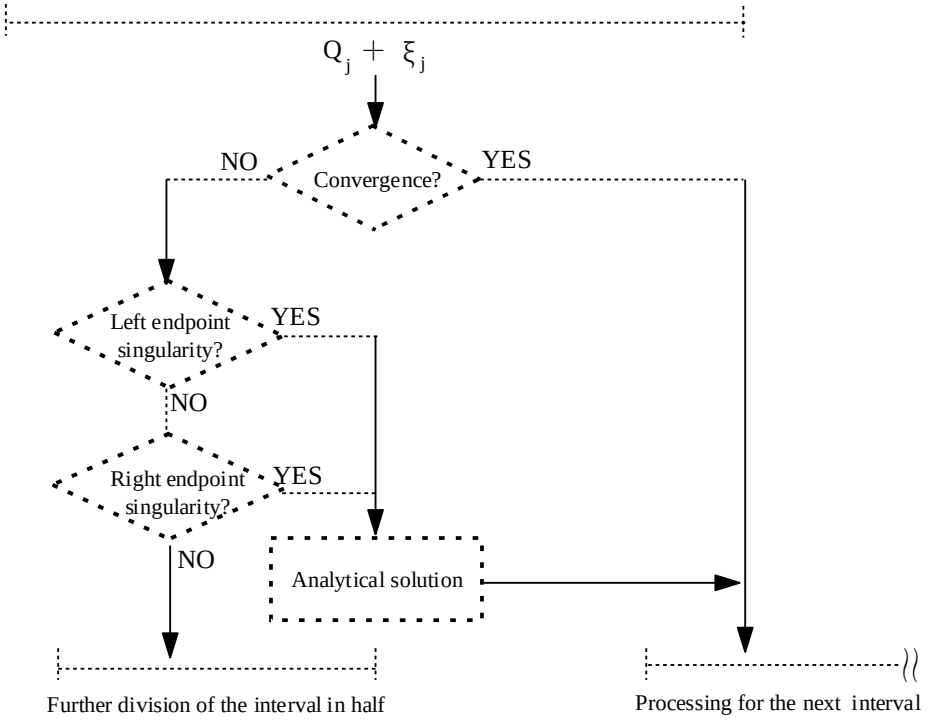
(3) Processing if a singular point is at a subdivision point

If a singular point is located at one of the points that divides the total interval into 2^m equal parts for a suitable positive integer m, then that singular point will become an endpoint of the interval at some step within the evaluation process.

Function values at interior points are used to detect singular points according to the flow shown in Figure 4−5. The detection operation occurs at this time when the interval width becomes $2^{-\ell}$ (where ℓ is a multiple of 5) times the width of the total interval.

The following equations are used for singular point detection and processing.

Figure 4−5 Calculation Procedure



- Algebraic singularity ($f(x) = \alpha X^{-p} + q$)

  Left endpoint singularity

  The following equation holds:

  $$\frac{f_2 - f_3}{f_3 - f_5} \approx \frac{f_3 - f_5}{f_5 - f_{10}} = d$$

  Analytical solution I_j

  $$I_j = \frac{2h_j(f_{10} - pq)}{1 - p}, \quad p = \frac{\log(d)}{\log(2)}, \quad q = \frac{2^{-p} f_5 - f_{10}}{2^{-p} - 1}$$

Right endpoint singularity
The following equation holds:

$$\frac{f_8 - f_7}{f_7 - f_5} \approx \frac{f_7 - f_5}{f_5 - f_0} = d$$

Analytical solution I_j

$$I_j = \frac{2h_j(f_0 - pq)}{1 - p}, \quad p = \frac{\log(d)}{\log(2)}, \quad q = \frac{2^{-p}f_5 - f_0}{2^{-p} - 1}$$

- Logarithmic singularity
  Left endpoint singularity
  The following equation holds:

$$f_2 - f_3 \approx f_3 - f_5 \approx f_5 - f_{10} = d$$

  Analytical solution I_j

$$I_j = 2h_j \left(f_5 + d\frac{1 - \log(2)}{\log(2)} \right)$$

  Right endpoint singularity
  The following equation holds:

$$f_8 - f_7 \approx f_7 - f_5 \approx f_5 - f_0 = d$$

  Analytical solution I_j
  Same equation as for the left endpoint singularity.

For both algebraic and logarithmic singularities, the almost equal expressions $\approx$ are considered to hold when the left and right sides of the expression differ by δ, where $\delta \leq \frac{\varepsilon'_0}{\sqrt{10h_j/h_0}}$ (ε'_0 : Required relative precision). In this case, singular point processing is performed. In addition, the error e_j is assumed to be:
$e_j = |\ (|\ \delta\ |\ +$Units for determining error$) \times I_j\ |$

(4) Processing for singular points not detected by the method described in (3)
If the singular point was not detected by the method described in (3), then a value approximated by the algebraic singularity $y = \alpha x^p$ $(0 < p < 1)$ is used. That is, k is assumed to be the point where the maximum absolute value is taken. Then $f_k = 0$ is assumed, and:

$$\delta_s = E_s \frac{p + 1}{(2 - 2^{-p})(1 - p)}$$

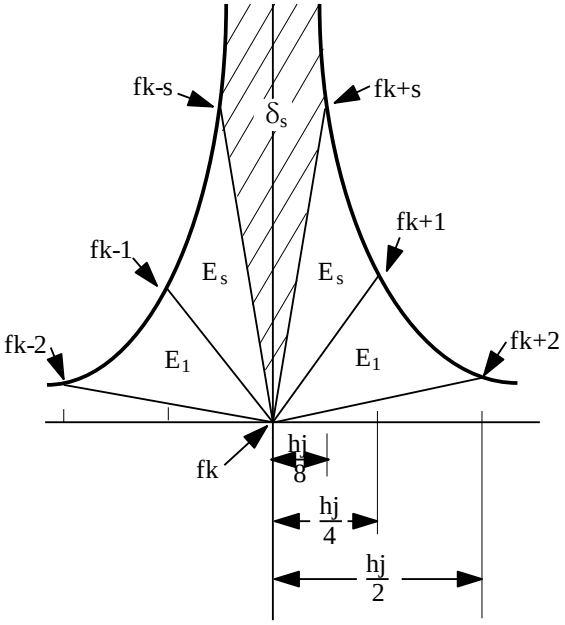
(This holds for either an endpoint or interior point.)

$$p = \frac{\log(2E_s/E_1)}{\log(2)}$$

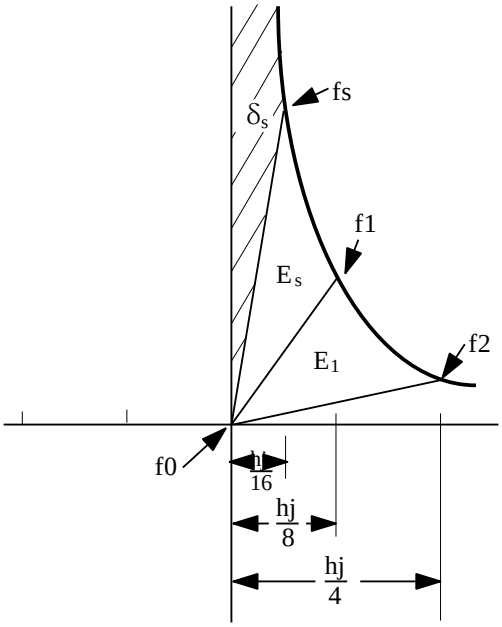
$$\left[\begin{array}{l} E_1 \text{ and } E_s \text{ are analytically calculated from function values.} \\ f_{k+s} \text{ and } f_{k-s} \text{ are function value used to calculate } \delta_s \\ \text{(See Figure 4–6(a), (b))} \end{array} \right]$$

Figure 4−6 Singular Point Processing

(a) Interior singular point



(b) End singular point



The integral value is calculated by adding δ_s obtained analytically to the integral value obtained by the trapezoidal rule for the portion where δ_s could not be added. In addition, the error is assumed to be the difference between the δ_s portion and the value obtained by the trapezoidal rule.

For singular points that cannot be detected by the methods described in (3) and (4), you can obtain a value that is as close as possible to the true value by an algorithm that improves the Aitken extrapolation method.

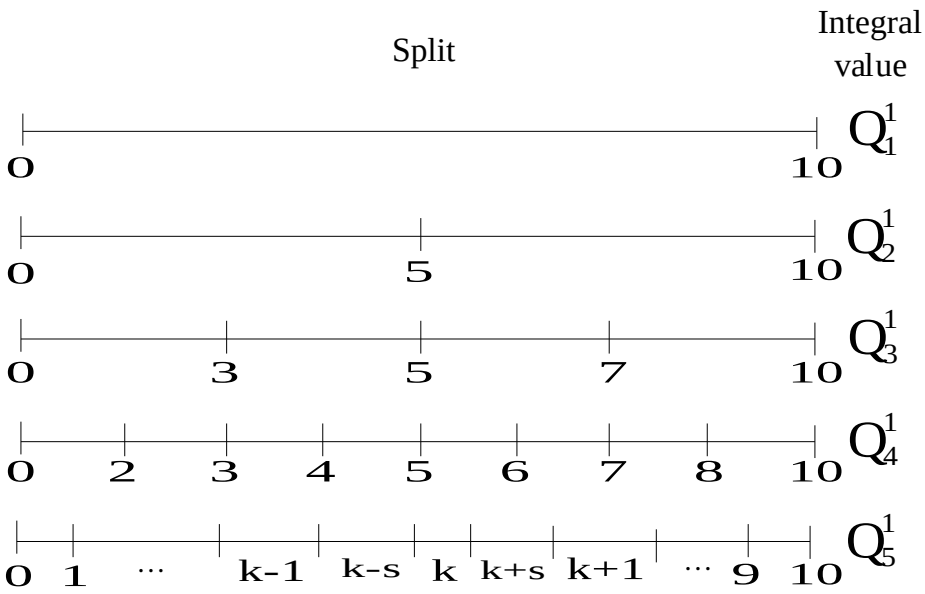
- Aitken extrapolation improvement

  Let the integral value for each subdivision be Q_n^1 for $(n = 1, \cdots, 5)$ as shown in Figure 4−7.

  Use the extrapolation formula shown below to create a table of Q_n^k for $(k = 1, 2, 3; n = 2k-1, \cdots, 5)$.

$$Q_n^k = Q_n^{k-1} - \frac{(Q_n^{k-1} - Q_{n-1}^{k-1})^2}{Q_n^{k-1} - 2Q_{n-1}^{k-1} + Q_{n-2}^{k-1}}$$

Figure 4−7 Various Subdivisions and Their Integral Values



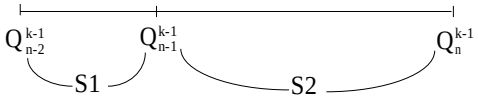
However, if $| Q_n^{k-1} - Q_{n-1}^{k-1} | \geq | Q_{n-1}^{k-1} - Q_{n-2}^{k-1} |$ then Q_n^k is Obtained from $Q_n^k = \frac{Q_n^{k-1} + mQ_{n-1}^{k-1}}{m+1}$ (m : Weight).

Finally, set integral value to Q_5^3 and the error is set to be $| Q_5^3 - Q_5^1 |$

Remarks

a. The weight is obtained from the following calculation.

$$m = \frac{| Q_n^{k-1} - Q_{n-1}^{k-1} |}{| Q_{n-1}^{k-1} - Q_{n-2}^{k-1} |} \frac{1}{15}$$



(If $S2$ is large compared to $S1$, then the weight on the Q_n^{k-1} side is decreased and a value close to Q_{n-1}^{k-1} is returned).

(5) Applications for an infinite interval

Use a variable transformation. Calculate the integrals over the interval $[0, 1]$ using the transformation shown below.

$$\begin{aligned}
\int_a^\infty f(x)dx &= \int_0^1 \frac{F(t)}{t^2}dt \\
F(t) &= f(a + \frac{1-t}{t}) \\
\int_{-\infty}^\infty f(x)dx &= \int_0^1 \frac{F(t) + G(t)}{t^2}dt \\
F(t) &= f(\frac{1-t}{t}), \quad G(t) = f(\frac{t-1}{t})
\end{aligned}$$

(6) Applications for a double integral
To evaluate the double integral:

$$I = \int_c^d \int_a^b f(x, y)dxdy$$

create a subroutine that obtains $F(y)$ defined as:

$$F(y) = \int_a^b f(x, y)dx$$

and calculate:

$$\int_c^d F(y)dy$$

while calling this subroutine.
If the integral area is not rectangular, to evaluate the double integral:

$$I = \int_c^d \int_{a(y)}^{b(y)} f(x, y)dxdy$$

create a subroutine that obtains $F(y)$ defined as:

$$A = a(y), \quad B = b(y), \quad F(y) = \int_A^B f(x, y)dx$$

and calculate:

$$\int_c^d F(y)dy$$

while calling this subroutine.

4.1.2.2 Gauss-Kronrod Method

A problem when using the group of Gauss formulas is that the previously calculated function values may not be reused when the dimension has been increased since the sampling points are the zeroes of a spherical polynomial. Therefore, Kronrod added $(n + 1)$ points to the n-point Gauss rule to create a $(3n + 1)$ degree integral rule based on $(2n + 1)$ points. However, since the n-point Gauss rule weight is not maintained, a new weight must be used for the $(2n + 1)$ points.
The non-adaptive subroutines start from the 10-point Gauss rule and modify integral values according to a Kronrod rule that increases the number of points to 21, 43, 87 or 175 points. If the differences between these modified values do not exceed the required precision, then the series is considered to have converged, and the updated value is returned as the integral value.
In a manner similar to the adaptive Newton-Cotes rule, the adaptive automatic integral subroutines continue to subdivide the integral interval until the error is within the allowable precision while adding the integral values over each subdivided interval to obtain the total integral value. However, in this case, the integral value and error are calculated by using formulas for Gauss-Kronrod pairs corresponding to 7-15 points, 10-21 points, 15-31 points, 20-41 points, 25-51 points or 30-61 points. In addition if there is no singularity information, the subroutine extrapolates the integral value of the ε-algorithm based on the 10-21 point formula.
If a trigonometric function or an algebraic or logarithmic function is used for the weight function, then adaptive

automatic integration is performed for the 7-15 point pair for the portions away from the singularities or where there is little oscillation, but the Clenshaw-Curtis rule is applied to the intervals containing singularities or where the function oscillates wildly.
To estimate the error e_j in interval j using the Gauss-Kronrod method, use the following equations:

$$\begin{aligned} e'_j &= \int_j \mid f(x) - \frac{Q_j}{h_j} \mid dx \\ e_j &= e'_j \min\left\{1.0, \frac{\xi_j}{e'_j}\right\} \end{aligned}$$

The values Q_j and ξ_j are the integral value and its revised value, respectively, in the interval j.

4.1.2.3 Clenshaw-Curtis method (functions having a weight function)

The Clenshaw-Curtis method approximates the integrand $f(x)$ by a Chebyshev polynomial of the form $\sum_{i=0}^{N}(a_i T_i(x))$, integrates this expansion to obtain the Chebyshev expansion $\sum_{i=0}^{N}(b_i T_i(x))$ corresponding to the integral value $Q(x) = \int_{-1}^{1} f(x)dx$, and obtains the integral value from this integrated expansion.
Expanding $f(x)$ as described above, we obtain:

$$f(x) = \frac{1}{2}a_0 + a_1 T_1(x) + a_2 T_2(x) \ + \cdots$$

The coefficients a_k, which are the Fourier-Chebyshev coefficients, are given by the following formula.

$$\begin{aligned} a_k &= \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{(1-x^2)^{\frac{1}{2}}}dx \\ &= \frac{2}{\pi}\int_{0}^{\pi} f(\cos(\theta))\cos(k\theta)d\theta \end{aligned}$$

Using the trapezoidal rule to approximate the calculation of a_k, we get:

$$\begin{aligned} a_k \approx \alpha_k &= \frac{1}{N}\left(f(x_0)T_k(x_0) + 2\sum_{j=1}^{N-1}(f(x_j)T_k(x_j)) + f(x_N)T_k(x_N)\right) \\ &= \frac{1}{N}\left(f(1) + 2\sum_{j=1}^{N-1}(f(\cos(\frac{\pi j}{N}))\cos(\frac{\pi k j}{N})) + (-1)^k f(-1)\right) \end{aligned}$$

If we truncate the polynomial for $f(x)$ at the N-th term, we get:

$$\begin{aligned} f(x) &\approx \frac{1}{2}a_0 + a_1 T_1(x) + a_2 T_2(x) + \cdots + a_N T_N(x) \\ &\approx \frac{1}{2}\alpha_0 + \alpha_1 T_1(x) + \cdots + \alpha_{N-1}T_{N-1}(x) + \frac{1}{2}\alpha_N T_N(x) \end{aligned}$$

The coefficients α_k can be obtained by a fast Fourier transform (FFT). In particular, Tolstov (1962) showed a simple processing method for $d = 12$. This library use $d = 12$ and $d = 24$.
When the weight function $w(x)$ is assigned to the function, the integration method is as follows.

$$\begin{aligned} &\int_{-1}^{1} w(x)f(x)dx \\ &= \frac{\alpha_0}{2}\int_{-1}^{1} w(x)dx + \sum_{k=1}^{N-1}\alpha_k \int_{-1}^{1}\{w(x)T_k(x)\}dx + \frac{\alpha_N}{2}\int_{-1}^{1}\{w(x)T_N(x)\}dx \end{aligned}$$

- When $w(x)$ is $\sin(\omega x)$

$$\begin{aligned}\int_{c_1}^{c_2} w(x)f(x)dx \quad \approx \quad & \frac{1}{2}(c_2-c_1)\Bigg[\frac{\alpha_0}{\lambda}\sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\sin(\lambda) \\ & + \sum_{k=1}^{N-1}\alpha_k\Bigg\{\cos\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\sin(\lambda x)T_k(x)dx \\ & + \sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\cos(\lambda x)T_k(x)dx\Bigg\} \\ & + \frac{\alpha_N}{2}\Bigg\{\cos\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\sin(\lambda x)T_N(x)dx \\ & + \sin\Big(\frac{(c_1+c_2)\omega}{2}\Big)\int_{-1}^{1}\cos(\lambda x)T_N(x)dx\Bigg\}\Bigg] \\ & \lambda = \frac{(c_2-c_1)\omega}{2}\end{aligned}$$

If we let $\int_{-1}^{1}\sin(\lambda x)T_k(x)dx$ be $S_k(\lambda)$ and $\int_{-1}^{1}\cos(\lambda x)T_k(x)dx$ be $C_k(\lambda)$, then $S_k(\lambda)$ and $C_k(\lambda)$ are expressed by the following recursive relations.

$$\begin{aligned}&\lambda^2(k-1)(k-2)S_{k+2}(\lambda) \; - \; 2(k^2-4)(\lambda^2-2k^2+2)S_k(\lambda) \; + \; \lambda^2(k+1)(k+2)S_{k-2}(\lambda) \\ &\quad = -8(k^2-4)\sin(\lambda) - 24\lambda\cos(\lambda)\end{aligned}$$

$$\begin{aligned}S_1(\lambda) &= 2(\sin(\lambda)-\lambda\cos(\lambda))\lambda^{-2} \\ S_3(\lambda) &= \lambda^{-2}\sin(\lambda)(18-48\lambda^{-2}) + \lambda^{-1}\cos(\lambda)(48\lambda^{-2}-2)\end{aligned}$$

$$\begin{aligned}&\lambda^2(k-1)(k-2)C_{k+2}(\lambda) \; - 2(k^2-4)(\lambda^2-2k^2+2)C_k(\lambda) \; + \; \lambda^2(k+1)(k+2)C_{k-2}(\lambda) \\ &\quad = 24\lambda\sin(\lambda) - 8(k^2-4)\cos(\lambda)\end{aligned}$$

$$\begin{aligned}C_0(\lambda) &= 2\lambda^{-1}\sin(\lambda) \\ C_2(\lambda) &= 8\lambda^{-2}\cos(\lambda) - \lambda^{-3}(2\lambda^2-8)\sin(\lambda) \\ C_4(\lambda) &= 32\lambda^{-4}(\lambda^2-12)\cos(\lambda) + 2\lambda^{-5}(\lambda^4-80\lambda^2+192)\sin(\lambda)\end{aligned}$$

$$S_{2k}(\lambda) \quad = \quad C_{2k+1}(\lambda) = 0 \;\; \text{for} \;\; (k=0,1,2\cdots)$$

These values can be similarly calculated when $w(x)$ is $\cos(\omega x)$.
If $\lambda \leq 2$, it is more efficient to use the Gauss-Kronrod rule than this method.

- When $w(x)$ is the function which has algebraic or logarithmic end-point singularities
If we let weight function be $u(x)$,

$$\begin{aligned}u(x) = (x-a)^{\gamma}(b-x)^{\delta}\{\log(x-a)\}^{\mu}\{\log(b-x)\}^{\nu} \\ (a \leq c_1 < c_2 \leq b, \mu, \nu = 0 \text{ or } 1)\end{aligned}$$

When c_1 is end-point and $c_1 = a$,

$$\int_a^{c_2} u(x)f(x)dx \approx \Big(\frac{c_2-a}{2}\Big)^{\gamma+1}\sum_{i=0}^{\mu}\{\log(c_2-a)\}^{\mu-i} \times\Big[\frac{\alpha_0}{2}\int_{-1}^{1}(1+x)^{\gamma}\Big\{\log\Big(\frac{1+x}{2}\Big)\Big\}^{i}dx+\sum_{k=1}^{N-1}\alpha_k G_{k,i}(\gamma)+\frac{\alpha_N}{2}G_{N,i}(\gamma)\Big]$$

here

$$G_{k,i}(\gamma)=\int_{-1}^{1}(1+x)^{\gamma}\left\{\log(\frac{1+x}{2})\right\}^{i}T_k(x)dx \quad \text{for} \quad (k=1,\cdots,N; i=0,1).$$

When c_2 is end-point and $c_2 = b$

$$\int_{c_1}^{b} u(x)f(x)dx \approx \Big(\frac{b-c_1}{2}\Big)^{\delta+1}\sum_{i=0}^{\nu}\{\log(b-c_1)\}^{\nu-i} \times\Big[\frac{\alpha_0}{2}\int_{-1}^{1}(1+x)^{\delta}\Big\{\log\Big(\frac{1+x}{2}\Big)\Big\}^{i}dx+\sum_{k=1}^{N-1}\alpha_k H_{k,i}(\delta)+\frac{\alpha_N}{2}H_{N,i}(\delta)\Big]$$

here

$$H_{k,i}(\delta)=(-1)^k G_{k,i}(\delta) \quad \text{for} \quad (k=1,\cdots,N; i=0,1).$$

- When $w(x)$ has interior point singularities for $\dfrac{1}{x-c}$

  If we let $v(x)=\dfrac{1}{x-c}$, then:

$$\int_{c_1}^{c_2} v(x)f(x)dx \approx \frac{\alpha_0}{2}\log|\frac{c'-1}{c'+1}|+\sum_{k=1}^{N-1}(\alpha_k V_k(c'))+\frac{\alpha_N}{2}V_N(c')$$

where:

$$c_1 < c < c_2$$
$$V_k(c')=\oint_{-1}^{+1}\frac{T_k(x)}{x-c'}dx, \qquad c'=\frac{2c-c_2-c_1}{c_2-c_1}$$

The values $V_k(c')$ are obtained by solving the following recursive relations.

$$V_{k+1}(c')-2c'V_k(c')+V_{k-1}(c')=\begin{cases}0 & k:\text{Odd Number}\\ \dfrac{4}{1-k^2} & k:\text{Even Number}\end{cases}$$

$$V_0(c') = \log|\frac{1-c'}{1+c'}|$$
$$V_1(c') = 2+c'V_0(c')$$

An adaptive-type integration is performed so that no subdivision is done at point c.

4.1.2.4 ε-algorithm

For a given series a_n for $(n = 0, 1, \cdots)$ let:

$$\begin{aligned} {\tau_n}^{(-1)} &= 0 \text{ for } (n = 0, 1, 2 \cdots) \\ {\tau_n}^{(0)} &= a_n \text{ for } (n = 0, 1, 2 \cdots) \end{aligned}$$

and starting form these values, sequentially generate the values ${\tau_n}^{(1)}, {\tau_n}^{(2)}, \cdots$ for $(k = 1, 2, 3, \cdots)$ from the equation:

$${\tau_n}^{(k)} = {\tau_{n-1}}^{(k-2)} + \frac{1}{{\tau_n}^{(k-1)} - {\tau_{n-1}}^{(k-1)}} \text{ for } (n = k, k+1, \cdots)$$

If we create the series, ${\tau_n}^{(2)}, {\tau_n}^{(6)}, {\tau_n}^{(8)}, \cdots$ in this way, it will converge to the value a_m $(m = \infty)$ faster than a_n.

$$\begin{array}{cccc} {\tau_1}^{(0)} & & & \\ {\tau_2}^{(0)} & {\tau_2}^{(1)} & & \\ {\tau_3}^{(0)} & {\tau_3}^{(1)} & {\tau_3}^{(2)} & \\ {\tau_4}^{(0)} & {\tau_4}^{(1)} & {\tau_4}^{(2)} & {\tau_4}^{(3)} \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

If you have no singularity information or if you are integrating an oscillatory function this method is used to obtain an approximate solution that is closer to the true solution by extrapolating the approximate integral solution for each step using the adaptive Gauss-Kronrod 10-21 point method or 7-15 point method. In addition, when integrating an oscillatory function over an infinite interval, it is used to extrapolate from the integral values over the number of finite intervals extending over the function cycle. (If you apply the ε-algorithm when you have no singularity information or when integrating an oscillatory function, updating proceeds by further dividing the smallest subdivided interval in half.)

4.1.2.5 Double exponential formula (integrating a function having endpoint or interior-point singularities)

The double exponential formula is an effective method of integral a function having singularities at the endpoints of the integration interval or for integrating over a semi-infinite or fully infinite interval. It obtains the integral value by converting to the function $f(\phi(u))\phi'(u)$ over the fully infinite interval $(-\infty, \infty)$, which is to be quickly reduced by a variable transformation, and applying the trapezoidal rule on a suitably truncated range. Values are further updated by dividing the step size in half until the integral value reaches the required precision. This method is described below.

(1) Let the initial step size be $h_0 = 0.25$ and let

$$W = f(\phi(0))\phi'(0)$$

(2) For $n = 1, 2, 3, \cdots$, let

$$W_n = W_{n-1} + f(\phi(nh_0))\phi'(nh_0) + f(\phi(-nh_0))\phi'(-nh_0)$$

and truncate additional terms when $| W_n - W_{n-1} | < \max(| W_n | \times$(Required relative precision), (Required absolute precision) $/(\frac{\pi}{2}h_0)) \times 0.01$
Let the truncated endpoints be $-Nh_0$ and Mh_0 and let $Q_0 = h_0 W_n$.

(3) Let the step size h_i be $\frac{1}{2}h_{i-1}$, and obtain integral values Q_i for $(i = 1, \cdots, 10)$ by the trapezoidal rule for the interval $[-Nh_0, Mh_0]$. In addition, let $Q_i - Q_{i-1} = \xi_i$.
When the following relationship holds:

$$\frac{4{\xi_i}^2}{\mid \xi_{i-1} \mid - \mid \xi_i \mid} + \mid \xi_i \mid < \max(\zeta_r Q_i, \zeta_a)$$

(ζ_r : Required relative precision, ζ_a : Required absolute precision)

Then, Q_i is returned as the integral value and $\frac{4{\xi_i}^2}{\mid \xi_{i-1} \mid - \mid \xi_i \mid} + \xi_i$ is returned as its absolute error.

(4) Estimate the integral value outside both the endpoints truncated in (b) by algebraically extrapolating from the values of terms added last and last but one, add the estimated value to the integral value, and then add the absolute value to the absolute error value.
The convergence determination method and error calculation formula carefully consider integration of a singular function. That is formula (4) of the adaptive Newton-Cotes rule is adapted to the non-adaptive trapezoidal rule to obtain $\varepsilon_i = \frac{{\xi_i}^2}{\mid \xi_{i-1} \mid - \mid \xi_i \mid}$.
If the convergence rate C always is fixed, the value of ε_i can be used to determine convergence. However, in reality, the convergence rate is not fixed and a series is assumed to have converged when the value of ε_i actually is a smaller value. Therefore, to estimate the error larger than the true error and to determine convergence safely, the calculation result for ε_i is multiplied by 4.0 to determine a safe rate, and the error is estimated by further adding $\mid \xi_i \mid$. This enables you to avoid obtaining a mistaken convergence (the actual error is larger than the estimated error and the calculation terminates before the required precision has been reached) according to the general method in which the error is assumed to be $\mid \xi_i \mid$.
If no canceling prevention transformation is performed in integration over a finite interval, then the function value is inaccurate if nh is large (about 2.5 for single precision or about 3.3 for double precision), $\phi(nh) \approx 1.0$, and there is a singularity within the range of ± 1.0 from the endpoints of $f(\phi(nh))$. Therefore, even if the integration after conversion has been obtained accurately, the conversion-time error is large, and this must be carefully considered in the integral value error. This program estimates error from the fact that convergence of (c) gets slower as the conversion-time error increases and adds this estimated error value to the estimated value for absolute error.
The convergence determination or error estimation methods described above can be used to integrate a function having no singularities without any problem other than the error estimate is a little large.
The value $\phi(nh)$ used here is as follows.
When $\int_{-1}^{1} f(x)dx$

$$\begin{aligned} x_n &= \phi(nh) = \tanh(\frac{\pi}{2}\sinh(nh)) \\ Q_h &= \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\tanh(\frac{\pi}{2}\sinh(nh)))\frac{\cosh(nh)}{\cosh^2(\pi/2\sinh(nh))} \end{aligned}$$

When $\int_{0}^{\infty} f(x)dx$

$$\begin{aligned} x_n &= \phi(nh) = \exp(\frac{\pi}{2}\sinh(nh)) \\ Q_h &= \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\exp(\frac{\pi}{2}\sinh(nh)))\cosh(nh)\exp(\frac{\pi}{2}\sinh(nh)) \end{aligned}$$

When $f(x)$ in $\int_0^\infty f(x)dx$ contains a factor of the form $\exp(-x)$

$$\begin{aligned} x_n &= \phi(nh) = \exp(\frac{\pi}{2}(nh - \exp(-nh))) \\ Q_h &= \frac{\pi}{2}h \sum_{n=-\infty}^{\infty} f(\exp\frac{\pi}{2}(nh - \exp(-nh)))(1 + \exp(-nh)) \exp\frac{\pi}{2} \\ & \quad (nh - \exp(-nh)) \end{aligned}$$

To obtain the integral value for a function having an interior-point singularity, the interval is divided at the singular point, integral values are obtained over each interval using the method described above for a function having an endpoint singularity, and the integral values are added together to obtain the total integral value.

Remarks

a. For canceling prevention

Canceling occurs if the denominator of $f(x)$ in the integral $\int_{-1}^{1} f(x)dx$ includes a factor of $(1+x)^\alpha, (1-x)^\beta$ $(0 < \alpha, \beta < 1)$. Therefore, we let $u = nh$ and the integral is divided so that:

- For $-1 \le x < 0$

$$\begin{aligned} 1 + x &= 1 + \tanh(\frac{\pi}{2}\sinh(u)) \\ &= \exp(\frac{\pi}{2}\sinh(u))/\cosh(\frac{\pi}{2}\sinh(u)) = -t_1 \end{aligned}$$

- For $0 \le x \le 1$

$$\begin{aligned} 1 - x &= 1 - \tanh(\frac{\pi}{2}\sinh(u)) \\ &= \exp(-\frac{\pi}{2}\sinh(u))/\cosh(\frac{\pi}{2}\sinh(u)) = t_2 \end{aligned}$$

and

$$\begin{aligned} \int_{-1}^{1} f(x)dx &= \int_{-1}^{0} f(x)dx + \int_{0}^{1} f(x)dx \\ &= \int_{-1}^{0} f(-1-t_1)dt_1 + \int_{0}^{1} f(1-t_2)dt_2 \end{aligned}$$

4.1.2.6 Integrating an oscillatory function over an infinite interval

If we assume the weight function is $\cos(\omega x)$ or $\sin(\omega x)$, the integrand that includes the weight function is $f(x)$, the period for which $f(x) = 0$ is λ, the phase shift is θ, and an arbitrary integer is m, then:

$$Q = \int_0^\infty f(x)dx, \quad f(m\lambda + \theta) = 0$$

Now, we assume M is a sufficiently large positive number and consider the following transformation:

$$x = M\phi(t), \quad \phi(-\infty) = 0, \quad \phi(+\infty) = 0$$

If we perform this transformation, the original integral is expressed as follows:

$$Q = \int_{-\infty}^{\infty} f(M\phi(t))M\phi'(t)dt$$

If we evaluate this by applying the trapezoidal rule with step-size h, we obtain the following expression:

$$Q \approx Mh \sum_{n=-\infty}^{\infty} f(M\phi(nh + \frac{\theta}{M}))\phi'(nh + \frac{\theta}{M}) \tag{4.6}$$

where we assume $\phi(t)$ is a function that satisfies the following:

$$\lim_{t\to\infty} \phi(t) = t$$

Then, if we take h so that $Mh = \lambda$, the following will hold for a large value of n :

$$f(M\phi(nh + \frac{\theta}{M})) \approx f(Mnh + \theta) = f(n\lambda + \theta) = 0$$

If we choose the following functions as $\phi(t)$ and $\phi'(t)$:

$$\begin{cases} \phi(t) &= \dfrac{t}{1 - \exp(-K \sinh t)} \\ \\ \phi'(t) &= \dfrac{1 - (1 + Kt \cosh t) \exp(-K \sinh t)}{(1 - \exp(-K \sinh t))^2} \end{cases} \qquad (\text{Assume } \ K = 6)$$

then as $t \to +\infty$, $\phi(t)$ approaches t and $f(M\phi(t)) \approx 0$, and as $t \to -\infty$, $\phi'(t)$ approaches 0. Also, since digits will be lost if this calculation is performed with t close to zero, the following approach is used.
If $\mid t \mid$ is less than the square root of the unit for determining error, then we choose the following functions as $\phi(t)$ and $\phi'(t)$.

$$\begin{cases} \phi(t) &= \dfrac{1}{K} \\ \phi'(t) &= 0.5 \end{cases}$$

The integration procedure is as follows.
Let the initial step size h in equation (4.6) be

$$h = \frac{3.23}{-\log((\text{Required absolute precision}))}$$

The summational interval range in equation (4.6) is expanded by 1 for both side until to satisfy below equation from Richardson's extrapolate equation and integral value is obtained.

$$\max\left\{\frac{t_1}{15}, \frac{19t_1 - t_2}{45}\right\} \times Mh < \frac{(\text{Required absolute precision})}{16}$$

where t_1 is sum of $f(M\phi(nh + \dfrac{\theta}{M}))\phi'(nh + \dfrac{\theta}{M})$ at current interval range both end-point and t_2 is a value of t_1 at previous step.
Let obtained integral value be S_1, halve the step size h repeatedly and obtain the integral value $S_2, S_3 \cdots$ similarly. When satisfy

$$\mid S_n - S_{n-1} \mid \leq \sqrt{\frac{(\text{Required absolute precision})}{10}}$$

let S_n be the integral value.

4.1.2.7 Multi-dimensional integration over a finite interval

First a function is numerically integrated in a dimension using the formula of the Gauss-Romberg rule with N sample points, and the obtained result is denoted by G_N. Next the formula is applied to numerically integrate it in the next dimension. This process is repeated for the successive dimensions. Using the Cartesian product and the product formula, the whole process is expressed as $G_N \times G_N \times \cdots$.
If the value of N is increased as $N_{n+1} = N_n + 2, N_1 = 2$ for $(n = 1, 2, \cdots)$, a series which approaches the true solution can be obtained. This series is accelerated with the following θ-algorithm, and an approximate solution is obtained.

θ-algorithm
Set $\theta_{-1}^{(n)} = 0, \theta_0^{(n)} = S^{(n)}$ ($S^{(n)}$ is the series for an approximate solution). Then

$$\theta_{2k+1}^{(n)} = \theta_{2k-1}^{(n+1)} + \frac{1}{\Delta\theta_{2k}^{n}}$$

Approximate solution

if not ($|\ \Delta\theta_{2k}^{(n)}\ | > |\ \Delta\theta_{2k}^{(n+1)}\ | > |\ \Delta\theta_{2k}^{(n+2)}\ |$) then

  if $\left(|\ \Delta\theta_{2k}^{(n+1)}\ | > |\ \Delta\theta_{2k}^{(n+2)}\ |\right)$ then

    $\theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + \dfrac{1}{\Delta\theta_{2k+1}^{(n+1)}}$ *

  else if $\left(\mathrm{sign}(\Delta\theta_{2k}^{(n+2)}) = \mathrm{sign}(\Delta\theta_{2k}^{(n+1)})\right)$ then ($\mathrm{sign}(x)$ gives a sign of value x)

    $\theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + 1.5\Delta\theta_{2k}^{(n+1)}$ **

  else

    $\theta_{2k+2}^{(n)} = \dfrac{\theta_{2k}^{(n+3)} + \theta_{2k}^{(n+1)}}{2}$ **

  end

else if $\left(|\ \Delta\theta_{2k+1}^{(n+1)}\ | \leq |\ \Delta\theta_{2k+1}^{(n)}\ |\right)$ then

  $\theta_{2k+2}^{(n)} = \theta_{2k}^{(n+2)} + \dfrac{1}{\Delta\theta_{2k+1}^{(n+1)}}$ *

else

  $\theta_{2k+2}^{(n)} = \theta_{2k}^{(n+1)} + \dfrac{\Delta\theta_{2k}^{(n+1)}\Delta\theta_{2k+1}^{(n+1)}}{\Delta^2\theta_{2k+1}^{(n)}}$

end

The estimated error in the approximate integration is

$$\varepsilon = \left\{\begin{matrix} 6 \text{ in double precision} \\ 9 \text{ in single precision} \end{matrix}\right\} \times \max(|\ \theta_{2k+2}^{(n)} - \theta_{2k}^{(n+2)}\ |, |\ \theta_{2k+2}^{(n)} - \theta_{2k}^{(n+3)}\ |)$$

for the processing without $*$, twice ε for those with $*$, and 10 times ε for those with $**$.

4.1.2.8 Integral of the product of arbitrary function and special functions

(1) Definite integral of the product of Chebyshev polynomial and
Bessel function of the order 0 $\displaystyle\int_0^1 J_0(\alpha x)T_n(2x-1)xdx$

From the Chebyshev expansion expression of $f(x)$

$$f(x) = \sum_{n=0}^{\infty} C_n T_n(2x-1)$$

the definite integral $\int_0^1 J_0(\alpha x)f(x)xdx$ is calculable as

$$\sum_{n=0}^{\infty} C_n \int_0^1 J_0(\alpha x)T_n(2x-1)xdx$$

by using applying the definite integral $\int_0^1 J_0(\alpha x)T_n(2x-1)xdx$.
Here, in $\alpha \leq 15.0$, it is more stable to use Gauss = Legendre's integration formula directly.

1. Obtaining $W_{0,1} = \int_0^1 J_0(\alpha x)dx$.
Using

$$\int_0^1 J_0(\alpha x)dx = (2N+1)^{-1}\sum_{k=0}^{2N}\frac{\sin(\alpha\cos(2\pi k/(2N+1)))}{\alpha\cos(2\pi k/(2N+1))}.$$

2. Obtaining $I_n = \int_0^1 J_0(\alpha x)T_n(2x-1)xdx$
It will be set to

$$I_n = J_1(\alpha)/\alpha + 2n^2 J_0(\alpha)/\alpha^2 - \alpha^{-2}(W_{n,3} + W_{n,2})$$

if $J_0(\alpha x)$ is expressed with this differentiation and the second degree differentiation using the differential equation of the 0th Bessel function and changing by integration by parts where

$$W_{n,1} = \int_0^1 J_0(\alpha x)T_n(2x-1)dx$$

$$W_{n,2} = \int_0^1 J_0(\alpha x)(T_n(2x-1))''xdx$$

$$W_{n,3} = \int_0^1 J_0(\alpha x)(T_n(2x-1))'dx$$

Furthermore, the relations

$$W_{0,1} = \int_0^1 J_0(\alpha x)dx$$

$$W_{0,2} = W_{0,3} = W_{1,2} = 0$$

$$W_{1,3} = 2W_{0,1}, W_{1,1} = 2I_0 - W_{0,1}, W_{2,2} = 16I_0, W_{2,3} = 16I_0 - 8W_{0,1}$$

$$W_{n,1} + 2W_{n-1,1} + W_{n-2,1} = 4I_{n-1}(n \geq 2)$$

$$W_{n,3}/n - W_{n-2,3}/(n-2) = 4W_{n-1,1}(n \geq 3)$$

$$W_{n,2}/n - W_{n-2,2}/(n-2) = (n-1)/nW_{n,3} + 2W_{n-1,3} + (n-1)/(n-2)W_{n-2,3}(n \geq 3)$$

which are obtained from the properties of Chebyshev polynomial $T_n(x)$, are used.

(2) Definite integral of the product of arbitrary function $f(x)$ and the 0th Bessel function $\int_0^1 J_0(\alpha x)f(x)xdx$
Dividing the interval [0,1] to small intervals, approximating $f(x)$ as the polynomial $P(x^2)$ in each interval $[\alpha_1, \alpha_3]$ with the condition that the values of $P(x)$ at the points $\alpha_j^2(j = 1, 2, 3)$ are $f(\alpha_j)$ and the degree of $P(x)$ is two or less, where α_2 is the middle point of each interval, calculating $\int_{\alpha_1}^{\alpha_3} J_0(\alpha x)P(x^2)xdx$ analytically, adds over all the subdivision intervals.

(3) Infinite integration $\int_{-\infty}^{\infty} e^{-x^2}f(x)dx$
$\int_{-\infty}^{\infty} e^{-x^2}f(x)dx$ can be approximated as

$$\sum_{1\leq j\leq n, -3\leq m\leq 3} w_j e^{-(z_j+2m)^2}f(z_j+2m)$$

$(z_j, w_j(j = 1, 2, \cdots, n)$ are Gauss points (zero points of Legendre polynomial of the degree n) and weights).

4.1.3 Reference Bibliography

(1) Fritsch, F. N. , Kahaner, D. K. and Lyness, J. N. , "Double Integration Using One-Dimensional Adaptive Quadrature Routines: A Software Interface Problem", ACM Trans Math. Softw. Vol. 7, pp.46-75, (1979).

(2) Patterson, "The Optimal Addition of Points to Quadrature Formulae", Math. Comp. Vol.22, (1968).

(3) Piessens and Branders, "A Note of the Optimal Addition of Abscissas to Quadrature Formulas of Gauss and Lobatto Type", Math. Comp. Vol.28, (1974).

(4) Wynn, "On the Convergence and Stability of the Epsilon Algorithm", J. SIAM Num. Anal. Vol.3, No.1, (1966).

(5) Monegato, "A Note on Extended Gaussian Quadrature Rules", Math. Comp. Vol.30, (1976).

4.2 INTEGRATION OVER A FINITE INTERVAL

4.2.1 DHEMNL, RHEMNL
Arbitrary Function

(1) **Function**

DHEMNL or RHEMNL automatically integrates the function over a finite interval. Even if the integrand has singularities, DHEMNL or RHEMNL determines its characteristics and automatically processes it. These subroutines are easy to use since the number of required input arguments has been minimized.

(2) **Usage**

Double precision:

```
CALL DHEMNL  (F, A, B, ER, Q, AE, IERR)
```

Single precision:

```
CALL RHEMNL  (F, A, B, ER, Q, AE, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ER	{D / R}	1	Input	Required relative precision (Default value : Unit for determining error ×64) (See Notes (c)) (See Section 4.1.1)
5	Q	{D / R}	1	Output	Integral value
6	AE	{D / R}	1	Output	Estimate value of absolute error (See Section 4.1.1)
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	There are five or more singular points.	Processing continues.
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value = 0.0.
1500	Restriction (b) was not satisfied.	Processing is performed with ER set to the default value.
2000	The interval was subdivided 500 times.	The minimum subdivision width is gradually widened so that an approximate solution can be obtained.
2400	A certain subdivided interval cannot be further subdivided.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3500	The result is unreliable (the error is larger than the result.)	Processing is aborted.
4000	Overflow occurred more than once in a single subdivided interval.	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If the integrand has an interior-point singularity, you should obtain the solution by having the singular point become one of the points dividing the entire interval into 2^n equal parts. In addition, if the required precision for double-precision calculations is not more lenient than the default value, the solution precision may worsen and output for the number of singular points may be greater than the actual number.
If the integrand has numerous peaks, you should increase the required precision by using the double-precision subroutine when obtaining the solution.
At other times or if you have no information about function characteristics, you should specify the precision you require as the required precision when obtaining the solution.

(c) If you input 0.0 for the variable ER, the default value is set.

(d) This subroutine uses an algorithm that is based on the adaptive Newton-Cotes 9-point rule but having more powerful singular point processing capabilities.

(7) **Example**

(a) Problem

Obtain the value of $\int_0^1 \sqrt{x}\log x dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHEMNL.

(Assume FHEMNL=0.0 when $x = 0.0$.)

A=0.0, B=1.0 and ER=0.0.

(c) Main program

```
      PROGRAM BHEMNL
! *** EXAMPLE OF DHEMNL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHEMNL
      CHARACTER FUNC*40
      DATA FUNC /'SQRT(X)*LOG(X)'/
      READ(5,*) A,B
      READ(5,*) ER
      WRITE(6,1000) FUNC,A,B,ER
      CALL DHEMNL(FHEMNL,A,B,ER,Q,AE,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE
 1000 FORMAT(' ',/,/,5X,'*** DHEMNL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ER =',G10.2)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2)
      END


      REAL(8) FUNCTION FHEMNL(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHEMNL = 0.0D0
      ELSE
         FHEMNL = SQRT(X)*LOG(X)
      ENDIF
      RETURN
      END
```

(d) Output results

```
*** DHEMNL ***

 FUNCTION = SQRT(X)*LOG(X)

 ** INPUT **

   A      =  0.00

   B      =  1.00

   ER =   0.0


 ** OUTPUT **

   IERR =    0


   INTEGRAL APPROXIMATION

     Q      = -0.4444444444D+00

   ESTIMATE OF ABSOLUTE ERROR

     AE     =  0.16E-14
```

4.2.2 DHNSNL, RHNSNL Smooth Function

(1) **Function**

DHNSNL or RHNSNL integrates a smooth function having no singularities over a finite interval.

(2) **Usage**

Double precision:

```
CALL DHNSNL  (F, A, B, ER, EA, Q, AE, NEV, IERR)
```

Single precision:

```
CALL RHNSNL  (F, A, B, ER, EA, Q, AE, NEV, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	Q	{D / R}	1	Output	Integral value
7	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
8	NEV	I	1	Output	Number of times integrand is evaluated
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by -1 or the integral value $= 0.0$
1500	Restriction (b) or (c) was not satisfied.	Processing is performed with ER or EA set to the default value.
2500	The solution precision will not reach the required precision	Processing is terminated without obtaining a solution having the required precision.
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set 0.0 is input.

(c) This subroutine uses a non-adaptive algorithm that starts with the 10-point Gauss rule and then updates the integral value according to the Kronrod rule in which the number of points is increased to 21, 43, 87 or 175 points.

(7) **Example**

(a) Problem

Obtain the value of $\int_0^1 (x^2 - 2x + 1)dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNSNL.
A=0.0, B=1.0, ER=0.0 and EA=0.0.

(c) Main program

```
      PROGRAM BHNSNL
! *** EXAMPLE OF DHNSNL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNSNL
      CHARACTER FUNC*40
      DATA FUNC /'X*X-2.*X+1.'/
      READ(5,*) A,B
      READ(5,*) ER,EA
      WRITE(6,1000) FUNC,A,B,ER,EA
      CALL DHNSNL(FHNSNL,A,B,ER,EA,Q,AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNSNL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNSNL(X)
      REAL(8) X
!
```

```
      FHNSNL = X*X-2.0D0*X+1.0D0
      RETURN
      END
```

(d) Output results

```
*** DHNSNL ***
 FUNCTION = X*X-2.*X+1.
 ** INPUT **
   A      =  0.00
   B      =  1.00
   ER     =   0.0
   EA     =   0.0

 ** OUTPUT **
   IERR =    0

   INTEGRAL APPROXIMATION
     Q      =  0.3333333333D+00
   ESTIMATE OF ABSOLUTE ERROR
     AE     =  0.15E-15
   NUMBER OF FUNCTION EVALUATIONS
     NEV    =   21
```

4.2.3 DHNOFL, RHNOFL
Function of the Type $f(x)\cdot$ ($\sin\omega x$ or $\cos\omega x$)

(1) **Function**

DHNOFL or RHNOFL integrates an oscillatory function that can be factored into $f(x)\cdot$ ($\sin\omega x$ or $\cos\omega x$) over a finite interval.

(2) **Usage**

Double precision:

```
CALL DHNOFL  (F, A, B, W, ITYPE, ER, EA, IDV, Q, AE, NEV, IWK, WK, IERR)
```

Single precision:

```
CALL RHNOFL  (F, A, B, W, ITYPE, ER, EA, IDV, Q, AE, 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	F	{D / R}	–	Input	Name defining the factor of integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	W	{D / R}	1	Input	ω of the weight function ($\sin\omega x$, $\cos\omega x$)
5	ITYPE	I	1	Input	Weight function type $1\cdots\int f(x)\cos(\omega x)dx$ $2\cdots\int f(x)\sin(\omega x)dx$
6	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
7	EA	{D / R}	1	Input	Required relative precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
8	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value:500)
9	Q	{D / R}	1	Output	Integral value
10	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
11	NEV	I	1	Output	Number of times integrand is evaluated
12	IWK	I	2,IDV	Work	Work area (a size of 1000 should be reserved if zero is entered for IDV)
13	WK	{D / R}	4,IDV	Work	Work area (a size of 2000 should be reserved if zero is entered for IDV)
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1 (except when 0 is input since the default value is assumed)

(e) ITYPE =1 or 2

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV.	Processing is terminated without obtaining a solution having the required precision.
2400	A certain subdivided interval cannot be further subdivided.	
2500	The solution precision will not reach the required precision.	
2700	Solutions obtained according to the $\varepsilon-$algorithm did not converge.	
3000	Restriction (e) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0.0 is input.

(c) This subroutine uses a 25-point modified Clenshaw-Curtis rule to integrate over intervals where the function oscillates wildly and combine this with the 13-point rule to calculate its error. In places where the function does not oscillate wildly, these subroutines calculate the integral value and its error according to the 7-15 point Gauss-Kronrod rule.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_0^{\frac{\pi}{2}} \sin x \cdot \frac{1}{\sqrt{1-0.25\sin^2 x}} dx$.

(b) Input data

Function subprogram name corresponding to factor of integrand $f(x)$: FHEMNL.

A=0.0, B=$\frac{\pi}{2}$, W=1.0, ITYPE=2, ER=0.0, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNOFL
! *** EXAMPLE OF DHNOFL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNOFL
      PARAMETER (IDV=0)
      DIMENSION IWK(2*500),WK(4*500)
      CHARACTER FUNC*40
      DATA FUNC /'SIN(X)/SQRT(1.-0.25*SIN(X)*SIN(X))'/
      READ(5,*) A,B
      READ(5,*) W
      READ(5,*) ITYPE
      READ(5,*) ER,EA
      WRITE(6,1000) FUNC,A,B,W,ITYPE,ER,EA,IDV
      CALL DHNOFL(FHNOFL,A,B,W,ITYPE,ER,EA,IDV,Q,&
                  AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNOFL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'W      =',F6.2,/,/,8X,'ITYPE  =',I6,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNOFL(X)
      REAL(8) X
!
      FHNOFL = 1.0D0/SQRT(1.0D0-0.25D0*SIN(X)*SIN(X))
      RETURN
      END
```

(d) Output results

```
 *** DHNOFL ***
  FUNCTION = SIN(X)/SQRT(1.-0.25*SIN(X)*SIN(X))
  ** INPUT **
    A      =  0.00
    B      =  1.57
    W      =  1.00
    ITYPE  =     2
    ER     =   0.0
    EA     =   0.0
    IDV    =     0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q      =  0.1098612289D+01
    ESTIMATE OF ABSOLUTE ERROR
      AE     =  0.10E-13
    NUMBER OF FUNCTION EVALUATIONS
      NEV    =   75
```

4.2.4 DHNEFL, RHNEFL

Function of the Type $f(x) \cdot ((x-a)^{\alpha}(b-x)^{\beta}\{\log(x-a)\}^{\gamma}\{\log(b-x)\}^{\delta})(a < x < b; \gamma, \delta = 0, 1)$

(1) **Function**

DHNEFL or RHNEFL integrates a function having an endpoint singularity that can be factored into $f(x) \cdot ((x-a)^{\alpha}(b-x)^{\beta}\{\log(x-a)\}^{\gamma}\{\log(b-x)\}^{\delta})(a < x < b; \gamma, \delta = 0, 1)$ over a finite interval.

(2) **Usage**

Double precision:

```
CALL DHNEFL (F, A, B, ALFA, BETA, ITYPE, ER, EA, IDV, Q, AE, NEV, IWK, WK,
             IERR)
```

Single precision:

```
CALL RHNEFL (F, A, B, ALFA, BETA, ITYPE, ER, EA, IDV, Q, AE, 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	F	{D / R}	–	Input	Name defining the factor of integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ALFA	{D / R}	1	Input	α of the weight function $(x-a)^{\alpha}(\alpha>-1)$
5	BETA	{D / R}	1	Input	β of the weight function $(b-x)^{\beta}(\beta>-1)$
6	ITYPE	I	1	Input	Weight function logarithm factor type $=1:(x-a)^{\alpha}\cdot(b-x)^{\beta}$ $=2:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(x-a)$ $=3:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(b-x)$ $=4:(x-a)^{\alpha}\cdot(b-x)^{\beta}\cdot\log(x-a)\cdot\log(b-x)$
7	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
8	EA	{D / R}	1	Input	Required absolute precision (Default value: positive minimum value $\times 2^{24}$) (See Section 4.1.1)
9	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value:500)
10	Q	{D / R}	1	Output	Integral value
11	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
12	NEV	I	1	Output	Number of times integrand is evaluated
13	IWK	I	IDV	Work	Work area (a size of 1000 should be reserved if zero is entered for IDV)
14	WK	{D / R}	$4\times$ IDV	Work	Work area (a size of 2000 should be reserved if zero is entered for IDV)
15	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1 (except when 0 is input since the default value is assumed)

(e) ITYPE = 1, 2, 3 or 4

(f) ALFA > −1.0 and BETA > −1.0

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV.	Processing is terminated without obtaining a solution having the required precision.
2400	A certain subdivided interval cannot be further subdivided.	
2500	The solution precision will not reach the required precision.	
3000	Restriction (e) or (f) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) This subroutine uses a 13-point and 25-point modified Clenshaw-Curtis rule to calculate the integral value and its error over the subdivided interval that includes the end-point and a 7-15 point Gauss-Kronrod rule over other subdivided intervals.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_0^1 \frac{\log(\frac{1}{x})}{\sqrt{x}} dx$.

(b) Input data

Function subprogram name corresponding to factor of integrand $f(x)$: FHNEFL.

(Assume FHNEFL=0.0 when $x = 0.0$.)

A=0.0, B=1.0, ALFA=−0.5, BETA=0.0, ITYPE=1, ER=0.0, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNEFL
! *** EXAMPLE OF DHNEFL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNEFL
      PARAMETER (IDV=0)
      DIMENSION IWK(500),WK(4*500)
      CHARACTER FUNC*40
      DATA FUNC /'LOG(1/X)/SQRT(X)'/
      READ(5,*) A,B
      READ(5,*) ALFA,BETA
      READ(5,*) ITYPE
      READ(5,*) ER,EA
      WRITE(6,1000) FUNC,A,B,ALFA,BETA,ITYPE,ER,EA,IDV
      CALL DHNEFL(FHNEFL,A,B,ALFA,BETA,ITYPE,ER,EA,IDV,&
                  Q,AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNEFL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ALFA   =',G10.2,/,/,8X,'BETA   =',G10.2,/,/,8X,'ITYPE  =',I6,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q    =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE   =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV  =',I5)
      END


      REAL(8) FUNCTION FHNEFL(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNEFL = 0.0D0
      ELSE
         FHNEFL = LOG(1.0D0/X)
      ENDIF
      RETURN
      END
```

(d) Output results

```
 *** DHNEFL ***
  FUNCTION = LOG(1/X)/SQRT(X)
  ** INPUT **
    A      =  0.00
    B      =  1.00
    ALFA   = -0.50
    BETA   =   0.0
    ITYPE  =     1
    ER     =   0.0
    EA     =   0.0
    IDV    =     0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q    =  0.4000000000D+01
    ESTIMATE OF ABSOLUTE ERROR
      AE   =  0.11E-12
    NUMBER OF FUNCTION EVALUATIONS
      NEV  = 4490
```

4.2.5 DHNIFL, RHNIFL
Function of the Type $f(x) \cdot (1/(x-c))$

(1) **Function**

DHNIFL or RHNIFL integrates a function having an interior-point singularity that can be factored into $f(x) \cdot (1/(x-c))$ over a finite interval.

(2) **Usage**

Double precision:

```
CALL DHNIFL  (F, A, B, C, ER, EA, IDV, Q, AE, NEV, IWK, WK, IERR)
```

Single precision:

```
CALL RHNIFL  (F, A, B, C, ER, EA, IDV, Q, AE, 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	F	{D / R}	–	Input	Name defining the factor of integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	C	{D / R}	1	Input	c of the weight function $1/(x-c)$
5	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
6	EA	{D / R}	1	Input	Required absolute precision (Default value: Unit for determining error $\times 2^{24}$) (See Section 4.1.1)
7	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value: 500)
8	Q	{D / R}	1	Output	Integral value
9	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
10	NEV	I	1	Output	Number of times integrand is evaluated
11	IWK	I	IDV	Work	Work area (a size of 1000 should be reserved if zero is entered for IDV)
12	WK	{D / R}	$4 \times$ IDV	Work	Work area (a size of 2000 should be reserved if zero is entered for IDV)
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1 (except when 0 is input since the default value is assumed)

(e) C ≠ A or B

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value = 0.0
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV.	Processing is terminated without obtaining a solution having the required precision.
2400	A certain subdivided interval cannot be further subdivided.	
2500	The solution precision will not reach the required precision.	
3000	Restriction (e) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) This subroutine uses a 13-point and 25-point modified Clenshaw-Curtis rule to calculate the integral value and its error over the subdivided interval that includes the point c and a 7-15 point Gauss-Kronrod rule over other subdivided intervals.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-1}^{5} \frac{1}{x(5x^3+6)}$.

(b) Input data

Function subprogram name corresponding to factor of integrand $f(x)$: FHNIFL.

A=−1.0, B=5.0, C=0.0, ER=1.0D−8, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNIFL
! *** EXAMPLE OF DHNIFL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNIFL
      PARAMETER (IDV=0)
      DIMENSION IWK(500),WK(4*500)
      CHARACTER FUNC*40
      DATA FUNC /'1./((5.*X**3+6.)*X)'/
      READ(5,*) A,B
      READ(5,*) C
      READ(5,*) ER,EA
      WRITE(6,1000) FUNC,A,B,C,ER,EA,IDV
      CALL DHNIFL(FHNIFL,A,B,C,ER,EA,IDV,Q,&
                  AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNIFL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'C      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END

      REAL(8) FUNCTION FHNIFL(X)
      REAL(8) X
!
      FHNIFL = 1.0D0/(5.0D0*X*X*X+6.0D0)
      RETURN
      END
```

(d) Output results

```
 *** DHNIFL ***
  FUNCTION = 1./((5.*X**3+6.)*X)
  ** INPUT **
    A      = -1.00
    B      =  5.00
    C      =  0.00
    ER     =  0.10E-07
    EA     =   0.0
    IDV    =     0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q      = -0.8994400696D-01
    ESTIMATE OF ABSOLUTE ERROR
      AE     =  0.35E-10
    NUMBER OF FUNCTION EVALUATIONS
      NEV    =  355
```

4.2.6 DHNPNL, RHNPNL
General Oscillatory or Peak-Type Function

(1) **Function**

DHNPNL or RHNPNL integrates a function having a weak singularity over a finite interval. You must determine the value of ISW corresponding to the singularity type.

(2) **Usage**

Double precision:

```
CALL DHNPNL  (F, A, B, ER, EA, IDV, Q, AE, NEV, ISW, IWK, WK, IERR)
```

Single precision:

```
CALL RHNPNL  (F, A, B, ER, EA, IDV, Q, AE, NEV, 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	F	{D / R}	–	Input	Name defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value:500)
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated
10	ISW	I	1	Input	Integer from 1 through 6, where ISW = 1 is applicable for a peak-type function and ISW = 6 is applicable for an oscillatory function. (See Notes (c))
11	IWK	I	IDV	Work	Work area (a size of 500 is reserved if zero is entered for IDV)
12	WK	{D / R}	$4 \times$ IDV	Work	Work area (a size of 2000 is reserved if zero is entered for IDV)
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER $\geq$ Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(c) EA $\geq$ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1 (except when 0 is input since the default value is assumed)

(e) $1 \leq \text{ISW} \leq 6$

(5) **Error indicator**

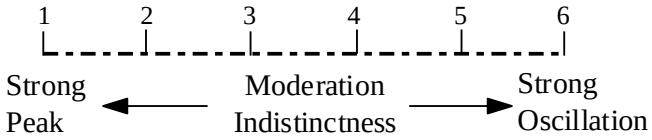
IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by -1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV	Processing is terminated without obtaining a solution having the required precision.
2400	A certain subdivided interval cannot be further subdivided.	
2500	The solution precision will not reach the required precision.	
3000	Restriction (e) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) The argument ISW should be used according to the criteria shown in the following figure.



(d) This subroutine uses an adaptive Gauss-Kronrod rule.

(7) **Example**

(a) Problem

Obtain the value of $\int_0^{0.9} \sin(10\pi x)dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNPNL.

A=0.0, B=0.9, ER=1.0D−10, EA=0.0, IDV=0 and ISW=6.

(c) Main program

```
      PROGRAM BHNPNL
! *** EXAMPLE OF DHNPNL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNPNL
      DIMENSION IWK(500),WK(4*500)
      CHARACTER FUNC*40
      DATA FUNC /'SIN(10.*PAI*X)'/
      READ(5,*) A,B
      READ(5,*) ER,EA
      READ(5,*) IDV,ISW
      WRITE(6,1000) FUNC,A,B,ER,EA,IDV,ISW
      CALL DHNPNL(FHNPNL,A,B,ER,EA,IDV,Q,&
                  AE,NEV,ISW,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNPNL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6,&
      /,/,8X,'ISW    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END

      REAL(8) FUNCTION FHNPNL(X)
      REAL(8) X,PAI
      DATA PAI /3.1415926535897932D0/
!
      FHNPNL = SIN(10.0D0*PAI*X)
      RETURN
      END
```

(d) Output results

```
*** DHNPNL ***
 FUNCTION = SIN(10.*PAI*X)
 ** INPUT **
   A      =  0.00
   B      =  0.90
   ER     =   0.0
   EA     =  0.10E-09
   IDV    =     0
   ISW    =     6

 ** OUTPUT **
   IERR =    0

   INTEGRAL APPROXIMATION
     Q      =  0.6366197724D-01
   ESTIMATE OF ABSOLUTE ERROR
     AE     =  0.26E-15
   NUMBER OF FUNCTION EVALUATIONS
     NEV    =   61
```

4.2.7 DHNENL, RHNENL
General Function Having an Endpoint Singularity

(1) **Function**

DHNENL or RHNENL integrates a general function having an endpoint singularity over a finite interval.

(2) **Usage**

Double precision:

```
CALL DHNENL  (F, A, B, ER, EA, ITMX, Q, AE, NEV, ISW, IERR)
```

Single precision:

```
CALL RHNENL  (F, A, B, ER, EA, ITMX, Q, AE, NEV, ISW, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	ITMX	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ITMX}}$; default value: 8)
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated
10	ISW	I	1	Input	ISW ≤ 0 : $f(x)$ is integrated directly. ISW ≥ 1 : A canceling prevention transformation is applied to $f(x)$.
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER $\geq$ Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(c) EA $\geq$ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) ITMX > 1 (except when 0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or ITMX set to the default value.
2000	Number of times integrand is evaluated reached ITMX	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If the integrand $f(x)$ contains a factor of $(x-a)^{\alpha}$ or $(b-x)^{\beta}$ $(-1 < \alpha, \beta < 0)$, then you must perform the following kind transformation to prevent canceling.
First, rewrite the function f as a function of y where:

$a < x < \frac{a+b}{2}$ in places where $x = a - y(-\frac{b-a}{2} < y < 0)$
$b > x \geq \frac{a+b}{2}$ in places where, $x = b - y(0 < y \leq \frac{b-a}{2})$

and set ISW = 1 and use the original interval (a, b).

For example, when calculating

$$\int_0^1 g(x)/\sqrt{x \cdot (1-x)dx}$$

this integral is expressed by:

$$\int_0^1 g(x)/\sqrt{x \cdot (1-x)}dx = \int_{-\frac{1}{2}}^0 g(-y)/\sqrt{(-y)(1+y)}dy + \int_0^{\frac{1}{2}} g(1-y)/\sqrt{(1-y)y}dy$$

so, input the following function `F` to this subroutine. Further set ISW = 1 and use *original* interval (0.0, 1.0).

Function subprogram:

```
      FUNCTION F(X)
!  Transformation to prevent canceling
      IF(X .GT. 0.0)THEN
        F = G(1-X)/SQRT(X*(1-X))
      ELSE IF(X .LT. 0.0)THEN
        F = G(-X)/SQRT(-X*(1+X))
      ENDIF
      RETURN
      END
      FUNCTION G(X)
      RETURN
      END
```

(G(X) is arbitrary function that specified by user.)

Main program:

```
          ≀

      CALL RHNENL (F, 0.0, 1.0, ER, EA, ITMX, Q, AE, NEV, 1, IERR)

          ≀
```

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine uses a double exponential formula (DE transformation formula).

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNENL.

(FHNENL=$\begin{cases} 1/\sqrt{(2-\mathrm{x})\mathrm{x}}(0 < \mathrm{x} \leq 1) \\ 1/\sqrt{-\mathrm{x}(2+\mathrm{x})}(-1 < \mathrm{x} < 0) \end{cases}$.)

A=−1.0, B=1.0, ER=0.0, EA=0.0, ITMX=0 and ISW=1.

(c) Main program

```
      PROGRAM BHNENL
! *** EXAMPLE OF DHNENL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNENL
      CHARACTER FUNC*40
      DATA FUNC /'1./SQRT(1.-X**2)'/
      READ(5,*) A,B
      READ(5,*) ER,EA
      READ(5,*) ITMX
      READ(5,*) ISW
      WRITE(6,1000) FUNC,A,B,ER,EA,ITMX,ISW
      CALL DHNENL(FHNENL,A,B,ER,EA,ITMX,Q,AE,&
                  NEV,ISW,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNENL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6,&
      /,/,8X,'ISW    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
```

```
     /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
     /,/,10X,'AE      =',G10.2,&
     /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
     END


     REAL(8) FUNCTION FHNENL(X)
     REAL(8) X
!
     IF(X.GE.0.0D0) THEN
        FHNENL = 1.0D0/SQRT((2.0D0-X)*X)
     ELSE
        FHNENL = 1.0D0/SQRT(-X*(2.0D0+X))
     ENDIF
     RETURN
     END
```

(d) Output results

```
*** DHNENL ***

 FUNCTION = 1./SQRT(1.-X**2)

 ** INPUT **

   A      = -1.00

   B      =  1.00

   ER     =   0.0

   EA     =   0.0

   ITMX   =     0

   ISW    =     1


 ** OUTPUT **

   IERR =    0


   INTEGRAL APPROXIMATION

     Q      =  0.3141592654D+01

   ESTIMATE OF ABSOLUTE ERROR

     AE     =  0.56E-14

   NUMBER OF FUNCTION EVALUATIONS

     NEV    =   65
```

4.2.8 DHNINL, RHNINL
General Function Having Interior-Point Singularities

(1) **Function**

DHNINL or RHNINL integrates a general function having interior-point singularities over a finite interval.

(2) **Usage**

Double precision:

CALL DHNINL (F, A, B, SP, NSP, ER, EA, ITMX, Q, AE, NEV, IERR)

Single precision:

CALL RHNINL (F, A, B, SP, NSP, ER, EA, ITMX, Q, AE, NEV, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	SP	{D / R}	NSP	Input	Singular point X-coordinate values
				Output	Coordinate values sorted in ascending order
5	NSP	I	1	Input	Number of singular points
6	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
7	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
8	ITMX	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ITMX}}$; default value: 8)
9	Q	{D / R}	1	Output	Integral value
10	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
11	NEV	I	1	Output	Number of times integrand is evaluated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) ITMX > 1 (except when 0 is input since the default value is assumed)

(e) NSP > 0

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or ITMX set to the default value.
2000	Number of times integrand is evaluated reached ITMX	Processing is terminated without obtaining a solution having the required precision.
2300	The integration precision is bad in a certain interval.	
2500	The solution precision will not reach the required precision.	
3000	Restriction (e) was not satisfied.	Processing is aborted.
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If the singularity is severe at a singular point, the required precision may not be achieved, and only two digits will be obtained for single precision, only four digits for double precision format. Therefore, if you require higher precision, subdivide the interval at the singular points, perform a canceling prevention transformation, and integrate using the subroutines for functions having endpoint singularities 4.2.7 $\left\{\begin{matrix}\text{DHNENL}\\\text{RHNENL}\end{matrix}\right\}$.

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine obtains the integral value between each pair of singular points by using a double exponential formula (DE transformation formula) and then they calculate the total integral value by adding these together.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-1}^{1} \frac{1}{\sqrt[3]{x^2}} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNINL.
(Assume FHNINL=0.0 when $x = 0.0$.)
A=−1.0, B=1.0, SP(1)=0.0, NSP=1, ER=1.0D−4, EA=0.0 and ITMX=0.

(c) Main program

```
      PROGRAM BHNINL
! *** EXAMPLE OF DHNINL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNINL
      PARAMETER (NSP=1)
      DIMENSION SP(NSP)
      CHARACTER FUNC*40
      DATA FUNC /'1./X**(2/3)'/
      READ(5,*) A,B
      READ(5,*) SP
      READ(5,*) ER,EA
      READ(5,*) ITMX
      WRITE(6,1000) FUNC,A,B,SP,ER,EA,ITMX
      CALL DHNINL(FHNINL,A,B,SP,NSP,ER,EA,ITMX,Q,&
                  AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) NSP,SP
      WRITE(6,1300) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNINL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'SP     =',G10.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'SORTED X-COORDINATE VALUE OF THE SINGULAR POINT',&
      /,/,10X,I2,5X,D18.10)
 1300 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNINL(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNINL = 0.0D0
      ELSE
         FHNINL = 1.0D0/(X*X)**(1.0D0/3.0D0)
      ENDIF
      RETURN
      END
```

(d) Output results

```
 *** DHNINL ***

  FUNCTION = 1./X**(2/3)

  ** INPUT **

    A      = -1.00

    B      =  1.00

    SP     =   0.0

    ER     =  0.10E-03

    EA     =   0.0
```

```
  ITMX   =     0

** OUTPUT **
  IERR =    0

  SORTED X-COORDINATE VALUE OF THE SINGULAR POINT
     1       0.0000000000D+00

  INTEGRAL APPROXIMATION
    Q      =  0.5999988818D+01
  ESTIMATE OF ABSOLUTE ERROR
    AE     =  0.32E-05
  NUMBER OF FUNCTION EVALUATIONS
    NEV    =   90
```

4.2.9 DHNANL, RHNANL
Singular Function for which Singularity Information is Unknown

(1) **Function**

DHNANL or RHNANL integrates a function having an endpoint or interior-point singularity for which singularity information is unknown over a finite interval. Calculations for these subroutines take a long time.

(2) **Usage**

Double precision:

```
CALL DHNANL  (F, A, B, ER, EA, IDV, Q, AE, NEV, IWK, WK, IERR)
```

Single precision:

```
CALL RHNANL  (F, A, B, ER, EA, IDV, Q, AE, 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	F	{D / R}	−	Input	Name defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	B	{D / R}	1	Input	Upper end of the integral interval
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value :500)
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated
10	IWK	I	IDV	Work	Work area (a size of 1000 should be reserved if zero is entered for IDV)
11	WK	{D / R}	4 × IDV	Work	Work area (a size of 2000 should be reserved if zero is entered for IDV)
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(c) EA ≥ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1 (except when 0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value over the interval (B, A) is multiplied by −1 or the integral value= 0.0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV	Processing is terminated without obtaining a solution having the required precision.
2400	A certain subdivided interval cannot be further subdivided.	
2500	The solution precision will not reach the required precision.	
2700	Solutions obtained according to the ε-algorithm did not converge.	
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) This subroutine obtains solutions based on the 10-21 point adaptive Gauss-Kronrod rule. Convergence is accelerated in the vicinity of singular points where convergence is difficult to obtain by extrapolating according to the ε-algorithm.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_0^1 \frac{\log x}{\sqrt{x}} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNANL.

(Assume FHNANL=0.0 when $x = 0.0$.)

A=0.0, B=1.0, ER=1.0D−8, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNANL
! *** EXAMPLE OF DHNANL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNANL
      PARAMETER (IDV=0)
      DIMENSION IWK(500),WK(4*500)
      CHARACTER FUNC*40
      DATA FUNC /'LOG(X)/SQRT(X)'/
      READ(5,*) A,B
      READ(5,*) ER,EA
      WRITE(6,1000) FUNC,A,B,ER,EA,IDV
      CALL DHNANL(FHNANL,A,B,ER,EA,IDV,Q,&
                  AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNANL ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,/,/,8X,'B      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNANL(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNANL = 0.0D0
      ELSE
         FHNANL = LOG(X)/SQRT(X)
      ENDIF
      RETURN
      END
```

(d) Output results

```
 *** DHNANL ***
  FUNCTION = LOG(X)/SQRT(X)
  ** INPUT **
    A      =  0.00
    B      =  1.00
    ER     =  0.10E-07
    EA     =   0.0
    IDV    =     0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q      = -0.4000000000D+01
    ESTIMATE OF ABSOLUTE ERROR
      AE     =  0.43E-12
    NUMBER OF FUNCTION EVALUATIONS
      NEV    =  357
```

4.2.10 DHBDFS, RHBDFS
Integral of Product with any Function $f(x)$ and Bessel Function $J_0(x)$

(1) **Function**

Evaluate the integrals of the product with any function $f(x)$ and Bessel function $\int_0^1 J_0(\alpha_i x) f(x) x dx$ for positive parameters α_i.

(2) **Usage**

Double precision:

CALL DHBDFS (M, NG, Z, ISW, F, B, WORK, IERR)

Single precision:

CALL RHBDFS (M, NG, Z, ISW, F, B, WORK, 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	M	I	1	Input	Number of parameter α_i
2	NG	I	1	Input	Number of data-points n
3	Z	{ D / R }	M	Input	Positive parameter α_i
4	ISW	I	1	Input	Integration method (See Notes (a))
5	F	–	–	Input	Name defining $f(x)$ (See Notes (b) and (c))
6	B	{ D / R }	M	Output	$\int_0^1 J_0(\alpha_i x) f(x) x dx$
7	WORK	{ D / R }	4×NG	Work	Work area
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) M $\geq$ 1

(b) NG $\geq$ 2

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000	The sequence did not converge in the step where Gauss points were obtained.	
5000	Overflow occurred in evaluating of Legendre polynomials.	

(6) **Notes**

(a) If ISW=0, then Gauss-NG points low is applied, but if ISW≠0, evaluate $f(x)$ in $[0,1]$ at the $2\times\mathrm{NG}-1$ points to calculate analytically.

(b) F should be defined as follows.
double precision

```
      SUBROUTINE SFUN(X,Y)
      REAL*8 X,Y
      Y=f(X)
      RETURN
      END
```

single precision

```
      SUBROUTINE SFUN(X,Y)
      REAL*4 X,Y
      Y=f(X)
      RETURN
      END
```

(c) $f(x)$ suffices to be defined in $0 \le x \le 1$.

(d) If Z has an element α greater than 15.0, it is better to use 4.2.11 $\left\{\begin{matrix}\mathrm{DHBSFC}\\ \mathrm{RHBSFC}\end{matrix}\right\}$ than to use this subroutine with setting ISW=0 (namely applying Gauss formula).

(e) It is desirable to set NG to a large value to get a good precision. On the other hand, the calculation time will increase if the order N becomes large.

(f) If the value α is not known in advance, a practical precision can be obtained by setting ISW=1 in almost cases.

(7) **Example**

(a) Problem
Set α=3.8352,4.1954,NG=50,ISW=0(Gauss formula),

$$f(r) = 0.854 - 0.483r(2.845 + 1.726r(1.429 + 0.396r(0.472 - 3.172r(1.741 - 2.621r$$

$$(2.459 - 1.637r(1.28 + 3.284r))))))$$

to evaluate $\int_0^1 J_0(\alpha x)f(x)xdx$, and compare the result with ISW=1.

(b) Main program

```
      PROGRAM BHBDFS
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER(M=2,NG=50)
      REAL(8)  AA(M), BB(M), CC(M), WORK(4*NG)
!
      EXTERNAL XTOY
```

```
!
      AA(1)=3.8352D0
      AA(2)=4.1954D0
!
      WRITE(6,90)
      WRITE(6,91)
      WRITE(6,6000) ' INPUT',AA(1),AA(2)
      CALL DHBDFS(M, NG, AA, 0, XTOY, BB, WORK, IERR)
      WRITE(6,92)
      WRITE(6,93) IERR
      WRITE(6,6000) 'G.L.  ',BB(1),BB(2)
      CALL DHBDFS(M, NG, AA, 1, XTOY, CC, WORK, IERR)
      WRITE(6,93) IERR
      WRITE(6,6000) '2-ND  ',CC(1),CC(2)
      WRITE(6,6000) 'DIST  ',CC(1)-BB(1),CC(2)-BB(2)
      STOP
   90 FORMAT(1X,' *** DHBDFS *** ',/,/)
   91 FORMAT(1X,' *** INPUT *** ')
   92 FORMAT(1X,/,/,1X,' *** OUTPUT *** ')
   93 FORMAT(1X,/,/,1X,'IERR = ',I4)
 6000 FORMAT(1X,/,/,1X,A6,3X,F12.8,3X,F12.8)
      END
      SUBROUTINE XTOY(X,Y)
      REAL(8)   X,Y
      Y=1.28D0 +3.284*X
      Y=2.459D0-1.637*X*Y
      Y=1.741D0-2.621*X*Y
      Y=0.472D0-3.172*X*Y
      Y=1.429D0+0.396*X*Y
      Y=2.845D0+1.726*X*Y
      Y=0.854D0-0.483*X*Y
      RETURN
      END
```

(c) Output results

```
 *** DHBDFS ***

 *** INPUT ***

 INPUT     3.83520000     4.19540000

 *** OUTPUT ***

IERR =    0

G.L.      -0.45758029    -0.48041798

IERR =    0

2-ND      -0.45758078    -0.48041845

DIST      -0.00000049    -0.00000047
```

4.2.11 DHBSFC, RHBSFC
Integral of the Product of Chebyshev Polynomial and Bessel Function of the Order 0

(1) **Function**

Evaluate the integral of the product of Chebyshev polynomial and Bessel function of the order 0 $\int_0^1 J_0(\alpha_i x)T_n(2x-1)xdx$ for $i = 1, \cdots$ and $n = 0, \cdots$.

(2) **Usage**

Double precision:

```
CALL DHBSFC ( N, M, Z, C, NC, WORK, IERR)
```

Single precision:

```
CALL RHBSFC ( N, M, Z, C, NC, WORK, 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	Upper bound of degree of Chebyshev polynomials n
2	M	I	1	Input	Number of parameters α_i
3	Z	{D / R}	M	Input	Parameters α_i
4	C	{D / R}	0 : NC, M	Output	$\int_0^1 J_0(\alpha_i x)T_n(2x-1)xdx (n = 0, \cdots, \mathrm{N})$
5	NC	I	1	Input	Adjustable dimension of array C
6	WORK	{D / R}	0:N,3	Work	Work area
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $1 \leq \mathrm{N} \leq \mathrm{NC}$

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

(c) Z is positive.

(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) In the case where Z has some elements less than 15.0, it is better for these values to apply 4.2.10 $\left\{\begin{matrix}\text{DHBDFS}\\ \text{RHBDFS}\end{matrix}\right\}$.

(b) N should be less than 36.

(7) **Example**

(a) Problem

Set α=3.8352,4.1954,N=7 to evaluate $\int_0^1 J_0(\alpha x)T_l(2x-1)xdx$. By this result and the Fourier coefficients of the function

$$f(r) = 0.854 - 0.483r(2.845 + 1.726r(1.429 + 0.396r(0.472 - 3.172r(1.741 - 2.621r$$

$$(2.459 - 1.637r(1.28 + 3.284r))))))$$

get the value of $\int_0^1 J_0(\alpha x)f(x)xdx$.

(b) Main program

```
      PROGRAM BHBSFC
      IMPLICIT REAL(8)(A-H,O-Z)
!
      PARAMETER(N=7,M=2,NG=50)
      REAL(8)&
         AA(M), CF(0:N,M), WORK(3*(N+1)),FR(0:N),B(M)
      CHARACTER*6 HEAD(0:N)
      DATA        HEAD/' N=0  ',' N=1  ',' N=2  ',' N=3  ',&
                       ' N=4  ',' N=5  ',' N=6  ',' N=7  '/
!
      EXTERNAL XTOY
!
      AA(1)=3.8352D0
      AA(2)=4.1954D0
!
      WRITE(6,90)
      WRITE(6,91)
      WRITE(6,6000) ' INPUT',AA(1),AA(2)
      CALL DHBSFC( N, M, AA, CF, N, WORK, IERR )
      WRITE(6,92)
      WRITE(6,93) IERR
      DO 1000 I=0, N
      WRITE(6,6000) HEAD(I),CF(I,1),CF(I,2)
 1000 CONTINUE
! *** FOURIER TRANSFORM
      PAI=ACOS(-1.D0)
      NN=NG
      DO 2000 I=0, N
         FR(I)=0
         DO 3000 K=1, NN
            X = (COS(PAI*(2*K-1)/(2*NN))+1)/2
            CALL XTOY(X,Y)
            FR(I)=FR(I)+Y*COS(I*PAI*(2*K-1)/(2*NN))
 3000    CONTINUE
         FR(I)=(2*FR(I))/NN
         IF(I.EQ.0) FR(I)=FR(I)/2
 2000 CONTINUE
! *** TSEBICHEF TO DINI(BESSEL)
      DO 4000 I=1, M
        B(I)=0
        DO 5000 J=0, N
          B(I)=B(I)+FR(J)*CF(J,I)
 5000   CONTINUE
 4000 CONTINUE
!
      WRITE(6,6000) 'COEFF ',B(1),B(2)
!
      STOP
   90 FORMAT(1X,' *** DHBSFC *** ',/,/)
   91 FORMAT(1X,' *** INPUT *** ',/,/)
   92 FORMAT(1X,' *** OUTPUT ***',/,/)
   93 FORMAT(1X,'IERR = ',I4,/,/)
 6000 FORMAT(1X,A6,3X,F10.6,3X,F10.6,/,/)
      END
      SUBROUTINE XTOY(X,Y)
      REAL(8)   X,Y
      Y=1.28D0 +3.284*X
      Y=2.459D0-1.637*X*Y
      Y=1.741D0-2.621*X*Y
      Y=0.472D0-3.172*X*Y
```

```
      Y=1.429D0+0.396*X*Y
      Y=2.845D0+1.726*X*Y
      Y=0.854D0-0.483*X*Y
      RETURN
      END
```

(c) Output results

```
 *** DHBSFC ***

 *** INPUT ***

 INPUT     3.835200     4.195400

 *** OUTPUT ***

IERR =    0

 N=0      -0.000367    -0.032670

 N=1      -0.093804    -0.101228

 N=2      -0.063934    -0.041917

 N=3       0.075850     0.089380

 N=4       0.044986     0.041434

 N=5       0.001470    -0.003408

 N=6       0.004264     0.003070

 N=7       0.004083     0.003811

COEFF     -0.457580    -0.480418
```

4.3 INTEGRATION OVER A SEMI-INFINITE INTERVAL

4.3.1 DHEMNH, RHEMNH
Arbitrary Function

(1) **Function**

DHEMNH or RHEMNH automatically integrates the function over a semi-infinite interval. Even if the integrand has singularities, DHEMNH or RHEMNH determines its characteristics and automatically processes it. These subroutines are easy to use since the number of required input arguments has been minimized.

(2) **Usage**

Double precision:

```
CALL DHEMNH  (F, A, ER, Q, AE, IERR)
```

Single precision:

```
CALL RHEMNH  (F, A, ER, Q, AE, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
4	Q	{D / R}	1	Output	Integral value
5	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ER ≥ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	There are five or more singular points.	Processing continues.
1500	Restriction (a) was not satisfied.	Processing is performed with ER set to the default value.
2000	Number of times integrand is evaluated reached 500	The minimum subdivision width is gradually widened so that an approximate solution can be obtained.
2400	A certain subdivided interval cannot be further subdivided.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.
4000	Overflow occurred more than once in a single subdivided interval.	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) When the integrand has singularities, if the required precision is not more lenient than the default value, the solution precision may worsen and the output for the number of singular points may be greater than the actual number.
If the integrand has numerous peaks, you should increase the required precision by using the double-precision subroutine when obtaining the solution.
If the integrand is oscillatory, you should use the subroutine 4.3.2 $\left\{\begin{matrix}\text{DHNOFH}\\\text{RHNOFH}\end{matrix}\right\}$.
At other times or if you have no information about function characteristics, you should specify the precision you require as the required precision when obtaining the solution.

(c) If you input 0.0 for the variable ER, the default value is set.

(d) This subroutine uses an algorithm that is based on the adaptive Newton-Cotes 9-point rule but having more powerful singular point processing capabilities.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_2^{\infty} \frac{1}{x^2} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHEMNH.

A=2.0 and ER=0.0.

(c) Main program

```
      PROGRAM BHEMNH
! *** EXAMPLE OF DHEMNH ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHEMNH
      CHARACTER FUNC*40
      DATA FUNC /'1./(X*X)'/
      READ(5,*) A
      READ(5,*) ER
      WRITE(6,1000) FUNC,A,ER
      CALL DHEMNH(FHEMNH,A,ER,Q,AE,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE
 1000 FORMAT(' ',/,/,5X,'*** DHEMNH ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,&
      /,/,8X,'ER     =',G10.2)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2)
      END


      REAL(8) FUNCTION FHEMNH(X)
      REAL(8) X
!
      FHEMNH = 1.0D0/(X*X)
      RETURN
      END
```

(d) Output results

```
 *** DHEMNH ***
  FUNCTION = 1./(X*X)
  ** INPUT **
    A      =  2.00
    ER     =   0.0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q      =  0.5000000000D+00
    ESTIMATE OF ABSOLUTE ERROR
      AE     =  0.89E-15
```

4.3.2 DHNOFH, RHNOFH
Function of the Type $f(x)\cdot$ ($\sin\omega x$ or $\cos\omega x$)

(1) **Function**

DHNOFH or RHNOFH integrates an oscillatory function that can be factored into $f(x)\cdot$ ($\sin\omega x$ or $\cos\omega x$) over a semi-infinite interval.

(2) **Usage**

Double precision:

```
CALL DHNOFH  (F, A, W, ITYPE, EA, ISY, IDV, Q, AE, NEV, IWK, WK, IERR)
```

Single precision:

```
CALL RHNOFH  (F, A, W, ITYPE, EA, ISY, IDV, Q, AE, 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	F	{D / R}	–	Input	Name defining the factor of integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	W	{D / R}	1	Input	ω of the weight function ($\sin\omega x$ or $\cos\omega x$)
4	ITYPE	I	1	Input	Weight function type $1\cdots\int f(x)\cos(\omega x)dx$ $2\cdots\int f(x)\sin(\omega x)dx$
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	ISY	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25\times 2^{-\mathrm{ISY}}$; default value: 8)
7	IDV	I	1	–	Unused
8	Q	{D / R}	1	Output	Integral value
9	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
10	NEV	I	1	Output	Number of times integrand is evaluated
11	IWK	I	1	Work	Work area (unused)
12	WK	{D / R}	1	Work	Work area (unused)
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EA $\geq$ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(b) $2 <$ ISY < 51 (except when 0 is input since the default value is assumed)

(c) ITYPE =1 or 2

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with EA, IDV, ISY set to the default value.
2100	The number of repetitions reached ISY	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3000	Restriction (c) was not satisfied.	Processing is aborted.
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument. 0.0 is input.

(c) This subroutine calculates with the variable conversion type formula for an oscillatory function semi-infinite integration.

(d) If the required absolute precision is not reached, then the solution is returned by making the convergence decision using 64× (Unit for determining error) as the relative precision.

(7) **Example**

(a) Problem
Obtain the value of $\displaystyle\int_0^\infty \frac{\sin x}{x}dx$.

(b) Input data
Function subprogram name corresponding to integrand $f(x)$: FHNOFH.
(Assume FHNOFH=0.0 when $x = 0.0$.)
A=0.0, W=1.0, ITYPE=2, EA=1.0D−8 and ISY=0.

(c) Main program

```
      PROGRAM BHNOFH
! *** EXAMPLE OF DHNOFH ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNOFH
      CHARACTER FUNC*40
      DATA FUNC /'SIN(X)/X'/
      READ(5,*) A
      READ(5,*) W
      READ(5,*) ITYPE
      READ(5,*) EA
      READ(5,*) ISY
      WRITE(6,1000) FUNC,A,W,ITYPE,EA,ISY
      CALL DHNOFH(FHNOFH,A,W,ITYPE,EA,ISY,IDV,Q,&
                  AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNOFH ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,&
      /,/,8X,'W      =',G10.2,/,/,8X,'ITYPE  =',I6,&
      /,/,8X,'EA     =',G10.2,/,/,8X,'ISY    =',I6,/,/,8X)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNOFH(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNOFH = 0.0D0
      ELSE
         FHNOFH = 1.0D0/X
      ENDIF
      RETURN
      END
```

(d) Output results

```
   *** DHNOFH ***

    FUNCTION = SIN(X)/X

    ** INPUT **

      A      =  0.00

      W      =   1.0

      ITYPE  =     2

      EA     =  0.10E-07

      ISY    =     0



    ** OUTPUT **

      IERR =    0

      INTEGRAL APPROXIMATION

        Q      =  0.1570796327D+01

      ESTIMATE OF ABSOLUTE ERROR

        AE     =  0.57E-09

      NUMBER OF FUNCTION EVALUATIONS

        NEV    =   73
```

4.3.3 DHNENH, RHNENH
General Function Having an Endpoint Singularity

(1) **Function**

DHNENH or RHNENH integrates an general function having an endpoint singularity over a semi-infinite interval.

(2) **Usage**

Double precision:

```
CALL DHNENH  (F, A, ER, EA, ITMX, Q, AE, NEV, ISW, IERR)
```

Single precision:

```
CALL RHNENH  (F, A, ER, EA, ITMX, Q, AE, NEV, ISW, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
4	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
5	ITMX	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ITMX}}$; default value: 8)
6	Q	{D / R}	1	Output	Integral value
7	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
8	NEV	I	1	Output	Number of times integrand is evaluated

No.	Argument	Type	Size	Input/ Output	Contents
9	ISW	I	1	Input	≤ 0: The integrand does not have a factor of e^{-x} or it is unknown if it has such a factor ≥ 1: The integrand has a factor of e^{-x}
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ER > Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(b) EA $\geq$ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(c) ITMX > 1 (except when 0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a), (b) or (c) was not satisfied.	Processing is performed with ER, EA or ITMX set to the default value.
2000	Processing was repeated ITMX times.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3100	After DE transformation, the function value did not become smaller sufficiently at both the + and − sides.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) This subroutine uses a double exponential formula (DE transformation formula).

(7) **Example**

(a) Problem

Obtain the value of $\int_0^\infty e^{-x}\log(x)dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNENH.

(Assume FHNENH=0.0 when $x = 0.0$.)

A=0.0, ER=1.0D−8, EA=0.0, ITMX=0 and ISW=1.

(c) Main program

```
      PROGRAM BHNENH
! *** EXAMPLE OF DHNENH ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNENH
      CHARACTER FUNC*40
      DATA FUNC /'EPS(-X)*LOG(X)'/
      READ(5,*) A
      READ(5,*) ER,EA
      READ(5,*) ITMX
      READ(5,*) ISW
      WRITE(6,1000) FUNC,A,ER,EA,ITMX,ISW
      CALL DHNENH(FHNENH,A,ER,EA,ITMX,Q,&
                  AE,NEV,ISW,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNENH ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6,&
      /,/,8X,'ISW    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNENH(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNENH = 0.0D0
      ELSE
         FHNENH = EXP(-X)*LOG(X)
      ENDIF
      RETURN
      END
```

(d) Output results

```
*** DHNENH ***
 FUNCTION = EPS(-X)*LOG(X)
 ** INPUT **
   A      =  0.00
   ER     =  0.10E-07
   EA     =   0.0
   ITMX   =     0
   ISW    =     1

 ** OUTPUT **
   IERR =    0

   INTEGRAL APPROXIMATION
     Q      = -0.5772156649D+00
   ESTIMATE OF ABSOLUTE ERROR
     AE     =  0.25E-11
   NUMBER OF FUNCTION EVALUATIONS
     NEV    =   81
```

4.3.4 DHNINH, RHNINH
General Function Having Interior-Point Singularities

(1) **Function**

DHNINH or RHNINH integrates a general function having interior-point singularities over a semi-infinite interval.

(2) **Usage**

Double precision:

CALL DHNINH (F, A, SP, NSP, ER, EA, ITMX, Q, AE, NEV, IERR)

Single precision:

CALL RHNINH (F, A, SP, NSP, ER, EA, ITMX, Q, AE, NEV, 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 defining the integrand $f(x)$
2	A	{D / R}	1	Input	Lower end of the integral interval
3	SP	{D / R}	NSP	Input	Singular point X-coordinate values
				Output	Coordinate values sorted in ascending order
4	NSP	I	1	Input	Number of singular points
5	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
6	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
7	ITMX	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ITMX}}$; default value: 8)
8	Q	{D / R}	1	Output	Integral value

No.	Argument	Type	Size	Input/ Output	Contents
9	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
10	NEV	I	1	Output	Number of times integrand is evaluated
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ER $\geq$ Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(b) EA $\geq$ Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(c) ITMX > 1 (except when 0 is input since the default value is assumed)

(d) NSP > 0

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a), (b) or (c) was not satisfied.	Processing is performed with ER, EA or ITMX set to the default value.
2000	Processing was repeated ITMX times in one interval.	Processing is terminated without obtaining a solution having the required precision.
2300	The integral precision is bad in a certain interval.	
2500	The solution precision will not reach the required precision.	
3000	Restriction (d) was not satisfied.	Processing is aborted.
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integral interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If the singularity is severe at a singular point, the required precision may not be achieved, and only two digits will be obtained for single precision, only four digits for double precision. Therefore, if you require higher precision, subdivide the interval at the singular points, perform a canceling prevention transformation for each finite interval, and integrate using the subroutines for functions having endpoint

singularities 4.2.7 $\begin{Bmatrix}\text{DHNENL}\\ \text{RHNENL}\end{Bmatrix}$, and then for the semi-infinite interval, integrate using the subroutine for functions having endpoint singularities 4.3.3 $\begin{Bmatrix}\text{DHNENH}\\ \text{RHNENH}\end{Bmatrix}$.

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine obtains the integral value between each pair of singular points by using a double exponential formula (DE transformation formula) for a semi-infinite interval and various finite intervals and then they calculate the total integral value by adding these together.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-1}^{\infty} \frac{x}{e^x - 1} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNINH.
(Assume FHNINH=0.0 when $x = 0.0$.)
A=−1.0, SP(1)=0.0, NSP=1, ER=1.0D−3, EA=0.0 and ITMX=0.

(c) Main program

```
      PROGRAM BHNINH
! *** EXAMPLE OF DHNINH ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNINH
      PARAMETER (NSP=1)
      DIMENSION SP(NSP)
      CHARACTER FUNC*40
      DATA FUNC /'X/(EXP(X)-1.)'/
      READ(5,*) A
      READ(5,*) SP
      READ(5,*) ER,EA
      READ(5,*) ITMX
      WRITE(6,1000) FUNC,A,SP,ER,EA,ITMX
      CALL DHNINH(FHNINH,A,SP,NSP,ER,EA,ITMX,Q,&
                  AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) NSP,SP
      WRITE(6,1300) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNINH ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',F6.2,&
      /,/,8X,'SP     =',G10.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'SORTED X-COORDINATE VALUE OF THE SINGULAR POINT',&
      /,/,10X,I2,5X,D18.10)
 1300 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNINH(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNINH = 0.0D0
      ELSE
         FHNINH = X/(EXP(X)-1.0D0)
      ENDIF
      RETURN
      END
```

(d) Output results

```
 *** DHNINH ***

  FUNCTION = X/(EXP(X)-1.)

  ** INPUT **

    A      = -1.00
```

```
  SP     =   0.0
  ER     =  0.10E-03
  EA     =   0.0
  ITMX   =     0

** OUTPUT **
  IERR =    0

  SORTED X-COORDINATE VALUE OF THE SINGULAR POINT
     1       0.0000000000D+00

  INTEGRAL APPROXIMATION
    Q      =  0.2922438686D+01
  ESTIMATE OF ABSOLUTE ERROR
    AE     =  0.33E-04
  NUMBER OF FUNCTION EVALUATIONS
    NEV    =   78
```

4.4 INTEGRATION OVER A FULLY INFINITE INTERVAL

4.4.1 DHEMNI, RHEMNI
Arbitrary Function

(1) **Function**

DHEMNI or RHEMNI automatically integrates the function over the fully infinite interval from $-\infty$ to ∞. The integrand can even have singularities at internal points. These subroutines are easy to use since the number of required input/output arguments has been minimized.

(2) **Usage**

Double precision:

```
CALL DHEMNI  (F, ER, Q, AE, IERR)
```

Single precision:

```
CALL RHEMNI  (F, ER, Q, AE, 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 defining the integrand $f(x)$
2	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64) (See Section 4.1.1)
3	Q	{D / R}	1	Output	Integral value
4	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
5	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ER $\geq$ Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) was not satisfied.	Processing is performed with ER set to the default value.
2000	Number of times integrand is evaluated reached 500.	The minimum subdivision width is gradually widened so that an approximate solution can be obtained.
2400	A certain subdivided interval cannot further subdivided.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.
4000	Overflow occurred more than once in a single subdivided interval.	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) When the integrand has singularities, if the required precision is not more lenient than the default value, the solution precision may worsen and the output for the number of singular points may be greater than the actual number.
If the integrand has numerous peaks, you should increase the required precision by using the double-precision subroutine when obtaining the solution.
If the integrand is oscillatory, you should use the subroutine 4.4.2 $\left\{\begin{matrix}\text{DHNOFI}\\\text{RHNOFI}\end{matrix}\right\}$.
At other times or if you have no information about function characteristics, you should specify the precision you require as the required precision when obtaining the solution.

(c) If you input 0.0 for the variable ER, the default value is set.

(d) This subroutine integrates over the fully infinite interval according to a variable transformation, using an algorithm that is based on the adaptive Newton-Cotes 9-point rule but having more powerful singular point processing capabilities.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-\infty}^{\infty} \frac{1}{1+x^2} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHEMNI.

ER=0.0.

(c) Main program

```
      PROGRAM BHEMNI
! *** EXAMPLE OF DHEMNI ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHEMNI
      CHARACTER FUNC*40
      DATA FUNC /'1./(1.+X*X)'/
      READ(5,*) ER
      WRITE(6,1000) FUNC,ER
      CALL DHEMNI(FHEMNI,ER,Q,AE,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE
 1000 FORMAT(' ',/,/,5X,'*** DHEMNI ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'ER     =',G10.2)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2)
      END


      REAL(8) FUNCTION FHEMNI(X)
      REAL(8) X
!
      FHEMNI = 1.0D0/(1.0D0+X*X)
      RETURN
      END
```

(d) Output results

```
   *** DHEMNI ***

    FUNCTION = 1./(1.+X*X)

    ** INPUT **

      ER     =   0.0


    ** OUTPUT **

      IERR =    0


      INTEGRAL APPROXIMATION

        Q      =  0.3141592654D+01

      ESTIMATE OF ABSOLUTE ERROR

        AE     =  0.56E-14
```

4.4.2 DHNOFI, RHNOFI
Function of the Type $f(x) \cdot (\sin \omega x$ or $\cos \omega x)$

(1) **Function**

DHNOFI or RHNOFI integrates an oscillatory function that can be factored into $f(x) \cdot (\sin \omega x$ or $\cos \omega x)$ over the fully infinite interval.

(2) **Usage**

Double precision:

```
CALL DHNOFI  (F, W, ITYPE, EA, ISY, IDV, Q, AE, NEV, IWK, WK, IERR)
```

Single precision:

```
CALL RHNOFI  (F, W, ITYPE, EA, ISY, IDV, Q, AE, 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	F	{D / R}	–	Input	Name defining the factor of integrand $f(x)$
2	W	{D / R}	1	Input	ω of the weight function ($\sin \omega x$ or $\cos \omega x$)
3	ITYPE	I	1	Input	Weight function type $1 \cdots \int f(x) \cos(\omega x) dx$ $2 \cdots \int f(x) \sin(\omega x) dx$
4	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum error $\times 2^{24}$) (See Section 4.1.1)
5	ISY	I	1	Input	Maximum number of repetitions between singular points (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ISY}}$; default value: 8)
6	IDV	I	1	–	Unused
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated.
10	IWK	I	1	Work	Work area (unused)
11	WK	{D / R}	1	Work	Work area (unused)
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EA $\geq$Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(b) $2 <$ ISY < 51
(except when 0 is input since the default value is assumed)

(c) ITYPE =1 or 2

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) or (b) was not satisfied.	Processing is performed with EA, or IDV set to the default value.
2100	The number of repetitions in the semi-infinite interval on one side reached ISY.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the required precision.	
3000	Restriction (c) was not satisfied.	Processing is aborted.
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	
3500	The result is unreliable (the error is larger than the result.)	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(c) This subroutine obtains the integral values over the intervals $(-\infty, 0]$ and $[0, \infty)$ based on the variable conversion formula for semi-infinite integration of an oscillatory function.

(d) If the required absolute precision is not reached, then the solution is returned by making the convergence decision using 64× (Unit for determining error) as the relative precision.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-\infty}^{\infty} \frac{\sin x}{x} dx$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNOFI.

(Assume FHNOFI=0.0 when $x = 0.0$.)

W=1.0, ITYPE=2, EA=1.0D−8 and ISY=0.

(c) Main program

```
      PROGRAM BHNOFI
! *** EXAMPLE OF DHNOFI ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNOFI
      CHARACTER FUNC*40
      DATA FUNC /'SIN(X)/X'/
      READ(5,*) W
      READ(5,*) ITYPE
      READ(5,*) EA
      READ(5,*) ISY
      WRITE(6,1000) FUNC,W,ITYPE,EA,ISY
      CALL DHNOFI(FHNOFI,W,ITYPE,EA,ISY,IDV,Q,&
                  AE,NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNOFI ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',&
      /,/,8X,'W      =',G10.2,/,/,8X,'ITYPE  =',I6,&
      /,/,8X,'EA     =',G10.2,/,/,8X,'ISY    =',I6,/,/,8X)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END

      REAL(8) FUNCTION FHNOFI(X)
      REAL(8) X
!
      IF(X.EQ.0.0D0) THEN
         FHNOFI = 0.0D0
      ELSE
         FHNOFI = 1.0D0/X
      ENDIF
      RETURN
      END
```

(d) Output results

```
*** DHNOFI ***
 FUNCTION = SIN(X)/X
 ** INPUT **
   W      =   1.0
   ITYPE  =     2
   EA     =  0.10E-07
   ISY    =     0


 ** OUTPUT **
   IERR =    0

   INTEGRAL APPROXIMATION
     Q      =  0.3141592654D+01
   ESTIMATE OF ABSOLUTE ERROR
     AE     =  0.33E-09
   NUMBER OF FUNCTION EVALUATIONS
     NEV    =  156
```

4.4.3 DHNINI, RHNINI
Function Having Interior-Point Singularities

(1) **Function**

DHNINI or RHNINI integrates a singular function over the fully infinite interval.

(2) **Usage**

Double precision:

```
CALL DHNINI  (F, SP, NSP, ER, EA, ITMX, Q, AE, NEV, IERR)
```

Single precision:

```
CALL RHNINI  (F, SP, NSP, ER, EA, ITMX, Q, AE, NEV, 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 defining the integrand $f(x)$
2	SP	{D / R}	NSP	Input	Singular point X-coordinate values
				Output	Coordinate values sorted in ascending order
3	NSP	I	1	Input	Number of singular points
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
6	ITMX	I	1	Input	Maximum number of repetitions between singular points in one interval (Minimum subdivision width after DE transformation is $0.25 \times 2^{-\mathrm{ITMX}}$; default value: 8)
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) ER ≥Unit for determining error ×64
(except when 0.0 is input since the default value is assumed)

(b) EA ≥Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(c) ITMX > 1 (except when 0 is input since the default value is assumed)

(d) NSP > 0

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a), (b) or (c) was not satisfied.	Processing is performed with ER, EA or ITMX set to the default value.
2000	A single interval was subdivided ITMX times.	Processing is terminated without obtaining a solution having the required precision.
2300	The integral precision is bad in a certain interval.	
2400	A certain subdivided interval cannot be further subdivided	
2500	The solution precision will not reach the required precision	
3000	Restriction (d) was not satisfied.	Processing is aborted.
3100	After DE transformation, the function value did not become smaller at both the + and − sides.	
3500	The result is unreliable (the error is larger than the result).	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. In addition, you must make sure that a function value overflow does not occur within the integration interval. (For example, you can set the function value at singular points to 0.0. (See Section 4.1.1)

(b) If the singularity is severe at a singular point, the required precision may not be achieved, and only two digits will be obtained for single precision, or only four digits for double precision.
Therefore, if you require higher precision, subdivide the interval at the singular points, perform a canceling prevention transformation for each finite interval, and integrate using the subroutines for functions having endpoint singularities 4.2.7 $\left\{\begin{matrix}\text{DHNENL}\\\text{RHNENL}\end{matrix}\right\}$.

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine obtains the integral value between each pair of singular points by using a double exponential formula (DE transformation formula) for semi-infinite intervals and various finite intervals and then they calculate the total integral value by adding these together.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_{-\infty}^{\infty} f(x)dx$ for $f(x) = \begin{cases} \frac{1}{x^2} & (|x| > 2) \\ x+2 & (|x| \leq 2) \end{cases}$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNINI.
SP(1)=−2.0, SP(2)=2.0, NSP=2, ER=1.0D−8, EA=0.0 and ITMX=0.

(c) Main program

```
      PROGRAM BHNINI
! *** EXAMPLE OF DHNINI ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNINI
      PARAMETER (NSP=2)
      DIMENSION SP(NSP)
      CHARACTER FUNC*60
      DATA FUNC /'1./(X*X) (ABS(X).GT.2.)  X+2. (ABS(X).LE.2.)'/
      READ(5,*) SP
      READ(5,*) ER,EA
      READ(5,*) ITMX
      WRITE(6,1000) FUNC,SP,ER,EA,ITMX
      CALL DHNINI(FHNINI,SP,NSP,ER,EA,ITMX,Q,&
                  AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) SP
      WRITE(6,1300) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNINI ***',/,/,6X,'FUNCTION = ',A60,&
      /,/,6X,'** INPUT **',&
      /,/,8X,'SP     =',G10.2,/,/,15X,'=',G10.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I5)
 1200 FORMAT(' ',/,/,8X,'SORTED X-COORDINATE VALUE OF THE SINGULAR POINT',&
      /,/,10X,' 1',5X,D18.10,/,/,10X,' 2',5X,D18.10)
 1300 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNINI(X)
      REAL(8) X
!
      IF(ABS(X).GT.2.0D0) THEN
         FHNINI = 1.0D0/(X*X)
      ELSE
         FHNINI = X+2.0D0
      ENDIF
      RETURN
      END
```

(d) Output results

```
*** DHNINI ***

 FUNCTION = 1./(X*X) (ABS(X).GT.2.)  X+2. (ABS(X).LE.2.)

 ** INPUT **

   SP     =  -2.0

          =   2.0

   ER     =   0.0

   EA     =   0.0

   ITMX   =     0

 ** OUTPUT **

   IERR =     0

   SORTED X-COORDINATE VALUE OF THE SINGULAR POINT
```

```
   1      -0.2000000000D+01
   2       0.2000000000D+01

INTEGRAL APPROXIMATION
  Q      =  0.9000000000D+01
ESTIMATE OF ABSOLUTE ERROR
  AE     =  0.30E-13
NUMBER OF FUNCTION EVALUATIONS
  NEV    =  350
```

4.4.4 DH2INT, RH2INT
Function of the Type $e^{-x^2} \cdot f(x)$

(1) **Function**

Evaluate the infinity integral with an exponential term with second degree

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx$$

(2) **Usage**

Double precision:

CALL DH2INT (N, F, W, WORK, IERR)

Single precision:

CALL RH2INT (N, F, W, WORK, 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 Gauss points
2	F	–	–	Input	Name defining $f(x)$
3	W	{ D / R }	1	Output	$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx$
4	WORK	{ D / R }	3×N	Work	Work area
5	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 2

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	The sequence did not converge in the step where Gauss points were obtained.	
5000	Overflow occurred in evaluating Legendre polynomials.	

(6) **Notes**

(a) F in double-precision should be created as **Example**.

```
SUBROUTINE SFUN(X,Y)
REAL(8) X,Y
Y=f(X)
RETURN
END
```

(b) $f(x)$ can be used if it is defined in $-7 < x < 7$. Namely, as

$$\int_{-\infty}^{-7} e^{-x^2} dx + \int_{7}^{\infty} e^{-x^2} dx < 7.415 \cdot 10^{-23},$$

in $x \leq -7$ and $x \geq 7$, the functional value $f(x)$ is omitted to get the relational precision $7.4 \cdot 10^{-16}$.

(7) **Example**

(a) Problem

Evaluate

$$\int_{-\infty}^{\infty} \frac{e^{-x^2}}{x^2+1} dx$$

(true value is $\pi e \operatorname{Erfc}(1)$).

(b) Main program

```
      PROGRAM BH2INT
      IMPLICIT REAL(8)(A-H,O-Z)
      PARAMETER(N=24,ONE=1.D0)
!     GAMMA6 = GAMMA(6.D0) = 5*4*3*2*1
      PARAMETER(GAMMA6=120.D0)
      REAL(8)    WORK(3*N)
!
      EXTERNAL  FN, FN1
!
      SMALLE = EXP  (ONE)
      PI     = ACOS(-ONE)
!     ERFC1  = ERFC (ONE)
      CALL WIERFC(1, ONE, ERFC1, IERR)
      WW = PI*SMALLE*ERFC1
!
      WRITE(6,90)
      WRITE(6,91)
      WRITE(6,92) N
      CALL DH2INT(N, FN, W, WORK, IERR)
      WRITE(6,93)
      WRITE(6,94) IERR
      WRITE(6,6000) W, WW
      CALL DH2INT(N, FN1,W, WORK, IERR)
      WRITE(6,94) IERR
      WRITE(6,6000) W, GAMMA6
      STOP
   90 FORMAT(1X,' *** DH2INT *** ',/,/)
   91 FORMAT(1X,' *** INPUT *** ',/,/)
   92 FORMAT(1X,' N = ',I2,/,/)
   93 FORMAT(1X,' *** OUTPUT *** ',/,/)
   94 FORMAT(1X,'IERR = ',I4,/,/)
 6000 FORMAT(1X, 'OUTPUT VALUE = ',E15.7,' TRUE = ',E15.7,/,/)
      END
      SUBROUTINE FN(X,Y)
      REAL(8)         X,Y
      Y=1.D0/(X*X+1.D0)
      RETURN
      END
      SUBROUTINE FN1(X,Y)
      REAL(8)         X,Y
      REAL(8)         Z,Z2
      Z=ABS(X)
      Z2=Z*Z
      Y=Z2**(6-1)
      Y=Y*Z
      RETURN
      END
```

(c) Output results

```
 *** DH2INT ***


 *** INPUT ***
```

```
 N = 24

 *** OUTPUT ***

IERR =    0

OUTPUT VALUE =   0.1343293E+01 TRUE =   0.1343293E+01

IERR =    0

OUTPUT VALUE =   0.1200000E+03 TRUE =   0.1200000E+03
```

4.5 INTEGRATION OVER A TWO-DIMENSIONAL FINITE INTERVAL

4.5.1 DHNRNM, RHNRNM
Two-Dimensional Integration over a Rectangular Area

(1) **Function**
DHNRNM or RHNRNM automatically performs two-dimensional integration over a rectangular area.

(2) **Usage**
Double precision:
CALL DHNRNM (F, A, B, C, D, ER, EA, IDV, Q, AE, NEV, IERR)
Single precision:
CALL RHNRNM (F, A, B, C, D, ER, EA, IDV, Q, AE, NEV, 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 defining the integrand $f(x, y)$
2	A	{D / R}	1	Input	Lower end of the integral interval in the X-axis direction
3	B	{D / R}	1	Input	Upper end of the integral interval in the X-axis direction
4	C	{D / R}	1	Input	Lower end of the integral interval in the Y-axis direction
5	D	{D / R}	1	Input	Upper end of the integral interval in the Y-axis direction
6	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
7	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
8	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value: 5000)
9	Q	{D / R}	1	Output	Integral value
10	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
11	NEV	I	1	Output	Number of times integrand is evaluated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $A < B, C < D$

(b) ER $\geq$Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(c) EA $\geq$Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1
(except when 0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value is multiplied by -1 or the integral value = 0. (If D < C and B < A, the integral value is used as it is.)
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV	The minimum subdivision width is gradually widened so that an approximate solution can be obtained.
2400	A certain subdivided interval cannot be further subdivided.	Processing is terminated without obtaining a solution having the required precision.
2500	The solution precision will not reach the require precision.	
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.
4000	Overflow occurred more than once in a single subdivided interval.	

(6) **Notes**

(a) You must declare the actual name of the first argument F in an EXTERNAL statement in the user program and you must create a function subprogram having that name. (See Section 4.1.1)

(b) If the integrand has a peak in an extremely narrow range, you should increase the required precision by using the double-precision subroutine when obtaining the solution. An appropriate value for the required relative precision is up to the square root of the value of the unit for determining error.

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine uses an algorithm based on the adaptive Newton-Cotes 9-point rule but having more powerful singular point processing capabilities and extended to two dimensions.

(7) **Example**

(a) Problem

Obtain the value of $\int_0^2 \int_0^2 (x+y)dxdy$.

(b) Input data

Function subprogram name corresponding to integrand $f(x, y)$: FHNRNM.
A=0.0, B=2.0, C=0.0, D=2.0, ER=0.0, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNRNM
! *** EXAMPLE OF DHNRNM ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNRNM
      CHARACTER FUNC*40
      DATA FUNC /'X+Y'/
      READ(5,*) A,B,C,D
```

```
      READ(5,*) ER,EA
      READ(5,*) IDV
      WRITE(6,1000) FUNC,A,B,C,D,ER,EA,IDV
      CALL DHNRNM(FHNRNM,A,B,C,D,ER,EA,IDV,&
                  Q,AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNRNM ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A =',F6.2,/,/,8X,'B =',F6.2,&
      /,/,8X,'C =',F6.2,/,/,8X,'D =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I5)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNRNM(X,Y)
      REAL(8) X,Y
!
      FHNRNM = X+Y
      RETURN
      END
```

(d) Output results

```
 *** DHNRNM ***
  FUNCTION = X+Y
  ** INPUT **
    A =  0.00
    B =  2.00
    C =  0.00
    D =  2.00
    ER     =   0.0
    EA     =   0.0
    IDV    =    0

  ** OUTPUT **
    IERR =    0

    INTEGRAL APPROXIMATION
      Q      =  0.8000000000D+01
    ESTIMATE OF ABSOLUTE ERROR
      AE     =  0.18E-13
    NUMBER OF FUNCTION EVALUATIONS
      NEV    =  441
```

4.5.2 DHNFNM, RHNFNM
Two-Dimensional Integration over an Area Indicated by the Function

(1) **Function**

DHNFNM or RHNFNM automatically performs two-dimensional integration over an arbitrary area in which the integral range in the X-axis direction is given as a function of y.

Calculate the value of integration $\int_c^d \int_{a(y)}^{b(y)} f(x,y)dxdy$

(2) **Usage**

Double precision:

```
CALL DHNFNM  (F, A, B, C, D, ER, EA, IDV, Q, AE, NEV, IERR)
```

Single precision:

```
CALL RHNFNM  (F, A, B, C, D, ER, EA, IDV, Q, AE, NEV, 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 defining the integrand $f(x, y)$
2	A	{D / R}	–	Input	Function $a(y)$ giving the lower end of the integral interval in the X-axis direction
3	B	{D / R}	–	Input	Function $b(y)$ giving the upper end of the integral interval in the X-axis direction
4	C	{D / R}	1	Input	Lower end of the integral interval in the Y-axis direction
5	D	{D / R}	1	Input	Upper end of the integral interval in the Y-axis direction
6	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error $\times 64$) (See Section 4.1.1)
7	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24}$) (See Section 4.1.1)
8	IDV	I	1	Input	Maximum number of subdivided intervals for normal processing (Default value: 5000)
9	Q	{D / R}	1	Output	Integral value
10	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
11	NEV	I	1	Output	Number of times integrand is evaluated
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) C < D

(b) ER ≥Unit for determining error $\times 64$
(except when 0.0 is input since the default value is assumed)

(c) EA ≥Positive minimum value $\times 2^{24}$
(except when 0.0 is input since the default value is assumed)

(d) IDV > 1
(except when 0 is input since the default value is assumed)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	Restriction (a) was not satisfied.	The integral value is multiplied by -1 or the integral value = 0.
1500	Restriction (b), (c) or (d) was not satisfied.	Processing is performed with ER, EA or IDV set to the default value.
2000	Number of times integrand is evaluated reached IDV	The minimum subdivision width is gradually widened so that an approximate solution can be obtained.
2400	A certain subdivided interval cannot be further subdivided.	Processing is terminated without obtaining a solution having the required precision
2500	The solution precision will not reach the required precision	
3500	The result is unreliable (the error is larger than the result).	Processing is aborted.
4000	Overflow occurred more than once in a single subdivided interval.	

(6) **Notes**

(a) You must declare the actual names of the first three arguments F, A, and B in an EXTERNAL statement in the user program and you must create function subprograms having the actual names of F, A, and B. (See Section 4.1.1)

(b) If the integrand has a peak in an extremely narrow range, you should increase the required precision by using the double-precision subroutine when obtaining the solution. An appropriate value for the required relative precision is up to the square root of the value of the unit for determining error.

(c) If a default value is shown in the "contents" column of the argument table, then the default value is set if 0 is input for an integer-type argument or 0.0 is input for a real-type argument.

(d) This subroutine uses an algorithm based on the adaptive Newton-Cotes 9-point rule but having more powerful singular point processing capabilities and extended to two dimensions.

(7) **Example**

(a) Problem

Obtain the value of $\displaystyle\int_0^2 \int_0^{\frac{\sqrt{4-y^2}}{2}} (x+y)dxdy$.

(b) Input data

Function subprogram name corresponding to integrand $f(x)$: FHNFNM.
Function subprogram name corresponding to function giving the lower end and upper end of the integral interval in the X-axis direction: FHAFNM and FHBFNM.
C=0.0, D=2.0, ER=1.0D−8, EA=0.0 and IDV=0.

(c) Main program

```
      PROGRAM BHNFNM
! *** EXAMPLE OF DHNFNM ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNFNM,FHNFN2,FHNFN3
```

```
      CHARACTER FUNC1*40,FUNC2*40,FUNC3*40
      DATA FUNC1 /'X+Y'/
      DATA FUNC2 /'0.0'/
      DATA FUNC3 /'SQRT(4-Y*Y)/2'/
      READ(5,*) C,D
      READ(5,*) ER,EA
      READ(5,*) IDV
      WRITE(6,1000) FUNC1,FUNC2,FUNC3,C,D,ER,EA,IDV
      CALL DHNFNM(FHNFNM,FHNFN2,FHNFN3,C,D,ER,EA,IDV,&
                  Q,AE,NEV,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE,NEV
 1000 FORMAT(' ',/,/,5X,'*** DHNFNM ***',/,/,6X,'FUNCTION',&
      /,/,8X,'FUNCTION1 = ',A40,/,/,8X,'FUNCTION2 = ',A40,&
      /,/,8X,'FUNCTION3 = ',A40,/,/,6X,'** INPUT **',&
      /,/,8X,'C =',F6.2,/,/,8X,'D =',F6.2,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'IDV    =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q      =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE     =',G10.2,&
      /,/,8X,'NUMBER OF FUNCTION EVALUATIONS',/,/,10X,'NEV    =',I5)
      END


      REAL(8) FUNCTION FHNFNM(X,Y)
      REAL(8) X,Y
!
      FHNFNM = X+Y
      RETURN
      END


      REAL(8) FUNCTION FHNFN2(Y)
      REAL(8) Y
!
      FHNFN2 = 0.0D0*Y
      RETURN
      END


      REAL(8) FUNCTION FHNFN3(Y)
      REAL(8) Y
!
      FHNFN3 = 0.5D0*SQRT(4.0D0-Y*Y)
      RETURN
      END
```

(d) Output results

```
*** DHNFNM ***
 FUNCTION
   FUNCTION1 = X+Y
   FUNCTION2 = 0.0
   FUNCTION3 = SQRT(4-Y*Y)/2
 ** INPUT **
   C =  0.00
   D =  2.00
   ER     =  0.10E-07
   EA     =   0.0
   IDV    =     0

 ** OUTPUT **
   IERR =    0

   INTEGRAL APPROXIMATION
     Q      =  0.2000000000D+01
   ESTIMATE OF ABSOLUTE ERROR
     AE     =  0.14E-09
   NUMBER OF FUNCTION EVALUATIONS
     NEV    = 3990
```

4.6 MULTI-DIMENSIONAL INTEGRATION OVER A FINITE INTERVAL

4.6.1 DHNRML, RHNRML
Multi-Dimensional Integration over a Hypercubic Space

(1) **Function**

DHNRML or RHNRML performed a multi-dimensional integration of a function over a hypercubic space of more than two dimensions. (If the function to be integrated is two-dimensional and has singularities, a subroutine for the two-dimensional integration is recommended.)

(2) **Usage**

Double precision:

CALL DHNRML (F, A, B, M, ER, EA, ITMX, Q, AE, NEV, IWK, WK, IERR)

Single precision:

CALL RHNRML (F, A, B, M, ER, EA, ITMX, Q, AE, 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	F	{D / R}	–	Input	Function subprogram name of integrand $f(x_1, \cdots, x_m)$
2	A	{D / R}	M	Input	a_i is lower limit of integration in x_i-axis direction. $i = 1, 2, \cdots, m$
3	B	{D / R}	M	Input	b_i is upper limit of integration in x_i-axis direction. $i = 1, 2, \cdots, m$
4	M	I	1	Input	Dimensionality of integration
5	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64×M) (See Section 4.1.1)
6	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24} \times$ M) (See Section 4.1.1)
7	ITMX	I	1	Input	Maximum number of repetitions (Default value: 60/M)
8	Q	{D / R}	1	Output	Integral value
9	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
10	NEV	I	1	Output	Number of times integrand is evaluated
11	IWK	I	M	Work	Work area
12	WK	{D / R}	3×M	Work	Work area
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $2 \leq \mathrm{M} \leq 9$

(b) ER $\geq$ Unit for determining error $\times 64 \times$ M
(except when 0.0 is input to set the default)

(c) EA $\geq$ Positive minimum value $\times 2^{24} \times$ M
(except when 0.0 is input to set the default)

(d) $4 \leq \mathrm{ITMX} \leq 30$ (except when 0 is input to set the default)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	$a_i > b_i$, $a_i = b_i$	$\int_{a_i}^{b_i} f dx_i = -\int_{b_i}^{a_i} f dx_i$. Changes the integral as above and integrates it. Or the integral is zero.
1500	Restriction (b), (c) or (d) was not satisfied.	Processes with default values.
2500	Repetition was terminated before the required precision of solution was reached.	Terminates processing without obtaining the required precision. (Solution with the best precision at the time is returned.)
3000	Restriction (a) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	Returns the obtained result.

(6) **Notes**

(a) The actual name of the first argument F must be defined in an EXTERNAL statement in the user's program. Furthermore a function subprogram with the name must be created must be created in advance. (See Section 4.1.1)
The function subprogram (in double-precision) should be created as follows:

```
REAL(8) FUNCTION F(X, M)
REAL(8) X]
DIMENSION X(M)
F=∼
RETURN
END
```

(b) The required relative precision should be approximately $\sqrt[M]{10^{-6}}$ if there is any singularity (a point where a function is not differentiable since its derivative is discontinuous or ∞) in the interval of integration, approximately MAX($\sqrt[M]{10^{-12}}, 10^{-4}\times$M) if there is any bad singularity (a point where a function becomes ∞) at the limits of integration or approximately $\sqrt{\text{Unit for determining error}}\times M^2/20$ in other cases.

(c) If a default value is available for an argument, input 0 for the integer type or 0.0 for the real type to set the default value.

(d) This subroutine obtains the value of an integral by accelerating, with the improved algorithm, the solution series which is obtained by increasing N of the N-point Gauss-Romberg rule in each dimension.

(7) **Example**

(a) Problem

Obtain the value of $\int_0^1 \int_0^1 \int_0^1 1/(3-\cos(\pi x)-\cos(\pi y)-\cos(\pi z))dxdydz$.

(b) Input data

Function subprogram name corresponding to integrand $f(x_1, \cdots, x_m)$: FHNRML.

A(1)=A(2)=A(3)=0.0, B(1)=B(2)=B(3)=1.0, M=3, ER=1.0D−4, EA=0.0 and ITMX=15.

(c) Main program

```
      PROGRAM BHNRML
! *** EXAMPLE OF DHNRML ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNRML
      DIMENSION A(3),B(3)
      DIMENSION IWK(3),WK(3*3)
      CHARACTER FUNC*40
      DATA FUNC /'1./(3.-COS(PAI*X)-COS(PAI*Y)-COS(PAI*Z))'/
      READ(5,*) (A(I),I=1,3)
      READ(5,*) (B(I),I=1,3)
      READ(5,*) M
      READ(5,*) ER
      READ(5,*) EA
      READ(5,*) ITMX
      WRITE(6,1000) FUNC,(A(I),I=1,3),(B(I),I=1,3),M,&
                    ER,EA,ITMX
      CALL DHNRML(FHNRML,A,B,M,ER,EA,ITMX,Q,AE,&
                  NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE
 1000 FORMAT(' ',/,/,5X,'*** DHNRML ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      =',3(F6.2,/,16X),&
      /,/,8X,'B      =',3(F6.2,/,16X),/,/,8X,'M      =',I3,&
      /,/,8X,'ER     =',G10.2,/,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q   =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE  =',G10.2)
      END


      REAL(8) FUNCTION FHNRML(X,M)
      REAL(8) X,PAI
      DIMENSION X(M)
      DATA PAI /3.14159265358979323 85D0/
!
      FHNRML = 1.0D0/(3.0D0-COS(PAI*X(1))-COS(PAI*X(2))-COS(PAI*X(3)))
      RETURN
      END
```

(d) Output results

```
 *** DHNRML ***

  FUNCTION = 1./(3.-COS(PAI*X)-COS(PAI*Y)-COS(PAI*Z))

  ** INPUT **

    A      =  0.00
              0.00
              0.00


    B      =  1.00
              1.00
              1.00


    M      =  3

    ER     =  0.10E-03

    EA     =   0.0

    ITMX   =    15


  ** OUTPUT **

    IERR =    0


    INTEGRAL APPROXIMATION

      Q   =  0.5054622128D+00
```

```
ESTIMATE OF ABSOLUTE ERROR
  AE  =  0.12E-04
```

4.6.2 DHNFML, RHNFML
Multi-Dimensional Integration over a Space Indicated by a Function

(1) **Function**

DHNFML, RHNFML performs a multi-dimensional integration of a function over a space indicated by a function of more than two dimensions. (If the function to be integrated is two-dimensional and has singularities, a subroutine for the two-dimensional integration is recommended.)

Performs the following integration

$$\int_{a_m}^{b_m} \int_{a_{m-1}}^{b_{m-1}} \cdots \int_{a_1}^{b_1} f(x_1, \cdots, x_m) dx_1 \cdots dx_m$$

here,

$$a_i = f_i(x_{i+1}, \cdots, x_m),\ b_i = g_i(x_{i+1}, \cdots, x_m); i = 1, 2, \cdots, m-1$$
$$a_m = f_m,\ b_m = g_m$$

(2) **Usage**

Double precision:

CALL DHNFML (F, R, M, ER, EA, ITMX, Q, AE, NEV, IWK, WK, IERR)

Single precision:

CALL RHNFML (F, R, M, ER, EA, ITMX, Q, AE, 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	F	{D / R}	–	Input	Function subprogram name of integrand $f(x_1, \cdots, x_m)$
2	R	–	–	Input	Name of subroutine which gives lower limit a_i and upper limit $b_i (i = 1, \cdots, m)$ of integration
3	M	I	1	Input	Dimensionality of integration
4	ER	{D / R}	1	Input	Required relative precision (Default value: Unit for determining error ×64×M) (See Section 4.1.1)
5	EA	{D / R}	1	Input	Required absolute precision (Default value: Positive minimum value $\times 2^{24} \times$ M) (See Section 4.1.1)
6	ITMX	I	1	Input	Maximum number of repetitions (Default value: 60/M)
7	Q	{D / R}	1	Output	Integral value
8	AE	{D / R}	1	Output	Estimated value of absolute error (See Section 4.1.1)
9	NEV	I	1	Output	Number of times integrand is evaluated
10	IWK	I	M	Work	Work area
11	WK	{D / R}	3×M	Work	Work area
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $2 \leq \mathrm{M} \leq 9$

(b) ER $\geq$ Unit for determining error $\times 64 \times$ M
(except when 0.0 is input to set the default)

(c) EA $\geq$ Positive minimum value $\times 2^{24} \times$ M
(except when 0.0 is input to set the default)

(d) $4 \leq \mathrm{ITMX} \leq 30$ (except when 0 is input to set the default)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1200	$a_i > b_i$, $a_i = b_i$	$\int_{a_i}^{b_i} f dx_i = -\int_{b_i}^{a_i} f dx_i$. Changes the integral as above and integrates it. Or the integral is zero.
1500	Restriction (b), (c) or (d) was not satisfied.	Processes with default values.
2500	Repetition was terminated before the required precision of solution was reached.	Terminates processing without obtaining the required precision. (Solution with the best precision at the time is returned.)
3000	Restriction (a) was not satisfied.	Processing is aborted.
3500	The result is unreliable (the error is larger than the result).	Returns the obtained result.

(6) **Notes**

(a) The actual name of the first argument F must be defined in an EXTERNAL statement in the user's program. Furthermore a function subprogram with the name must be created must be created in advance. (See Section 4.1.1)
The function subprogram (in double-precision) should be created as follows:

```
REAL(8) FUNCTION F(X, M)
REAL(8) X
DIMENSION X(M)
F=∼
RETURN
END
```

(b) The actual name of the second argument R must be defined in an EXTERNAL statement in the user's program. Furthermore a subroutine subprogram with the name must be created in advance. The subroutine subprogram (in double-precision) should be created as follows:

- Example of subroutine

```
SUBROUTINE R(I, X, A, B, M)
REAL(8) X, A, B
DIMENSION X(M), A(M), B(M)

IF (I .EQ. 1) THEN
    A(1)=f1(X(2),···,X(M))
    B(1)=g1(X(2),···,X(M))
ELSE IF (I .EQ. 2) THEN
    A(2)=f2(X(3),···,X(M))
    B(2)=g2(X(3),···,X(M))
        .
        .
        .
```

ELSE IF (I .EQ. M−1) THEN
A(M−1)=$f_{M-1}(X(M))$
B(M−1)=$g_{M-1}(X(M))$
ELSE IF (I .EQ. M−1) THEN
A(M)=f_M
B(M)=g_M
ENDIF
RETURN
END

(c) The required relative precision should be approximately $\sqrt[M]{10^{-6}}$ if there is any singularity (a point where a function is not differentiable since its derivative is discontinuous or ∞) in the interval of integration, approximately MAX($\sqrt[M]{10^{-12}}, 10^{-4}\times$M) if there is any bad singularity (a point where a function becomes ∞) at the limits of integration or approximately $\sqrt{\text{Unit for determining error}}\times M^2/20$ in other cases.

(d) If a default value is available for an argument, input 0 for the integer type or 0.0 for the real type to set the default value.

(e) This subroutine obtains the value of an integral by accelerating, with the improved θ-algorithm, the solution series which is obtained by increasing N of the N-point Gauss-Romberg rule in each dimension.

(7) **Example**

(a) Problem

$$\int_0^1 \int_0^{\sqrt{1-z^2}} \int_0^{\sqrt{1-y^2-z^2}} \sqrt{1-x^2-y^2-z^2}\,dxdydz$$

(b) Input data
Function subprogram name corresponding to integrand $f(x_1,\cdots,x_m)$: FHNFML.
Name of subroutine which gives lower limit and upper limit of integration: FHNSUB.
M=3, ER=1.0D−8, EA=0.0 and ITMX=15.

(c) Main program

```
      PROGRAM BHNFML
! *** EXAMPLE OF DHNFML ***
      IMPLICIT REAL(8) (A-H,O-Z)
      EXTERNAL FHNFML
      EXTERNAL FHNFM2
      DIMENSION IWK(3),WK(3*3)
      CHARACTER FUNC*40
      DATA FUNC /'SQRT(1.-X*X-Y*Y-Z*Z)'/
      READ(5,*) M
      READ(5,*) ER
      READ(5,*) EA
      READ(5,*) ITMX
      WRITE(6,1000) FUNC,M,ER,EA,ITMX
      CALL DHNFML(FHNFML,FHNFM2,M,ER,EA,ITMX,Q,AE,&
                  NEV,IWK,WK,IERR)
      WRITE(6,1100) IERR
      WRITE(6,1200) Q,AE
 1000 FORMAT(' ',/,/,5X,'*** DHNFML ***',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'M      =',I3,/,/,8X,'ER     =',G10.2,&
      /,/,8X,'EA     =',G10.2,/,/,8X,'ITMX   =',I6)
 1100 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1200 FORMAT(' ',/,/,8X,'INTEGRAL APPROXIMATION',/,/,10X,'Q   =',D18.10,&
      /,/,8X,'ESTIMATE OF ABSOLUTE ERROR',&
      /,/,10X,'AE  =',G10.2)
      END


      REAL(8) FUNCTION FHNFML(X,M)
      REAL(8) X
      DIMENSION X(M)
!
      FHNFML = SQRT(1.0D0-X(1)*X(1)-X(2)*X(2)-X(3)*X(3))
      RETURN
```

```
      END


      SUBROUTINE FHNFM2(I,X,A,B,M)
      REAL(8) X,A,B
      DIMENSION X(M),A(M),B(M)
!
      IF(I.EQ.1) THEN
         A(1) = 0.0D0
         B(1) = SQRT(1.0D0-X(2)*X(2)-X(3)*X(3))
      ELSEIF(I.EQ.2) THEN
         A(2) = 0.0D0
         B(2) = SQRT(1.0D0-X(3)*X(3))
      ELSE
         A(3) = 0.0D0
         B(3) = 1.0D0
      ENDIF
      RETURN
      END
```

(d) Output results

```
*** DHNFML ***

 FUNCTION = SQRT(1.-X*X-Y*Y-Z*Z)

 ** INPUT **

   M      =  3

   ER     =  0.10E-07

   EA     =   0.0

   ITMX   =    15


 ** OUTPUT **

   IERR =    0


   INTEGRAL APPROXIMATION

     Q    =  0.3084251376D+00

   ESTIMATE OF ABSOLUTE ERROR

     AE   =  0.16E-08
```

Chapter 5

APPROXIMATION AND INTERPOLATION

5.1 INTRODUCTION

This chapter describes subroutines that fit a function to given data points. This library provides the following subroutines for obtaining the optimum coefficients of the function that approximates user-assigned data points according to a least squares approximation.

(1) Least squares approximation orthogonal polynomials

(2) Least squares approximation nonlinear functions

(3) Two-dimensional arbitrary data least squares approximation polynomials

(4) Two-dimensional lattice data least squares approximation polynomials

The subroutine that obtains the least squares approximation orthogonal polynomials determines the coefficients of an orthogonal polynomial that minimizes the sum of the squares of the differences of the function values y_i and the orthogonal polynomial values for x at $x = x_i$ when function values y_i $(i = 1, 2, \cdots, n)$ at n data points x_i $(i = 1, 2, \cdots, n)$ are given.
The subroutine that obtains the least squares approximation nonlinear functions determines the user-defined function that minimizes the sum of the squares of the differences of the function values y_i and the values of the user-defined function at $x = x_i$ when function values y_i $(i = 1, 2, \cdots, n)$ at n data points x_i $(i = 1, 2, \cdots, n)$ are given.
The subroutine that obtains the two-dimensional arbitrary data least squares approximation polynomials determines the coefficients of a polynomial for x and y that minimizes the sum of the squares of the differences of the function values z_i and the values of the polynomial for x and y at $(x, y) = (x_i, y_i)$ when n two-dimensional coordinate points (x_i, y_i) $(i = 1, 2, \cdots, n)$ and function values z_i at those points are given.
The subroutine that obtains the two-dimensional lattice data least squares approximation polynomials determines the coefficients of a polynomial for x and y that minimizes the sum of the squares of the differences of the function values z_{ij} and the values of the polynomial for x and y at $(x, y) = (x_i, y_i)$ when all function values z_{ij} at $nx \times ny$ two-dimensional lattice points (x_i, y_i) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ are given.
For interpolation, this library provides the following subroutines for obtaining interpolation values and interpolation polynomial coefficients for user-assigned data points.

(1) Unequally spaced discrete point interpolation value

(2) Unequally spaced discrete point interpolation value and interpolation coefficients

(3) Discrete point interpolation value on two-dimensional cross section lines

(4) Discrete point interpolation value on two-dimensional lattice

The subroutine that obtains the unequally spaced discrete point interpolation value obtains the Y coordinate value at the interpolation point by using Aitken's scheme to interpolate n given data points (x_i, y_i) $(i = 1, 2, \cdots, n)$ for a given interpolation point x.

The subroutine that obtains the unequally spaced discrete point interpolation value and interpolation coefficients obtains the Y coordinate value at the interpolation point and the interpolation polynomial coefficients by using Newton's method to interpolate n given data points (x_i, y_i) $(i = 1, 2, \cdots, n)$ for a given interpolation point x.
The subroutine that obtains the discrete point interpolation value on two-dimensional cross section lines establishes nx straight lines (called cross section lines here) $x = x_i$ $(i = 1, 2, \cdots, nx)$ parallel to the Y-axis in the XY-plane and obtains the interpolation value at an arbitrary point according to a cubic spline function by assigning ny_i data points on each of those cross sections and function values at those points. This subroutine also obtains spline coefficients used for interpolating data on the given cross section lines.
The subroutine that obtains the discrete point interpolation value on a two-dimensional lattice obtains the interpolation value of the function at a single point (x, y) when all function values z_{ij} on two-dimensional lattice points (x_i, y_j) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ are given.
For a Chebyshev approximation, the following subroutine is provided for obtaining the Chebyshev coefficients of the function that obtains the best fit approximation of a function given by the user.

(1) Chebyshev approximation

The Chebyshev approximation subroutine determines the coefficients of the Chebyshev polynomial that obtains the relative minimum of the difference between the Chebyshev polynomial value and function value y_i for $x = x_i$ $(i = 0, 2, \cdots, n)$ when a function $f(x)$ is given in the finite interval $[a, b]$. That is, if T_k is the Chebyshev polynomial for the Chebyshev coefficients c_k $(k = 0, 1, \cdots, m)$ that were obtained, the polynomial coefficients d_k $(k = 0, 1, \cdots, m)$ are obtained so that

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(c_k T_k(y))$$

Also, the degree of the optimal approximation polynomial is obtained by the this subroutine.

5.1.1 Notes

(1) The coefficient initial values should be taken as close as possible to the optimal coefficients.

(2) An appropriate value for the required precision is on the order of the square root of the unit for determining error.

(3) The appropriate required maximum error for the Chebyshev approximation is on the order of (Chebyshev coefficients truncation error) $\times 10^2$. (See Section 5.6.1)

5.1.2 Algorithms Used

5.1.2.1 Least squares approximation orthogonal polynomials

When the data points (x_i, y_i) and weight function $w(x_i)$ $(i = 1, 2, \cdots, n)$ are given, y_i will be approximated by the following polynomial of degree m :

$$f(x) = a_0 x^m + a_1 x^{m-1} + \cdots + a_{m-1} x + a_m \tag{5.1}$$

If (x_i, y_i) are not distributed in the neighborhood of the origin, then to minimize error, a coordinate conversion is performed for the calculations so that the midpoint of the distribution of x_i will be 0.0 and the minimum value of y_i will be 0.0.
Now, orthogonal polynomials $P_j(x)$ $(j = 0, 1, \cdots, m)$ are defined as polynomials that satisfy the following relationship:

$$\sum_{i=1}^{n} w(x_i) P_u(x_i) P_v(x_i) = \sum_{i=1}^{n} w(x_i) \{P_u(x_i)\}^2 \delta_{uv} \tag{5.2}$$

where δ_{uv} is the Kronecker delta defined as follows:

$$\delta_{uv} = \begin{cases} 1 & (u = v) \\ 0 & (u \neq v) \end{cases}$$

$f(x)$ is expressed as the following linear combination of the orthogonal polynomials $P_j(x)$ $(j = 0, 1, \cdots, m)$:

$$f(x) = \sum_{j=0}^{m} b_j P_j(x) \tag{5.3}$$

(1) Determination of b_j and $P_j(x_i)$ $(i = 1, 2, \cdots, n;\ j = 0, 1, \cdots, m)$
The coefficients b_j are determined according to the least squares method. That is b_j are determined so that the following function:

$$H(b_0, \cdots, b_m) \equiv \sum_{i=1}^{n} w(x_i) \{y_i - f(x_i)\}^2$$

is minimized. This function is minimized when:

$$\frac{\partial H}{\partial b_k} = 0 \tag{5.4}$$

By using the orthogonality condition of (5.2) in (5.4), the coefficients b_j satisfy the following:

$$b_j = \frac{\sum_{i=1}^{n} w(x_i) y_i P_j(x_i)}{\sum_{i=1}^{n} w(x_i) \{P_j(x_i)\}^2} \ (j = 0, 1, \cdots, m) \tag{5.5}$$

The orthogonal polynomials $P_j(x)$ can be constructed according to the following recurrence relations as j-degree polynomials of x :

$$\begin{aligned} P_{-1}(x) &= 0 \\ P_0(x) &= 1 \\ P_{j+1}(x) &= (x - \alpha_{j+1}) P_j(x) - \beta_j P_{j-1}(x) \ (j = 0, \cdots, m-1) \end{aligned} \tag{5.6}$$

where the coefficients α_{j+1} and β_j are expressed as follows:

$$\begin{aligned} \alpha_{j+1} &= \frac{\sum_{i=1}^{n} w(x_i) x_i \{P_j(x_i)\}^2}{\sum_{i=1}^{n} w(x_i) \{P_j(x_i)\}^2} \\ \beta_j &= \frac{\sum_{i=1}^{n} w(x_i) \{P_j(x_i)\}^2}{\sum_{i=1}^{n} w(x_i) \{P_{j-1}(x_i)\}^2} \end{aligned} \tag{5.7}$$

from the orthogonality condition given in (5.2).
The following values can be obtained sequentially from (5.6) and (5.7):

$$P_{-1}(x_i), P_0(x_i) \to \alpha_1, \beta_0 \to P_1(x_i) \to \alpha_2, \beta_1 \to \cdots$$

The coefficients b_j $(j = 0, 1, \cdots, m)$ are obtained by using these values and (5.5).

(2) Determining the coefficients a_k $(k = 0, 1, \cdots, m)$
If $P_j(x)$ is expressed as follows:

$$P_j(x) = \sum_{k=0}^{j} c_{j,k} x^k = c_{j,j} x^j + c_{j,j-1} x^{j-1} + \cdots + c_{j,1} x + c_{j,0}$$

then, $c_{j+1,k+1}$ can be written as:

$$c_{j+1,k+1} = c_{j,k} - \alpha_{j+1} c_{j,k+1} - \beta_j c_{j-1,k+1} \ (j = 0, 1, \cdots, m-1;\ k = 0, 1, \cdots, j)$$

(Where, $c_{0,0} = 1$, $c_{-1,0} = 0$).
The value of $c_{j,k}$ are obtained from the above. Also, since $f(x)$ can be written as:

$$f(x) = \sum_{j=0}^{m} b_j \sum_{k=0}^{j} c_{j,k} x^k = \sum_{k=0}^{m} \left(\sum_{j=k}^{m} b_j c_{j,k} \right) x^k$$

the coefficients a_k can be obtained as follows:

$$a_{m-k+1} = \sum_{j=k}^{m} b_j c_{j,k}$$

from the b_j and $c_{j,k}$ that had been obtained as described earlier.

5.1.2.2 Nonlinear least square method

Given n coordinate values (x_i, y_i) $(i = 1, \cdots, n)$ and an approximation function $f(x, \boldsymbol{a})$ that has m parameters $\boldsymbol{a} = \{a_i\}$ $(i = 1, \cdots, m)$, this algorithm obtains a values $\boldsymbol{a}$ for which the following sum of the squares:

$$S(\boldsymbol{a}) = \sum_{i=1}^{n} (y_i - f(x_i, \boldsymbol{a}))^2$$

is a minimum, where the vector function $\boldsymbol{h}(\boldsymbol{a}) = \{h_i(\boldsymbol{a})\}$ is defined as follows:

$$h_i(\boldsymbol{a}) = y_i - f(x_i, \boldsymbol{a}) \ (i = 1, \cdots, n)$$

$S(\boldsymbol{a})$ can be written as:

$$S(\boldsymbol{a}) = \|\boldsymbol{h}(\boldsymbol{a})\|_2^2$$

where $\|\boldsymbol{a}\| = \sqrt{\boldsymbol{a}^T\boldsymbol{a}}$ (the notation ^{T} indicates transpose).
To obtain a solution that minimizes $S(\boldsymbol{a})$, a search is begun from an initial value $\boldsymbol{a} = \boldsymbol{a}_0$, and Powell's hybrid method is used to sequentially correct the solutions $\boldsymbol{a} = \boldsymbol{a}_1, \boldsymbol{a}_2, \cdots$. The hybrid method determines the correction vector at each step as a linear combination of the correction vectors according the steepest descent method and the Gauss-Newton method obtained by linearizing a nonlinear function for the coefficient $\boldsymbol{a}$. An independence check always is performed for the correction vector so that they do not end up being confined within a subspace. Also, the value of the Jacobian matrix is determined from function information and the value from the previous step, without directly calculating the Jacobian matrix in each step.

(1) Correction vector $\Delta\boldsymbol{a}$ calculation
If the vector function $\boldsymbol{h}(\boldsymbol{a} + \Delta\boldsymbol{a})$ is approximated in a linear range from $\Delta\boldsymbol{a}$ as follows:

$$\boldsymbol{h}_L(\boldsymbol{a} + \Delta\boldsymbol{a}) = \boldsymbol{h}(\boldsymbol{a}) + A\Delta\boldsymbol{a}$$

then consider the problem of minimizing:

$$S_L(\boldsymbol{a} + \Delta\boldsymbol{a}) = \|\boldsymbol{h}(\boldsymbol{a}) + A\Delta\boldsymbol{a}\|_2^2$$

where A is the Jacobian matrix $\dfrac{\partial\boldsymbol{h}}{\partial\boldsymbol{a}}$ of $\boldsymbol{h}$.
The correction vector $\Delta\boldsymbol{a}_S$ at each step according to the steepest descent method is given by:

$$\Delta\boldsymbol{a}_S = \frac{\|\boldsymbol{b}\|_2^2}{\|A\boldsymbol{b}\|_2^2}\boldsymbol{b}$$

where $\boldsymbol{b} = -A^T\boldsymbol{h}(\boldsymbol{a})$.
The correction vector $\Delta\boldsymbol{a}_G$ at each step according to the Gauss-Newton method is obtained by solving the following normal equation:

$$A^TA\Delta\boldsymbol{a}_G = \boldsymbol{b}$$

This library uses the QR decomposition algorithm to solve this equation.
In general, the following relationship holds:

$$\|\Delta\boldsymbol{a}_S\|_2 \leq \|\Delta\boldsymbol{a}_G\|_2$$

The correction vector at each step is determined as follows by taking a linear combination of $\Delta\boldsymbol{a}_S$ and $\Delta\boldsymbol{a}_G$ depending on the step size d.

(a) If $d \leq \|\Delta\boldsymbol{a}_S\|_2$, then:

$$\Delta\boldsymbol{a} = d\frac{\Delta\boldsymbol{a}_S}{\|\Delta\boldsymbol{a}_S\|_2}$$

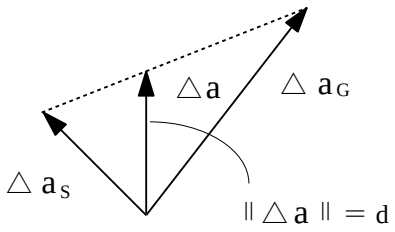


Figure 5−1

(b) If $\|\Delta \boldsymbol{a}_S\|_2 < d < \|\Delta \boldsymbol{a}_G\|_2$, (See Figure 5−1) then:

$$\Delta \boldsymbol{a} = \alpha \Delta \boldsymbol{a}_S + \beta \Delta \boldsymbol{a}_G \ (\alpha > 0, \beta > 0, \|\Delta \boldsymbol{a}\|_2 = d)$$

(c) If $\|\Delta \boldsymbol{a}_G\|_2 \leq d$

$$\Delta \boldsymbol{a} = \Delta \boldsymbol{a}_G$$

(2) Step size d determination

The initial value of the step is assumed to be $d = \|\Delta \boldsymbol{a}_S\|_2$. 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 of S in the linear approximation is represented by:

$$\Delta S_L = S_L(\boldsymbol{a} + \Delta \boldsymbol{a}) - S_L(\boldsymbol{a})$$

and the actual amount of change is represented by:

$$\Delta S = S(\boldsymbol{a} + \Delta \boldsymbol{a}) - S(\boldsymbol{a})$$

(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 λ, 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 $\delta \boldsymbol{h}$ defined as:

$$\delta \boldsymbol{h} = \boldsymbol{h}(\boldsymbol{a} + \Delta \boldsymbol{a}) - (\boldsymbol{h}(\boldsymbol{a}) + A\Delta \boldsymbol{a})$$

S_P and S_S are as follows:

$$\begin{aligned} S_P &= \sum_{i=1}^{n} |h_i(\boldsymbol{a} + \Delta \boldsymbol{a})\delta h_i| \\ S_S &= \|\delta \boldsymbol{h}\|_2^2 \end{aligned}$$

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 μ is calculated as follows:

$$\begin{aligned} \mu &= \min(2, \lambda, \tau) \\ \tau &= \frac{\lambda}{\mu} \end{aligned}$$

where the initial value for τ 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. According to the hybrid method, 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.
m mutually independent vectors from the vectors used to correct the Jacobian matrix during the past $2m$ iterations are made to be orthogonal and are stored in $\Omega = (\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m)$. The array $\boldsymbol{j}$ of size n is used to store information indicating the number of iterations earlier in which $\boldsymbol{\omega}_i$ was the correction vector. That is, this information indicates that $\boldsymbol{\omega}_i$ is the vector that was taken j_i iterations earlier. Ω is initialized as the unit matrix, and $\boldsymbol{j}$ is initialized with values $j_i = m - i + 1$ $(i = 1, \cdots, m)$.
When a solution is corrected, the following steps are performed.

(a) When $\Delta\boldsymbol{a} = \Delta\boldsymbol{a}_G$
Regardless of its independence, $\Delta\boldsymbol{a}$ is taken as the correction vector.

(b) When $\Delta\boldsymbol{a} = \Delta\boldsymbol{a}_G$ does not occur
If $j_1 < 2m$ or if $\Delta\boldsymbol{a}$ is independent of $(\boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots, \boldsymbol{\omega}_m)$, then 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:

$$A' = A + \delta\boldsymbol{h}\frac{\Delta\boldsymbol{a}^T}{\|\Delta\boldsymbol{a}\|_2^2}$$

However, if $\|\Delta\boldsymbol{a}\|_2 < d_{min}$ or if $j_1 = 2m$ and $\Delta\boldsymbol{a}$ is not independent of $(\boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots, \boldsymbol{\omega}_m)$, then $\Delta\boldsymbol{a} = d_{min}\boldsymbol{\omega}_1$ is set.

(5) Revision of Ω and $\boldsymbol{j}$
Ω and $\boldsymbol{j}$ are revised as follows.
When $\Delta\boldsymbol{a} = d_{min}\boldsymbol{\omega}_1$ has been set, the following values should be set:

$$\begin{aligned}
\boldsymbol{\omega}_i &= \boldsymbol{\omega}_{i+1} \qquad (i = 1, \cdots, m-1) \\
\boldsymbol{\omega}_m &= \boldsymbol{\omega}_1 \\
j_i &= j_{i+1} + 1 \qquad (i = 1, \cdots, m-1) \\
j_m &= 1
\end{aligned}$$

Otherwise, the following is performed.
The minimum value k is obtained for which $(\boldsymbol{\omega}_{k+1}, \boldsymbol{\omega}_{k+2}, \cdots, \boldsymbol{\omega}_m, \Delta\boldsymbol{a})$ are mutually independent.
$(\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_{k-1}, \boldsymbol{\omega}_{k+1}, \boldsymbol{\omega}_{k+2}, \cdots, \boldsymbol{\omega}_m, \Delta\boldsymbol{a})$ are made to be orthogonal, and these are again assumed to be

$(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m)$. The following values are then set:

$$\begin{aligned} j_i &= j_i + 1 \qquad (i = 1, \cdots, k-1) \\ j_i &= j_{i+1} + 1 \qquad (i = k, \cdots, m-1) \\ j_m &= 1 \end{aligned}$$

In this way, the relationship $j_1 \leq 2m$ always holds.

(6) Convergence decision
Convergence is assumed to have occurred when the following relationship holds and $\boldsymbol{a} + \Delta\boldsymbol{a}$ is assumed to be the solution:

$$\|\Delta\boldsymbol{a}\|_\infty \leq e_r \max(1, \|\boldsymbol{a} + \Delta\boldsymbol{a}\|_\infty)$$

where e_r is the required relative precision, and:

$$\|\boldsymbol{a}\|_\infty = \max_i |a_i|$$

5.1.2.3 Two-dimensional arbitrary data least squares approximation polynomials

Obtain the following least squares approximation polynomial for given discrete points (x_k, y_k, z_k) $(k = 1, 2, \cdots, n)$ in a space:

$$f(x, y) = \sum_{j=1}^{m+1} \sum_{i=1}^{m+2-j} a_{i,j} x^{i-1} y^{j-1}$$

where, $a_{i,j}$ are the coefficients of the polynomial, x and y are independent variables, and m is the maximum degree of the polynomial for x and y.
Let the residual sum of squares of z_k and $f(x_k, y_k)$ be represented by χ^2, where χ^2 is given by:

$$\chi^2 = \sum_{k=1}^{n} \left(z_k - \sum_{j=1}^{m+1} \sum_{i=1}^{m+2-j} a_{i,j} x_k^{i-1} y_k^{j-1} \right)^2$$

The coefficients $a_{i,j}$ are determined by the following condition for minimizing χ^2 :

$$\frac{\partial \chi^2}{\partial a_{i,j}} = 0$$

Obtain the following system of normal equations from this:

$$\sum_{jc=1}^{m+1} \sum_{ic=1}^{m+2-jc} \sum_{k=1}^{n} a_{ic,jc} x_k^{ic+ir-2} y_k^{jc+jr-2} = \sum_{k=1}^{n} x_k^{ir-1} y_k^{jr-1} z_k$$
$$(jr = 1, \cdots, m+1;\ ir = 1, \cdots, m+2-jr)$$

Now, define G_k, $\boldsymbol{a}$ and $\boldsymbol{b}_k$ as:

$$\begin{aligned} G_k &= (g_{k,(jr,ir),(jc,ic)}) \\ &= (x_k^{ic+ir-2} y_k^{jc+jr-2}) \qquad \begin{pmatrix} jr = 1, \cdots, m+1; & jc = 1, \cdots, m+1 \\ ir = 1, \cdots, m+2-jr; & ic = 1, \cdots, m+2-jc \end{pmatrix} \\ \boldsymbol{a} &= (a_{(jc,ic)}) \qquad \begin{pmatrix} jc = 1, \cdots, m+1 \\ ic = 1, \cdots, m+2-jc \end{pmatrix} \\ \boldsymbol{b}_k &= (b_{k,(jr,ir)}) = (x_k^{ir-1} y_k^{jr-1} z_k) \qquad \begin{pmatrix} jr = 1, \cdots, m+1 \\ ir = 1, \cdots, m+2-jr \end{pmatrix} \end{aligned}$$

and G and $\boldsymbol{b}$ as:

$$G = \sum_{k=1}^{n} G_k$$

$$\boldsymbol{b} = \sum_{k=1}^{n} \boldsymbol{b}_k$$

Then, the system of normal equations is expressed as:

$$G\boldsymbol{a} = \boldsymbol{b}$$

Set $\tilde{G}\tilde{\boldsymbol{a}} = \tilde{\boldsymbol{b}}$ by transforming $\tilde{G}$ to an $\{\frac{(m+1)(m+2)}{2}\}$-order square matrix and $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ to $\{\frac{(m+1)(m+2)}{2}\}$-order column vectors, and obtain the desired polynomial by solving this for $\tilde{\boldsymbol{a}}$.
The polynomial coefficients $a_{i,j}$ are stored in a one-dimensional matrix as shown in Figure 5−2 for computational convenience. The subscript correspondence relationship is expressed as:

$$a(i,j) = \tilde{a}(i + ((j-1) \times (2 \times m - j + 4))/2)$$

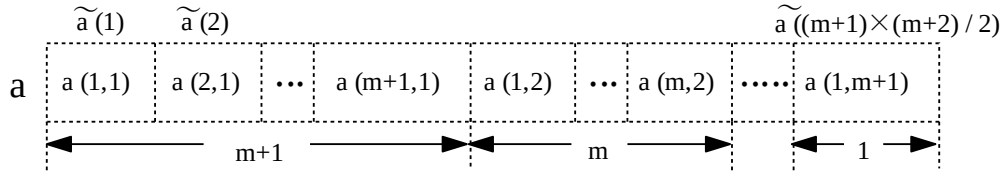


Figure 5−2

5.1.2.4 Two-dimensional lattice data least squares approximation polynomials

If all Z coordinate values $Z_{i,j}$ on two-dimensional lattice points (x_i, y_j) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ are given, obtain the following least squares approximation polynomial that approximates this function:

$$f(x,y) = \sum_{i=1}^{ix+1} \sum_{j=1}^{iy+1} a_{i,j} x^{i-1} y^{j-1} \tag{5.8}$$

Assume that $f(x,y)$ for determining the coefficients $a_{i,j}$ is expressed as follows as a linear combination of the products of the orthogonal polynomials $\Phi_{r-1}(x)$ and $\Psi_{s-1}(y)$, which are defined below:

$$f(x,y) = \sum_{r=1}^{ix+1} \sum_{s=1}^{iy+1} \Gamma_{rs} \Phi_{r-1}(x) \Psi_{s-1}(y) \tag{5.9}$$

Since $\Phi_{r-1}(x)$ and $\Psi_{s-1}(y)$ are $(r-1)$-degree and $(s-1)$-degree polynomials for x and y, respectively, if we let $C_{r,k}$ and $B_{s,l}$ be coefficients of the orthogonal polynomials for x and y, then $\Phi_{r-1}(x)$ and $\Psi_{s-1}(y)$ can be written as the following expressions:

$$\Phi_{r-1}(x) = \sum_{k=1}^{r} C_{r,k} x^{k-1} \tag{5.10}$$

$$\Psi_{s-1}(y) = \sum_{l=1}^{s} B_{s,l} y^{l-1} \tag{5.11}$$

Substitute (5.10) and (5.11) in (5.9) as follows:

$$
\begin{aligned}
f(x,y) &= \sum_{r=1}^{ix+1}\sum_{s=1}^{iy+1}\Gamma_{rs}\Phi_{r-1}(x)\Psi_{s-1}(y) \qquad (5.12)\\
&= \sum_{r=1}^{ix+1}\sum_{s=1}^{iy+1}\Gamma_{rs}\sum_{k=1}^{r}C_{r,k}x^{k-1}\sum_{l=1}^{s}B_{s,l}y^{l-1}\\
&= \sum_{r=1}^{ix+1}\sum_{s=1}^{iy+1}\Gamma_{rs}\sum_{k=r}^{ix+1}C_{r,k}x^{r-1}\sum_{l=s}^{iy+1}B_{s,l}y^{l-1} \quad \text{(See Figure 5−3)}\\
f(x,y) &= \sum_{i=1}^{ix+1}\sum_{j=1}^{iy+1}\left(\Gamma_{ij}\sum_{k=1}^{ix+1}\sum_{l=j}^{iy+1}C_{i,k}B_{j,l}\right)x^{i-1}y^{j-1}
\end{aligned}
$$

By comparing this with (5.8), we obtain the coefficients $a_{i,j}$ as follows:

$$a_{i,j} = \Gamma_{ij}\sum_{k=1}^{ix+1}\sum_{l=j}^{iy+1}C_{i,k}B_{j,l}$$

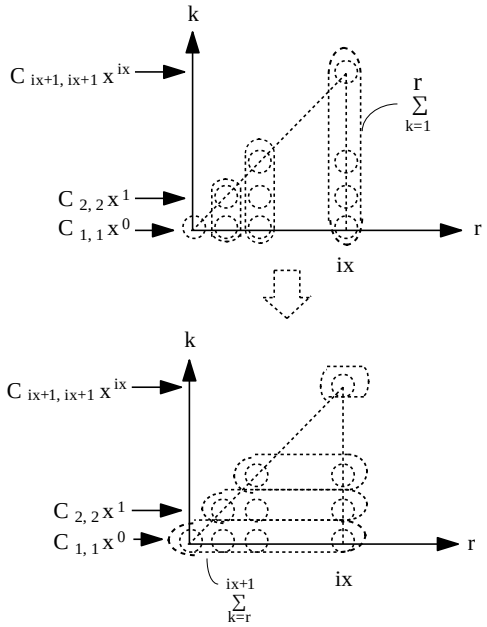


Figure 5−3

(1) Orthogonal polynomials
Orthogonal polynomials for x and y are considered here. Define polynomials that satisfy the following conditions:

$$\sum_{i=1}^{nx}\Phi_r(x_i)\Phi_s(x_i) = D_r\delta_{rs} \qquad (5.13)$$

$$\sum_{j=1}^{ny}\Psi_r(y_j)\Psi_s(y_j) = d_r\delta_{rs} \qquad (5.14)$$

at data points x_i $(i = 1, 2, \cdots, nx)$ and y_j $(j = 1, 2, \cdots, ny)$.
Where δ_{rs} is the Kronecker delta defined as follows:

$$\delta_{rs} = \begin{cases} 1 & (r = s) \\ 0 & (r \neq s) \end{cases}$$

The orthogonal polynomials $\Phi_r(x)$ and $\Psi_s(y)$ for x and y that satisfy the orthogonality conditions shown above can be obtained from the following recurrence relations.
Orthogonal polynomial of x

$$\Phi_r(x) = (x - \alpha_{r-1})\Phi_{r-1}(x) - \beta_{r-1}\Phi_{r-2}(x) \ (r = 2, 3, \cdots) \tag{5.15}$$

where,

$$\begin{aligned} \alpha_{r-1} &= \sum_{i=1}^{nx} x_i \frac{\Phi_{r-1}^2(x_i)}{D_{r-1}} \\ \beta_{r-1} &= \frac{D_{r-1}}{D_{r-2}} \\ \Phi_0(x) &= 1 \\ \Phi_1(x) &= x - \bar{x} \\ \bar{x} &= \sum_{i=1}^{nx} \frac{x_i}{nx} \end{aligned}$$

Orthogonal polynomial of y

$$\Psi_s(y) = (y - \alpha'_{s-1})\Psi_{s-1}(y) - \beta'_{s-1}\Psi_{s-2}(y) \ (s = 2, 3, \cdots) \tag{5.16}$$

where,

$$\begin{aligned} \alpha'_{s-1} &= \sum_{j=1}^{ny} y_j \frac{\Psi_{s-1}^2(y_j)}{d_{s-1}} \\ \beta'_{s-1} &= \frac{d_{s-1}}{d_{s-2}} \\ \Psi_0(y) &= 1 \\ \Psi_1(y) &= y - \bar{y} \\ \bar{y} &= \sum_{j=1}^{ny} \frac{y_j}{ny} \end{aligned}$$

(2) Calculation of Γ_{rs}
First, fix $y = y_j$ and obtain the least squares curved line for x.
Write the least squares curved line as follows using coefficients $\lambda_{r,j}$:

$$q_j(x) \equiv f(x, y_j) = \sum_{r=1}^{ix+1} \lambda_{r,j}\Phi_{r-1}(x) \ (j = 1, 2, \cdots, ny) \tag{5.17}$$

Minimize the residual sum of squares at lattice point $x = x_i$, which is given by the following expression:

$$Q = \sum_{i=1}^{nx} \{q_j(x_i) - z_{i,j}\}^2 \tag{5.18}$$

That is, let $\frac{\partial Q}{\partial \lambda_{r,j}} = 0$ and obtain the following expression by using the orthogonality conditions given in (5.13):

$$\lambda_{r,j} = \sum_{i=1}^{nx} z_{i,j} \frac{\Phi_{r-1}(x_i)}{D_{r-1}} \ (r = 1, 2, \cdots, ix+1) \tag{5.19}$$

By comparing (5.9) with (5.17), we obtain the coefficients as follows:

$$\lambda_{r,j} = \sum_{s=1}^{iy+1} \Gamma_{r,s} \Psi_{s-1}(y_j) \tag{5.20}$$

Multiply both sides of (5.20) by $\Psi_{s-1}(y_j)$ and take the sum from $j = 1$ to ny. Use the orthogonality conditions given in (5.14) to obtain:

$$\Gamma_{rs} = \sum_{j=1}^{ny} \lambda_{r,j} \frac{\Psi_{s-1}(y_j)}{d_{s-1}} \ (s = 1, 2, \cdots, iy+1) \tag{5.21}$$

Γ_{rs} is determined by using (5.21).

(3) Calculation of $C_{r,k}$ and $B_{s,l}$
Obtain the following recurrence relation for $C_{r,k}$ by substituting (5.10) in (5.15) and setting the coefficients of the powers of x to zero so that this holds for an arbitrary x :

$$C_{r,k} = C_{r-1,k-1} - \alpha_{r-1} C_{r-1,k} - \beta_{r-1} C_{r-2,k}$$

(where $C_{r,k} = 0$ when $r < k$ and $C_{0,0} = 1$).
Similarly, the recurrence relation for $B_{s,l}$ is given as follows:

$$B_{s,l} = B_{s-1,l-1} - \alpha'_{s-1} B_{s-1,l} - \beta'_{s-1} B_{s-2,l}$$

(where $B_{s,l} = 0$ when $s < l$ and $B_{0,0} = 1$).
These expressions can be used to calculate $C_{r,k}$ and $B_{s,l}$.

5.1.2.5 Unequally spaced discrete point interpolation value

If data points (x_i, y_i) $(i = 1, \cdots, n)$ and interpolation point $x = u$ are given, obtain (u_i, v_i) by rearranging the x_i in ascending order of distance from u and simultaneously rearranging the y_i corresponding to them.
To obtain an interpolation polynomial by interpolating two points (u_1, v_1) and (u_2, v_2), use the following basic linear interpolation formula:

$$y(x) = \frac{v_1(x - u_2) - v_2(x - u_1)}{u_1 - u_2} \tag{5.22}$$

to obtain the interpolation polynomial of the first degree.
Also, to obtain an interpolation polynomial by interpolating three points $(u_1, v_1), (u_2, v_2)$, and (u_3, v_3), first perform a linear interpolation between (u_1, v_1) and (u_2, v_2) by using (5.22) to obtain the following first degree interpolation polynomial:

$$y_1^1(x) = \frac{v_1(x - u_2) - v_2(x - u_1)}{u_1 - u_2}$$

Next, perform a linear interpolation between (u_2, v_2) and (u_3, v_3) by using (5.22) to obtain the following first degree interpolation polynomial:

$$y_1^2(x) = \frac{v_2(x-u_3) - v_3(x-u_2)}{u_2 - u_3}$$

By using (5.22) to perform a linear interpolation between the $y_1^1(x)$ and $y_1^2(x)$ that were obtained, the following second degree interpolation polynomial is obtained:

$$y_2^1(x) = \frac{y_1^1(x)(x-u_3) - y_1^2(x)(x-u_1)}{u_1 - u_3}$$

The third degree interpolation polynomial, fourth degree interpolation polynomial, and so on can be created by repeatedly using (5.22) in a similar manner.
In general, the j-th degree interpolation polynomial (where, $j = 2, \cdots, n$) is expressed as follows:

$$y_1^k(x) = \frac{v_k(x-u_{k+1}) - v_{k+1}(x-u_k)}{u_k - u_{k+1}} \ (k = 1, \cdots, j) \tag{5.23}$$

$$y_m^k(x) = \frac{y_{m-1}^k(x)(x-u_{k+m}) - y_{m-1}^{k+1}(x)(x-u_k)}{u_k - u_{k+m}} \tag{5.24}$$
$$(m = 2, \cdots, j;\ k = 1, \cdots, j+1-m)$$

Therefore, the value of the n-th degree interpolation polynomial for $x = u$ (interpolation value) is obtained by calculating (5.23) and (5.24) for $x = u$.
Next, we will explain how to determine the degree of the interpolation polynomial.
First, let Z_j $(= y_j^1(u))$ be the interpolation value that was interpolated using j points in the vicinity of $x = u$.
Define the interpolation difference D_j as follows:

$$D_j \equiv |Z_{j-1} - Z_j| \ (j = 2, \cdots, n)$$

and calculate $D_2, \cdots, D_n$. If the following relationship always holds for the user-assigned required absolute precision ε,

$$D_j > \varepsilon \ (j = 2, \cdots, n)$$

let the value l for which the following holds be the degree of the interpolation polynomial:

$$D_l = \min_j(D_j)$$

and let Z_l be the interpolation value at that time. Also, D_l is output as the absolute error of the interpolation value.
If the following relationship holds for some j $(j \leq n)$,

$$D_j \leq \varepsilon$$

let the minimum j for which this relationship holds be the degree of the interpolation polynomial, and let Z_j be the interpolation value at that time. Also, D_j is output as the absolute error of the interpolation value.

5.1.2.6 Unequally spaced discrete point interpolation value and interpolation coefficients

If data points (x_i, y_i) $(i = 1, \cdots, n)$ and interpolation point $x = u$ are given, obtain (u_i, v_i) by rearranging the x_i in ascending order of distance from u and simultaneously rearranging the y_i corresponding to them.
Let the m-th degree polynomial be expressed as follows:

$$f(x) = c_1 + c_2(x-u_1) + \cdots + c_{m+1}(x-u_1)\cdots(x-u_m) \ (1 \leq m \leq n-1) \tag{5.25}$$

and obtain the coefficients $c_1, \cdots, c_{m+1}$.
Now, if interpolation points $x = u_1, \cdots, u_n$ and function $f(x)$ are given, define divided difference of $f(x)$ inductively as follows:

$$\begin{aligned} f[i] &= f(u_i)\ (i = 1, \cdots, n) \\ f[i_1, \cdots, i_{k+1}] &= \frac{f[i_2, \cdots, i_{k+1}] - f[i_1, \cdots, i_k]}{u_{i_{k+1}} - u_{i_1}}\ (k = 1, \cdots, n) \end{aligned}$$

($i_1, \cdots, i_{k+1}$ are positive integers, they are different from each other, and they are equal to or smaller than n.)
If we set $x = u_1$ in (5.25), we get:

$$c_1 = f(u_1) = f[1]$$

If we substitute this in (5.25) and rearrange the terms, we get:

$$\frac{f(x) - f[1]}{x - u_1} = c_2 + c_3(x - u_2) + \cdots + c_{m+1}(x - u_2) \cdots (x - u_m)$$

If we set $x = u_2$, we get:

$$c_2 = \frac{f(u_2) - f[1]}{u_2 - u_1} = \frac{f[2] - f[1]}{u_2 - u_1} = f[1, 2]$$

If we repeat the above operations in a similar manner, we can represent the coefficients c_i as:

$$c_i = f[1, \cdots, i]\ (i = 1, \cdots, m + 1)$$

Therefore, we can obtain c_i $(i = 1, 2, \cdots, m + 1)$ by sequentially calculating the divided difference and we can calculate the interpolation value at the given interpolation point from these and expression (5.25).

5.1.2.7 Discrete point interpolation value on two-dimensional cross section lines

Consider nx straight lines $x = x_i$ $(i = 1, 2, \cdots, nx)$ perpendicular to the X-axis in the XY-plane. If ny_i $(i = 1, \cdots, nx)$ points $(x_i, y_{j,i})$ $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny_i)$ on each straight line and Z coordinate values $z_{i,j}$ $(i = 1, \cdots, nx;\ j = 1, \cdots, ny_i)$ at each point are given, obtain the interpolation value at the point (x_l, y_l) $(x_1 \leq x_l \leq x_{nx};\ \min(y_{j,i}) \leq y_l \leq \max(y_{j,i}))$ as follows. (See Figure 5−4)

(Step 1)

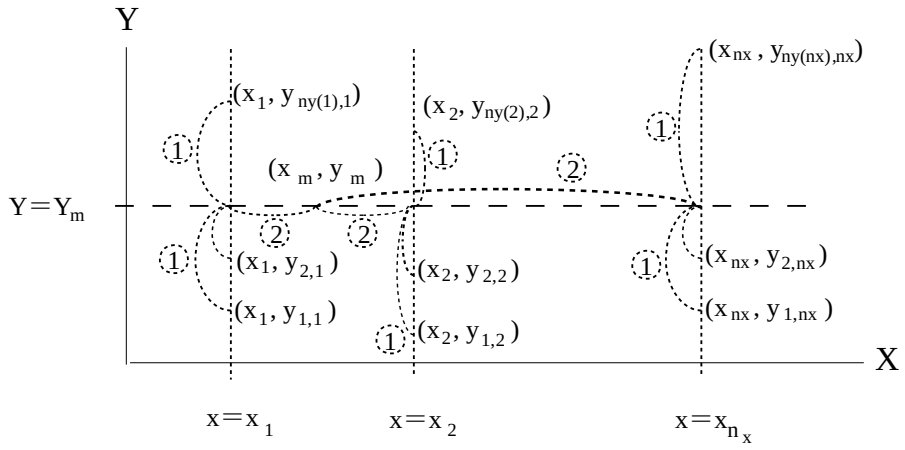
On each cross section, obtain the cubic spline by treating the Y coordinate value of the data point as the abscissa and the function value as the ordinate. (See Section 6.1.2) Using the spline coefficients that were just obtained, obtain interpolation values at the intersections of each cross section line and the straight line $y = y_l$. (See Section 6.1.2) The value may be an extrapolation value depending on the value of y_l.

(Step 2)

Next, in exactly the same way, obtain the spline coefficients by treating the X coordinate value on the cross section line as the abscissa and the interpolation value on the cross section line at the y_l that was just obtained as the ordinate, and obtain the interpolation value at $x = x_l$. The interpolation value at the point (x_l, y_l) can be determined as described above.

5.1.2.8 Discrete point interpolation value on two-dimensional lattice

If all Z coordinate values $z_{i,j}$ on two-dimensional lattice points (x_i, y_j) $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ are given, use Aitken's scheme to obtain the interpolation value for an arbitrary point (x_l, y_l) within the lattice. (See Section



①: Calculation in step 1
②: Calculation in step 2

Figure 5−4

5.1.2)
First, obtain the interpolation value at x_l by treating the X coordinate value on each line parallel to the X-axis as the abscissa and the function value as the ordinate. Next, obtain the interpolation value at y_l by treating the Y coordinate value as the abscissa and the interpolation value at x_l that was just obtained as the ordinate.

5.1.2.9 Chebyshev approximation

The norm of the continuous function of $f(x)$ on the closed interval $[a, b]$ is defined by

$$\|f\| = \max_{x\in[a,b]} |f(x)|$$

This is called the maximum value norm or uniform norm.
If we look at polynomials $P_n(x) = a_0 + a_1x + \cdots + a_nx^n$ of degree n (or a rational expression in which the numerator and denominator are polynomials of degree m and n, respectively) as approximations of the function $f(x)$, the polynomial (rational expression) that minimizes the degree of approximation $\|f - P_n\|$ is called the Chebyshev approximation or simply the best (fit) approximation.
Although the best (fit) approximation is good for representing an approximation as a formula, when the procedure for obtaining the approximation is complicated or new calculations are required to add terms, skill and a great deal of effort are required to create the approximation. Although various techniques can be used in the best (fit) approximation, the technique that uses the Chebyshev expansion described below can be calculated relatively easily.
The polynomial conditions for the best (fit) approximation are that the relative maximum values of the error functions are equal and their signs are represented by alternating plus and minus signs. The Chebyshev polynomial satisfies these conditions. The method of sequentially obtaining the Chebyshev polynomial is described below.

(1) Obtaining the Chebyshev coefficients
The Chebyshev polynomial of degree n, which is written as $T_n(x)$, is given by the following explicit function.

$$T_n(x) = \cos(n \arccos x) \ (n \geq 0) \tag{5.26}$$

Although this appears at first glance to be a trigonometric function, using the trigonometric identity function in (5.26) leads to the following expressions for $T_n(x)$.

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ &\vdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \ (n \geq 1) \end{aligned}$$

The Chebyshev polynomials are orthogonal with weight $(1-x^2)^{-\frac{1}{2}}$ on the interval $[-1, 1]$. In particular, the following relationships hold.

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}}dx = \begin{cases} 0 & (i \neq j) \\ \frac{\pi}{2} & (i = j \neq 0) \\ \pi & (i = j = 0) \end{cases} \tag{5.27}$$

The polynomial $T_n(x)$ has n zero points on the interval $[-1, 1]$. The positions of those zero points are

$$x = \cos\left[\frac{\pi(k-\frac{1}{2})}{n}\right] \ (k = 1, 2, \cdots, n) \tag{5.28}$$

In the same interval, $n+1$ extrema, which are at the following points,

$$x = \cos\left[\frac{\pi k}{n}\right] \ (k = 0, 1, \cdots, n)$$

appear alternately as relative maximum and relative minimum values. All relative maximum values are $T_n(x) = 1$, and all relative minimum values are $T_n(x) = -1$. This property of the Chebyshev polynomials helps reduce the error in the polynomial approximation of the function.
The Chebyshev polynomials satisfy discrete orthogonality relationships at the same time as the continuous orthogonality relationships of (5.27). That is, if x_k $(k = 1, \cdots, m)$ are m zero points of $T_m(x)$ given by (5.28), then the following holds if $i, j < m$

$$\sum_{k=1}^{m}(T_i(x_k)T_j(x_k)) = \begin{cases} 0 & (i \neq j) \\ \frac{m}{2} & (i = j \neq 0) \\ m & (i = j = 0) \end{cases} \tag{5.29}$$

The following property is obtained from the relationships in (5.26), (5.28) and (5.29).
When $f(x)$ is an arbitrary function defined on the interval $[-1, 1]$, if the $N+1$ coefficients c_j $(j = 0, 1, \cdots, N)$ are defined as follows for a sufficiently large N

$$\begin{aligned} c_j &= \frac{2}{N}\sum_{k=1}^{N}(f(x_k)T_j(x_k)) \\ &= \frac{2}{N}\sum_{k=1}^{N}(f\left(\cos\left[\frac{\pi(k-\frac{1}{2})}{N}\right]\right)\cos\left[\frac{\pi j(k-\frac{1}{2})}{N}\right]) \end{aligned} \tag{5.30}$$

then the following holds

$$f(x) \quad \sim \quad \left[\sum_{j=0}^{N}(c_j T_j(x))\right] - \frac{1}{2}c_0 \tag{5.31}$$

In particular, at all N given zero points of $T_N(x)$ (that is when $x = x_k$), the left side of (5.31) equals the right side.
For a fixed N, (5.31) is a polynomial of x, and that polynomial approximates the function $f(x)$ on the interval $[-1, 1]$ (which includes all zero points of $T_N(x)$).
By ignoring c_k $(k = m + 1, \cdots, N)$, (5.31) can be reduced to the most accurate polynomial among the polynomials of the same degree, which is represented by the following expression.

$$f(x) \quad \sim \quad \left[\sum_{k=0}^{m}(c_k T_k(x))\right] - \frac{1}{2}c_0 \tag{5.32}$$

This is because the difference between (5.32) and (5.31) does not become greater than the sum of the ignored c_k $(k = m + 1, \cdots, N)$ since $T_k(x)$ all take a value between ± 1. In particular, if we consider the case when the c_k quickly decrease, most of the error is due to the oscillatory function $c_{m+1}T_{m+1}(x)$ having $m + 2$ equal extrema that are almost uniformly distributed on the interval $[-1, 1]$. This property, which is called extending the error uniformly, is important in approximating the function.
To generalize the expression in (5.30), perform the variable transformation

$$y \quad \equiv \quad \frac{x - \frac{1}{2}(b + a)}{\frac{1}{2}(b - a)} \tag{5.33}$$

to let the domain of the function $f(x)$ being approximate be $[a, b]$. This transformation enables a function $f(x)$ on an arbitrary interval to be approximated by a Chebyshev polynomial of y $(-1 \leq y \leq 1)$.

(2) Polynomial approximation according to Chebyshev coefficients
Next, by transforming the Chebyshev coefficients c_k that were obtained to polynomial coefficients of the original variable x, we obtain the following approximation polynomial

$$f(x) \sim \sum_{k=0}^{m}(g_k x^k) \qquad a \leq x \leq b \tag{5.34}$$

However, although the coefficients g_k of expression (5.34) reflect the original Chebyshev approximation, calculating this expression requires a higher calculation precision than calculating the Chebyshev summation (5.32). This is because of the following reason.
For example, if we consider a Chebyshev polynomial of degree 30, although T_{30} is given as

$$T_{30}(x) \quad = \quad 2^{29}x^{30} + \cdots \tag{5.35}$$

it is actually expressed as $T_{30}(x) = \cos(30 \arccos(x))$. Since $-1 \leq T_{30}(x) \leq 1$, even though calculations involving large coefficients occurred, the value of T_{30} remained between ± 1. At this time, canceling occurred in the calculations involving large numbers, causing precision to drop. Therefore, the Chebyshev approximation should be represented as a polynomial only when the degree m of the Chebyshev approximation polynomial does not exceed 7 or 8. Note that even in this case, the precision decreases by two places in the

calculation in (5.35).
By performing the following procedure sequentially, the coefficients g_k can be derived from the transformed Chebyshev coefficients c_k $(k = 0, 1, \cdots, m)$ for a suitable value of m.
When the coefficients c_k $(k = 0, 1, \cdots, m)$ are given, obtain the coefficients d_k $(k = 0, 1, \cdots, m)$ so that the following relationships holds

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(c_k T_k(y))$$

Here, the following Clenshaw recurrence is used.

$$\begin{aligned} d_{m+2} &\equiv d_{m+1} \equiv 0 \\ d_j &= 2xd_{j+1} - d_{j+2} + c_j \ (j = m, m-1, \cdots, 1) \\ f(x) &\equiv d_0 = xd_1 - d_2 + \frac{1}{2}c_0 \end{aligned}$$

Next, when the coefficients d_k $(k = 0, 1, \cdots, m)$ are given, obtain the coefficients g_k $(k = 0, 1, \cdots, m)$ so that the following relationships holds

$$\sum_{k=0}^{m}(d_k y^k) = \sum_{k=0}^{m}(g_k x^k)$$

(Here, x and y are related according to (5.33). That is, the interval $-1 \leq y \leq 1$ is mapped to the interval $a \leq x \leq b$.)
First, perform the following transformation.

$$g_k = d_k \left(\frac{b-a}{2}\right)^k$$

Next, obtain g_k by synthetic division.
Here, let's try to obtain the value of $g_k(z)$ at $x = z$.
First, assume the polynomial to be obtained is

$$g_k(x) = \sum_{k=0}^{m}(g_k x^k) \tag{5.36}$$

If we assume that this has been divided by $(x - z)$, it can be written as

$$g_k(x) = (x - z)\sum_{k=0}^{m-1}(d_k x_{k-1}) \tag{5.37}$$

If we compare the coefficients of x_k in the two expressions above, we obtain

$$\begin{aligned} g_0 &= d_0 \\ g_k &= zg_{k-1} + d_k \ (k = 1, 2, \cdots, m) \end{aligned}$$

On the other hand, since the following clearly holds

$$g_m(z) = d_m$$

if we let $z = \dfrac{b+a}{2}$, then from (5.33), starting from $d_0 = g_0$, the values of g_k are obtained by the recurrence of (5.36) and (5.37).

5.1.3 Reference Bibliography

(1) 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).

(2) Ralston, A. and Rabinowitz, P. , "A First Course in Numerical Analysis", McGraw-Hill, Inc. , (1978).

(3) Balch, Stephen, J. and Thompson, Garth, T. , "An Efficient Algorithm For Polynomial Surface Fitting", Computers & Geosciences Vol.15, No.1, pp.107-119, (1989).

(4) William H. Press, "NUMERICAL RECIPES IN FORTRAN", CAMBRIDGE UNIVERSITY PRESS.

5.2 INTERPOLATION

5.2.1 DPDOPL, RPDOPL
Unequally Spaced Discrete Point Interpolation Value

(1) **Function**

DPDOPL or RPDOPL obtains the interpolation value f_l at the interpolation point x_l by using Aitken's scheme to interpolate n given points $(x_i, y_i)(i = 1, \cdots, n)$.

(2) **Usage**

Double precision:

CALL DPDOPL (X, Y, N, XL, EPS, FL, DL, WK, IERR)

Single precision:

CALL RPDOPL (X, Y, N, XL, EPS, FL, DL, 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	X	{D / R}	N	Input	X coordinates x_i of data points
2	Y	{D / R}	N	Input	Y coordinates y_i of data points
3	N	I	1	Input	Number of data points n
4	XL	{D / R}	1	Input	Interpolation point x_l
5	EPS	{D / R}	1	Input	Required absolute precision (Default value: Unit for determining error×64) (See Section 5.1.2.5)
6	FL	{D / R}	1	Output	Interpolation value f_l at x_l
7	DL	{D / R}	1	Output	Absolute error of interpolation value (See Section 5.1.2.5)
8	WK	{D / R}	N × 5	Work	Work area
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS ≥ Unit for determining error ×64

(b) N ≥ 2

(c) X(i) ≠ X(j) (i ≠ j)

(d) $\min_{i=1,\cdots,N}(X(i)) \leq XL \leq \max_{i=1,\cdots,N}(X(i))$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) was not satisfied.	Processing is performed with the default value set.
2500	The precision could not be guaranteed since the absolute error of the interpolation value was not less than or equal to the required precision.	Processing is terminated without obtaining a solution having the required precision.
3000	Restriction (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3020	Restriction (d) was not satisfied.	
4000	Overflow occurred during the interpolation value calculation.	

(6) **Notes**

(a) To obtain multiple interpolation values in the vicinity of x_l, use the subroutine 5.2.2 $\left\{\begin{matrix}\text{DPDAPN}\\\text{RPDAPN}\end{matrix}\right\}$ to obtain interpolation polynomial coefficients. Using that subroutine will be more efficient than using this subroutine. However, to obtain multiple interpolation values that are distant from x_l, using this subroutine is more efficient since the value obtained by 5.2.2 $\left\{\begin{matrix}\text{DPDAPN}\\\text{RPDAPN}\end{matrix}\right\}$ will deviate significantly from the true value due to the Runge phenomenon.

(7) **Example**

(a) Problem
Given the following input data:

i	x_i	y_i
1	0.0	0.0
2	60.0	0.0824
3	120.0	0.2747
4	180.0	0.6502

obtain the interpolation value at XL.

(b) Input data
Data points(X, Y), N=4, XL=150.0 and
EPS $= 5.0 \times 10^{-3}$.

(c) Main program

```
      PROGRAM BPDOPL
! *** EXAMPLE OF DPDOPL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(N=4)
      DIMENSION X(N),Y(N),WK(N*5)
      READ(5,*) (X(I),Y(I),I=1,N)
      READ(5,*) XL
      READ(5,*) EPS
      WRITE(6,1000) N
      WRITE(6,1100) (I,X(I),Y(I),I=1,N)
```

```
      WRITE(6,1110) XL
      WRITE(6,1120) EPS
      CALL DPDOPL(X,Y,N,XL,EPS,FL,DL,WK,IERR)
      WRITE(6,1200) IERR
      WRITE(6,1300) DL
      WRITE(6,1400) FL
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DPDOPL ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NUMBER OF DATA POINTS = ',I3)
 1100 FORMAT(' ',/,/,6X,'DATA POINTS (X,Y)',&
            /,/,9X,'I',7X,'X(I)',10X,'Y(I)',&
            /,4(8X,I2,F13.4,F14.4,/))
 1110 FORMAT(' ',/,8X,'INTERPOLATION POINT',/,11X,D13.4)
 1120 FORMAT(' ',/,8X,'ABSOLUTE ERROR',/,11X,D13.4)
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1300 FORMAT(' ',/,8X,'ABSOLUTE ERROR',/,11X,D18.9)
 1400 FORMAT(' ',/,8X,'INTERPOLATED VALUE',/,11X,D18.9)
      END
```

(d) Output results

```
*** DPDOPL ***

 ** INPUT **

   NUMBER OF DATA POINTS =   4


 DATA POINTS (X,Y)

    I       X(I)          Y(I)
    1        0.0000        0.0000
    2       60.0000        0.0824
    3      120.0000        0.2747
    4      180.0000        0.6502


   INTERPOLATION POINT
         0.1500D+03

   ABSOLUTE ERROR
         0.5000D-02

 ** OUTPUT **

   IERR =    0

   ABSOLUTE ERROR
         0.458125000D-02

   INTERPOLATED VALUE
         0.434968750D+00
```

5.2.2 DPDAPN, RPDAPN
Unequally Spaced Discrete Point Interpolation Value and Interpolation Coefficients

(1) **Function**

DPDAPN or RPDAPN obtains the interpolation value f_l at the interpolation point x_l and the coefficients $c_i (i = 1, \cdots, m+1)$ of the interpolation polynomial

$$f(x) = c_1 + \sum_{i=2}^{m+1} \prod_{j=1}^{i-1} c_i (x - u_j)$$

by using Newton's method to interpolate n given points (x_i, y_i) $(i = 1, \cdots, n)$.

(2) **Usage**

Double precision:

CALL DPDAPN (X, Y, N, XL, M, C, FL, U, WK, IERR)

Single precision:

CALL RPDAPN (X, Y, N, XL, M, C, FL, U, 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	X	{D / R}	N	Input	X coordinates x_i of cross section lines
2	Y	{D / R}	N	Input	Y coordinates y_i of cross section lines
3	N	I	1	Input	Number of data points n
4	XL	{D / R}	1	Input	Interpolation point x_l
5	M	I	1	Input	Degree m of interpolation polynomial
6	C	{D / R}	M,M+1	Output	Newton's divided differences (interpolation polynomial coefficients c_i = C(1, i) (See Notes (a))
7	FL	{D / R}	1	Output	Interpolation value f_l at x_l
8	U	{D / R}	N	Output	Sorted X coordinate values u_j
9	WK	{D / R}	N × 2	Work	Work area
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $M > 0$

(b) $N \geq M + 1$

(c) $X(i) \neq X(j)$ $(i \neq j)$

(d) $\min_{i=1,\cdots,N}(X(i)) \leq XL \leq \max_{i=1,\cdots,N}(X(i))$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3020	Restriction (d) was not satisfied.	
4000	Overflow occurred during the interpolation value calculation.	

(6) **Notes**

(a) The coefficients $C(1,i)$ $(i = 1, 2, \cdots, M+1)$ that are obtained are coefficients of the polynomial expressed as: $YL = C(1,1) + C(1,2) \times (XL - U(1))$
$+C(1,3) \times (XL - U(1)) \times (XL - U(2))$
$+\cdots$
$+C(1,M+1) \times (XL - U(1)) \times \cdots \times (XL - U(M))$
Therefore, to obtain the approximation value in the vicinity of XL by using these coefficients, perform the following calculation, for example.
Sample calculation (obtains the approximation value Y at point X using C(1, I))

```
Y=C(1, M+1)
DO 20 I=M, 1, -1
  Y=C(1, I)+(X-U(I))*Y
20 CONTINUE
```

Although the method described above is efficient for obtaining an approximation value of a value in the vicinity of XL, if this method is used to obtain an approximation value of a value distant from XL, the precision may be poor. Therefore, this subroutine should be used again or subroutine 5.2.1 $\left\{\begin{matrix}\text{DPDOPL}\\\text{RPDOPL}\end{matrix}\right\}$ should be used.

(b) In this subroutine, only $M + 1$ input data X(i) in the vicinity of XL are efficient.

(7) **Example**

(a) Problem
Given the following input data:

i	x_i	y_i
1	-1.0	9.0
2	0.0	6.0
3	-3.0	-6.0
4	1.0	-4.0
5	2.0	-6.0

obtain the coefficients of the interpolation polynomial for these points and the interpolation value.

(b) Input data

Data points(X, Y), N=5, XL=-2.0 and M=4.

(c) Main program

```
      PROGRAM BPDAPN
! *** EXAMPLE OF DPDAPN ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(N=5,M=4)
      DIMENSION X(N),Y(N),C(M,M+1),U(N),WK(N*2)
      READ(5,*) (X(I),Y(I),I=1,N)
      READ(5,*) XL
      WRITE(6,1000) N,M
      WRITE(6,1100) (I,X(I),Y(I),I=1,N)
      WRITE(6,1110) XL
      CALL DPDAPN(X,Y,N,XL,M,C,FL,U,WK,IERR)
      WRITE(6,1200) IERR
      WRITE(6,1210) (I,U(I),I=1,M+1)
      WRITE(6,1300) FL
      WRITE(6,1400) (I,C(1,I),I=1,M+1)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DPDAPN ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NUMBER OF DATA POINTS = ',I3,&
             /,/,8X,'INTERPOLATING POLYNOMIAL''S DEGREE = ',I3)
 1100 FORMAT(' ',/,/,6X,'DATA POINTS (X,Y)',&
             /,/,9X,'I',7X,'X(I)',10X,'Y(I)',&
             /,5(8X,I2,F13.4,F14.4,/))
 1110 FORMAT(' ',/,8X,'INTERPOLATION POINT',/,11X,D13.4)
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1210 FORMAT(' ',/,8X,'SORTED X',/,/,(9X,'U(',I3,') = ',D18.9))
 1300 FORMAT(' ',/,8X,'INTERPOLATED VALUE',/,11X,D18.9)
 1400 FORMAT(' ',/,8X,'COEFFICIENT',/,/,(9X,'C(',I3,') = ',D18.9))
      END
```

(d) Output results

```
 *** DPDAPN ***

  ** INPUT **

    NUMBER OF DATA POINTS =   5

    INTERPOLATING POLYNOMIAL'S DEGREE =   4


  DATA POINTS (X,Y)

     I       X(I)          Y(I)
     1      -1.0000        9.0000
     2       0.0000        6.0000
     3      -3.0000       -6.0000
     4       1.0000       -4.0000
     5       2.0000       -6.0000


    INTERPOLATION POINT
         -0.2000D+01

  ** OUTPUT **

    IERR =    0

    SORTED X

     U(  1) =   -0.100000000D+01
     U(  2) =   -0.300000000D+01
     U(  3) =    0.000000000D+00
     U(  4) =    0.100000000D+01
     U(  5) =    0.200000000D+01

    INTERPOLATED VALUE
          0.200000000D+01

    COEFFICIENT

     C(  1) =    0.900000000D+01
     C(  2) =    0.750000000D+01
     C(  3) =   -0.350000000D+01
     C(  4) =    0.000000000D+00
     C(  5) =    0.500000000D+00
```

5.3 SURFACE INTERPOLATION

5.3.1 DPLOPL, RPLOPL

Discrete Point Interpolation Value on Two-Dimensional Cross Section Lines

(1) **Function**

DPLOPL or RPLOPL establishes straight lines (called cross section lines here) $x = x_i$ $(i = 1, 2, \cdots, nx)$ parallel to the Y axis in the X, Y plane and obtains the interpolation value f_l at an arbitrary point (x_l, y_l) by assigning data points $(y_{j,i}, z_{j,i})$ $(j = 1, \cdots, ny_i)$ on each of those x_i cross sections and interpolating these according to a cubic spline function. This subroutine also obtains spline coefficients used for interpolating data on the given cross section lines.

(2) **Usage**

Double precision:

CALL DPLOPL (X, NX, Y, Z, MY, NY, XL, YL, FL, CSP, ISW, WK, IERR)

Single precision:

CALL RPLOPL (X, NX, Y, Z, MY, NY, XL, YL, FL, CSP, ISW, 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	X	{D / R}	NX	Input	X Coordinates x_i of cross section lines
2	NX	I	1	Input	Number of cross section lines (dimension of array X) nx
3	Y	{D / R}	MY,NX	Input	Y coordinate values $y_{i,j}$ of data points on cross section line x_i
4	Z	{D / R}	MY,NX	Input	Z coordinate values $z_{i,j}$ of data points on cross section line x_i
5	MY	I	1	Input	Maximum number of data points on cross section lines MAX(ny_i)
6	NY	I	NX	Input	Number of data points ny_i on each cross section line
7	XL	{D / R}	1	Input	Interpolation point X coordinate value x_l
8	YL	{D / R}	1	Input	Interpolation point Y coordinate value y_l
9	FL	{D / R}	1	Output	Interpolation value f_l at interpolation point (x_l, y_l)
10	CSP	{D / R}	See Contents	Output	Spline coefficients for data on each cross section line **Size**: $3 \times (\mathrm{MY}-1), \mathrm{NX}$
11	ISW	I	1	Input/ Output	Processing switch. Enter 0 as the initial value. After processing terminates, 1 is returned. If ISW $\neq$ 0, processing is performed using the spline coefficients of the previous execution.
12	WK	{D / R}	See Contents	Work	Work area **Size**: $5 \times \mathrm{NX} - 3$
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $NX \geq 2,\ NY(i) \geq 2\ (i = 1, \cdots, NX)$

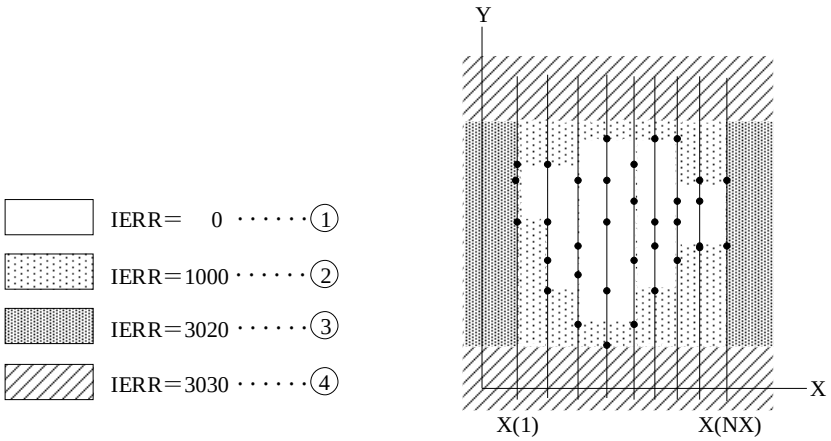
(b) $X(1) < X(2) < \cdots\cdots < X(NX)$ (ascending order)
$Y(1,i) < Y(2,i) < \cdots < Y(NY(i),i)$
$(i = 1, \cdots, NX)$ (ascending order)

(c) $X(1) \leq XL \leq X(NX)$

(d) $\min\limits_{1 \leq i \leq NX} Y(j,i) \leq YL \leq \max\limits_{1 \leq i \leq NX} Y(j,i)$

- Relationship between interpolation point and IERR
The Figure 5−5 shows X-Y coordinates of data points. The vertical lines (on which black dots are displayed within the figure) indicate cross section lines. The IERR values are determined according to the position of the interpolation point (XL, YL). (Within the figure, the boundary is included in the area for which the value is smaller.)

Figure 5−5



(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The interpolation point is in the range of indicated by ② in Figure 5−5.	A value extrapolated by using the spline coefficients at the end points is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied. (The interpolation point is in the range indicated by ③ in Figure 5−5)	
3030	Restriction (d) was not satisfied. (The interpolation point is in the range indicated by ④ in Figure 5−5)	

(6) **Notes**

(a) The spline coefficients CSP and the following subroutine can be used to obtain the volume of the region that surrounds the entered data points:

6.2.7 {DGIIPC / RGIIPC}, 6.2.20 {DGIICZ / RGIICZ}

For: S···Area of cross section (Real; Size NX)
V···Volume of solid (Real; Size 1)
C···Work (Real; Size 3, (NX − 1))

the main portion of the program is as follows
(double precision example):

```
      DO 10 I=1, NX
      CALL DGIICZ(Y(1, I), Z(1, I), NY(I), CSP(1, I),
     &           Y(1, I), Y(NY(I),I), S(I), IERR)
 10   CONTINUE
      CALL DGIIPC(X, S, NX, X(1), X(NX), V, C, IERR)
```

(b) The first time this subroutine is called, set ISW to 0. If you do not, the results will be invalid. For the second and subsequent time this subroutine is called, when interpolating a different interpolation point using the same data, retain the contents of ISW, set new values for XL and YL, and call this subroutine again. (At this time, the value of ISW has been changed to 1.) In this case, processing will be faster than the first time the subroutine was called since processing is performed using spline coefficients that have already been obtained.

≀
XL=(Interpolation point X coordinate value ①)
YL=(Interpolation point Y coordinate value ①)
ISW=0 ····· (Set ISW)
CALL {DPLOPL / RPLOPL} (X, ···, XL, YL, ···, ISW, ···) ····· (First time)
≀
XL=(Interpolation point X coordinate value ②)
····· (Do not set ISW)
YL=(Interpolation point Y coordinate value ②) ⋮
·····
CALL {DPLOPL / RPLOPL} (X, ···, XL, YL, ···, ISW, ···) ····· (Second time)
≀

(7) **Example**

(a) Problem

Given the following input data:

$$x_1 = 1.0$$
$$x_2 = 2.0$$
$$x_3 = 3.0$$
$$x_4 = 4.0$$
$$x_5 = 5.0$$

j	1	2	3	4	5	6
$y_{j,1}$ =	0.0,	1.0,	2.0,	3.0		
$z_{j,1}$ =	3.0,	2.82843,	2.23607,	0.0		
$y_{j,2}$ =	0.0,	1.0,	2.0,	3.0,	4.0	
$z_{j,2}$ =	4.0,	3.87298,	3.46410,	2.64575,	0.0	
$y_{j,3}$ =	0.0,	1.0,	2.0,	3.0,	4.0,	4.58258
$z_{j,3}$ =	4.58258,	4.47214,	4.12311,	3.46410,	2.23607,	0.0
$y_{j,4}$ =	0.0,	1.0,	2.0,	3.0,	4.0,	4.89898
$z_{j,4}$ =	4.89898,	4.79583,	4.47214,	3.87298,	2.82843,	0.0
$y_{j,5}$ =	0.0,	1.0,	2.0,	3.0,	4.0,	5.0
$z_{j,5}$ =	5.0,	4.89898,	4.58258,	4.0,	3.0,	0.0

obtain the interpolation value at $(x_l, y_l) = (1.6, 2.3)$ and $(3.2, 1.8)$.

(b) Input data

$\mathrm{X, NX} = 5, \mathrm{Y, Z, MY} = 6, \mathrm{XL(1)} = 1.6, \mathrm{YL(1)} = 2.3, \mathrm{XL(2)} = 3.2,$
$\mathrm{YL(2)} = 1.8, \mathrm{NY(1)} = 4, \mathrm{NY(2)} = 5, \mathrm{NY(3)} = 6, \mathrm{NY(4)} = 6$ and
$\mathrm{NY(5)} = 6.$

(c) Main program

```
      PROGRAM BPLOPL
! *** EXAMPLE OF DPLOPL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (NX=5,MY=6)
      DIMENSION X(NX),NY(NX),Y(MY,NX),Z(MY,NX),CSP(15,5),WK(22)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) (NY(I),I=1,NX)
      READ(5,*) ((Y(J,I),J=1,NY(I)),I=1,NX)
      READ(5,*) ((Z(J,I),J=1,NY(I)),I=1,NX)
      READ(5,*) XL1,YL1
      READ(5,*) XL2,YL2
      WRITE(6,1000) NX,(NY(I),I=1,NX),MY,XL1,YL1,XL2,YL2
      WRITE(6,1100) (X(I),I=1,NX)
      WRITE(6,1200)
      DO 10 I=1,NX
         WRITE(6,1300) (Y(J,I),J=1,NY(I))
   10 CONTINUE
      WRITE(6,1400)
      DO 20 I=1,NX
         WRITE(6,1300) (Z(J,I),J=1,NY(I))
   20 CONTINUE
      ISW = 0
      CALL DPLOPL(X,NX,Y,Z,MY,NY,XL1,YL1,FL1,CSP,ISW,WK,IERR)
      WRITE(6,1500) IERR
      IF(IERR.GE.3000) GO TO 9999
      WRITE(6,1600)
      WRITE(6,1700) FL1
      CALL DPLOPL(X,NX,Y,Z,MY,NY,XL2,YL2,FL2,CSP,ISW,WK,IERR)
      WRITE(6,1800) IERR
      IF(IERR.GE.3000) GO TO 9999
      WRITE(6,1900)
      WRITE(6,2000) FL2
 9999 STOP
! ***** FORMAT *****
 1000 FORMAT(' ',/,5X,'*** DPLOPL ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NX =',I3,/,/,8X,'NY(I) =',5I3,/,/,8X,'MY =',I3,&
             /,/,8X,'XL1 =',D17.10,/,/,8X,'YL1 =',D17.10,&
```

```
           /,/,8X,'XL2 =',D17.10,/,/,8X,'YL2 =',D17.10)
1100 FORMAT(' ',/,6X,'X =',5(2X,F9.5))
1200 FORMAT(' ',/,6X,'Y =',/)
1300 FORMAT(9X,6(2X,F9.5))
1400 FORMAT(' ',/,/,6X,'Z(X,Y) =',/)
1500 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR =',I4)
1600 FORMAT(' ',/,8X,'( INTERPOLATED VALUE AT (XL1,YL1) )',/)
1700 FORMAT(9X,'FL1 = ',D17.10)
1800 FORMAT(' ',/,/,6X,'-- (XL2,YL2) --',/,/,8X,'IERR =',I4)
1900 FORMAT(' ',/,8X,'( INTERPOLATED VALUE AT (XL2,YL2) )',/)
2000 FORMAT(9X,'FL2 = ',D17.10)
     END
```

(d) Output results

```
*** DPLOPL ***

 ** INPUT **

   NX =  5

   NY(I) =  4  5  6  6  6

   MY =  6

   XL1 = 0.1600000000D+01

   YL1 = 0.2300000000D+01

   XL2 = 0.3200000000D+01

   YL2 = 0.1800000000D+01

 X =    1.00000    2.00000    3.00000    4.00000    5.00000

 Y =

        0.00000    1.00000    2.00000    3.00000
        0.00000    1.00000    2.00000    3.00000    4.00000
        0.00000    1.00000    2.00000    3.00000    4.00000    4.58258
        0.00000    1.00000    2.00000    3.00000    4.00000    4.89898
        0.00000    1.00000    2.00000    3.00000    4.00000    5.00000


 Z(X,Y) =

        3.00000    2.82843    2.23607    0.00000
        4.00000    3.87298    3.46410    2.64575    0.00000
        4.58258    4.47214    4.12311    3.46410    2.23607    0.00000
        4.89898    4.79583    4.47214    3.87298    2.82843    0.00000
        5.00000    4.89898    4.58258    4.00000    3.00000    0.00000


 ** OUTPUT **

   IERR =   0

   ( INTERPOLATED VALUE AT (XL1,YL1) )

    FL1 =  0.2849715886D+01


 -- (XL2,YL2) --

   IERR =   0

   ( INTERPOLATED VALUE AT (XL2,YL2) )

    FL2 =  0.4313073896D+01
```

5.3.2 DPGOPL, RPGOPL
Discrete Point Interpolation Value on a Two-Dimensional Lattice

(1) **Function**

DPGOPL or RPGOPL obtains the interpolation value f_l at a single arbitrary point (x_l, y_l) within a lattice when all Z coordinate values $z_{i,j}$ on two-dimensional lattice points (x_i, y_i) $(i = 1, 2, \cdots, nx;\ j = 1, 2, \cdots, ny)$ are given.

(2) **Usage**

Double precision:

```
CALL DPGOPL  (X, NX, Y, NY, Z, EPS, XL, YL, FL, WK, IERR)
```

Single precision:

```
CALL RPGOPL  (X, NX, Y, NY, Z, EPS, XL, YL, FL, 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	X	{D / R}	NX	Input	X coordinate values x_i of data points
2	NX	I	1	Input	Dimension nx of array X
3	Y	{D / R}	NY	Input	Y coordinate values y_i of data points
4	NY	I	1	Input	Dimension ny of array Y
5	Z	{D / R}	NX, NY	Input	Z(I, J) is the Z coordinate value $z_{i,j}$ at data point (x_i, y_j)
6	EPS	{D / R}	1	Input	Required absolute precision (Default value: Unit for determining error × 64)
7	XL	{D / R}	1	Input	Interpolation point X coordinate value x_l
8	YL	{D / R}	1	Input	Interpolation point Y coordinate value y_l
9	FL	{D / R}	1	Output	interpolation value f_l at interpolation point (x_l, y_l)
10	WK	{D / R}	See Contents	Work	Work area **Size**: NY + 5 × MAX(NX, NY)
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) EPS $\geq$ Unit for determining error $\times 64$

(b) NX ≥ 2, NY ≥ 2

(c) X(i) $\neq$ X(j), Y(i) $\neq$ Y(j) (i $\neq$ j)

(d) $\min\limits_{1\leq i\leq NX}(X(i)) \leq XL \leq \max\limits_{1\leq i\leq NX}(X(i))$
$\min\limits_{1\leq i\leq NX}(Y(i)) \leq YL \leq \max\limits_{1\leq i\leq NX}(Y(i))$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Restriction (a) was not satisfied.	Processing is performed with the default value set.
2500	The precision could not be guaranteed since the absolute error of the interpolation value was not less than or equal to the required precision.	Processing is terminated without obtaining a solution having the required precision.
3000	Restriction (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3020	Restriction (d) was not satisfied.	
4000	Overflow occurred during the calculation.	

(6) **Notes**
None

(7) **Example**

(a) Problem
Given the following input data:

$x_1 = 1.0, \quad y_1 = 1.0$
$x_2 = 1.2, \quad y_2 = 1.2$
$x_3 = 1.4, \quad y_3 = 1.4$
$x_4 = 1.6, \quad y_4 = 1.6$
$\qquad\quad y_5 = 1.8$

		$\xrightarrow{j}$				
	$z_{i,j}$	y_1	y_2	y_3	y_4	y_5
	x_1	2.0,	1.56,	1.04,	0.44,	-0.24
$i\downarrow$	x_2	2.88,	2.44,	1.92,	1.32,	0.64
	x_3	3.92,	3.48,	2.96,	2.36,	1.68
	x_4	5.12,	4.68,	4.16,	3.56,	2.88

obtain the interpolation value at $(x_l, y_l) = (1.3, 1.5)$.

(b) Input data
X, NX=4, Y, NY=5, Z, EPS=0.1, XL and YL.

(c) Main program

```
      PROGRAM BPGOPL
! *** EXAMPLE OF DPGOPL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (NX=4,NY=5)
      DIMENSION   X(NX),Y(NY),Z(NX,NY),WK(30)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) (Y(J),J=1,NY)
      READ(5,*) ((Z(I,J),J=1,NY),I=1,NX)
      READ(5,*) EPS
      READ(5,*) XL
      READ(5,*) YL
      WRITE(6,1000) NX,NY,EPS,XL,YL
      WRITE(6,1100) (X(I),I=1,NX)
      WRITE(6,1200) (Y(J),J=1,NY)
      WRITE(6,1300)
      DO 10 I=1,NX
         WRITE(6,1400) (Z(I,J),J=1,NY)
   10 CONTINUE
      CALL DPGOPL(X,NX,Y,NY,Z,EPS,XL,YL,FL,WK,IERR)
      WRITE(6,1500) IERR
      IF(IERR.GE.3000) GO TO 9999
      WRITE(6,1600)
      WRITE(6,1700) FL
 9999 STOP
! ***** FORMAT *****
 1000 FORMAT(' ',/,5X,'*** DPGOPL ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NX =',I3,/,/,8X,'NY =',I3,/,/,8X,'EPS =',D17.10,&
             /,/,8X,'XL =',D17.10,/,/,8X,'YL =',D17.10)
 1100 FORMAT(' ',/,15X,'X =',4(2X,F9.5))
 1200 FORMAT(' ',/,15X,'Y =',5(2X,F9.5))
 1300 FORMAT(' ',/,/,6X,'Z(X,Y) =',/)
 1400 FORMAT(9X,5(2X,F9.5))
 1500 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR =',I4)
 1600 FORMAT(' ',/,/,8X,'( INTERPOLATED VALUE AT (XL,YL) )',/)
 1700 FORMAT(9X,'FL = ',D17.10)
      END
```

(d) Output results

```
   *** DPGOPL ***

    ** INPUT **

      NX =  4

      NY =  5

      EPS = 0.1000000000D+00

      XL = 0.1300000000D+01

      YL = 0.1500000000D+01

             X =    1.00000    1.20000    1.40000    1.60000

             Y =    1.00000    1.20000    1.40000    1.60000    1.80000


    Z(X,Y) =

           2.00000    1.56000    1.04000    0.44000   -0.24000
           2.88000    2.44000    1.92000    1.32000    0.64000
           3.92000    3.48000    2.96000    2.36000    1.68000
           5.12000    4.68000    4.16000    3.56000    2.88000


    ** OUTPUT **

      IERR =   0


      ( INTERPOLATED VALUE AT (XL,YL) )

       FL =  0.2130000000D+01
```

5.4 LEAST SQUARES APPROXIMATION

5.4.1 DNDAAO, RNDAAO

Least Squares Approximation Orthogonal Polynomial Having Automatically Determined Degree

(1) **Function**

DNDAAO or RNDAAO takes (x_i, y_i) $(i = 1, \cdots, n)$ as given coordinate values and obtains the degree m, coefficients a_j $(j = 1, \cdots, m+1)$, and $f(x_i)$ values of the optimum polynomial $f(x) = \sum_{j=1}^{m+1} a_j x^{m+1-j}$ that minimizes the weighted sum of the squares of the differences of the approximate value $f(x_i)$ and y_i, which is expressed as $\sum_{i=1}^{n} w(x_i)\{y_i - f(x_i)\}^2$.

(2) **Usage**

Double precision:

```
CALL DNDAAO  (X, Y, W, N, A, M, SX, F, WK, IERR)
```

Single precision:

```
CALL RNDAAO  (X, Y, W, N, A, M, SX, F, 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	X	{D / R}	N	Input	X coordinates x_i of data points
2	Y	{D / R}	N	Input	Y coordinates y_i of data points
3	W	{D / R}	N	Input	Weight function values W(i) = w(x_i) at data points (See Notes (a))
4	N	I	1	Input	Number of data points n
5	A	{D / R}	M + 1	Output	Coefficients a_i of approximation polynomial (A(1): Coefficient of highest degree term, $\cdots$, A(M+1): Constant term)
6	M	I	1	Output	Degree m of optimum approximation polynomial
7	SX	{D / R}	1	Output	If IERR = 0, SX = 0 If IERR = 1000, A is the array of polynomial coefficients for (X + SX)
8	F	{D / R}	N	Output	Approximate value of $f(x_i)$ of Y coordinate at x_i
9	WK	{D / R}	N × 8	Work	Work area
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 2

(b) W(i) $\geq$ 0 (i = $1, \cdots,$ N)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Since the X coordinate range does not include the origin, the approximation values are not accurately obtained by using the general polynomial coefficients.	Polynomial coefficients for $f(x+s)$ are output to A and s is output to SX. (See Notes (b))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	The number of data points for which the weight is not 0.0 and the X coordinates are considered to differ is less than the optimum degree plus one. (X coordinates are considered to differ here if the difference of the X coordinate at one point with that at another point is greater than the entire range times the square root of the unit for determining error.	
4000	Overflow or division by zero occurred during the polynomial coefficient calculation.	

(6) **Notes**

(a) If $w(x_k)$ is greater than $w(x_j)$ $(j \neq k)$, then the value of F at $x = x_k$ is closer to the value of y_k than the value of F at $x = x_j$ is to the value of y_j.

(b) The origin at which $x = 0$ must be included in the X, Y coordinate distribution range. Since the precision of the approximation value calculations by a polynomial in terms of x will decrease significantly if the origin is not included, the following processing is performed.

- IERR = 1000 is returned.
- The amount s that the X coordinates are shifted is returned in SX.
- The coefficients of the polynomial for $f(x+s) = \sum_{i=1}^{m+1} a'_i (x+s)^{m+1-i}$ are returned in A.

Therefore, the approximation polynomial value calculation will be, for example, as follows.
Sample calculation (obtains the approximation polynomial value Y at point X)

```
      IF (IERR .EQ. 1000)X=X+SX
      Y=A(1)
      DO 10 I=2, M+1
        Y=Y*X+A(I)
   10 CONTINUE
```

However, the Y value obtained by this calculation for input data point X may differ in the error range from the F value obtained by using the subroutine. The reason is that the F value is obtained by using an intermediate orthogonal polynomial that is used for obtaining the approximation polynomial within the subroutine.

(7) **Example**

(a) Problem

Given the following input data:

i	x_i	y_i	$w(x_i)$
1	0.0	5.312	1.0
2	0.3	6.044	1.0
3	0.7	8.276	1.0
4	1.0	11.000	1.0
5	1.2	13.496	1.0

obtain the coefficients of the approximation polynomial.

(b) Input data

Data points (X(i), Y(i)), weight function values W(i) and N = 5.

(c) Main program

```
      PROGRAM BNDAAO
! *** EXAMPLE OF DNDAAO ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(N=5)
      DIMENSION X(N),Y(N),W(N),A(N),F(N),WK(N*8)
      READ(5,*) (X(I),Y(I),W(I),I=1,N)
      WRITE(6,1000) N
      WRITE(6,1100) (I,X(I),Y(I),W(I),I=1,N)
      CALL DNDAAO(X,Y,W,N,A,M,SX,F,WK,IERR)
      WRITE(6,1200) IERR,M
      WRITE(6,1300) SX
      WRITE(6,1400) (I,A(I),I=1,M+1)
      WRITE(6,1500) (I,F(I),I=1,N)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DNDAAO ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NUMBER OF DATA POINTS = ',I3)
 1100 FORMAT(' ',/,/,6X,'DATA POINTS (X,Y) , WEIGHT FUNCTION VALUE',&
            /,/,9X,'I',7X,'X(I)',10X,'Y(I)',6X,'W(I)',&
            /,5(8X,I2,F13.4,F14.4,F10.4,/))
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4,&
            /,/,8X,'OPTIMAL DEGREE = ',I3)
 1300 FORMAT(' ',/,8X,'SX = ',D17.10)
 1400 FORMAT(' ',/,8X,'COEFFICIENTS OF POLYNOMIAL',&
            /,/,(9X,'A(',I2,') = ',D24.10))
 1500 FORMAT(' ',/,8X,'APPROXIMATE VALUE',&
            /,/,(9X,'F(',I2,') = ',D24.10))
      END
```

(d) Output results

```
 *** DNDAAO ***

  ** INPUT **

    NUMBER OF DATA POINTS =   5

  DATA POINTS (X,Y) , WEIGHT FUNCTION VALUE

     I       X(I)          Y(I)      W(I)
     1       0.0000        5.3120    1.0000
     2       0.3000        6.0440    1.0000
     3       0.7000        8.2760    1.0000
     4       1.0000       11.0000    1.0000
     5       1.2000       13.4960    1.0000


  ** OUTPUT **

    IERR =    0

    OPTIMAL DEGREE =   4

    SX =  0.0000000000D+00

    COEFFICIENTS OF POLYNOMIAL

     A( 1) =         0.1238095238D+01
     A( 2) =        -0.1961904762D+01
     A( 3) =         0.5469523810D+01
     A( 4) =         0.9422857143D+00
     A( 5) =         0.5312000000D+01
```

```
APPROXIMATE VALUE

 F( 1) =         0.5312000000D+01
 F( 2) =         0.6044000000D+01
 F( 3) =         0.8276000000D+01
 F( 4) =         0.1100000000D+02
 F( 5) =         0.1349600000D+02
```

5.4.2 DNDAPO, RNDAPO
Least Squares Approximation Orthogonal Polynomials

(1) **Function**

DNDAPO or RNDAPO takes (x_i, y_i) $(i = 1, \cdots, n)$ as given coordinate values and obtains the coefficients a_j $(j = 1, \cdots, m+1)$ and $f(x_i)$ values of the optimum m-th degree polynomial $f(x) = \sum_{j=1}^{m+1} a_j x^{m+1-j}$ that minimizes the weighted sum of the squares of the differences of the approximate value $f(x_i)$ and y_i, which is expressed as $\sum_{i=1}^{n} w(x_i)\{y_i - f(x_i)\}^2$.

(2) **Usage**

Double precision:

```
CALL DNDAPO  (X, Y, W, N, A, M, SX, F, WK, IERR)
```

Single precision:

```
CALL RNDAPO  (X, Y, W, N, A, M, SX, F, 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	X	{D / R}	N	Input	X coordinates x_i of data points
2	Y	{D / R}	N	Input	Y coordinates y_i of data points
3	W	{D / R}	N	Input	Weight function values W(i) = w(x_i) of data points (See Notes (a))
4	N	I	1	Input	Number of data points
5	A	{D / R}	M + 1	Output	Coefficients a_i of approximate polynomial (A(1): Coefficient of highest degree term, $\cdots$, A(M+1): Constant term)
6	M	I	1	Input	Degree of approximate polynomial
7	SX	{D / R}	1	Output	If IERR = 0, SX = 0 If IERR = 1000, A is the array of polynomial coefficients for (X+SX)

No.	Argument	Type	Size	Input/ Output	Contents
8	F	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	N	Output	Approximate value $f(x_i)$ of Y at X
9	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	N × 8	Work	Work area
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) M>0

(b) $\text{N}' \geq \text{M}+1$ (N′:Number of data points for which the weight is not 0.0 and the X coordinate spread exceeds $\sqrt{\varepsilon}$ of the entire data distribution range) (ε: Unit for determining error)

(c) $\text{W(i)} \geq 0$ $(\text{i} = 1, \cdots, \text{N})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Since the X coordinate range does not include the origin, approximation values are not accurately obtained by using the general polynomial coefficients.	Polynomial coefficients for $f(x+s)$ are output in A and s is output in SX. (See Notes (c))
3000	Restriction (a) or (c) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	Overflow occurred during the calculation.	

(6) **Notes**

(a) If $w(x_k)$ is larger than $w(x_j)$ $(j \neq k)$, then the value of F at $x = x_k$ is closer to the value of y_k than the value of F at $x = x_j$ is to the value of y_j.

(b) If data is input for which the spread of the X coordinate values does not exceed $\sqrt{\varepsilon}$ of the X coordinate data distribution range, then the data must satisfy restriction (b).

(c) The origin at which $x = 0$ must be included in the X, Y coordinate distribution range. Since the precision of approximation value calculations by a polynomial in terms of x will decrease significantly if the origin is not included, the following processing is performed.

- IERR = 1000 is returned.
- The amount s that the X coordinates are shifted is returned in SX.
- The coefficients a'_i of the polynomial for $f(x+s) = \sum_{i=1}^{m+1} a'_i (x+s)^{m+1-i}$ are returned in A.

Therefore, the approximation polynomial value calculation will be, for example, as follows.
Sample calculation (obtains the approximation polynomial value Y at point X)

```
IF (IERR .EQ. 1000)X=X+SX
Y=A(1)
DO 10 I=2, M+1
```

```
        Y=Y*X+A(I)
   10 CONTINUE
```

However, the Y value obtained by this calculation for input data point X may differ in the error range from the F value obtained by using the subroutine. The reason is that the F value is obtained by using an intermediate orthogonal polynomial that is used for obtaining the approximation polynomial within the subroutine.

(d) Since a decrease in precision occurs more readily if a larger value is taken for the degree M of the approximate polynomial, double precision should be used.

(7) **Example**

(a) Problem
Given the following input data:

i	x_i	y_i	$w(x_i)$
1	0.0	5.312	1.0
2	0.3	6.044	1.0
3	0.7	8.276	1.0
4	1.0	11.000	1.0
5	1.2	13.496	1.0

obtain the coefficients of the approximate polynomial.

(b) Input data
Data points (x_i, y_i), weight function values $w(x_i)$, N=5 and M=3.

(c) Main program

```
      PROGRAM BNDAPO
! *** EXAMPLE OF DNDAPO ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(M=3,N=5)
      DIMENSION X(N),Y(N),W(N),A(M+1),F(N),WK(N*8)
      READ(5,*) (X(I),Y(I),W(I),I=1,N)
      WRITE(6,1000) M,N
      WRITE(6,1100) (I,X(I),Y(I),W(I),I=1,N)
      CALL DNDAPO(X,Y,W,N,A,M,SX,F,WK,IERR)
      WRITE(6,1200) IERR
      WRITE(6,1300) SX
      WRITE(6,1400) (I,A(I),I=1,M+1)
      WRITE(6,1500) (I,F(I),I=1,N)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DNDAPO ***',/,/,6X,'** INPUT **',&
             /,/,8X,'DEGREE OF APPROXIMATE POLYNOMIAL = ',I3,&
             /,/,8X,'NUMBER OF DATA POINTS = ',I3)
 1100 FORMAT(' ',/,/,6X,'DATA POINTS (X,Y) , WEIGHT FUNCTION VALUE',&
             /,/,9X,'I',7X,'X(I)',10X,'Y(I)',6X,'W(I)',&
             /,7(8X,I2,F13.4,F14.4,F10.4,/))
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1300 FORMAT(' ',/,8X,'SX = ',D17.10)
 1400 FORMAT(' ',/,8X,'COEFFICIENT OF X',/,/,(9X,'A(',I2,') = ',D24.10))
 1500 FORMAT(' ',/,8X,'APPROXIMATE VALUE',/,/,(9X,'F(',I2,') = ',D24.10))
      END
```

(d) Output results

```
*** DNDAPO ***

 ** INPUT **

   DEGREE OF APPROXIMATE POLYNOMIAL =   3

   NUMBER OF DATA POINTS =   5


 DATA POINTS (X,Y) , WEIGHT FUNCTION VALUE

    I       X(I)          Y(I)      W(I)
    1       0.0000        5.3120    1.0000
    2       0.3000        6.0440    1.0000
    3       0.7000        8.2760    1.0000
    4       1.0000       11.0000    1.0000
```

```
   5       1.2000       13.4960    1.0000

** OUTPUT **

  IERR =    0

  SX =  0.0000000000D+00

  COEFFICIENT OF X

   A( 1) =         0.1017511521D+01
   A( 2) =         0.3280184332D+01
   A( 3) =         0.1414691244D+01
   A( 4) =         0.5308516129D+01

  APPROXIMATE VALUE

   F( 1) =         0.5308516129D+01
   F( 2) =         0.6055612903D+01
   F( 3) =         0.8255096774D+01
   F( 4) =         0.1102090323D+02
   F( 5) =         0.1348787097D+02
```

5.4.3 DNDANL, RNDANL
Least Squares Approximation Nonlinear Functions

(1) **Function**

DNDANL or RNDANL fits an approximate function $f(x_i, \boldsymbol{a})$ to given coordinate values (x_i, y_i) $(i = 1, \cdots, n)$ and optimizes parameters a_i $(i = 1, \cdots, m)$ where a_i are components of $\boldsymbol{a}$ so that the sum of the squares of the residuals $y_i - (x_i, \boldsymbol{a})$ is minimized.

(2) **Usage**

Double precision:

```
CALL DNDANL  (F, X, Y, N, ER, NEV, A, M, YF, S, IWK, WK, IERR)
```

Single precision:

```
CALL RNDANL  (F, X, Y, N, ER, NEV, A, M, YF, S, 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	F	{D / R}	-	Input	Name of function subprogram F(X, A) of approximation function $f(x, \boldsymbol{a})$. (See Notes (a))
2	X	{D / R}	N	Input	X coordinate values of data points x_i
3	Y	{D / R}	N	Input	Y coordinate values of data points y_i
4	N	I	1	Input	Number of data points n
5	ER	{D / R}	1	Input	Required precision (Default value: $2 \times \sqrt{(\text{Unitfordeterminingerror})}$)
6	NEV	I	1	Input	Maximum number of evaluations of function $f(x, \boldsymbol{a})$ (Default value: $100 \times \text{N} \times \text{M}$)
				Output	Actual number of function evaluations
7	A	{D / R}	M	Input	Coefficient initial values $\boldsymbol{a}_0$
				Output	Optimum coefficients $\boldsymbol{a}^*$
8	M	I	1	Input	Number of coefficients m
9	YF	{D / R}	N	Output	Value of approximate function $f(x_i, \boldsymbol{a}^*)$ at x_i
10	S	{D / R}	1	Output	Minimum sum of squares value $s = \sum_{i=1}^{n} (y_i - f(x_i, \boldsymbol{a}^*))^2$
11	IWK	I	$4 \times \text{M}$	Work	Work area
12	WK	{D / R}	See Contents	Work	Work area **Size**: $\text{N} \times (2 \times \text{M} + 1) + \text{M} \times (\text{M} + 4)$
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $0 < \text{M} \leq \text{N}$

(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 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 values of A, YF, 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 2M 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 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 should be created as follows:

```
REAL(8) FUNCTION   F(X, A)
REAL(8)   A]
DIMENSION   A(*)
      F = f(x, a) (expression in terms of X and coefficients A(i))
RETURN
END
```

(b) Convergence is determined according to the following condition, and the solution is assumed to be $\boldsymbol{a} + \Delta\boldsymbol{a}$:

$$\|\Delta\boldsymbol{a}\| \leq \mathrm{ER} \times \max(1, \|\boldsymbol{a} + \Delta\boldsymbol{a}\|)$$

where $\Delta\boldsymbol{a}$ is the correction vector for $\boldsymbol{a}$ and $\|\boldsymbol{a}\| = \max|a_i|$. 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) The x_i and y_i coordinate values must be of nearly equal order. If the order differs, then the a_i values are obtained by scaling the x_i values so that they are on the same order as y_i, and then the a_i are transformed so that they become a_i for the original x_i values.

Example: Obtain the optimum a_1, a_2, a_3, a_4 for the function:

$$y = a_i[1 - \exp\{-(x - a_3)/a_2\}] + a_4$$

to fit a Weibull distribution curve.

(i) Assume:

$$y_{\min} = \min_{i=1,\cdots,n} y_i$$
$$x_{\min} = \min_{i=1,\cdots,n} x_i$$

Let:

$$y'_i = y_i - y_{\min}\ (i = 1, \cdots, n),\ a'_4 = y_{\min}$$
$$x'_i = x_i - x_{\min}\ (i = 1, \cdots, n),\ a'_3 = x_{\min}$$

Now, let:

$$y_{\max} = \max_{i=1,\cdots,n} y'_i$$
$$x_{\max} = \max_{i=1,\cdots,n} x'_i$$

(If $a_3 = a_4 = 0$, these operations are unnecessary.)

(These above operations perform a coordinate transformation.)

(ii) Let $s = y_{\max}/x_{\max}$ and set as follows:

$$x''_i = sx'_i (i = 1, \cdots, m) \text{(Scaling)}$$

(iii) Let the initial values of a_1 and a_2 be on the order of $y_{\max}$ and let the initial values of a_3 and a_4 be zero. Perform a nonlinear least squares approximation using data (x''_i, y'_i) $(i = 1, \cdots, n)$, for $f(x, \boldsymbol{a}) = a'_1[1 - \exp\{-(x - a'_3)/a'_2\}] + a'_4$, and obtain coefficients a'_1, a'_2, a'_3, and a'_4.

(iv) The coefficients a_2, a_3 and a_4 are as follows:

$$a_1 = a'_1$$
$$a_2 = a'_2/s$$
$$a_3 = a'_3/s + x_{\min}$$
$$a_4 = a'_4 + y_{\min}$$

(7) **Example**

(a) Problem

Obtain the optimum values of x_0, w, h, a_0, and a_1 by fitting the function:

(	−5.0,	2.7)
(	−4.0,	2.9)
(	−3.0,	3.1)
(	−2.0,	3.4)
(	1.0,	3.9)
(	0.0,	4.7)
(	1.0,	6.0)
(	2.0,	7.8)
(	3.0,	7.9)
(	4.0,	6.3)
(	5.0,	5.2)

to the following 11 data points(x, y):

$$f(x) = \frac{hw^2}{(x - x_0)^2 + w^2} + a_0 + a_1 x$$

Assume the following initial values:

$x_0 = 0.0$, $w = 1.0$, $h = 6.0$, $a_0 = 3.5$ and $a_1 = 0.2$.

(b) Input data

Subroutine name: FNDANL.

Array X and Y, N=11, ER=0.0 and NEV=0.

Array A:

Assign x_0, w, h, a_0, and a_1 sequentially to A(1), A(2), A(3), A(4) and A(5).

M=5.

(c) Main program

```
      PROGRAM BNDANL
! *** EXAMPLE OF DNDANL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (M = 5, N = 11)
      DIMENSION IWK(3*M)
      DIMENSION A(M),YF(N),WK(N*(2*M+1)+M*(M+4))
      DIMENSION X(N),Y(N)
      EXTERNAL FNDANL
!
      WRITE(6,1000)
      READ(5,*) X
      READ(5,*) Y
      READ(5,*) NEV
      READ(5,*) ER
      READ(5,*) A
      WRITE(6,1100) M,N,NEV,ER
      WRITE(6,1200) (I,X(I),Y(I),I=1,N)
      WRITE(6,1300) (I,A(I),I=1,M)
      CALL DNDANL(FNDANL,X,Y,N,ER,NEV,A,M,YF,S,IWK,WK,IERR)
      WRITE(6,1400) IERR,NEV,(I,A(I),I=1,M),S,(I,YF(I),I=1,N)
!
 1000 FORMAT(' ',/,' *** DNDANL ***')
 1100 FORMAT('  ** INPUT **',/,&
             5X,'M      =',I5,/,&
             5X,'N      =',I5,/,&
             5X,'NEV    =',I5,/,&
             5X,'ER     =',D18.10)
 1200 FORMAT(5X,'(( COORDINATES (X,Y) ))',/,&
             5X,'    I      X(I)     Y(I)',/,&
           (5X,I5,4X,F5.1,4X,F5.1))
 1300 FORMAT(5X,'(( INITIAL VALUE OF COEFFICIENTS ))',/,&
           (5X,'   A(',I2,')  =',F5.1))
 1400 FORMAT('  ** OUTPUT **',/,&
             5X,'IERR   =',I5,/,&
             5X,'NEV    =',I5,/,&
             5X,'(( OPTIMIZED COEFFICIENTS ))',/,&
          5(5X,'   A(',I2,') =',D18.10,/),&
             5X,'(( LEAST SQUARES ))',/,&
             5X,'   S     =',D18.10,/,&
             5X,'(( FUNCTION VALUE ))',/,&
         11(5X,'   YF(',I2,') =',D18.10,/))
      END


      REAL(8) FUNCTION FNDANL(X,A)
      REAL(8) X,A,F1,F2
      DIMENSION A(*)
!
      F1 = A(3)*A(2)*A(2)/((X-A(1))*(X-A(1))+A(2)*A(2))
      F2 = A(4)+A(5)*X
      FNDANL = F1+F2
      RETURN
      END
```

(d) Output results

```
*** DNDANL ***
 ** INPUT **
    M      =    5
    N      =   11
    NEV    =    0
    ER     =  0.0000000000D+00
    (( COORDINATES (X,Y) ))
        I      X(I)      Y(I)
        1     -5.0       2.7
        2     -4.0       2.9
        3     -3.0       3.1
        4     -2.0       3.4
        5     -1.0       3.9
        6      0.0       4.7
        7      1.0       6.0
        8      2.0       7.8
        9      3.0       7.9
       10      4.0       6.3
       11      5.0       5.2
    (( INITIAL VALUE OF COEFFICIENTS ))
       A( 1)  =  0.0
       A( 2)  =  1.0
       A( 3)  =  6.0
       A( 4)  =  3.5
```

```
      A( 5)  =  0.2
** OUTPUT **
   IERR   =    0
   NEV    =  759
   (( OPTIMIZED COEFFICIENTS ))
      A( 1) =  0.2492552430D+01
      A( 2) =  0.1800461441D+01
      A( 3) =  0.4973661261D+01
      A( 4) =  0.2959458530D+01
      A( 5) =  0.1078684881D+00
   (( LEAST SQUARES ))
      S      =  0.3605711487D-02
   (( FUNCTION VALUE ))
      YF( 1) =  0.2691637221D+01
      YF( 2) =  0.2883155010D+01
      YF( 3) =  0.3118433889D+01
      YF( 4) =  0.3432009279D+01
      YF( 5) =  0.3895849126D+01
      YF( 6) =  0.4664779965D+01
      YF( 7) =  0.6015182707D+01
      YF( 8) =  0.7802543266D+01
      YF( 9) =  0.7890714197D+01
      YF(10) =  0.6314899027D+01
      YF(11) =  0.5190794081D+01
```

5.5 LEAST SQUARES SURFACE APPROXIMATION

5.5.1 DNRAPL, RNRAPL

Two-Dimensional Arbitrary Data Least Squares Approximation Polynomial

(1) **Function**

DNRAPL or RNRAPL takes arbitrary data points on a plane $(x_k, y_k, z_k)(k = 1, \cdots, n)$ and obtains the coefficients $a_{i,j}$ and the approximate values $f(x_k, y_k)$ $(k = 1, \cdots, n)$ at the input data points of the m-th degree approximation polynomial for x and y:

$$f(x, y) = \sum_{j=1}^{m+1} \sum_{i=1}^{m+2-j} a_{i,j} x^{i-1} y^{j-1}$$

so that the sum of the squares of the differences of the polynomial values $f(x_k, y_k)$ and the Z coordinate values z_k at the data points is minimized.

(2) **Usage**

Double precision:

```
CALL DNRAPL  (X, Y, Z, N, M, A, F, IW, WK, IERR)
```

Single precision:

```
CALL RNRAPL  (X, Y, Z, N, M, A, F, IW, 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	X	{ D / R }	N	Input	X coordinates x_k of data points
2	Y	{ D / R }	N	Input	Y coordinates y_k of data points
3	Z	{ D / R }	N	Input	Z coordinates z_k of data points
4	N	I	1	Input	Number of data points
5	M	I	1	Input	Degree m for x and y of the approximation polynomial
6	A	{ D / R }	See Contents	Output	Coefficients $a_{i,j}$ of approximation polynomial **Size**: $(\mathrm{M}+1)\times(\mathrm{M}+2)/2$
7	F	{ D / R }	N	Output	Approximate value $f(x_k, y_k)$ of Z coordinate at (x_k, y_k)
8	IW	I	See Contents	Work	Work area **Size**: $(\mathrm{M}+1)\times(\mathrm{M}+2)/2$
9	WK	{ D / R }	See Contents	Work	Work area **Size**: $(\mathrm{M}+2)^4/4$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq (\mathrm{M}+1)\times(\mathrm{M}+2)/2$

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

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The degree of the approximation exceeded 10. (The approximation error may be large. (See Notes (c))	Processing continues using the assigned degree.
2100	There existed the diagonal element which was close to zero in the LU decomposition. The precise solutions or the inverse matrix may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied, or $\mathrm{N}' < \mathrm{N}^{'} < (\mathrm{M}+1)\times(\mathrm{M}+2)/2$ was satisfied where N' is the number of data points for which $(\mathrm{X(K)}, \mathrm{Y(K)})$ differed.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000+i	The pivot became 0.0 during the processing of the i-th step of the LU decomposition subroutine used to solve the normal equation internally.	

(6) **Notes**

(a) If all input data points (x_k, y_k) $(k = 1, \cdots, n)$ lie on a single straight line, they cannot define a surface. The result obtained in this case is a single indefinite solution.

(b) In this subroutine, the polynomial coefficients $a_{i,j}$ is stored in the one-dimensional array A to increase the efficiency. The correspondence relationship between $a_{i,j}$ and A is as follows:

$$a_{i,j} = \mathrm{A(i + \{(j-1)\times(2\times m - j + 4)\}/2)}$$

This corresponds as follows in the figure below. First, look at the terms listed in the first row from left to right. The first term, which is the constant term, corresponds to A(1). Then, sequentially, A(2) =coefficient of x, A(3) = coefficient of $x^2, \cdots, \mathrm{A(m+1)}$ = coefficient of x^m. Since we have reached the right end of the first row, move to the next row and look at the terms in a similar manner. $\mathrm{A(m+2)}$ = coefficient of $y, \cdots, \mathrm{A(2m+1)}$ = coefficient of yx^{m-1}. Proceeding in this way, terms are associated up until the $(m+1)$th row in which $\mathrm{A((m+1)\times(m+2))}$ = coefficient of y^m.

$$\begin{array}{lllll} 1 & x & x^2 & x^3 & \cdots \quad x^m \\ y & xy & x^2y & \cdots & x^{m-1}y \\ y^2 & xy^2 & \cdots & \cdot & \\ \cdots & & \cdot & & \\ \cdots & \cdot & & & \\ y^m & & & & \end{array}$$

(c) Since this subroutine approximates the given points on the surface by only one polynomial, meaningless results may be produced depending on the data. In particular, if the degree of the approximation polynomial is large, you should check whether or not the output results are appropriate. The fitting rate, which is defined by the following equation, is one characteristic for checking whether or not the

output results are appropriate. The fitting rate can be calculated by using the following program (double precision example, main portion only).

ZS $\cdots \parallel \mathrm{f_k} \parallel_2 = \sqrt{\sum_{\mathrm{k=1}}^{\mathrm{N}} \mathrm{Z(K)}^2}$ (Real; Size: 1)

RS $\cdots \mid x \mid = \sqrt{\sum_{\mathrm{k=1}}^{\mathrm{N}} (\mathrm{F(K)} - \mathrm{Z(K)})^2}$ (Real; Size: 1) DF $\cdots$ Work variable (Real; Size: 1)

FIT $\cdots$ fitting Fitting rate $= \dfrac{\parallel f_k \parallel_2}{\mid x \mid + \parallel f_k \parallel_2} \times 100$ (Real; Size: 1)

The fitting rate can be calculated by using the follow program (double precision example, main portion only).

```
      ZS = 0.0D0
      RS = 0.0D0
      DO  10  K = 1, N
          ZS = ZS + Z(K) * Z(K)
          DF = F(K) - Z(K)
          RS = RS + DF * DF
10    CONTINUE
      ZS = SQRT(ZS)
      RS = SQRT(RS)
      FIT = 1.0D2
      IF((ZS.NE.0.0D0).OR.(RS.NE.0.0D0))  THEN
          FIT = ZS/(ZS + RS) * 1.0D2
      END  IF
```

If the fitting rate is low, use this subroutine after subdividing the approximation interval and shifting or scaling the input data to the vicinity of the origin.

(d) In the following cases, the approximation polynomial coefficient values A may differ depending on the operating system or whether the calculations are single precision or double precision.

- The degree is high
- The input data values oscillate severely

(e) Method of using the polynomial coefficients A to obtain an approximate value at a point other than the input data points. For:

XL $\cdots$ X coordinate value of approximation point	(Real; Size: 1)
YL $\cdots$ Y coordinate value of approximation point	(Real; Size: 1)
FL $\cdots$ Approximate value	(Real; Size: 1)
S $\cdots$ Work	(Real; Size: 1)
W $\cdots$ Work	(Real; Size: M+1)
ID $\cdots$ Work index of A	(Integer; Size: 1)
I, J $\cdots$ DO loop variables	(Integer; Size: 1)

the main portion of the program is as follows (double precision example):

```
      FL = A(1)
      IF(M.EQ.0)  GO  TO  999
      S = 1.0D0
```

```
      W(1) = 1.0D0
      DO  10  I = 2, M + 1
          S = S * XL
          W(I) = S
          FL = FL + S * A(I)
   10 CONTINUE
          DO  30  J = 2, M + 1
              DO  20  I = 1, M + 2 - J
                  W(I) = W(I) * YL
                  ID = I + (J - 1) * (2 × (M + 1) - J + 2)/2
                  FL = FL + W(I) * A(ID)
   20     CONTINUE
   30 CONTINUE
  999 CONTINUE
```

(7) **Example**

(a) Problem

Given the following input data:

k	x_k	y_k	z_k
1	6.95	−0.48	48.24
2	2.44	9.70	−17.57
3	0.89	−0.70	0.67
4	7.27	−7.51	38.75
5	−7.36	−1.18	53.82
6	−0.07	−4.72	−5.56
7	4.55	−5.84	12.18
8	−1.26	7.45	−12.29

obtain the approximation polynomial coefficients and approximate values at the input points.

(b) Input data

X, Y, Z, N = 8 and M = 2.

(c) Main program

```
      PROGRAM BNRAPL
! *** EXAMPLE OF DNRAPL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=8)
      DIMENSION   X(N),Y(N),Z(N),A(6),F(N),IW(6),WK(64)
      READ(5,*) (X(K),K=1,N)
      READ(5,*) (Y(K),K=1,N)
      READ(5,*) (Z(K),K=1,N)
      READ(5,*) M
      WRITE(6,1000) N,M
      WRITE(6,1100)
      DO 10 K=1,N
         WRITE(6,1200) X(K),Y(K),Z(K)
   10 CONTINUE
      CALL DNRAPL(X,Y,Z,N,M,A,F,IW,WK,IERR)
      WRITE(6,1300) IERR
      IF(IERR.GT.1000) GO TO 9999
      WRITE(6,1400)
      DO 20 K=1,(M+1)*(M+2)/2
         WRITE(6,1500) K,A(K)
   20 CONTINUE
      WRITE(6,1600)
      DO 30 K=1,N
         WRITE(6,1700) K,F(K)
   30 CONTINUE
 9999 STOP
```

```
! ***** FORMAT *****
 1000 FORMAT(' ',/,5X,'*** DNRAPL ***',/,/,6X,'** INPUT **',&
             /,/,8X,'N =',I3,/,/,8X,'M =',I3)
 1100 FORMAT(' ',/,/,T18,'X',T37,'Y',T56,'Z',/)
 1200 FORMAT(9X,3(D17.10,2X))
 1300 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR =',I4)
 1400 FORMAT(' ',/,/,8X,'( COEFFICIENT OF POLYNOMIAL FOR X,Y )',/)
 1500 FORMAT(9X,'A(',I2,') =',D17.10)
 1600 FORMAT(' ',/,/,8X,'( APPROXIMATE VALUE OF Z )',/)
 1700 FORMAT(9X,' F(',I2,') = ',D17.10)
      END
```

(d) Output results

```
*** DNRAPL ***

 ** INPUT **

   N =  8

   M =  2


            X                 Y                 Z

     0.6950000000D+01  -0.4800000000D+00   0.4824000000D+02
     0.2440000000D+01   0.9700000000D+01  -0.1757000000D+02
     0.8900000000D+00  -0.7000000000D+00   0.6700000000D+00
     0.7270000000D+01  -0.7510000000D+01   0.3875000000D+02
    -0.7360000000D+01  -0.1180000000D+01   0.5382000000D+02
    -0.7000000000D-01  -0.4720000000D+01  -0.5560000000D+01
     0.4550000000D+01  -0.5840000000D+01   0.1218000000D+02
    -0.1260000000D+01   0.7450000000D+01  -0.1229000000D+02


 ** OUTPUT **

   IERR =   0


   ( COEFFICIENT OF POLYNOMIAL FOR X,Y )

    A( 1) = 0.1041814661D-02
    A( 2) =-0.1831308069D-03
    A( 3) = 0.9998981083D+00
    A( 4) =-0.6891729227D-03
    A( 5) = 0.6835795168D-04
    A( 6) =-0.2499564336D+00


   ( APPROXIMATE VALUE OF Z )

     F( 1) =  0.4823986023D+02
     F( 2) = -0.1756987956D+02
     F( 3) =  0.6708593014D+00
     F( 4) =  0.3875110083D+02
     F( 5) =  0.5381983778D+02
     F( 6) = -0.5559399793D+01
     F( 7) =  0.1217789338D+02
     F( 8) = -0.1229027217D+02
```

5.5.2 DNGAPL, RNGAPL
Two-Dimensional Lattice Data Least Squares Approximation Polynomial

(1) **Function**

If all Z coordinate values $z_{i,j}$ on two-dimensional lattice points (x_i, y_j) $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ are given, DNGAPL or RNGAPL obtains the coefficients $a_{i,j}$ and the approximate values $f(x_i, y_j)$ $(i = 1, \cdots, nx;\ j = 1, \cdots, ny)$ at the input lattice points of the approximation polynomial for x and y:

$$f(x,y) = \sum_{i=1}^{ix+1} \sum_{j=1}^{iy+1} a_{i,j} x^{i-1} y^{j-1}$$

so that the sum of the squares of the differences of the polynomial values $f(x_i, y_j)$ and the Z coordinate values $z_{i,j}$ is minimized.

(2) **Usage**

Double precision:

```
CALL DNGAPL  (X, NX, Y, NY, Z, IX, IY, A, F, WK, IERR)
```

Single precision:

```
CALL RNGAPL  (X, NX, Y, NY, Z, IX, IY, A, F, 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	X	{D / R}	NX	Input	X coordinates x_i of data points
2	NX	I	1	Input	Dimension nx of array X
3	Y	{D / R}	NY	Input	Y coordinates y_i of data points
4	NY	I	1	Input	Dimension ny of array Y
5	Z	{D / R}	NX,NY	Input	Z coordinates $z_{i,j}$ at data points (x_i, y_j)
6	IX	I	1	Input	Highest degree ix for x of the approximation polynomial
7	IY	I	1	Input	Highest degree iy for y of the approximation polynomial
8	A	{D / R}	See Contents	Output	Coefficients $a_{i,j}$ of approximation polynomial for x, y **Size**: $(\mathrm{IX}+1)\times(\mathrm{IY}+1)$
9	F	{D / R}	NX,NY	Output	Approximate value $f(x_i, y_j)$ of Z coordinate at (x_i, y_j)
10	WK	{D / R}	See Contents	Work	Work area **Size**: $9\times(\mathrm{NX}+2\times\mathrm{NY})$
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NX} > 2, \mathrm{NY} > 2$

(b) $\mathrm{X(i)} \neq \mathrm{X(j)} \quad (\mathrm{i} \neq \mathrm{j})$
$\mathrm{Y(i)} \neq \mathrm{Y(j)} \quad (\mathrm{i} \neq \mathrm{j})$

(c) $0 < \mathrm{IX} \leq \min\,(8, \mathrm{NX}-1)$
$0 < \mathrm{IY} \leq \min\,(8, \mathrm{NY}-1)$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	

(6) **Notes**

(a) When Z coordinate values corresponding to two-dimensional lattice points (x_i, y_j) are $z_{i,j}$ $(i = 1, \cdots, \mathrm{nx};\ j = 1, \cdots, \mathrm{ny})$, $z_{i,j}$ are stored in array Z as follows.
Z(i, j) = $\mathrm{z_{i,j}}$

(b) The coefficients $a_{i,j}$ are stored in array A in the order
$a_{1,1}, \cdots, a_{1,iy+1}, a_{2,1}, \cdots, a_{2,iy+1}, \cdots, a_{ix+1,1}, \cdots, a_{ix+1,iy+1}$.

(c) Method of using the polynomial coefficients A to obtain an approximate value at a point other than the input data points.

XL	···X coordinate value of approximation point	(Real; Size: 1)
YL	···Y coordinate value of approximation point	(Real; Size: 1)
FL	···Approximate value	(Real; Size: 1)
S	···Work	(Real; Size: 1)
ID	···Work index of A	(Integer; Size: 1)
IE	···Work index of A	(Integer; Size: 1)
K, L	···DO loop variables	(Integer; Size: 1)

the main portion of the program is as follows:

```
      ID = (IX + 1) × (IY + 1)
      FL = A(ID)
      DO   10   L = 1, IY
           FL = FL * YL + A(ID − L)
10    CONTINUE
      DO   30   K = IX, 1, −1
           IE = K * (IY + 1)
           S = A(IE)
           DO   20   L = 1, IY
                S = S * YL + A(IE − L)
20         CONTINUE
           FL = FL * XL + S
30    CONTINUE
```

(d) Since this subroutine approximates the given points on the surface by only one polynomial, meaningless results may be produced depending on the data. In particular, if the degree of the approximation polynomial is large, you should check whether or not the output results are appropriate. The fitting rate (See Section 5.5.1) is one characteristic for checking whether or not the output results are appropriate. If the fitting rate is low, use this subroutine after subdividing the approximation interval and shifting or scaling the input data to the vicinity of the origin.

(7) **Example**

(a) Problem

Given the following lattice coordinates and corresponding z coordinate values as input data:

$x_1 = -3.0 \quad y_1 = -2.0$
$x_2 = -1.0 \quad y_2 = -1.0$
$x_3 = 1.0 \quad y_3 = 0.0$
$x_4 = 3.0 \quad y_4 = 1.0$
$\quad\quad\quad\quad y_5 = 2.0$

		j →				
	$z_{i,j}$	y_1	y_2	y_3	y_4	y_5
	x_1	28.0	29.5	27.0	20.5	10.0
$i\downarrow$	x_2	−2.0	2.5	3.0	−0.5	−8.0
	x_3	−8.0	−0.5	3.0	2.5	−2.0
	x_4	10.0	20.5	27.0	29.5	28.0

obtain the approximation polynomial coefficients and approximate values at the input lattice points.

(b) Input data

X, NX=4, Y, NY=5, Z, IX=2 and IY=2.

(c) Main program

```
      PROGRAM BNGAPL
! *** EXAMPLE OF DNGAPL ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (NX=4,NY=5)
      DIMENSION   X(NX),Y(NY),Z(NX,NY),A(9),F(NX,NY),WK(126)
      READ(5,*) (X(I),I=1,NX)
      READ(5,*) (Y(J),J=1,NY)
      READ(5,*) ((Z(I,J),J=1,NY),I=1,NX)
      READ(5,*) IX,IY
      WRITE(6,1000) NX,NY,IX,IY,(X(I),I=1,NX),(Y(J),J=1,NY)
      WRITE(6,1100)
      DO 10 I=1,NX
         WRITE(6,1200) (Z(I,J),J=1,NY)
   10 CONTINUE
      CALL DNGAPL(X,NX,Y,NY,Z,IX,IY,A,F,WK,IERR)
      WRITE(6,1300) IERR
      IF(IERR.NE.0) GO TO 9999
      WRITE(6,1400)
      LA = (IX+1)*(IY+1)
      DO 20 K=1,LA
         WRITE(6,1500) K,A(K)
   20 CONTINUE
      WRITE(6,1600)
      DO 30 I=1,NX
         WRITE(6,1200) (Z(I,J),J=1,NY)
   30 CONTINUE
 9999 STOP
! ***** FORMAT *****
 1000 FORMAT(' ',/,5X,'*** DNGAPL ***',/,/,6X,'** INPUT **',&
             /,/,8X,'NX =',I3,/,/,8X,'NY  =',I3,&
             /,/,8X,'IX  =',I3,/,/,8X,'IY  =',I3,/,/,6X,'COORDINATES',&
             /,/,9X,'X  =',4(2X,F9.5),/,/,9X,'Y  =',5(2X,F9.5))
 1100 FORMAT(' ',/,/,6X,'Z(X,Y) =')
 1200 FORMAT(' ',/,/,8X,5(2X,F9.5))
 1300 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR = ',I4)
 1400 FORMAT(' ',/,/,8X,'( COEFFICIENT OF POLYNOMIAL FOR X,Y )')
 1500 FORMAT(' ',/,11X,'A(',I2,') =',D17.10)
 1600 FORMAT(' ',/,/,6X,'F(X,Y) =')
      END
```

(d) Output results

```
 *** DNGAPL ***

  ** INPUT **

    NX =  4

    NY  =  5

    IX  =  2

    IY  =  2
```

```
COORDINATES
   X  =   -3.00000   -1.00000    1.00000    3.00000
   Y  =   -2.00000   -1.00000    0.00000    1.00000    2.00000

Z(X,Y) =

     28.00000   29.50000   27.00000   20.50000   10.00000

     -2.00000    2.50000    3.00000   -0.50000   -8.00000

     -8.00000   -0.50000    3.00000    2.50000   -2.00000

     10.00000   20.50000   27.00000   29.50000   28.00000

** OUTPUT **
  IERR =    0

  ( COEFFICIENT OF POLYNOMIAL FOR X,Y )
     A( 1) = 0.0000000000D+00
     A( 2) = 0.0000000000D+00
     A( 3) =-0.2000000000D+01
     A( 4) = 0.0000000000D+00
     A( 5) = 0.1500000000D+01
     A( 6) = 0.0000000000D+00
     A( 7) = 0.3000000000D+01
     A( 8) = 0.0000000000D+00
     A( 9) = 0.0000000000D+00

F(X,Y) =

     28.00000   29.50000   27.00000   20.50000   10.00000

     -2.00000    2.50000    3.00000   -0.50000   -8.00000

     -8.00000   -0.50000    3.00000    2.50000   -2.00000

     10.00000   20.50000   27.00000   29.50000   28.00000
```

5.6 CHEBYSHEV APPROXIMATION

5.6.1 DNCBPO, RNCBPO
Chebyshev Approximation

(1) **Function**

DNCBPO or RNCBPO obtains the Chebyshev coefficients c_j, which satisfy $F(x) \sim [\sum_{j=0}^{\mathrm{N}} \mathrm{C}_j \mathrm{T}_j(x)] - \frac{1}{2}\mathrm{C}_0$, by computing the Chebyshev approximation of a function $f(x)$ defined on the finite interval $[a, b]$ (when ISW=0). Also, this subroutine obtains the polynomial coefficients y_k for which $f(x) \sim \sum_{k=0}^{m} y_k x^k$ (when ISW=1).

The coefficients c_j are defined as $\mathrm{C}_j = \frac{\mathrm{N}}{2} \sum_{k=1}^{\mathrm{N}} [F[\cos(\pi(k - \frac{1}{2})/\mathrm{N})) \cos(\pi j(k - \frac{1}{2})/\mathrm{N})]$.

(2) **Usage**

Double precision:

```
CALL DNCBPO  (F, A, B, N, CEPS, AEPS, C, NC, ISW, WK, IERR)
```

Single precision:

```
CALL RNCBPO  (F, A, B, N, CEPS, AEPS, C, NC, ISW, 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	F	{D / R}	−	Input	Function f to be Chebyshev approximated (See Note (a))
2	A	{D / R}	1	Input	Lower end of approximation range a
3	B	{D / R}	1	Input	Upper end of approximation range b
4	N	I	1	Input	Required degree n of Chebyshev coefficients
5	CEPS	{D / R}	1	Input	Chebyshev coefficient truncation error (See Note (c))
6	AEPS	{D / R}	1	Input	Admissible constant (See Note (b))
				Output	Estimating residual (See Note (b))
7	C	{D / R}	0: N	Output	Chebyshev coefficients $C_j(j = 0, 1, \cdots, n)$ or polynomial coefficients $y_k(k$=0, 1, $\cdots, m)$
8	NC	I	1	Output	Truncation degree m (See Note (c))
9	ISW	I	1	Input	ISW=0: Obtain only Chebyshev coefficients ISW=1: Obtain Chebyshev coefficients and compute polynomial approximation according to those coefficients (See Note (c))
10	WK	{D / R}	2∗(N+1)	Work	Work area
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) A < B

(b) N ≥ 0

(c) ISW = 0 or ISW=1

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Restriction (a) was not satisfied.	When A=B: C (0) =F(A) and C (i) =0.0 (i = 1, ⋯, N) are returned. When A > B: A and B are switched and processing continues.
3000	Restriction (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3500	The error between the function and approximation polynomial is greater than the admissible constant AEPS.	The error is output and the result that was obtained is returned.

(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 should be created as follows:

```
REAL(8) FUNCTION F (X)
REAL(8) X
    F = f(X)
RETURN
END
```

(b) The admissible constant is the maximum error predicted between the function $f(x)$ and the Chebyshev approximated function. The estimating residual is the maximum error estimated between the approximation function and original function $f(x)(\mathrm{A} \leq x \leq \mathrm{B})$ when the interval $[\mathrm{A}, \mathrm{B}]$ is subdivided.

(c) The term "truncate" is used here with the following meaning. In order to compute a Chebyshev approximation optimally, the Chebyshev coefficients that were obtained are evaluated to obtain the degree NC(= m), which is the limit of the coefficients that cannot be ignored, and coefficients for degrees NC+1 and above are discarded.
The truncation error is the systematic error that always occurs when the formula containing the series is truncated at a specific term or according to a predetermined method.
The series is truncated only when the residual to be estimated is restrained by a certain predetermined admissible constant, and the more random error that occurs at that time is considered as part of the rounding error.

(7) **Example**

(a) Problem
Given the function $f(x) = x^2(x^2 - 2)\sin(x)$ on the interval $[-1, 1]$, obtain the coefficients of the Chebyshev approximation polynomial.

(b) Input data
Function: DNCHEV

Lower end of approximation range: A= − 1.0

Upper end of approximation range: B= 1.0

Required degree of Chebyshev coefficients: N=10

Chebyshev coefficient truncation error: CEPS=0.01

Admissible constant: AEPS=0.0001

Whether or not to compute polynomial approximation according to Chebyshev coefficients: ISW=1

(c) Main program

```
      PROGRAM BNCBPO
! *** EXAMPLE OF DNCBPO ***
      IMPLICIT NONE
      INTEGER ISW,IERR,N,NC,I
      REAL(8) CEPS,AEPS
      CHARACTER F*40
      PARAMETER (N=10,ISW=1)
      REAL(8) DNCHEV,A,B,C(0:N),WK(2*(N+1))
      EXTERNAL DNCHEV
      DATA F/'X**2*(X**2-2*X)*SIN(X)'/
!
      WRITE(6,1000)
      READ(5,*) A,B
      READ(5,*) CEPS,AEPS
      WRITE(6,1100) F,A,B,N,CEPS,AEPS,ISW
      CALL DNCBPO(DNCHEV,A,B,N,CEPS,AEPS,C,NC,ISW,WK,IERR)
      WRITE(6,1200) IERR
      WRITE(6,1300) NC
      WRITE(6,1400) AEPS
      WRITE(6,1500) (I,C(I),I=0,NC)
!
 1000 FORMAT(' ',/,' *** DNCBPO ***')
 1100 FORMAT(' ',/,/,6X,'FUNCTION = ',A40,&
      /,/,6X,'** INPUT **',/,/,8X,'A      = ',D18.10,/,/,8X,'B      = ',D18.10,&
      /,/,8X,'N      =   ',I2,/,/,8X,'CEPS   = ',D18.10,&
      /,/,8X,'AEPS   = ',D18.10,/,/,8X,'ISW    =   ',I1)
 1200 FORMAT(' ',/,/,6X,'** OUTPUT **',/,/,8X,'IERR     = ',I4)
 1300 FORMAT(' ',/,/,8X,'NC       =   ',I2)
 1400 FORMAT(' ',/,/,8X,'AEPS     =  ',D18.10)
 1500 FORMAT(' ',/,/,6X,'*** COEFFICIENTS OF POLYNOMIAL *** ',/,' ',&
             (/,9X,'C(',I2,') = ',D18.10))
!
      STOP
      END
      REAL(8) FUNCTION DNCHEV(X)
      REAL(8) X
      DNCHEV=X**2*(X**2-2)*SIN(X)
      RETURN
      END
```

(d) Output results

```
*** DNCBPO ***

     FUNCTION = X**2*(X**2-2*X)*SIN(X)
     ** INPUT **
       A      =  -0.1000000000D+01
       B      =   0.1000000000D+01
       N      =   10
       CEPS   =   0.1000000000D-03
       AEPS   =   0.1000000000D-01
       ISW    =   1

     ** OUTPUT **
       IERR     =    0

       NC       =    7

       AEPS     =    0.3190389782D-04

     *** COEFFICIENTS OF POLYNOMIAL ***
        C( 0) =  -0.6591949209D-16
        C( 1) =  -0.2897819470D-03
```

```
C( 2) =   0.6453171331D-15
C( 3) =  -0.1996122151D+01
C( 4) =  -0.1776356839D-14
C( 5) =   0.1319288235D+01
C( 6) =   0.1332267630D-14
C( 7) =  -0.1643790612D+00
```

Chapter 6

SPLINE FUNCTIONS

6.1 INTRODUCTION

This chapter describes subroutines that perform interpolations or approximations of given data points using spline functions and that compute the derivatives and integrals of obtained spline function.
This library provides functions which perform interpolations or approximations of given data points using cubic spline functions (one-variable function), bicubic spline functions (two-variable function) and B–spline functions. In addition, an error detection and correction subroutine is provided for when your input data is contaminated with isolated incongruous data values.
Normally, subroutines for a spline function are designed to obtain interpolation values, integrals or derivatives of spline function after obtaining spline coefficients. When obtaining a spline function, the following three category is set for distinguishing interpolation or approximation level.

(1) Interpolation
Create a spline function that passes through all given data points. In this case, all knots of a spline function is equal to given data points. This library provides several functions corresponding to methods for endpoints conditions decision.

(2) Smoothed interpolation
Create a spline function whose all knots abscissa values is equal to that of given data points. Obtained spline function may not passes through all given data points. It is not suitable for interpolation function for given data points, but smoother approximation function is obtained than one for simple interpolation. This method is effective when you want to obtain an approximation by revising all data points using the same compensation rate. Two methods are available. You can either enter a control variable corresponding to the compensation rate or automatically obtain the optimum compensation curve by a statistical procedure.

(3) Least squares interpolation
Create a spline function whose all knot positions are adjustable parameter. In this case, spline coefficients are obtained by applying least squares approximation in each interval between knots. More suitable spline function is obtained for some purpose than one for smoothed interpolation. Especially, when there are an extremely large number of data points, the least squares approximate spline function is effective. There are two methods of determining knot positions. You can specify them as input parameters or have them found automatically so that the total least squares error is minimized.

6.1.1 Notes

(1) There are innumerable functions which approximate (or interpolate) given data points. Usually obtained values from interpolations or approximations differ what kind of approximation function is chosen. What kind of approximation function is prefer depend on your purpose. A spline function is especially effective approximation function when you approximate or interpolate data points by smooth approximation function. There are several kind of spline functions and you must use proper one.

(2) The subroutines having names that begin [L]D/R store the spline coefficients (1st through 3rd order terms) in argument C as shown below:

Storage status within array C(3, NX−1)

$$\begin{array}{|ccccc|} \hat{C}_{1,1} & \hat{C}_{1,2} & \hat{C}_{1,3} & \cdots & \hat{C}_{1,\mathrm{NX}-1} \\ \hat{C}_{2,1} & \hat{C}_{2,2} & \hat{C}_{2,3} & \cdots & \hat{C}_{2,\mathrm{NX}-1} \\ \hat{C}_{3,1} & \hat{C}_{3,2} & \hat{C}_{3,3} & \cdots & \hat{C}_{3,\mathrm{NX}-1} \end{array} \quad \updownarrow 3$$

$\leftarrow$ − − − − − − − −NX−1− − − − − − − − $\rightarrow$

Remarks

a. $\hat{C}_{1,j}$, $\hat{C}_{2,j}$ and $\hat{C}_{3,j}$ $(j = 1, 2, \cdots, \mathrm{NX}-1)$: the spline coefficients

However, for the subroutines having names that begin [L]W/V the spline coefficients are stored as follows:

Storage status within array C(NX, 3)

$$\begin{array}{|ccc|} C_{1,1} & C_{1,2} & C_{1,2} \\ C_{2,2} & C_{2,2} & C_{2,3} \\ \vdots & \vdots & \vdots \\ C_{\mathrm{NX}-1,1} & C_{\mathrm{NX}-1,2} & C_{\mathrm{NX}-1,3} \\ 0.0 & 0.0 & 0.0 \end{array} \quad \updownarrow \mathrm{NX}$$

$\leftarrow$ − − − − − − − 3 − − − − − − − $\rightarrow$

Remarks

a. $C_{i,1}$, $C_{i,2}$ and $C_{i,3}$ $(i = 1, 2, \cdots, \mathrm{NX}-1)$: the spline coefficients; $C_{i,k} = \hat{C}_{k,i}$ $(k = 1, 2, 3)$

Zeros are stored in C(NX, 1), C(NX, 2), and C(NX, 3) and the actual spline coefficients are stored in C(i, j) i = 1, $\cdots$, NX − 1; j = 1, $\cdots$, 3. So, the size of the array C must be (NX, 3). This change is for solving the simultaneous linear equation to obtain the spline coefficients effectively on vector computers.

(3) Since input data may contain error data, you should call the subroutine that detects and corrects error data.

(4) Cubic spline coefficients for one-dimensional data differ according to the endpoint condition or smoothing method used. Therefore, to perform calculations using different conditions or smoothing methods, you must call the subroutine that matches those conditions.

(5) To obtain values such as interpolation values, derivative values by cubic spline interpolation using the same endpoint conditions or smoothing methods repeatedly, processing will be more efficient if you call the subroutine that obtains cubic spline coefficients once and then repeatedly call the subroutine that obtains corresponding values by cubic spline coefficients.

(6) The bicubic spline subroutine 6.3.1 $\begin{Bmatrix}\mathrm{DGISXB}\\ \mathrm{RGISXB}\end{Bmatrix}$ and 6.3.3 $\begin{Bmatrix}\mathrm{DGIIZB}\\ \mathrm{RGIIZB}\end{Bmatrix}$ do not obtain bicubic spline coefficients.

6.1.2 Algorithms Used

6.1.2.1 Cubic aperiodic spline function (inputting endpoint conditions)

A spline function is an interval-wise polynomial that connect m-th and lower order polynomials defined on each subinterval under the condition that the function values and $(m-1)$-th and lower derivatives are continuous on the entire interval. In general, the quadratic differential coefficients M_i are obtained as solutions of the following simultaneous linear equations.

$$\begin{bmatrix} 2 & \lambda_1 & & & & & \\ \mu_2 & 2 & \lambda_2 & & & 0 & \\ & \mu_3 & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \lambda_{n-2} & \\ & 0 & & & \mu_{n-1} & 2 & \lambda_{n-1} \\ & & & & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{6.1}$$

If we assume that the X coordinates of the knots in each interval are given by:

$$-\infty < \xi_1 < \xi_2 < \cdots < \xi_n < \infty$$

the function values at each knot are:

$$y_1, \cdots, y_n$$

and the distance between knots is:

$$h_i = \xi_{i+1} - \xi_i$$

then:

$$\begin{aligned} \lambda_i &= \frac{h_i}{h_{i-1}+h_i} && \text{for } (i = 2, \cdots, n-1) \\ \mu_i &= 1 - \lambda_i && \text{for } (i = 2, \cdots, n-1) \\ d_i &= \frac{6}{h_{i-1}+h_i}\left\{\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right\} && \text{for } (i = 2, \cdots, n-1) \end{aligned} \tag{6.2}$$

λ_1, μ_n, d_1 and d_n are determined from the endpoint conditions as described below.
If we let $f(x)$ represent the spline function, then:

(1) If $f'(\xi_1)$ and $f'(\xi_n)$ (slopes at the endpoints) are known:

$$\begin{aligned} &\lambda_1 = 1.0, \quad \mu_n = 1.0 \\ &d_1 = \frac{6}{h_1}\left\{\frac{y_2 - y_1}{h_1} - f'(\xi_1)\right\} \\ &d_n = \frac{6}{h_{n-1}}\left\{f'(\xi_n) - \frac{y_n - y_{n-1}}{h_{n-1}}\right\} \end{aligned} \tag{6.3}$$

(2) If $f''(\xi_1)$ and $f''(\xi_n)$ are know:

$$\begin{aligned} &\lambda_1 = \mu_n = 0.0 \\ &d_1 = 2f''(\xi_1) \\ &d_n = 2f''(\xi_n) \end{aligned} \tag{6.4}$$

If the endpoint conditions are not known, you can assume $f''(\xi_1) = f''(\xi_n) = 0.0$ and use the so-called cubic natural spline.

(3) If $f'''(\xi_1)$ and $f'''(\xi_n)$ are known:

$$\begin{aligned} &\lambda_1 = \mu_n = -2.0 \\ &d_1 = -2h_1 f'''(\xi_1) \\ &d_n = 2h_{n-1} f'''(\xi_n) \end{aligned} \tag{6.5}$$

It the endpoint conditions are not known, you can assume $f'''(\xi_1) = f'''(\xi_n) = 0.0$. This is called the P-spline.

You can solve these simultaneous linear equations as follows:

$b_i = 2$ for $(i = 1, \cdots, n)$

(Forward process)

$$\left.\begin{aligned} b_i &= b_i - \frac{\mu_i \lambda_{i-1}}{b_{i-1}} \\ d_i &= d_i - \frac{\mu_i d_{i-1}}{b_{i-1}} \end{aligned}\right\} \text{ for } (i = 2, \cdots, n)$$

$M_n = d_n - b_n$

(Backward process)

$$M_i = \frac{d_i - \lambda_i M_{i+1}}{b_i} \text{ for } (i = n-1, \cdots, 1)$$

Obtaining the spline coefficients $c_{1\sim3,i}$ from the derived M_i, we get:

$$\begin{aligned} c_{1,i} &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{6}(M_{i+1} - M_i) - \frac{h_i M_i}{2} \\ c_{2,i} &= \frac{M_i}{2} \\ c_{3,i} &= \frac{M_{i+1} - M_i}{6h_i} \end{aligned} \tag{6.6}$$

6.1.2.2 Cubic periodic spline function

The simultaneous linear equations for obtaining the quadratic differential coefficients M_i of the periodic spline function are as follows.

$$\begin{bmatrix} 2 & \lambda_2 & \cdots & \cdots & 0 & \cdots & \mu_2 \\ \mu_3 & 2 & \lambda_3 & & & 0 & \vdots \\ \vdots & \mu_4 & \ddots & \ddots & & & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & & \ddots & \ddots & \lambda_{n-2} & \vdots \\ \vdots & 0 & & & \mu_{n-1} & 2 & \lambda_{n-1} \\ \lambda_n & \cdots & 0 & \cdots & \cdots & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_2 \\ M_3 \\ \vdots \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ \vdots \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix} \tag{6.7}$$

$M_1 = M_n, y_1 = y_n$

Use the following algorithm to solve these simultaneous linear equations.

$$
\begin{aligned}
b_1 &= 2 \\
b_i &= 2 - \lambda_i \frac{\mu_{i+1}}{b_{i-1}} && \text{for } (i = 2, \cdots, n-2) \\
p_i &= -\frac{\lambda_{i+1}}{b_i} && \text{for } (i = 1, \cdots, n-2) \\
q_1 &= -\frac{\mu_2}{2} \\
q_i &= -q_{i-1} \frac{\mu_{i+1}}{b_{i-1}} && \text{for } (i = 2, \cdots, n-2) \\
r_1 &= \frac{d_2}{2} \\
r_i &= \frac{d_{i+1} - r_{i-1}\mu_{i+1}}{b_{i-1}} && \text{for } (i = 2, \cdots, n-2) \\
t_i &= p_i t_{i+1} + q_i && (t_{n-1} = 1, \text{ for } (i = n-2, \cdots, 1)) \\
v_i &= p_i v_{i+1} + r_i && (v_{n-1} = 0, \text{ for } (i = n-2, \cdots, 1)) \\
M_n &= \frac{d_n - \lambda_n v_1 - \mu_n v_{n-2}}{\lambda_n t_1 + \mu_n t_{n-2} + 2} \\
M_i &= t_{i-1} M_n + v_{i-1} && \text{for } (i = n, \cdots, 2) \\
M_1 &= M_n \\
y_1 &= y_n
\end{aligned}
$$

The procedure for obtaining the spline coefficients from the derived M_i is similar to that described for aperiodic spline functions.

6.1.2.3 Cubic aperiodic spline functions (endpoint condition input is unnecessary)

In this case, use the algorithm that uses the first differential coefficients $m_i = f'(\xi_i)$. The simultaneous linear equations for obtaining m_i are as follows.

$$
\begin{bmatrix}
2 & \mu_1 & & & & & \\
\lambda_2 & 2 & \mu_2 & & & 0 & \\
 & \lambda_3 & \ddots & \ddots & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & \ddots & \ddots & \mu_{n-3} & \\
 & 0 & & & \lambda_{n-2} & 2 & \mu_{n-2} \\
 & & & & & \lambda_{n-1} & 2
\end{bmatrix}
\begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix}
=
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix}
\tag{6.8}
$$

λ and μ are similar to the case when endpoint conditions are input.

$$
c_i = 3 \left\{ \lambda_i \frac{(y_i - y_{i-1})}{h_{i-1}} + \mu_i \frac{(y_{i+1} - y_i)}{h_i} \right\} \quad \text{for } (i = 2, \cdots, n-2) \tag{6.9}
$$

Assume "not-a-knot" endpoint conditions. That is, assume that the third derivative $f'''(x)$ is continuous at x_1 through x_3 and x_{n-2} through x_n and the first and second derivatives are continuous at x_3 and x_{n-2}. The equations at the endpoints then become:

$$
\begin{aligned}
\mu_1 &= \frac{2(h_1 + h_2)}{h_2} \\
\lambda_{n-1} &= \frac{2h_{n-1}}{h_{n-1} + h_{n-2}} \\
c_1 &= \frac{2[(h_1 + 2(h_1 + h_2))(y_2 - y_1)/h_1 + {h_1}^2 (y_3 - y_2)/{h_2}^2]}{h_1 + h_2} \\
c_{n-1} &= \frac{2[{h_{n-2}}^2 (y_n - y_{n-1})/h_{n-1} + h_{n-1}(3h_{n-2} + 2h_{n-1})(y_{n-1} - y_{n-2})/h_{n-2}]}{(h_{n-1} + h_{n-2})^2}
\end{aligned}
\tag{6.10}
$$

Obtaining the spline coefficients $c_{1\sim3,i}$ from the derived m_i, we get:

$$\begin{cases} c_{1,i} = m_i \\ c_{2,i} = \dfrac{y_{i+1}-y_i}{{h_i}^2} - \dfrac{m_i}{h_i} - c_{3,i}h_i \\ c_{3,i} = \dfrac{m_{i+1}+m_i}{{h_1}^2} - 2\dfrac{y_{i+1}-y_i}{{h_i}^3} \end{cases} \tag{6.11}$$

6.1.2.4 Cubic spline smoothing by specifying a control variable

(See Reference Bibliography (5))

Approximate the data set by a smooth curve on which the data points are knots and consider a natural spline ($f''(\xi_1) = f''(\xi_n) = 0$) that minimizes the following function:

$$\begin{aligned} S_i &= \int_{\xi_1}^{\xi_n} (f''(x))^2 dx \\ S_m &= \sum_{i=1}^{n} \left(\frac{f(\xi_1) - y_i}{\delta y_i} \right)^2 \\ & S_i + pS_m \to \min \end{aligned} \tag{6.12}$$

where let S_f, which is the value that enables S_m to be obtained instead of specifying p, be the value of the control variable that is input. By varying the value of p under the condition of equation (6.12), determine the value of p such that $S_m = S_f$ be fulfilled.

Let δy_i be the estimated value of the deviation from y_i and choose values of S_f in the range:

$$N - \sqrt{2N} \le S_f \le N + \sqrt{2N}$$

N is the number of the data.

Let the components $q_{i,j}$ of the tridiagonal matrix Q be:

$$q_{i-1,i} = \frac{1}{h_{i-1}}, \quad q_{i,i} = -\frac{1}{h_{i-1}} - \frac{1}{h_i}, \quad q_{i+1,i} = \frac{1}{h_i} \tag{6.13}$$

the components $t_{i,j}$ of the tridiagonal matrix T be:

$$t_{i,j} = \frac{2}{3}(h_{i-1} + h_i), \quad t_{i,i+1} = t_{i+1,i} = \frac{h_i}{3} \tag{6.14}$$

and the components of the diagonal matrix D be:

$$d_i = \delta y_i \tag{6.15}$$

Using these notations, the algorithm for obtaining the spline coefficients is as follows.

① Start from $\bar{p} = 0$.

② Obtain the tridiagonal matrix $Q^T DQ$, the matrix $Q^T \boldsymbol{y}$, and the tridiagonal matrix T. ($\boldsymbol{y}$ is a vector having y_i as its component)

③ Perform the Cholesky decomposition $\bar{p}Q^T D^2 Q + T = R^T R$ and obtain the solution $\boldsymbol{u}$ by solving the simultaneous linear equation $R^T R\boldsymbol{u} = Q^T \boldsymbol{y}$ using forward substitution and back substitution.

④ Let $\boldsymbol{v} = \dfrac{\boldsymbol{y} - \boldsymbol{s}}{\bar{p}} = D^2 Q\boldsymbol{u}$ and obtain $\boldsymbol{v}$. ($\boldsymbol{s}$ is a vector having spline function value $f(\xi_i)$ at each knot.)
Obtain:

$$e = \boldsymbol{v}Q\boldsymbol{u} = \frac{(\boldsymbol{y} - \boldsymbol{s})^2}{\bar{p}D^2}, \quad S'_m = \bar{p}^2 e = \sum \left(\frac{f(\xi_i) - y_i}{\delta y_i} \right)^2$$

⑤ If $S'_m > S_f$, obtain a new $\bar{p}$ by the processing described below and then return to ③. Use Cholesky forward substitution to obtain the solution $\boldsymbol{g}$ of $R^T\boldsymbol{g} = Q^T D^2 Q\boldsymbol{u}$.

$$\begin{aligned} f &= \boldsymbol{g}^T\boldsymbol{g} \\ h &= e - \bar{p}f \\ \bar{p} &= \bar{p} + \frac{S_f - S'_m}{(\sqrt{S_f/e} + \bar{p})h} \end{aligned}$$

⑥ If $S'_m \leq S_f$, then end by calculating the spline coefficients as described below.

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{y} - \bar{p}\boldsymbol{v} \\ c_{2,i} &= u_i \\ h_i &= x_{i+1} - x_i \\ c_{3,i} &= \frac{u_{i+1} - u_i}{3h_i} \\ c_{1,i} &= \frac{f_{i+1} - f_i}{h_i} - c_{2,i}h_i - c_{3,i}{h_1}^2 \end{aligned}$$

6.1.2.5 Cubic spline automatic smoothing

(See Reference Bibliography (4))
To obtain the best approximation curve for the data set, perform the smoothing using the algorithm described below to obtain the value p (See preceding section) that minimizes the cross-validation function.

① Obtain $T^{1/2}$ as follows. (See equation (6.14) for matrix T)

$$\begin{aligned} T^{1/2} &= UXU^T \quad \text{(Singular-value decomposition)} \\ &\quad \left(\begin{array}{l} U : \text{Matrix consisting of eigenvectors of } T. \\ X : \text{Diagonal matrix having square roots of the eigenvalues} \\ \quad (\lambda_i) \text{ of } T \text{ as diagonal elements.} \end{array} \right) \\ T^{-1/2} &= UEU^T \\ &\quad (E : \text{Diagonal matrix having } \frac{1}{\sqrt{\lambda_i}} \text{ as diagonal components.}) \end{aligned}$$

② Obtain F from $F = QT^{-1/2}$. (See equation (6.13) for matrix Q)

③ Obtain $F = UWV^T$ by performing a singular-value decomposition of F. If $\boldsymbol{s}$ (function value at knot) $= A\boldsymbol{y}$, then:

$$\begin{aligned} I - A &= Q(Q^TQ + pT)^{-1}Q^T \\ &= F(F^TF + pI)^{-1}F^T \quad \text{(Assume Weight} = 1.) \\ I - A &= U \begin{pmatrix} \frac{{d_1}^2}{{d_1}^2 + p} & & 0 \\ & \ddots & \\ 0 & & \frac{{d_{n-2}}^2}{{d_{n-2}}^2 + p} \end{pmatrix} U^T \quad (d_i : \text{Diagonal element of } W) \end{aligned}$$

④ Let the approximation function $V(\hat{p})$ of the cross validation function be:

$$V(\hat{p}) = \frac{1}{n}\sum_{j=1}^{n-2}\left(\frac{{d_j}^2}{{d_j}^2+p}\right){z_j}^2 \Big/ \left[\frac{1}{n}\sum_{j=1}^{n-2}\left(\frac{{d_j}^2}{{d_j}^2+p}\right)\right]^2$$

where $\boldsymbol{z} = U^T\boldsymbol{y}$

Obtain the minimum value of $V(\hat{p})$ by using the Davidon's method to search for extremum values. Calculate the spline coefficients by obtaining the quadratic differential coefficients and the function values at knots from the value of $\hat{p}$ at the point where the minimum occurs. (See preceding section)
In the Davidon's method of searching for extremum values, first determine two points between which minimum point exists. Then interpolate the function by cubic polynomial using the function values and its derivatives at these points and obtain minimum value.

6.1.2.6 Cubic spline coefficients (least squares method with specification of knot locations)

(See Reference Bibliography (6))
Obtain the spline function by specifying interpolation intervals according to knots (ξ_i) so that the least squares error between the input data (y_i) and spline function values $f(x_i)$ is minimized.
Let $S = \sqrt{\sum_{i=1}^{n} w_i(f(x_i)-y_i)^2}$ be the least squares error, where the weight w_i at each data point has the value as follows.

$$\begin{aligned} w_1 &= \frac{x_2 - x_1}{x_n - x_1} \\ w_i &= \frac{x_{i+1} - x_{i-1}}{x_n - x_1} \quad \text{for } (i = 2, \cdots, n-1) \\ w_n &= \frac{x_n - x_{n-1}}{x_n - x_1} \end{aligned} \tag{6.16}$$

Assume the spline function is represented by the concurrently orthonormal base functions $\{\Psi_i\}_{i=1}^m$.

$$\begin{aligned} &< \Psi_i, \Psi_j >= \delta_{ij}, \quad \text{for } (i, j = 1, \cdots, m) \\ &(\delta \text{ indicates the Kronecker delta and } <> \text{ indicates the inner product.}) \end{aligned} \tag{6.17}$$

The function value u is $u = \sum_{i=1}^{m} < u, \Psi_i > \Psi_i$ where $< u, \Psi_i >$ is the coefficient of Ψ_i.
Consider the cubic spline base $\{\Phi_i\}_{i=1}^m$ and use the modified Gram-Schmidt orthogonalization method to obtain the orthonormal polynomial base from them. This method is expressed as follows.

$$\left.\begin{aligned} {\Phi_i}^{(1)} &= \Phi_i \\ {\Phi_i}^{(j+1)} &= {\Phi_i}^{(j)} - < {\Phi_i}^{(j)}, \Psi_j > \Psi_j \quad \text{for } (j = 1, \cdots, i-1) \\ \Psi_i &= \frac{{\Phi_i}^{(i)}}{\|{\Phi_i}^{(i)}\|} \quad (\|{\Phi_i}^{(i)}\| = (< {\Phi_i}^{(i)}, {\Phi_i}^{(i)} >)^{1/2}) \end{aligned}\right\} \text{ for } (i = 1, \cdots, m) \tag{6.18}$$

If orthogonal polynomials are created as described above, then the approximation function for the added knots should be updated by adding the new orthogonal polynomial base and their coefficients. To obtain the function values and first differential coefficients at the added knots, values obtained from the new polynomial components should be added to the values before the knots were added.
This algorithm is described below divided into individual processing steps.

① Calculate weights.

② Use orthogonal polynomials to obtain the cubic least squares approximation polynomial on the single interval from the first knot ξ_0 to the last knot ξ_m. Let this bases be Ψ_1 through Ψ_4. In addition, obtain the least squares error for this.

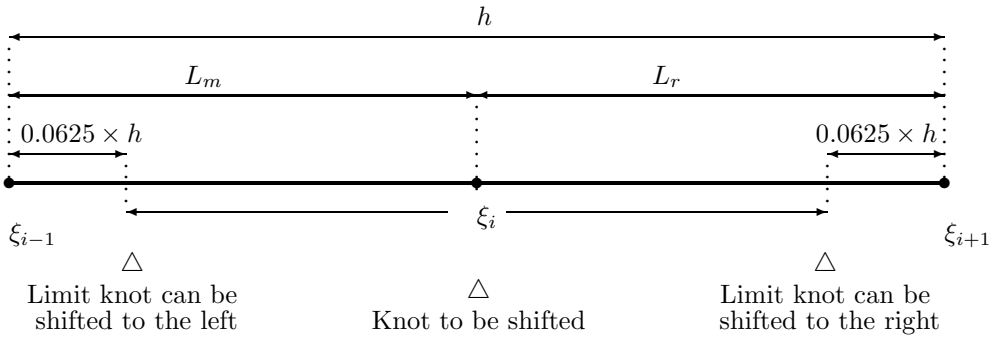
③ Next, add knots one at a time and calculate the cubic spline Φ_i; corresponding to the added knots, and obtain the orthonormal polynomial base by the modified Gram-Schmidt orthogonalization method. At the same time, calculate the function values and first differential coefficient values at the added knots and update the least squares error.

④ Finally, when all knots have been added, obtain the spline polynomial from the function values and first differential coefficient values.

⑤ When adding and modifying knots, create orthonormal polynomials from the polynomials and, at the end, regenerate the spline polynomial.

6.1.2.7 Cubic spline coefficients (least squares method with knot positions automatically determined)

(See Reference Bibliography (7))
The number of knots is fixed in advance and the least squares error E_S is minimized by moving interior knots ξ_i for $(i = 2, 3, \cdots, n-1)$.
The algorithm method is as follows. Fix the knots at the two endpoints and move knots sequentially from the rightmost knot to the leftmost knot to optimum positions so that the least squares error E_i on each interval is minimized.
The following figure shows the distance knots are shifted to the left or right.



Amount of shift to the left $=L_m\times$ (Average over the entire interval of ((Distance preceding knot was shifted)$/h$)). Initial value is $0.4\times L_m$

Amount of shift to the right $=L_r\times$ (Average over the entire interval of ((Distance preceding knot was shifted)$/h$)). Initial value is $0.4\times L_r$

6.1.2.8 Interpolation values according to cubic spline coefficients

Search the interval $\xi_i \leq x < \xi_{i+1}$ for the interpolation point. Then calculate the interpolation value by using the following interpolation equation.

$$f(x) = s_i + c_{1,i}(x - \xi_i) + c_{2,i}(x - \xi_i)^2 + c_{3,i}(x - \xi_i)^3 \tag{6.19}$$

s_i: The spline function value $f(\xi_i)$ at the knot. This equals y_i if neither smoothing nor the least squares methods is used.

6.1.2.9 Derivatives according to cubic spline coefficients

In the same way as getting interpolation values, search the interval for locations to differentiate and use the following equations.

$$\begin{aligned} f'(x) &= c_{1,i} + 2c_{2,i}(x-\xi_i) + 3c_{3,i}(x-\xi_i)^2 \\ f''(x) &= 2c_{2,i} + 6c_{3,i}(x-\xi_i) \end{aligned} \tag{6.20}$$

6.1.2.10 Integrals according to cubic spline coefficients

Let $[a, b]$ be the integration interval where a and b are in the following intervals.

$$\xi_i \le a < \xi_{i+1}, \xi_j \le b < \xi_{j+1}$$

Then:

$$\begin{aligned} \int_a^b f(x)dx &= \int_{\xi_i}^{\xi_{i+1}} f(x)dx - \int_{\xi_i}^{a} f(x)dx + \int_{\xi_{i+1}}^{\xi_j} f(x)dx + \int_{\xi_j}^{b} f(x)dx \\ &= \sum_{k=i}^{j-1} \int_{\xi_k}^{\xi_{k+1}} f(x)dx - \int_{\xi_i}^{a} f(x)dx + \int_{\xi_j}^{b} f(x)dx \end{aligned} \tag{6.21}$$

where:

$$\begin{aligned} \int_{\xi_k}^{\xi_{k+1}} f(x)dx &= s_i h_i + \frac{c_{1,i}}{2}{h_i}^2 + \frac{c_{2,i}}{3}{h_i}^3 + \frac{c_{3,i}}{4}{h_i}^4 \\ &= \frac{h_i}{2}(y_{i+1} + y_i) - \frac{{h_i}^3}{12}(c_{2,i+1} + c_{2,i}) \\ \int_{\xi_k}^{a} f(x)dx &= s_i(a-\xi_k) + \frac{c_{1,i}}{2}(a-\xi_k)^2 + \frac{c_{2,i}}{3}(a-\xi_k)^3 + \frac{c_{3,i}}{4}(a-\xi_k)^4 \end{aligned}$$

6.1.2.11 Bicubic spline coefficients

A bicubic spline is defined as an extension of a cubic spline. First, create a lattice partitioning of the X, Y plane according to the following expressions.

$$\begin{aligned} a < x_1 < x_2 < \cdots < x_m = b \\ c < y_1 < y_2 < \cdots < y_m = d \end{aligned} \tag{6.22}$$

Let $z_{i,j}$ be the function value at x_i, y_i.

(1) First, fix $y = y_k$ and interpolate by a cubic spline using data points $(x_i, z_{i,k})$ for $(i = 1, 2, \cdots, m)$ and "not-a-knot" endpoint conditions. Use the obtained cubic coefficient to obtain the derivative value $\dfrac{\partial z_{i,k}}{\partial x}$ for $(i = 1, \cdots, m)$ at the point (x_i, y_k).
Do this for $k = 1, \cdots, n$.

(2) Similarly, fix $x = x_k$, interpolate by a cubic spline using data points $(y_j, z_{k,j})$ for $(j = 1, \cdots, n)$ and "not-a-knot" endpoint conditions, and obtain the derivative values

$$\frac{\partial z_{k,i}}{\partial y} \text{ for } (i = 1, \cdots, n; k = 1, \cdots, m)$$

(3) Assume "not-a-knot" endpoint conditions at the four corners and interpolate by a cubic spline using the derivatives $\frac{\partial z_{i,j}}{\partial x}, \frac{\partial z_{i,j}}{\partial y}$ for $(i = 1, \cdots, m; j = 1, \cdots, n)$ obtained in steps 1 and 2 as data values. Use the obtained cubic spline to obtain the derivatives $\frac{\partial^2 z_{i,j}}{\partial x \partial y}$ for $(i = 1, \cdots, m; j = 1, \cdots, n)$ at the points (x_i, y_j).

You can simplify steps (1) through (3) described above by calculating cubic spline interpolation coefficients using first differential coefficients as follows.

(1) Obtain solutions $z_x(i, j)$ of the simultaneous linear equations:

$$h_i z_x(i+1, j) + 2(h_i + h_{i+1}) z_x(i, j) + h_{i+1} z_x(i-1, j) = 3 \left\{ \frac{h_i}{h_{i+1}} (z_{i+1,j} - z_{i,j}) + \frac{h_{i+1}}{h_i} (z_{i,j} - z_{i-1,j}) \right\} \quad (i = 2, \cdots, m)$$

for $j = 1, \cdots, n$, and:

$$h_i z_{xy}(i+1, j) + 2(h_i + h_{i+1}) z_{xy}(i, j) + h_{i+1} z_{xy}(i-1, j) = 3 \left\{ \frac{h_i}{h_{i+1}} (z_y(i+1, j) - z_y(i, j)) + \frac{h_{i+1}}{h_i} (z_y(i, j) - z_y(i-1, j)) \right\} \quad (i = 2, \cdots, m)$$

for $j = 1$ and $j = n$ (two endpoints).

(2) Obtain solutions $z_y(i, j)$ of:

$$k_j z_y(i, j+1) + 2(k_j + k_{j+1}) z_y(i, j) + k_{j+1} z_y(i, j+1) = 3 \left\{ \frac{k_j}{k_{j+1}} (z_{i,j+1} - z_{i,j}) + \frac{k_{j+1}}{k_j} (z_{i,j} - z_{i,j-1}) \right\} \quad (j = 1, \cdots, n)$$

for $i = 1, \cdots, m$.

(3) Obtain solutions $z_{xy}(i, j)$ of:

$$k_j z_{xy}(i, j+1) + 2(k_j + k_{j+1}) z_{xy}(i, j) + k_{j+1} z_{xy}(i, j+1) = 3 \left\{ \frac{k_j}{k_{j+1}} (z_x(i, j+1) - z_x(i, j)) + \frac{k_{j+1}}{k_j} (z_x(i, j) - z_x(i, j-1)) \right\} \quad (j = 1, \cdots, n)$$

for $i = 1, \cdots, m$.

6.1.2.12 Bicubic spline interpolation values

The function value at the point (x, y) is represented by the following equation.

$$f(x, y) = \sum_{\ell=0}^{3} \sum_{r=0}^{3} \alpha_{\ell,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i} \right)^{\ell} \left(\frac{y - y_j}{y_{j+1} - y_j} \right)^{r} \tag{6.23}$$

$\alpha_{\ell,r}^{i,j}$ are obtained from the following matrix calculations:

$$
\begin{aligned}
&\Gamma_{ij} = A(h_i) K_{ij} A(k_j)^T \\
&\Gamma_{ij} \quad = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \quad \text{(Individual elements } \alpha_{\ell r}\text{)} \\
&A(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ -3 & -2t & 3 & -t \\ 2 & t & -2 & t \end{bmatrix} \\
&K_{ij} = \begin{bmatrix} z(i,j) & z_y(i,j) & z(i,j+1) & z_y(i,j+1) \\ z_x(i,j) & z_{xy}(i,j) & z_x(i,j+1) & z_{xy}(i,j+1) \\ z(i+1,j) & z_y(i+1,j) & z(i+1,j+1) & z_y(i+1,j+1) \\ z_x(i+1,j) & z_{xy}(i+1,j) & z_x(i+1,j+1) & z_{xy}(i+1,j+1) \end{bmatrix}
\end{aligned}
\tag{6.24}
$$

where:

$$
\begin{aligned}
&x_i < x < x_{i+1} \quad , \quad y_j < y < y_{j+1} \\
&h_i = x_{i+1} - x_i \quad , \quad k_j = y_{j+1} - y_j \\
&z(i,j) = z_{i,j} \quad , \quad z_x(i,j) = \frac{\partial z_{i,j}}{\partial x} \quad , \quad z_y(i,j) = \frac{\partial z_{i,j}}{\partial y} \quad , \quad z_{xy}(i,j) = \frac{\partial^2 z_{i,j}}{\partial x \partial y}
\end{aligned}
$$

In this library, the following values are set as spline coefficients.

$$
\begin{aligned}
C(1,I,1,J) &= z_{i,j} &&= z(i,j) \\
C(2,I,1,J) &= \frac{\partial z_{i,j}}{\partial x} &&= z_x(i,j) \\
C(1,I,2,J) &= \frac{\partial z_{i,j}}{\partial y} &&= z_y(i,j) \\
C(2,I,2,J) &= \frac{\partial^2 z_{i,j}}{\partial x \partial y} &&= z_{xy}(i,j)
\end{aligned}
$$

Therefore, $\alpha_{\ell,r}^{i,j}$ is not used to obtain spline function values. Instead, these values are obtained directly by combining the calculations in equations (6.23) and (6.24).

$$
\begin{aligned}
m &\leftarrow 1 \\
h_x &\leftarrow x_{i+1} - x_i \\
h_y &\leftarrow y_{j+1} - y_j \\
U &\leftarrow (x - x_i)/h_x \\
V &\leftarrow (y - y_j)/h_y
\end{aligned}
$$

for $k = j, j+1$
  for $\ell = 1, 2$
    $\alpha_0 \leftarrow C(1, i, \ell, k)$
    $\alpha_1 \leftarrow h_x C(2, i, \ell, k)$
    $\alpha_2 \leftarrow 3(C(1, i+1, \ell, k) - \alpha_0) - h_x C(2, i+1, \ell, k) - 2\alpha_1$
    $\alpha_3 \leftarrow 2(\alpha_0 - C(1, i+1, \ell, k)) + h_x C(2, i+1, \ell, k) + \alpha_1$
    $S_m \leftarrow \alpha_0 + U(\alpha_1 + U(\alpha_2 + U\alpha_3))$
    $m \leftarrow m + 1$

$$
\begin{aligned}
\alpha_0 &\leftarrow S_1 \\
\alpha_1 &\leftarrow h_y S_2 \\
\alpha_2 &\leftarrow 3(S_3 - \alpha_0) - h_y S_4 - 2\alpha_1 \\
\alpha_3 &\leftarrow 2(\alpha_0 - S_3) + h_y S_4 + \alpha_1 \\
f(x, y) &\leftarrow \alpha_0 + V(\alpha_1 + V(\alpha_2 + V\alpha_3))
\end{aligned}
$$

6.1.2.13 Bicubic spline mixed partial derivatives

$$
\begin{aligned}
f(x, y) &= \sum_{\ell=0}^{3}\sum_{r=0}^{3} \alpha_{\ell,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i}\right)^{\ell} \left(\frac{y - y_j}{y_{j+1} - y_j}\right)^{r} \\
\frac{\partial f}{\partial x} &= \sum_{\ell=1}^{3}\sum_{r=0}^{3} -\ell\alpha_{\ell,r}^{i,j} \frac{(x - x_i)^{\ell-1}}{(x_{i+1} - x_i)^{\ell}} \left(\frac{y - y_j}{y_{j+1} - y_j}\right)^{r} \qquad (6.25) \\
\frac{\partial f}{\partial y} &= \sum_{\ell=0}^{3}\sum_{r=1}^{3} -r\alpha_{\ell,r}^{i,j} \left(\frac{x - x_i}{x_{i+1} - x_i}\right)^{\ell} \frac{(y - y_j)^{r-1}}{(y_{j+1} - y_j)^{r}} \qquad (6.26) \\
\frac{\partial^2 f}{\partial x \partial y} &= \sum_{\ell=1}^{3}\sum_{r=1}^{3} \ell r\alpha_{\ell,r}^{i,j} \frac{(x - x_i)^{\ell-1}}{(x_{i+1} - x_i)^{\ell}} \frac{(y - y_j)^{r-1}}{(y_{j+1} - y_j)^{r}} \qquad (6.27)
\end{aligned}
$$

These calculations are similar to those performed for interpolation values.

6.1.2.14 Bicubic spline double integral

Compute the double integral for the rectangular interval formed from integration intervals $[A, B]$ in the X direction and $[C, D]$ in the Y direction by summing integrals computed for subintervals or lattices contained in this interval. Use the following equation to compute the integral for a single interval.

$$
\begin{aligned}
&\int_{x_i}^{x}\int_{y_j}^{y} f(x, y)dydx \\
&= \sum_{\ell=0}^{3}\sum_{r=0}^{3} \alpha_{\ell,r}^{i,j} \frac{(x_{i+1} - x_i)(y_{j+1} - y_j)}{(\ell+1)(r+1)} \left(\frac{x - x_i}{x_{i+1} - x_i}\right)^{\ell+1} \left(\frac{y - y_j}{y_{j+1} - y_j}\right)^{r+1}
\end{aligned} \qquad (6.28)
$$

6.1.2.15 Plane data interpolation

For interpolation first, perform a spline interpolation assuming the distance between data points and X coordinate as abscissa value and ordinate value, respectively, then determine the interpolated X coordinate. Next, perform a spline interpolation assuming the abscissa value as the above and Y coordinate as ordinate value, then determine the interpolated y ordinate. Output the interpolation values x and y obtained in the above.

For smoothing, perform a spline interpolation using the cubic spline automatic smoothing subroutine. When the string of input data points takes the shape of an open curve, obtain the spline coefficient with the method requiring no endpoint condition input. When the string of input data points takes the shape of a closed curve, obtain the spline coefficient with the method of periodic conditions.

6.1.2.16 Interpolation using a B–spline function (one–dimensional)

Assume that the data (x_i, y_i) $(i = 0, 1, \cdots, N)$ is given in the region $R : a = x_0 \leq x \leq x_N = b$. At this time, use the following spline function of order m (degree $m - 1$):

$$S(x_i) = y_i \quad (i = 0, 1, \cdots, N) \tag{6.29}$$

having the following (internal) knots that were determined in advance:

$$\xi_1 \leq \xi_2 \leq \cdots \leq \xi_n \tag{6.30}$$

to perform interpolation. Since interpolation can be uniquely performed at this time, the condition:

$$N + 1 = m + n \tag{6.31}$$

and the Shoenberg–Whitney conditions must hold.
Here, the Shoenberg–Whitney conditions are that the following inequalities are satisfied:

$$\left.\begin{array}{l} x_0 < \xi_1 < x_m, \\ x_1 < \xi_2 < x_{m+1}, \\ \cdots\cdots\cdots\cdots \\ x_{N-m} < \xi_n < x_N \end{array}\right\} \tag{6.32}$$

A B–spline of order m (degree $m - 1$) is used as the basis function of the spline function $S(x)$. To create the required basis function, we introduce the following $2m$ additional knots:

$$\left.\begin{array}{l} \xi_{1-m} = \xi_{2-m} = \cdots = \xi_0 = a, \\ \xi_{n+1} = \xi_{n+2} = \cdots = \xi_{n+m} = b \end{array}\right\} \tag{6.33}$$

This corresponds to entering m overlaid knots on the endpoints of the interval $[a, b]$.
Once this is done, $S(x)$ is expressed in $\xi_0 \leq x \leq \xi_{n+1}$ as follows:

$$S(x) = \sum_{i=1}^{n+m} c_i^* M_{mi}(x) = \sum_{i=1}^{n+m} c_i N_{mi}(x) \tag{6.34}$$

Here, $M_{mi}(x)$ is the B–spline of order m defined for the knots $\xi_{i-m}, \xi_{i-m+1}, \cdots, \xi_i$. Also, $N_{mi}(x)$, which is the normalized B–spline, is defined by:

$$N_{mi}(x) = (\xi_i - \xi_{i-m}) M_{mi}(x) \tag{6.35}$$

The value of the B–spline $M_{mi}(x)$ can be easily calculated by using the de Boor–Cox algorithm. The de Boor–Cox algorithm is a computational technique that uses the following asymptotic expressions:

$$M_{rj}(x) = \frac{(x - \xi_{j-r}) M_{r-1,j-1}(x) + (\xi_j - x) M_{r-1,j}(x)}{\xi_j - \xi_{j-r}} \quad (r = 2, 3, \cdots, m) \tag{6.36}$$

$$M_{1j} = \begin{cases} \dfrac{1}{\xi_j - \xi_{j-1}} & (\xi_{j-1} \leq x < \xi_j) \\ \\ 0 & (\text{Otherwise}) \end{cases} \tag{6.37}$$

By substituting (6.34) in (6.29), we obtain the following simultaneous linear equations:

$$\sum_{i=1}^{n+m} c_i N_{mi}(x_j) = y_j \quad (j = 0, 1, \cdots, N) \tag{6.38}$$

Let these equations be expressed in matrix representation as:

$$\boldsymbol{Ac} = \boldsymbol{y} \tag{6.39}$$

Equation (6.39) can be effectively solved by using Gaussian elimination with partial pivoting. When the solution is substituted in (6.34), the interpolation spline $S(x)$ is determined.

6.1.2.17 Interpolation using a B–spline function (multi–dimensional)

Lets extend the interpolation method for one–dimensional data using a B–spline, which was described in (17), to the case for two–dimensional data. Assume that the function $f_{ij} = f(x_i, y_j)$ is given at the knots $(x_i, y_j)(i = 0, 1, \cdots, I; j = 0, 1, 2, \cdots, J)$ in the rectangular region $R : a = x_0 \leq x \leq x_I = b; c = y_0 \leq y \leq y_J = d$. At this time, use the following spline function of two variables of order m (degree $m - 1$) and order n (degree $n - 1$):

$$S(x_i, y_j) = f_{ij} \quad (i = 0, 1, \cdots, I; j = 0, 1, \cdots, J) \tag{6.40}$$

having the following (internal) knots in the x direction:

$$\xi_1 \leq \xi_2 \leq \cdots \leq \xi_h \tag{6.41}$$

and the following (internal) knots in the y direction:

$$\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_k \tag{6.42}$$

to perform interpolation. Assume that the following relationships hold for the x direction:

$$I + 1 = h + m, \tag{6.43}$$

$$\left.\begin{array}{l} x_0 < \xi_1 < x_m \\ x_1 < \xi_2 < x_{m+1} \\ \cdots\cdots\cdots\cdots\cdots \\ x_{I-m} < \xi_h < x_I \end{array}\right\} \tag{6.44}$$

and the following relationships hold for the y direction:

$$J + 1 = k + n, \tag{6.45}$$

$$\left.\begin{array}{l} y_0 < \zeta_1 < y_n \\ y_1 < \zeta_2 < y_{n+1} \\ \cdots\cdots\cdots\cdots\cdots \\ y_{J-n} < \zeta_k < y_J \end{array}\right\} \tag{6.46}$$

The spline function $S(x, y)$ can be formed by using a set of basis functions. These basis functions can be created by taking tensor products of one–dimensional basis functions. To create the required basis function, we introduce the following $2m$ additional knots in the x direction:

$$\left.\begin{array}{l} \xi_{1-m} = \cdots = \xi_0 = a, \\ \xi_{h+1} = \cdots = \xi_{h+m} = b \end{array}\right\} \tag{6.47}$$

and the following $2n$ additional knots in the y direction:

$$\left.\begin{aligned}\zeta_{1-n} = \cdots = \zeta_0 = c,\\ \zeta_{k+1} = \cdots = \zeta_{k+n} = d\end{aligned}\right\} \tag{6.48}$$

At this time, the product $M_{mi}(x)M_{nj}(y)$ $(i = 1, 2, \cdots, h+m; j = 1, 2, \cdots, k+n)$ of $M_{mi}(x), (i = 1, 2, \cdots, h+m)$, which is the B–spline of order m (degree $m-1$) for the knot $\boldsymbol{\xi} = (\xi_{1-m}, \xi_{2-m}, \cdots, \xi_{h+m})$, and $M_{nj}(y), (j = 1, 2, \cdots, k+n)$, which is the B–spline of order n (degree $n-1$) for the knot $\boldsymbol{\zeta} = (\zeta_{1-n}, \zeta_{2-n}, \cdots, \zeta_{k+n})$, becomes the basis function of the spline function $S(x, y)$. That is, $S(x, y)$ is expressed in $\xi_0 \le x \le \xi_{h+1}, \zeta_0 \le y \le \zeta_{k+1}$ as follows:

$$S(x,y) = \sum_{i=1}^{h+m}\sum_{j=1}^{k+n} c^*_{ij} M_{mi}(x) M_{nj}(y) = \sum_{i=1}^{h+m}\sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y) \tag{6.49}$$

Here, $N_{mi}(x)$ and $N_{nj}(y)$, which are normalized B–splines of order m (degree $m-1$) and order n (degree $n-1$) respectively, satisfy the following relationships:

$$\left.\begin{aligned}N_{mi}(x) = (\xi_i - \xi_{i-m}) M_{mi}(x)\\ N_{nj}(y) = (\zeta_j - \zeta_{j-n}) M_{nj}(y)\end{aligned}\right\} \tag{6.50}$$

The values of these B–splines can be easily calculated by using the de Boor–Cox algorithm described in (17).
If we substitute (6.49) in (6.40), we obtain:

$$\sum_{r=1}^{h+m}\sum_{s=1}^{k+n} c_{rs} N_{mr}(x_i) N_{ns}(y_j) = f_{ij} \quad (i = 0, 1, \cdots, I; j = 0, 1, \cdots, J) \tag{6.51}$$

Equation (6.51), which represents simultaneous linear equations for which c_{rs} is unknown, has a unique solution because (6.43) to (6.46)are assumed.
Equation (6.51) can be expressed in matrix representation as:

$$A\boldsymbol{c} = \boldsymbol{f} \tag{6.52}$$

where, $\boldsymbol{c}$ and $\boldsymbol{f}$ can be written as follows:

$$\boldsymbol{c} = [c_{11}, c_{21}, \cdots, c_{h+m,1}, \cdots, c_{h+m,k+n}]^T \tag{6.53}$$

$$\boldsymbol{f} = [f_{00}, f_{10}, \cdots, f_{I0}, \cdots, f_{IJ}]^T \tag{6.54}$$

Equation (6.52) can be solved by using Gaussian elimination with partial pivoting. When the solution is substituted in (6.49), the interpolation spline $S(x, y)$is determined.
The interpolation method for two–dimensional data described above can be extended in a similar manner for interpolation of multidimensional data of three or more dimensions.

6.1.2.18 B–spline smoothing (one–dimensional data)

Assume that data is given in the interval $[a, b]$ and that F_k is defined as follows:

$$F_k = f(x_k) + e_k \quad (k = 1, 2, \cdots, N) \tag{6.55}$$

where, $f(x)$ is the underlying function of the data (unknown function) and the e_k are mutually independent errors that obey a normal distribution with mean 0 and variance σ^2.

Use the least squares method to apply equation (6.34) to the data (6.55). Assume that the knots are given. The sum of the squares of the residuals is as follows:

$$Q = \sum_{k=1}^{N} \{S(x_k) - F_k\}^2 \tag{6.56}$$

Partially differentiate (6.56) with respect to parameter $c_i (i = 1, 2, \cdots, n + m)$ and set it to 0 to obtain the normalized equation:

$$\boldsymbol{Ac} = \boldsymbol{d} \tag{6.57}$$

Here $\boldsymbol{c}$ and $\boldsymbol{d}$ are as follows:

$$\boldsymbol{c} = (c_1, c_2, \cdots, c_{n+m})^T, \tag{6.58}$$

$$\begin{aligned} \boldsymbol{d} = \{ &\sum_{x_k \in [\xi_{1-m}, \xi_1]} N_{m1}(x_k) F_k, \sum_{x_k \in [\xi_{2-m}, \xi_2]} N_{m2}(x_k) F_k, \cdots, \\ &\sum_{x_k \in [\xi_n, \xi_{n+m}]} N_{m,n+m}(x_k) F_k \}^T \end{aligned} \tag{6.59}$$

T represents the transpose.
The element in row i and column j of A is expressed as follows:

$$a_{ij} = \sum N_{mi}(x_k) N_{mj}(x_k) \tag{6.60}$$

The values of the B–spline $N_{mi}(x)$ can be easily calculated as described in (17). Also, by taking into account that the coefficient matrix A is a band matrix, (6.57) can be effectively solved by using Cholesky's method. If its solution $\boldsymbol{c}$ is substituted in (6.34), the approximation function $S(x)$ is determined.
To obtain a good approximation, the number of knots and their positions must be determined appropriately. Here, we assume that the standard AIC for applications is used, and that the best has been selected from among several applications that are considered. (That is, we assume that a good number of knots and good positions are obtained.)
This AIC, which is also called Akaike's Information Criterion, is defined as follows:

$$AIC = (-2) \log_e (\text{maximum likelihood}) + 2(\text{number of parameters}) \tag{6.61}$$

Here, the number of parameters is the number that can freely change within the statistical model. The model that minimizes the AIC value is considered to be the best model. When the AIC is used, a subjective judgement is completely unnecessary, and the best model from among multiple models that are considered can be automatically determined.
Let's consider the following regression model:

$$F_k = S(x_k) + e_k \quad (k = 1, 2, \cdots, N) \tag{6.62}$$

The parameters of this model are the coefficients $c_i (i = 1, 2, \cdots, n + m)$ of $S(x)$ and the variance σ^2 of the error. Since the knots of $S(x)$ are determined before the application of the model, the number of parameters cannot be increased. Therefore, from (6.61), AIC is as follows:

$$AIC = N \log_e Q + 2(n + m) \tag{6.63}$$

The model that minimizes (6.63) is considered to be the best approximation function. Therefore, we can try varying the number of knots and their positions to search for the application that minimizes AIC as much as possible. If there is a number of knots and corresponding positions for which that application can be satisfied, then we can assume that the best approximation was obtained.

6.1.2.19 B–spline smoothing (multi–dimensional data)

Let's extend the one–dimensional smoothing using a B–spline, which was described in (19), to the case for two–dimensional data. Assume that data is given in the rectangular region $R = [a, b] \times [c, d]$ on the $x - y$ plane and that F_{ij} is defined as follows:

$$F_{ij} = f(x_i, y_j) + e_r \quad (i = 1, 2, \cdots, I; j = 1, 2, \cdots, J; r = 1, 2, \cdots, IJ) \tag{6.64}$$

where, $f(x, y)$ is the underlying function of the data (unknown function) and the e_r are mutually independent errors that obey a normal distribution with mean 0 and variance σ^2. The sample points (x_i, y_j) are given at grid points.
Use the least squares method to apply equation (6.49) to the data (6.64). Assume that the knots are given in each direction. The sum of the squares of the residuals is as follows:

$$Q = \sum_{r=1}^{N} \{S(x_r, y_r) - F_r\}^2 \tag{6.65}$$

Partially differentiate (6.65) with respect to parameter $c_{ij} \quad (i = 1, 2, \cdots, h + m; j = 1, 2, \cdots, k + n)$ and set it to 0 to obtain the normalized equation:

$$A\boldsymbol{c} = \boldsymbol{d} \tag{6.66}$$

Here $\boldsymbol{c}$ and $\boldsymbol{d}$ are as follows:

$$\boldsymbol{c} = (c_{11}, c_{21}, \cdots, c_{h+m,1}, \cdots, c_{h+m,k+n})^T \tag{6.67}$$

$$\begin{aligned}
\boldsymbol{d} = \quad \{ & \sum_{x_r \in [\xi_{1-m}, \xi_1] \cap y_r \in [\zeta_{1-n}, \zeta_1]} N_{m1}(x_r) N_{n1}(y_r) F_r, \\
& \sum_{x_r \in [\xi_{2-m}, \xi_2] \cap y_r \in [\zeta_{1-n}, \zeta_1]} N_{m2}(x_r) N_{n1}(y_r) F_r, \\
& \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots, \\
& \sum_{x_r \in [\xi_h, \xi_{h+m}] \cap y_r \in [\zeta_{1-n}, \zeta_1]} N_{m,h+m}(x_r) N_{n1}(y_r) F_r, \\
& \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots, \\
& \sum_{x_r \in [\xi_h, \xi_{h+m}] \cap y_r \in [\zeta_k, \zeta_{k+n}]} N_{m,h+m}(x_r) N_{n,k+n}(y_r) F_r \}^T
\end{aligned} \tag{6.68}$$

By taking into account that the coefficient matrix A is a band matrix, (6.66) can be effectively solved by using Cholesky's method.
Let's consider the following regression model:

$$F_{ij} = f(x_i, y_j) + e_r \quad (i = 1, 2, \cdots, I; j = 1, 2, \cdots, J; r = 1, 2, \cdots, IJ) \tag{6.69}$$

From (6.61), AIC is as follows:

$$AIC = N \log_e Q + 2(h + m)(k + n) \tag{6.70}$$

We can try varying the number of knots and their positions to search for the application that minimizes AIC as much as possible. If there is a number of knots and corresponding positions for which that application can be satisfied, then we can assume that the best approximation was obtained. The smoothing for two–dimensional data described above can be extended in a similar manner for smoothing of multidimensional data of three or more dimensions.

6.1.3 Reference Bibliography

(1) De Boor, C. , "A Practical Guide to Splines", Springer-Verlag New York, (1978).

(2) Ahlberg, J. , Nilson, E. and Walsh, J. , "The Theory of Splines and Their Applications", Academic Press New York, (1967).

(3) Guerra, V. and Tapia, R. A. , "A Local Procedure for Error Detection and Data Smoothing", MRC Technical Summary Report #1452, Mathematics Research Center, University of Wisconsin-Madison, (1974).

(4) Craven, P. and Wahaba, G. , "Smoothing Noisy Data with Spline Functions", Numer. Math. , Vol. 31, pp. 377-403, (1979).

(5) Reinsch, C. H. , "Smoothing by Spline Functions", Numer. Math. , Vol. 10, pp. 177-183, (1967).

(6) De Boor, C. and Rice, J. R. , "Least Squares Cubic Spline Approximation I - Fixed Knots", Computer Sciences Department TR20, Purdue Univ. , (1968).

(7) De Boor, C. and Rice, J. R. , "Least Squares Cubic Spline Approximation II - Variable Knots", Computer Sciences Department TR21, Purdue Univ. , (1968).

(8) STONE, HAROLD. S. , "Parallel Tridiagonal Equation Solvers", ACM Transactions on Mathematical Software, Vol. 1, No. 4, 289 (1975).

(9) Hockney, R. W. and Jesshope, C. R. , "Parallel Computer"

6.2 CUBIC SPLINE (CURVED LINE INTERPOLATION)

6.2.1 DGISPC, RGISPC
Interpolation Values and Cubic Spline Coefficients

(1) **Function**

DGISPC or RGISPC obtains cubic spline coefficients for "not-a-knot" endpoint conditions when endpoint conditions need not be entered and calculates interpolation values at specified points. The knots are set equal to sample points.

(2) **Usage**

Double precision:

CALL DGISPC (X, Y, N, XL, FL, M, C, IERR)

Single precision:

CALL RGISPC (X, Y, N, XL, FL, M, C, 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}	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)) for $i = 1, \cdots, \mathrm{N}$ (X(i) < X(i + 1), i ≠ N)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	XL	{D / R}	M	Input	Abscissa values of points where interpolation values are calculated
5	FL	{D / R}	M	Output	Cubic spline interpolation values at XL(i)
6	M	I	1	Input	Number of interpolation values
7	C	{D / R}	$3, (\mathrm{N}-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t (X(j) ≤ t < X(j + 1)): $\mathrm{f(t)} = ((\mathrm{C(3,j)} \times \mathrm{D} + \mathrm{C(2,j)}) \times \mathrm{D} + \mathrm{C(1,j)}) \times \mathrm{D} + \mathrm{Y(j)}$ where: $\mathrm{D} = \mathrm{t} - \mathrm{X(j)}$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq 2$

(b) $\mathrm{X(1)} < \mathrm{X(2)} < \cdots < \mathrm{X(N)}$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\left\{\begin{matrix}\text{DGISCX}\\\text{RGISCX}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{DGIDCY}\\\text{RGIDCY}\end{matrix}\right\}$ or 6.2.20 $\left\{\begin{matrix}\text{DGIICZ}\\\text{RGIICZ}\end{matrix}\right\}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation $y_i = x_i e^{-4.0x_i}$ to find y_i $(i = 1, 2, \cdots, 9)$ at each point
$\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$
and perform cubic spline interpolation using these points as sample points. In addition, obtain the value of the cubic spline function at the uniformly spaced points $xl_j = 0.1 \times j (j = 1, 2, \cdots, 10)$.

(b) Input data
X(i)=x_i, Y(i)=y_i (i = $1, \cdots$, N), XL(j)=xl_j (j = $1, \cdots$, M), N = 9 and M = 10.

(c) Main program

```
      PROGRAM BGISPC
! *** EXAMPLEOF DGISPC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=9,M=10)
      DIMENSION X(N),Y(N),XL(M),FL(M),C(3,N-1)
      READ(5,*) X
      READ(5,*) XL
      READ(5,*) Y
      WRITE(6,1000)   N,M,(I,X(I),Y(I),I=1,9)
      WRITE(6,1001)   (I,XL(I),I=1,10)
      CALL DGISPC(X,Y,N,XL,FL,M,C,IERR)
      WRITE(6,1300)   IERR
      WRITE(6,1400)   (I,FL(I),I=1,M)
      WRITE(6,1500)
      DO 20  J=1,N-1
      WRITE(6,1600)  J,(C(I,J),I=1,3)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGISPC ***',/,/,6X,'** INPUT **',/,/,8X,'N  =',&
             I3,/,/,8X,'M  =',I3,/,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',4X,&
             'X(I)',5X,'Y(I)',/,9(8X,I2,F7.2,F11.4,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',3X,'XL(J)',/,&
             10(8X,I2,F7.2,/))
 1300 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1400 FORMAT(' ',/,(9X,'FL(',I2,')=',D20.10))
 1500 FORMAT(' ',/,9X,'C(I,J)',/,26X,'I=1',19X,'I=2',19X,'I=3',/)
 1600 FORMAT(11X,'J=',I1,3D22.10)
      END
```

(d) Output results

```
 *** DGISPC ***

  ** INPUT **
```

```
  N  =  9

  M  = 10

COORDINATES (X,Y)

   I    X(I)     Y(I)
   1   0.00     0.0000
   2   0.10     0.0670
   3   0.23     0.0917
   4   0.34     0.0873
   5   0.47     0.0717
   6   0.59     0.0557
   7   0.73     0.0394
   8   0.92     0.0232
   9   1.00     0.0183


SPECIFIED POINTS

   J   XL(J)
   1   0.10
   2   0.20
   3   0.30
   4   0.40
   5   0.50
   6   0.60
   7   0.70
   8   0.80
   9   0.90
  10   1.00


** OUTPUT **

  IERR=   0

   FL( 1)=    0.6700000000D-01
   FL( 2)=    0.9006470964D-01
   FL( 3)=    0.9036744658D-01
   FL( 4)=    0.8081846936D-01
   FL( 5)=    0.6764006631D-01
   FL( 6)=    0.5442716983D-01
   FL( 7)=    0.4259556695D-01
   FL( 8)=    0.3263937277D-01
   FL( 9)=    0.2459374176D-01
   FL(10)=    0.1830000000D-01

   C(I,J)
                      I=1                   I=2                   I=3

     J=1      0.9628817827D+00     -0.3294844482D+01      0.3660266545D+01
     J=2      0.4137208827D+00     -0.2196764518D+01      0.3660266545D+01
     J=3      0.2813762187D-01     -0.7692605656D+00      0.1362069450D+01
     J=4     -0.9165658153D-01     -0.3197776471D+00      0.7827027013D+00
     J=5     -0.1351157428D+00     -0.1452359362D-01      0.2448083836D+00
     J=6     -0.1280256831D+00      0.7360742448D-01      0.6592205430D-01
     J=7     -0.1035393875D+00      0.1012946873D+00     -0.2686318577D-01
     J=8     -0.6795668932D-01      0.8598267140D-01     -0.2686318577D-01
```

6.2.2 DGISSC, RGISSC
Smoothed Interpolation Values and Cubic Spline Coefficients

(1) **Function**

DGISSC or RGISSC automatically obtains the optimum smoothed cubic spline coefficients and calculates interpolation values at specified points. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

```
CALL DGISSC  (X, YD, N, XL, FL, M, Y, C, ISW, WK, IERR)
```

Single precision:

```
CALL RGISSC  (X, YD, N, XL, FL, M, Y, C, ISW, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point (X(i), YD(i)) for $i = 1, \cdots, N$ (X(i) < X(i + 1), i ≠ N)
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XL	{D / R}	M	Input	Abscissa values of points where interpolation values are calculated
5	FL	{D / R}	M	Output	Smoothed cubic spline interpolation values at XL(i)
6	M	I	1	Input	Number of interpolation values
7	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
8	C	{D / R}	3, (N − 1)	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t (X(j) ≤ t < X(j + 1)): f(t) = ((C(3, j) × D + C(2, j)) × D + C(1, j)) × D + Y(j) where: D = t − X(j)
9	ISW	I	1	Input	Processing switch ISW=1: When the abscissa value spacing may not be uniform. When sample points are nearly uncorrelated, limit value N up to 20. ISW=2: When the abscissa value spacing is uniform.
10	WK	{D / R}	See Contents	Work	Work area **Size**: N × (2 × N + 4) (when ISW=1) 6 × N (when ISW=2)
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 4

(b) X(1) < X(2) < ⋯ < X(N) (Ascending order)

(c) If ISW=2, abscissa values must be uniformly spaced.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	The minimum cross-validation value could not be found. (The data is uncorrelated.)	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem

Use the equation $y_i = \sin(3\pi x_i/2) + e_i$ (e_i:Uniform random number in the interval $[-0.2, 0.2]$) to find values y_i at each point $x_i = (i-1)/24$ ($i = 1, 2, \cdots, 25$) and perform smoothed cubic spline interpolation using these points as sample points. In addition, obtain the value of the cubic spline function at $xl_j = 0.1 \times (j-1) (j = 1, 2, \cdots, 10)$.

(b) Input data

X(i)=x_i, YD(i)=y_i (i = $1, \cdots,$ N), XL(j)=xl_j (j = $1, \cdots,$ M), N=25, ISW=2 and M=10.

(c) Main program

```
      PROGRAM BGISSC
! *** EXAMPLE OF DGISSC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=25,M=10,ISW=2)
      DIMENSION X(N),YD(N),XL(M),FL(M),Y(N),C(3,N-1),WK(6*N)
      READ(5,*) X
      READ(5,*) YD
      READ(5,*) XL
      WRITE(6,1000)   N,M,ISW,(I,X(I),YD(I),I=1,25)
      WRITE(6,1001)   (I,XL(I),I=1,10)
      CALL DGISSC(X,YD,N,XL,FL,M,Y,C,ISW,WK,IERR)
      WRITE(6,1300)   IERR
      WRITE(6,1400)   (I,FL(I),I=1,M)
      WRITE(6,1500)   (I,Y(I),I=1,N)
      WRITE(6,1600)
      DO 20  J=1,N-1
      WRITE(6,1700)  J,(C(I,J),I=1,3)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGISSC ***',/,/,6X,'** INPUT **',/,/,8X,&
      'N   =',I3,/,/,8X,'M   =',I3,/,/,8X,'ISW =',I3,/,/,6X,'COORDINATES (X,YD)',&
       /,/,9X,'I',5X,'X(I)',7X,'YD(I)',/,25(8X,I2,F10.4,F11.4,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',3X,&
       'XL(J)',/,10(8X,I2,F7.2,/))
 1300 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1400 FORMAT(' ',/,(9X,'FL(',I2,')=',D21.10))
 1500 FORMAT(' ',/,(9X,'Y(',I2,')=',D22.10))
 1600 FORMAT(' ',/,9X,'C(I,J)',/,27X,'I=1',19X,'I=2',19X,'I=3',/)
 1700 FORMAT(11X,'J=',I2,3D22.10)
      END
```

(d) Output results

```
*** DGISSC ***

 ** INPUT **

   N   = 25

   M   = 10

   ISW =  2

 COORDINATES (X,YD)

    I     X(I)       YD(I)
    1    0.0000     0.0481
    2    0.0416     0.1472
    3    0.0833     0.4416
    4    0.1250     0.5794
    5    0.1666     0.6412
    6    0.2083     0.7395
    7    0.2500     0.7260
    8    0.2916     1.0909
    9    0.3333     0.8045
   10    0.3750     1.0209
   11    0.4166     0.8506
   12    0.4583     1.0020
   13    0.5000     0.5168
   14    0.5416     0.6915
   15    0.5833     0.3485
   16    0.6250     0.3854
   17    0.6666    -0.1024
   18    0.7083    -0.1352
   19    0.7500    -0.3388
   20    0.7916    -0.5710
   21    0.8333    -0.5254
   22    0.8750    -0.8405
   23    0.9166    -0.7953
   24    0.9583    -1.0837
   25    1.0000    -1.1848


 SPECIFIED POINTS

    J   XL(J)
    1   0.00
    2   0.10
    3   0.20
    4   0.30
    5   0.40
    6   0.50
    7   0.60
    8   0.70
    9   0.80
   10   0.90


 ** OUTPUT **

   IERR=   0

    FL( 1)=     0.9502274321D-01
    FL( 2)=     0.4468124930D+00
    FL( 3)=     0.7323488324D+00
    FL( 4)=     0.9006450999D+00
    FL( 5)=     0.8960335094D+00
    FL( 6)=     0.6952556081D+00
    FL( 7)=     0.3411800920D+00
    FL( 8)=    -0.8609161540D-01
    FL( 9)=    -0.4897553892D+00
    FL(10)=    -0.8487507981D+00

    Y( 1)=      0.9502274321D-01
    Y( 2)=      0.2445740596D+00
    Y( 3)=      0.3906840363D+00
    Y( 4)=      0.5268362807D+00
    Y( 5)=      0.6479571394D+00
    Y( 6)=      0.7512835941D+00
    Y( 7)=      0.8341775208D+00
    Y( 8)=      0.8925288529D+00
    Y( 9)=      0.9199921903D+00
    Y(10)=      0.9153973040D+00
    Y(11)=      0.8761381156D+00
    Y(12)=      0.8021604666D+00
    Y(13)=      0.6952556081D+00
    Y(14)=      0.5626220076D+00
    Y(15)=      0.4079605341D+00
    Y(16)=      0.2374932368D+00
    Y(17)=      0.5779584896D-01
    Y(18)=     -0.1211775851D+00
    Y(19)=     -0.2940767779D+00
    Y(20)=     -0.4579219747D+00
    Y(21)=     -0.6125125122D+00
    Y(22)=     -0.7613707264D+00
```

```
Y(23)=     -0.9065999263D+00
Y(24)=     -0.1051376070D+01
Y(25)=     -0.1195843966D+01

C(I,J)
                  I=1                  I=2                  I=3

  J= 1      0.3600899832D+01     0.2132481200D-13    -0.5898332782D+01
  J= 2      0.3570203920D+01    -0.7369966811D+00    -0.1813855083D+02
  J= 3      0.3414416040D+01    -0.3003408607D+01    -0.1173825797D+02
  J= 4      0.3103144243D+01    -0.4470103941D+01    -0.5130837195D+01
  J= 5      0.2704082848D+01    -0.5111202048D+01    -0.5980230354D+01
  J= 6      0.2247197597D+01    -0.5858431831D+01    -0.7461464285D+01
  J= 7      0.1720359516D+01    -0.6790741793D+01    -0.2105971054D+02
  J= 8      0.1045092463D+01    -0.9422152626D+01     0.3876151258D+01
  J= 9      0.2803992897D+00    -0.8937827526D+01    -0.1064157140D+02
  J=10     -0.5195022632D+00    -0.1026749187D+02     0.2620441166D+01
  J=11     -0.1361147121D+01    -0.9940067749D+01    -0.5897783170D+00
  J=12     -0.2192224070D+01    -0.1001376055D+02     0.2453066414D+02
  J=13     -0.2898708639D+01    -0.6948654065D+01     0.2098246695D+01
  J=14     -0.3466611895D+01    -0.6686478141D+01     0.1829860575D+02
  J=15     -0.3928366515D+01    -0.4400067353D+01     0.1082423418D+02
  J=16     -0.4238560998D+01    -0.3047579292D+01     0.2941656810D+02
  J=17     -0.4339335605D+01     0.6280208913D+00     0.9279458398D+01
  J=18     -0.4238729609D+01     0.1787489218D+01     0.7516797823D+01
  J=19     -0.4050713082D+01     0.2726713106D+01     0.1894951599D+01
  J=20     -0.3813716235D+01     0.2963487308D+01    -0.1231930310D+02
  J=21     -0.3630969459D+01     0.1424190386D+01    -0.1368993061D+01
  J=22     -0.3519458869D+01     0.1253134703D+01    -0.1131578552D+02
  J=23     -0.3473961992D+01    -0.1607726983D+00     0.2674956687D+01
  J=24     -0.3473433435D+01     0.1734631398D+00    -0.1388260423D+01
```

6.2.3 DGISMC, RGISMC
Least Squares Interpolation Values and Cubic Spline Coefficients

(1) **Function**

DGISMC or RGISMC obtains least squares approximation cubic spline coefficients and calculates interpolation values at specified points. In addition optimum knot positions are obtained.

(2) **Usage**

Double precision:

```
CALL DGISMC (X, YD, N, XK, NXK, ITMX, XL, FL, M, S, Y, C, IWK, WK1, WK2,
             IERR)
```

Single precision:

```
CALL RGISMC (X, YD, N, XK, NXK, ITMX, XL, FL, M, S, Y, C, IWK, WK1, WK2,
             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}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XK	{D / R}	NXK	Input	Initial estimates of knot positions $XK(i) < XK(i+1), i = 1, \cdots, NXK - 1$ $XK(1) \leq X(1)$ $XK(NXK) \geq X(N)$
				Output	Optimum knot positions
5	NXK	I	1	Input	Number of knots
6	ITMX	I	1	Input	Maximum number of iterations (15 is suitable)
				Output	Actual iteration count
7	XL	{D / R}	M	Input	Abscissa values of points where interpolation values are calculated.
8	FL	{D / R}	M	Output	Least squares approximation cubic spline interpolation values at XL(i)
9	M	I	1	Input	Number of interpolation values

No.	Argument	Type	Size	Input/ Output	Contents
10	S	{D R}	1	Output	Least squares error of cubic spline approximation
11	Y	{D R}	NXK	Output	0th order terms of cubic spline coefficients
12	C	{D R}	See Contents	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t (XK(j) $\leq$ t $<$ XK(j + 1)): f(t) = ((C(3, j) $\times$ D + C(2, j)) $\times$ D + C(1, j)) $\times$ D + Y(j) where: D = t − XK(j) **Size**: 3, (NXK − 1)
13	IWK	I	75	Work	Work area
14	WK1	{D R}	See Contents	Work	Work area **Size**: N $\times$ (NXK + 6)
15	WK2	{D R}	3811	Work	Work area
16	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 2, 2 $\leq$ NXK $\leq$ 28, and ITMX $\leq$ 20

(b) X(1) $<$ X(2) $<$ $\cdots$ $<$ X(N) (Ascending order)

(c) Sample points are distributed throughout the range determined by the end point knots.

(d) XK(1) $<$ XK(2) $<$ $\cdots$ $<$ XK(NXK) (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) $<$ X(1) or XL(i) $>$ X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
1500	Calculations ended after ITMX iterations without obtaining the minimum least squares error (S). (The data is nearly uncorrelated.)	Interpolation is performed using the current spline coefficients and least squares error.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) Different cubic spline coefficients may be obtained and convergence status may be changed depending on the initial estimates of the knot positions.

(b) Usually values Y(1) and Y(NXK) become extrapolated ones using cubic spline coefficients.

(c) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array XK, Y, C and variable NXK must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

to find y_i at each point $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ and perform least squares approximation cubic spline interpolation with four knots $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ using these points as sample points. In addition, obtain the value of the cubic spline function at $xl_j = 0.25 \times k (k = 1, 2, \cdots, 7)$.

(b) Input data
X(i)=x_i, YD(i)=y_i (i = $1, \cdots,$ N), XK(j)=ξ_j (j = $1, \cdots,$ NXK), XL(j)=xl_j (k = $1, \cdots,$ M),
N = 21, NXK = 4, M = 7 and ITMX = 15.

(c) Main program

```
      PROGRAM BGISMC
! *** EXAMPLEOF DGISMC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=21,NXK=4,M=7)
      DIMENSION X(N),YD(N),XK(NXK),XL(M),FL(M),Y(NXK),C(3,NXK-1),&
                IWK(75),WK1(210),WK2(3811)
      READ(5,*) XK
      READ(5,*) XL
      READ(5,*) X
      READ(5,*) YD
      ITMX = 15
      WRITE(6,1000)   N,NXK,M,ITMX,(I,X(I),YD(I),I=1,21)
      WRITE(6,1001)   (I,XL(I),I=1,7)
      WRITE(6,1002)   (K,XK(K),K=1,4)
      CALL DGISMC(X,YD,N,XK,NXK,ITMX,XL,FL,M,S,Y,C,&
                  IWK,WK1,WK2,IERR)
      WRITE(6,1500)   IERR
      WRITE(6,1600)   (I,FL(I),I=1,M)
      WRITE(6,1700)   (I,Y(I),I=1,NXK)
      WRITE(6,1800)
      DO 20   J=1,NXK-1
      WRITE(6,1810)  J,(C(I,J),I=1,3)
   20 CONTINUE
      WRITE(6,1900)   (I,XK(I),I=1,NXK)
      WRITE(6,2000)   ITMX
      WRITE(6,2100)   S
 1000 FORMAT(' ',/,/,5X,'*** DGISMC ***',/,/,6X,'** INPUT **',/,/,8X,'N    =',&
       I3,/,/,8X,'NXK  =',I3,/,/,8X,'M    =',I3,/,/,8X,'ITMX =',I3,/,/,6X,&
       'COORDINATES (X,YD)',/,/,9X,'I    X(I)   YD(I)',/,21(8X,I2,2F7.2,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',4X,&
       'XL(J)',/,7(8X,I2,F8.2,/))
 1002 FORMAT(' ',/,6X,'LOCATIONS OF KNOTS',/,/,9X,'K',4X,&
       'XK(K)',/,4(8X,I2,F8.2,/))
 1500 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1600 FORMAT(' ',/,(9X,'FL(',I1,')=',D24.10))
 1700 FORMAT(' ',/,(9X,'Y(',I2,')=',D24.10))
 1800 FORMAT(' ',/,9X,'C(I,J)',/,27X,'I=1',19X,'I=2',19X,'I=3',/)
 1810 FORMAT(11X,'J=',I2,3D22.10)
 1900 FORMAT(' ',/,8X,'( OPTIMAL LOCATIONS OF KNOTS )',&
             /,/,(9X,'XK(',I1,')=',D22.10))
```

```
 2000 FORMAT(' ',/,8X,'( NUMBER OF ITERATIONS )',/,/,9X,'ITMX=',I2)
 2100 FORMAT(' ',/,8X,'( LEAST SQUARES ERROR )',/,/,9X,'S =',D23.10)
      END
```

(d) Output results

```
*** DGISMC ***

 ** INPUT **

   N    = 21

   NXK  =  4

   M    =  7

   ITMX = 15

 COORDINATES (X,YD)

    I    X(I)  YD(I)
    1   0.00   1.00
    2   0.10   0.90
    3   0.20   0.80
    4   0.30   0.70
    5   0.40   0.60
    6   0.50   0.50
    7   0.60   0.60
    8   0.70   0.70
    9   0.80   0.80
   10   0.90   0.90
   11   1.00   1.00
   12   1.10   0.90
   13   1.20   0.80
   14   1.30   0.70
   15   1.40   0.60
   16   1.50   0.50
   17   1.60   0.40
   18   1.70   0.30
   19   1.80   0.20
   20   1.90   0.10
   21   2.00   0.00


 SPECIFIED POINTS

    J    XL(J)
    1    0.25
    2    0.50
    3    0.75
    4    1.00
    5    1.25
    6    1.50
    7    1.75


 LOCATIONS OF KNOTS

    K    XK(K)
    1    0.00
    2    0.33
    3    1.33
    4    2.00


 ** OUTPUT **

   IERR=   0

    FL(1)=         0.7428298577D+00
    FL(2)=         0.5382544060D+00
    FL(3)=         0.7726883504D+00
    FL(4)=         0.9163869695D+00
    FL(5)=         0.7856196126D+00
    FL(6)=         0.5104972415D+00
    FL(7)=         0.2213361798D+00

    Y( 1)=         0.9837783573D+00
    Y( 2)=         0.5580871037D+00
    Y( 3)=         0.8013007574D+00
    Y( 4)=         0.4845275096D-01

    C(I,J)
                        I=1                   I=2                   I=3

      J= 1     -0.3454003900D+00     -0.3855853842D+01      0.5529117634D+01
      J= 2      0.6481571877D+00      0.5598937312D+01     -0.1431286281D+02
      J= 3      0.1148362744D+01     -0.3141652192D+01      0.1390040784D+01

   ( OPTIMAL LOCATIONS OF KNOTS )

    XK(1)=       0.0000000000D+00
    XK(2)=       0.5700000000D+00
    XK(3)=       0.7735602432D+00
```

```
 XK(4)=      0.2000000000D+01
( NUMBER OF ITERATIONS )
 ITMX= 3
( LEAST SQUARES ERROR )
 S =       0.2888606848D-01
```

6.2.4 DGIDPC, RGIDPC
Derivative Values and Cubic Spline Coefficients

(1) **Function**

DGIDPC or RGIDPC obtains cubic spline coefficients for "not-a-knot" endpoint conditions when endpoint conditions need not be entered and calculates first and second derivatives at specified points. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGIDPC  (X, Y, N, XL, DL, DDL, M, C, IERR)
```

Single precision:

```
CALL RGIDPC  (X, Y, N, XL, DL, DDL, M, C, 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}	N	Input	Abscissa values X(i) of sample point $(X(i), Y(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	XL	{D / R}	M	Input	Abscissa values of points where derivatives of cubic spline function are calculated
5	DL	{D / R}	M	Output	First derivatives of cubic spline function at XL(i) $DL(i) = (3.0 \times C(3,j) \times D + 2.0 \times C(2,j)) \times D + C(1,j)$ where: $X(j) \leq XL(i) < X(j+1)$ $D = XL(i) - X(j)$
6	DDL	{D / R}	M	Output	Second derivatives of cubic spline function at XL(i) $DDL(i) = 6.0 \times C(3,j) \times D + 2.0 \times C(2,j)$ where: $X(j) \leq XL(i) < X(j+1)$ $D = XL(i) - X(j)$
7	M	I	1	Input	Number of points where derivatives are calculated
8	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation

$$y_i = x_i e^{-4.0x_i}$$

to find y_i $(i = 1, 2, \cdots, 9)$ at each point $\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$ and perform cubic spline interpolation using these points as sample points. In addition, obtain the first and second derivatives of the cubic spline function at the uniformly spaced points $xl_j = 0.1\times(j-1)(j = 1, 2, \cdots, 10)$.

(b) Input data
X(i)=x_i, Y(i)=y_i (i = $1, \cdots$, N), XL(j)=xl_j (j = $1, \cdots$, M), N = 9 and M = 10.

(c) Main program

```
      PROGRAM BGIDPC
! *** EXAMPLE OF DGIDPC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=9,M=10)
      DIMENSION X(N),Y(N),XL(M),DL(M),DDL(M),C(3,N-1)
      READ(5,*) X
      READ(5,*) XL
      READ(5,*) Y
      WRITE(6,1000)   N,M,(I,X(I),Y(I),I=1,9)
      WRITE(6,1001)   (I,XL(I),I=1,10)
      CALL DGIDPC(X,Y,N,XL,DL,DDL,M,C,IERR)
      WRITE(6,1400)  IERR
      WRITE(6,1500)  (I,DL(I),I=1,M)
      WRITE(6,1600)  (I,DDL(I),I=1,M)
      WRITE(6,1700)
      WRITE(6,1800)
      DO 20  J=1,N-1
      WRITE(6,1810)  J,(C(I,J),I=1,3)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIDPC ***',/,/,6X,'** INPUT **',/,/,8X,'N  =',&
             I3,/,/,8X,'M  =',I3,/,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',4X,&
             'X(I)',5X,'Y(I)',/,9(8X,I2,F7.2,F11.4,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',3X,'XL(J)',/,&
             10(8X,I2,F7.2,/))
 1400 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1500 FORMAT(' ',/,8X,'( THE VALUE OF THE FIRST DERIVATIVE )',&
             /,/,(9X,'DL(',I2,')=',D20.10))
 1600 FORMAT(' ',/,8X,'( THE VALUE OF THE SECOND DERIVATIVE )',&
             /,/,(9X,'DDL(',I2,')=',D19.10))
 1700 FORMAT(' ',/,8X,'( SPLINE COEFFICIENTS )')
 1800 FORMAT(' ',/,9X,'C(I,J)',/,26X,'I=1',19X,'I=2',19X,'I=3',/)
 1810 FORMAT(11X,'J=',I1,3D22.10)
      END
```

(d) Output results

```
*** DGIDPC ***

 ** INPUT **

   N  =  9

   M  = 10

 COORDINATES (X,Y)

    I    X(I)     Y(I)
    1   0.00     0.0000
    2   0.10     0.0670
    3   0.23     0.0917
    4   0.34     0.0873
    5   0.47     0.0717
    6   0.59     0.0557
    7   0.73     0.0394
    8   0.92     0.0232
    9   1.00     0.0183


 SPECIFIED POINTS

    J   XL(J)
    1   0.10
    2   0.20
    3   0.30
    4   0.40
    5   0.50
    6   0.60
    7   0.70
    8   0.80
    9   0.90
   10   1.00


 ** OUTPUT **

   IERR=   0

   ( THE VALUE OF THE FIRST DERIVATIVE )

    DL( 1)=    0.4137208827D+00
    DL( 2)=    0.8417597547D-01
    DL( 3)=   -0.5953643640D-01
    DL( 4)=   -0.1215767100D+00
    DL( 5)=   -0.1353261758D+00
    DL( 6)=   -0.1265337580D+00
    DL( 7)=   -0.1094390792D+00
    DL( 8)=   -0.8975302008D-01
    DL( 9)=   -0.7142823200D-01
    DL(10)=   -0.5471523507D-01

   ( THE VALUE OF THE SECOND DERIVATIVE )

    DDL( 1)=  -0.4393529036D+01
    DDL( 2)=  -0.2197369109D+01
    DDL( 3)=  -0.9664519622D+00
    DDL( 4)=  -0.3577823218D+00
    DDL( 5)=   0.1501832180D-01
    DDL( 6)=   0.1511701722D+00
    DDL( 7)=   0.1907234048D+00
    DDL( 8)=   0.1913068365D+00
    DDL( 9)=   0.1751889251D+00
    DDL(10)=   0.1590710136D+00

   ( SPLINE COEFFICIENTS )

    C(I,J)
                    I=1                  I=2                  I=3

      J=1      0.9628817827D+00    -0.3294844482D+01     0.3660266545D+01
      J=2      0.4137208827D+00    -0.2196764518D+01     0.3660266545D+01
      J=3      0.2813762187D-01    -0.7692605656D+00     0.1362069450D+01
      J=4     -0.9165658153D-01    -0.3197776471D+00     0.7827027013D+00
      J=5     -0.1351157428D+00    -0.1452359362D-01     0.2448083836D+00
      J=6     -0.1280256831D+00     0.7360742448D-01     0.6592205430D-01
      J=7     -0.1035393875D+00     0.1012946873D+00    -0.2686318577D-01
      J=8     -0.6795668932D-01     0.8598267140D-01    -0.2686318577D-01
```

6.2.5 DGIDSC, RGIDSC
Smoothed Derivative Values and Cubic Spline Coefficients

(1) **Function**

DGIDSC or RGIDSC automatically obtains the optimum smoothed spline coefficients and the first and second derivative at specified points. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

```
CALL DGIDSC  (X, YD, N, XL, DL, DDL, M, Y, C, ISW, WK, IERR)
```

Single precision:

```
CALL RGIDSC  (X, YD, N, XL, DL, DDL, M, Y, C, ISW, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point (X(i), YD(i)), i = 1, ···, N (X(i) < X(i + 1), i ≠ N)
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XL	{D / R}	M	Input	Abscissa values of points where derivatives of cubic spline function are calculated
5	DL	{D / R}	M	Output	First derivatives of cubic spline function at XL(i) DL(i) = (3.0×C(3, j)×D+2.0×C(2, j))×D + C(1, j) where: X(j) ≤ XL(i) < X(j + 1) D = XL(i) − X(j)
6	DDL	{D / R}	M	Output	Second derivatives of cubic spline function at XL(i) DDL(i) = 6.0 × C(3, j) × D + 2.0 × C(2, j) where: X(j) ≤ XL(i) < X(j + 1) D = XL(i) − X(j)
7	M	I	1	Input	Number of points where derivatives are calculated
8	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
9	C	{D / R}	3, (N − 1)	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t (X(j) ≤ t < X(j + 1)): f(t) = ((C(3, j) × D + C(2, j)) × D + C(1, j)) × D + Y(j) where: D = t − X(j)
10	ISW	I	1	Input	Processing switch ISW=1: When the abscissa value spacing may not be uniform. When sample points are nearly uncorrelated, limit value N up to 20. ISW=2: When the abscissa value spacing is uniform.
11	WK	{D / R}	See Contents	Work	Work area Size: N × (2 × N + 4) (when ISW=1) 6 × N (when ISW=2)
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 4

(b) X(1) < X(2) < $\cdots$ < X(N) (Ascending order)

(c) If ISW=2, abscissa values must be uniformly spaced.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	The minimum cross-validation value could not be found. (The data is uncorrelated.)	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation

$$y_i = \sin(3\pi x_i/2) + e_i \ (e_i : \text{Uniform random number in the interval } [-0.2, 0.2])$$

to find values y_i at each point $x_i = (i-1)/24$ $(i = 1, 2, \cdots, 25)$ and perform smoothed cubic spline interpolation using these points as sample points. In addition, obtain the first and second derivatives of the cubic spline function at the uniformly spaced points $xl_j = 0.1 \times (j-1)(j = 1, 2, \cdots, 10)$.

(b) Input data
X(i)=x_i, YD(i)=y_i (i = $1, \cdots,$ N), XL(j)=xl_j (j = $1, \cdots,$ M), N=25, ISW=2 and M=10.

(c) Main program

```
      PROGRAM BGIDSC
! *** EXAMPLE OF DGIDSC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=25,M=10,ISW=2)
      DIMENSION X(N),YD(N),XL(M),DL(M),DDL(M),&
                Y(N),C(3,N-1),WK(6*N)
      READ(5,*) X
      READ(5,*) YD
      READ(5,*) XL
      WRITE(6,1000)   N,M,ISW,(I,X(I),YD(I),I=1,25)
      WRITE(6,1001)   (I,XL(I),I=1,10)
      CALL DGIDSC(X,YD,N,XL,DL,DDL,M,Y,C,ISW,WK,IERR)
      WRITE(6,1400)   IERR
```

```
      WRITE(6,1700)    (I,DL(I),I=1,M)
      WRITE(6,1800)    (I,DDL(I),I=1,M)
      WRITE(6,1500)    (I,Y(I),I=1,N)
      WRITE(6,1600)
      DO 20  J=1,N-1
      WRITE(6,1610)  J,(C(I,J),I=1,3)
   20 CONTINUE
 1000 FORMAT(' ',/,/,5X,'*** DGIDSC ***',/,/,6X,'** INPUT **',/,/,8X,&
      'N   =',I3,/,/,8X,'M   =',I3,/,/,8X,'ISW =',I3,/,/,6X,'COORDINATES (X,YD)',&
       /,/,9X,'I',5X,'X(I)',7X,'YD(I)',/,25(8X,I2,F10.4,F11.4,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',3X,&
       'XL(J)',/,10(8X,I2,F7.2,/))
 1400 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1500 FORMAT(' ',/,(9X,'Y(',I2,')=',D20.10))
 1600 FORMAT(' ',/,9X,'C(I,J)',/,27X,'I=1',19X,'I=2',19X,'I=3',/)
 1610 FORMAT(11X,'J=',I2,3D22.10)
 1700 FORMAT(' ',/,8X,'( THE VALUE OF THE FIRST DERIVATIVE )',&
             /,/,(9X,'DL(',I2,')=',D20.10))
 1800 FORMAT(' ',/,8X,'( THE VALUE OF THE SECOND DERIVATIVE )',&
             /,/,(9X,'DDL(',I2,')=',D19.10))
      END
```

(d) Output results

```
*** DGIDSC ***

 ** INPUT **

   N   = 25

   M   = 10

   ISW =  2

 COORDINATES (X,YD)

    I     X(I)       YD(I)
    1    0.0000     0.0481
    2    0.0416     0.1472
    3    0.0833     0.4416
    4    0.1250     0.5794
    5    0.1666     0.6412
    6    0.2083     0.7395
    7    0.2500     0.7260
    8    0.2916     1.0909
    9    0.3333     0.8045
   10    0.3750     1.0209
   11    0.4166     0.8506
   12    0.4583     1.0020
   13    0.5000     0.5168
   14    0.5416     0.6915
   15    0.5833     0.3485
   16    0.6250     0.3854
   17    0.6666    -0.1024
   18    0.7083    -0.1352
   19    0.7500    -0.3388
   20    0.7916    -0.5710
   21    0.8333    -0.5254
   22    0.8750    -0.8405
   23    0.9166    -0.7953
   24    0.9583    -1.0837
   25    1.0000    -1.1848


 SPECIFIED POINTS

    J   XL(J)
    1   0.00
    2   0.10
    3   0.20
    4   0.30
    5   0.40
    6   0.50
    7   0.60
    8   0.70
    9   0.80
   10   0.90


 ** OUTPUT **

   IERR=   0

   ( THE VALUE OF THE FIRST DERIVATIVE )

    DL( 1)=    0.3600899832D+01
    DL( 2)=    0.3304281144D+01
    DL( 3)=    0.2342640634D+01
    DL( 4)=    0.8876208027D+00
    DL( 5)=   -0.1027963530D+01
    DL( 6)=   -0.2898708639D+01
    DL( 7)=   -0.4066272453D+01
    DL( 8)=   -0.4266328432D+01
    DL( 9)=   -0.3766537398D+01
```

```
 DL(10)=   -0.3478019232D+01

( THE VALUE OF THE SECOND DERIVATIVE )

 DDL( 1)=   0.4264962400D-13
 DDL( 2)=  -0.7182990662D+01
 DDL( 3)=  -0.1142084226D+02
 DDL( 4)=  -0.1864894723D+02
 DDL( 5)=  -0.2014191757D+02
 DDL( 6)=  -0.1389730813D+02
 DDL( 7)=  -0.7715546441D+01
 DDL( 8)=   0.3115645246D+01
 DDL( 9)=   0.5306081741D+01
 DDL(10)=   0.8089015773D+00

 Y( 1)=    0.9502274321D-01
 Y( 2)=    0.2445740596D+00
 Y( 3)=    0.3906840363D+00
 Y( 4)=    0.5268362807D+00
 Y( 5)=    0.6479571394D+00
 Y( 6)=    0.7512835941D+00
 Y( 7)=    0.8341775208D+00
 Y( 8)=    0.8925288529D+00
 Y( 9)=    0.9199921903D+00
 Y(10)=    0.9153973040D+00
 Y(11)=    0.8761381156D+00
 Y(12)=    0.8021604666D+00
 Y(13)=    0.6952556081D+00
 Y(14)=    0.5626220076D+00
 Y(15)=    0.4079605341D+00
 Y(16)=    0.2374932368D+00
 Y(17)=    0.5779584896D-01
 Y(18)=   -0.1211775851D+00
 Y(19)=   -0.2940767779D+00
 Y(20)=   -0.4579219747D+00
 Y(21)=   -0.6125125122D+00
 Y(22)=   -0.7613707264D+00
 Y(23)=   -0.9065999263D+00
 Y(24)=   -0.1051376070D+01
 Y(25)=   -0.1195843966D+01

 C(I,J)
                  I=1                  I=2                  I=3

   J= 1      0.3600899832D+01     0.2132481200D-13    -0.5898332782D+01
   J= 2      0.3570203920D+01    -0.7369966811D+00    -0.1813855083D+02
   J= 3      0.3414416040D+01    -0.3003408607D+01    -0.1173825797D+02
   J= 4      0.3103144243D+01    -0.4470103941D+01    -0.5130837195D+01
   J= 5      0.2704082848D+01    -0.5111202048D+01    -0.5980230354D+01
   J= 6      0.2247197597D+01    -0.5858431831D+01    -0.7461464285D+01
   J= 7      0.1720359516D+01    -0.6790741793D+01    -0.2105971054D+02
   J= 8      0.1045092463D+01    -0.9422152626D+01     0.3876151258D+01
   J= 9      0.2803992897D+00    -0.8937827526D+01    -0.1064157140D+02
   J=10     -0.5195022632D+00    -0.1026749187D+02     0.2620441166D+01
   J=11     -0.1361147121D+01    -0.9940067749D+01    -0.5897783170D+00
   J=12     -0.2192224070D+01    -0.1001376055D+02     0.2453066414D+02
   J=13     -0.2898708639D+01    -0.6948654065D+01     0.2098246695D+01
   J=14     -0.3466611895D+01    -0.6686478141D+01     0.1829860575D+02
   J=15     -0.3928366515D+01    -0.4400067353D+01     0.1082423418D+02
   J=16     -0.4238560998D+01    -0.3047579292D+01     0.2941656810D+02
   J=17     -0.4339335605D+01     0.6280208913D+00     0.9279458398D+01
   J=18     -0.4238729609D+01     0.1787489218D+01     0.7516797823D+01
   J=19     -0.4050713082D+01     0.2726713106D+01     0.1894951599D+01
   J=20     -0.3813716235D+01     0.2963487308D+01    -0.1231930310D+02
   J=21     -0.3630969459D+01     0.1424190386D+01    -0.1368993061D+01
   J=22     -0.3519458869D+01     0.1253134703D+01    -0.1131578552D+02
   J=23     -0.3473961992D+01    -0.1607726983D+00     0.2674956687D+01
   J=24     -0.3473433435D+01     0.1734631398D+00    -0.1388260423D+01
```

6.2.6 DGIDMC, RGIDMC
Least Squares Method Derivative Values and Cubic Spline Coefficients

(1) **Function**

DGIDMC or RGIDMC obtains least squares approximation cubic spline coefficients and calculates first and second derivatives at specified points. In addition optimum knot positions are obtained.

(2) **Usage**

Double precision:

CALL DGIDMC (X, YD, N, XK, NXK, ITMX, XL, DL, DDL, M, S, Y, C, IWK, WK1, WK2, IERR)

Single precision:

CALL RGIDMC (X, YD, N, XK, NXK, ITMX, XL, DL, DDL, M, S, Y, C, IWK, WK1, WK2, 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}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XK	{D / R}	NXK	Input	Initial estimates of knot positions $XK(i) < XK(i+1), i = 1, \cdots, NXK - 1$ $XK(1) \leq X(1)$ $XK(NXK) \geq X(N)$
				Output	Optimum knot positions
5	NXK	I	1	Input	Number of knots
6	ITMX	I	1	Input	Maximum number of iterations (15 is suitable)
				Output	Actual iteration count
7	XL	{D / R}	M	Input	Abscissa values of points where derivatives of cubic spline function are calculated

No.	Argument	Type	Size	Input/ Output	Contents
8	DL	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	M	Output	First derivatives of cubic spline function at XL(i) DL(i) = $(3.0\times\text{C}(3,\text{j})\times\text{D}+2.0\times\text{C}(2,\text{j}))\times\text{D}+\text{C}(1,\text{j})$ where: $\text{XK}(\text{j})\le\text{XL}(\text{i})<\text{XK}(\text{j}+1)$ $\text{D}=\text{XL}(\text{i})-\text{XK}(\text{j})$
9	DDL	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	M	Output	Second derivatives of cubic spline function at XL(i) $\text{DDL}(\text{i})=6.0\times\text{C}(3,\text{j})\times\text{D}+2.0\times\text{C}(2,\text{j})$ where: $\text{XK}(\text{j})\le\text{XL}(\text{i})<\text{XK}(\text{j}+1)$ $\text{D}=\text{XL}(\text{i})-\text{XK}(\text{j})$
10	M	I	1	Input	Number of points where derivatives are calculated
11	S	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	1	Output	Least squares error of cubic spline approximation
12	Y	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	NXK	Output	0th order terms of cubic spline coefficients
13	C	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($\text{XK}(\text{j})\le\text{t}<\text{XK}(\text{j}+1)$): $\text{f}(\text{t})=((\text{C}(3,\text{j})\times\text{D}+\text{C}(2,\text{j}))\times\text{D}+\text{C}(1,\text{j}))\times\text{D}+\text{Y}(\text{j})$ where: $\text{D}=\text{t}-\text{XK}(\text{j})$ **Size**: $3,(\text{NXK}-1)$
14	IWK	I	75	Work	Work area
15	WK1	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $\text{N}\times(\text{NXK}+6)$
16	WK2	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	3811	Work	Work area
17	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\text{N}\ge 2$, $2\le\text{NXK}\le 28$, and $\text{ITMX}\le 20$

(b) $\text{X}(1)<\text{X}(2)<\cdots<\text{X}(\text{N})$ (Ascending order)

(c) Sample points are distributed throughout the range determined by the endpoint knots.

(d) $\text{XK}(1)<\text{XK}(2)<\cdots<\text{XK}(\text{NXK})$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
1500	Calculations ended after ITMX iterations without obtaining the minimum least squares error (S). (The data is nearly uncorrelated.)	Interpolation is performed using the current cubic spline coefficients and least squares error.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) Different cubic spline coefficients may be obtained and convergence status may be changed depending on the initial estimates of the knot positions.

(b) Usually values Y(1) and Y(NXK) become extrapolated ones using cubic spline coefficients.

(c) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array XK, Y, C and variable NXK must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

to find y_i at each point $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ and perform least squares approximation cubic spline interpolation with four knots $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ using these points as sample points. In addition, obtain the first and second derivatives of the cubic spline function at the uniformly spaced points $xl_j = 0.25 \times k (k = 1, 2, \cdots, 7)$.

(b) Input data
X(i) = $\mathrm{x_i}$, YD(i) = $\mathrm{y_i}$ (i = $1, \cdots,$ N),
XK(j) = ξ_{j} (j = $1, \cdots,$ NXK),
XL(j) = $\mathrm{xl_j}$ (k = $1, \cdots,$ M),
N = 21, NXK = 4, M = 7 and ITMX = 15.

(c) Main program

```
      PROGRAM BGIDMC
!  *** EXAMPLE OF DGIDMC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=21,NXK=4,M=7)
      DIMENSION X(N),YD(N),XK(NXK),XL(M),DL(M),DDL(M),&
                Y(NXK),C(3,NXK-1),IWK(75),WK1(210),WK2(3811)
      READ(5,*) XK
      READ(5,*) XL
      READ(5,*) X
      READ(5,*) YD
      ITMX = 15
      WRITE(6,1000)   N,NXK,M,ITMX,(I,X(I),YD(I),I=1,21)
      WRITE(6,1001)   (I,XL(I),I=1,7)
      WRITE(6,1002)   (K,XK(K),K=1,4)
      CALL DGIDMC(X,YD,N,XK,NXK,ITMX,XL,DL,DDL,M,S,&
                  Y,C,IWK,WK1,WK2,IERR)
      WRITE(6,1500) IERR
      WRITE(6,1800) (I,DL(I),I=1,M)
      WRITE(6,1900) (I,DDL(I),I=1,M)
      WRITE(6,1600) (I,Y(I),I=1,NXK)
      WRITE(6,1700)
      DO 20  J=1,NXK-1
      WRITE(6,1710)  J,(C(I,J),I=1,3)
   20 CONTINUE
      WRITE(6,2000) (I,XK(I),I=1,NXK)
      WRITE(6,2100) ITMX
      WRITE(6,2200) S
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIDMC ***',/,/,6X,'** INPUT **',/,/,8X,'N    =',&
       I3,/,/,8X,'NXK  =',I3,/,/,8X,'M    =',I3,/,/,8X,'ITMX =',I3,/,/,6X,&
       'COORDINATES (X,YD)',/,/,9X,'I    X(I)  YD(I)',/,21(8X,I2,2F7.2,/))
 1001 FORMAT(' ',/,6X,'SPECIFIED POINTS',/,/,9X,'J',4X,&
       'XL(J)',/,7(8X,I2,F8.2,/))
 1002 FORMAT(' ',/,6X,'LOCATIONS OF KNOTS',/,/,9X,'K',4X,&
       'XK(K)',/,4(8X,I2,F8.2,/))
 1500 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1600 FORMAT(' ',/,(9X,'Y(',I2,')=',D24.10))
 1700 FORMAT(9X,'C(I,J)',/,26X,'I=1',19X,'I=2',19X,'I=3',/)
 1710 FORMAT(11X,'J=',I2,3D22.10)
 1800 FORMAT(' ',/,8X,'( THE VALUE OF THE FIRST DERIVATIVE )',&
             /,/,(9X,'DL(',I2,')=',D21.10))
 1900 FORMAT(' ',/,8X,'( THE VALUE OF THE SECOND DERIVATIVE )',&
             /,/,(9X,'DDL(',I2,')=',D20.10))
 2000 FORMAT(' ',/,8X,'( OPTIMAL LOCATIONS OF KNOTS )',&
             /,/,(9X,'XK(',I1,')=',D24.10))
 2100 FORMAT(' ',/,8X,'( NUMBER OF ITERATIONS )',/,/,9X,'ITMX=',I2)
 2200 FORMAT(' ',/,8X,'( LEAST SQUARES ERROR )',/,/,9X,'S =',D24.10)
      END
```

(d) Output results

```
 *** DGIDMC ***

  ** INPUT **

    N    = 21

    NXK  =  4

    M    =  7

    ITMX = 15

  COORDINATES (X,YD)

     I    X(I)  YD(I)
     1   0.00   1.00
     2   0.10   0.90
     3   0.20   0.80
     4   0.30   0.70
     5   0.40   0.60
     6   0.50   0.50
     7   0.60   0.60
     8   0.70   0.70
     9   0.80   0.80
    10   0.90   0.90
    11   1.00   1.00
    12   1.10   0.90
    13   1.20   0.80
    14   1.30   0.70
    15   1.40   0.60
    16   1.50   0.50
    17   1.60   0.40
    18   1.70   0.30
    19   1.80   0.20
    20   1.90   0.10
    21   2.00   0.00


  SPECIFIED POINTS
```

```
   J    XL(J)
   1    0.25
   2    0.50
   3    0.75
   4    1.00
   5    1.25
   6    1.50
   7    1.75


LOCATIONS OF KNOTS

   K    XK(K)
   1    0.00
   2    0.33
   3    1.33
   4    2.00


** OUTPUT **

  IERR=   0

  ( THE VALUE OF THE FIRST DERIVATIVE )

   DL( 1)=    -0.1236617755D+01
   DL( 2)=    -0.5441600670D-01
   DL( 3)=     0.1272564355D+01
   DL( 4)=    -0.6060430150D-01
   DL( 5)=    -0.8986570050D+00
   DL( 6)=    -0.1215444415D+01
   DL( 7)=    -0.1010966530D+01

  ( THE VALUE OF THE SECOND DERIVATIVE )

   DDL( 1)=    0.5819687665D+00
   DDL( 2)=    0.8875645218D+01
   DDL( 3)=   -0.4260017215D+01
   DDL( 4)=   -0.4394741402D+01
   DDL( 5)=   -0.2309680226D+01
   DDL( 6)=   -0.2246190502D+00
   DDL( 7)=    0.1860442126D+01

   Y( 1)=        0.9837783573D+00
   Y( 2)=        0.5580871037D+00
   Y( 3)=        0.8013007574D+00
   Y( 4)=        0.4845275096D-01
   C(I,J)
                    I=1                   I=2                   I=3

     J= 1     -0.3454003900D+00     -0.3855853842D+01      0.5529117634D+01
     J= 2      0.6481571877D+00      0.5598937312D+01     -0.1431286281D+02
     J= 3      0.1148362744D+01     -0.3141652192D+01      0.1390040784D+01

  ( OPTIMAL LOCATIONS OF KNOTS )

   XK(1)=        0.0000000000D+00
   XK(2)=        0.5700000000D+00
   XK(3)=        0.7735602432D+00
   XK(4)=        0.2000000000D+01

  ( NUMBER OF ITERATIONS )

   ITMX= 3

  ( LEAST SQUARES ERROR )

   S =        0.2888606848D-01
```

6.2.7 DGIIPC, RGIIPC
Integral Values and Cubic Spline Coefficients

(1) **Function**

DGIIPC or RGIIPC obtains cubic spline coefficients for "not-a-knot" endpoint conditions when endpoint conditions need not be entered and calculates the integral over the specified range. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGIIPC  (X, Y, N, A, B, Q, C, IERR)
```

Single precision:

```
CALL RGIIPC  (X, Y, N, A, B, Q, C, 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 }	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)) for $i = 1, \cdots, N$ ($X(i) < X(i+1), i \neq N$)
2	Y	{ D / R }	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	A	{ D / R }	1	Input	Lower limit of the integration range
5	B	{ D / R }	1	Input	Upper limit of the integration range
6	Q	{ D / R }	1	Output	Integral value of cubic spline function over abscissa interval [A, B]
7	C	{ D / R }	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 2

(b) X(1) $<$ X(2) $<\cdots<$ X(N) (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	A or B was outside the interpolation interval range.	A function extrapolated from the cubic spline coefficients at the endpoints is integrated.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\left\{\begin{matrix}\text{DGISCX}\\\text{RGISCX}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{DGIDCY}\\\text{RGIDCY}\end{matrix}\right\}$ or 6.2.20 $\left\{\begin{matrix}\text{DGIICZ}\\\text{RGIICZ}\end{matrix}\right\}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem

Use the equation $y_i = x_i e^{-4.0x_i}$ to find y_i $(i = 1, 2, \cdots, 9)$ at each point
$\{x_i\} = \{0.0, 0.1, 0.23, 0.34, 0.47, 0.59, 0.73, 0.92, 1.0\}$
and perform cubic spline interpolation using these points as sample points. In addition, obtain the integral value of the cubic spline function over abscissa interval $[0.2, 0.5]$.

(b) Input data

X(i)=x_i, Y(i)=y_i (i = $1, \cdots,$ N), A=0.2, B=0.5 and N=9.

(c) Main program

```
      PROGRAM BGIIPC
! *** EXAMPLE OF DGIIPC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=9,A=0.2,B=0.5)
      DIMENSION X(N),Y(N),C(3,N-1)
      READ(5,*) X
      READ(5,*) Y
      WRITE(6,1000)   N,A,B,(I,X(I),Y(I),I=1,9)
      CALL DGIIPC(X,Y,N,A,B,Q,C,IERR)
      WRITE(6,1200)  IERR
      WRITE(6,1300)  Q
      WRITE(6,1400)
      WRITE(6,1500)
      DO 20  J=1,N-1
      WRITE(6,1510)  J,(C(I,J),I=1,3)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIIPC ***',/,/,6X,'** INPUT **',/,/,8X,'N  =',I3,&
        /,/,8X,'LIMITS OF INTEGRATION    A =',F8.4,/,/,33X,'B =',F8.4,/,/,6X,&
        'COORDINATES (X,Y)',/,/,9X,'I',4X,&
        'X(I)',5X,'Y(I)',/,9(8X,I2,F7.2,F11.4,/))
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1300 FORMAT(' ',/,8X,'( INTEGRAL )',/,/,9X,'Q=',D24.10)
 1400 FORMAT(' ',/,8X,'( SPLINE COEFFICIENTS )')
 1500 FORMAT(' ',/,9X,'C(I,J)',/,26X,'I=1',19X,'I=2',19X,'I=3',/)
 1510 FORMAT(11X,'J=',I1,3D22.10)
      END
```

(d) Output results

```
*** DGIIPC ***

 ** INPUT **

   N  =  9

   LIMITS OF INTEGRATION    A =  0.2000

                            B =  0.5000

 COORDINATES (X,Y)

    I    X(I)     Y(I)
    1   0.00     0.0000
    2   0.10     0.0670
    3   0.23     0.0917
    4   0.34     0.0873
    5   0.47     0.0717
    6   0.59     0.0557
    7   0.73     0.0394
    8   0.92     0.0232
    9   1.00     0.0183


 ** OUTPUT **

   IERR=   0

   ( INTEGRAL )

    Q=        0.2518278750D-01

   ( SPLINE COEFFICIENTS )

    C(I,J)
                      I=1                   I=2                   I=3

      J=1      0.9628817827D+00     -0.3294844482D+01      0.3660266545D+01
      J=2      0.4137208827D+00     -0.2196764518D+01      0.3660266545D+01
      J=3      0.2813762187D-01     -0.7692605656D+00      0.1362069450D+01
      J=4     -0.9165658153D-01     -0.3197776471D+00      0.7827027013D+00
      J=5     -0.1351157428D+00     -0.1452359362D-01      0.2448083836D+00
      J=6     -0.1280256831D+00      0.7360742448D-01      0.6592205430D-01
      J=7     -0.1035393875D+00      0.1012946873D+00     -0.2686318577D-01
      J=8     -0.6795668932D-01      0.8598267140D-01     -0.2686318577D-01
```

6.2.8 DGIISC, RGIISC
Smoothed Integral Value and Cubic Spline Coefficients

(1) **Function**

DGIISC or RGIISC automatically obtains the optimum smoothed spline coefficients and calculates the value of the integral over the specified range. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

```
CALL DGIISC  (X, YD, N, A, B, Q, Y, C, ISW, WK, IERR)
```

Single precision:

```
CALL RGIISC  (X, YD, N, A, B, Q, Y, C, ISW, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	A	{D / R}	1	Input	Lower limit of the integration range
5	B	{D / R}	1	Input	Upper limit of the integration range
6	Q	{D / R}	1	Output	Integral value of cubic spline function over abscissa interval $[A, B]$
7	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
8	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
9	ISW	I	1	Input	Processing switch ISW=1: When the abscissa value spacing may not be uniform. When sample points are nearly uncorrelated, limit value N up to 20. ISW=2: When the abscissa value spacing is uniform.
10	WK	{D / R}	See Contents	Work	Work area **Size**: $N \times (2 \times N + 4)$ (when ISW=1) $6 \times N$ (when ISW=2)
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(c) If ISW=2, abscissa values must be uniformly spaced.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	A or B was outside the interpolation interval range.	A function extrapolated from the cubic spline coefficients at the endpoints is integrated.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	The minimum cross-validation value could not be found. (The data is uncorrelated.)	

(6) **Notes**

(a) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\begin{Bmatrix}\text{DGISCX}\\ \text{RGISCX}\end{Bmatrix}$, 6.2.19 $\begin{Bmatrix}\text{DGIDCY}\\ \text{RGIDCY}\end{Bmatrix}$ or 6.2.20 $\begin{Bmatrix}\text{DGIICZ}\\ \text{RGIICZ}\end{Bmatrix}$, respectively. In this case, contents of array X, Y, C and variable N must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem
Use the equation $y_i = \sin(3\pi x_i/2) + e_i$ (e_i:Uniform random number in the interval $[-0.2, 0.2]$) to find values y_i at each point $x_i = (i-1)/24$ ($i = 1, 2, \cdots, 25$) and perform smoothed cubic spline interpolation using these points as sample points. In addition, obtain the integral value of the cubic spline function over abscissa interval $[0.2, 0.5]$.

(b) Input data
X(i) = x_i, YD(i) = y_i (i = $1, \cdots, N$), A = 0.2, B = 0.5, N = 25, ISW = 2 and M = 10.

(c) Main program

```
      PROGRAM BGIISC
! *** EXAMPLE OF DGIISC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=25,ISW=2,A=0.2,B=0.5)
      DIMENSION X(N),YD(N),Y(N),C(3,N-1),WK(6*N)
      READ(5,*) X
      READ(5,*) YD
      WRITE(6,1000)   N,ISW,A,B,(I,X(I),YD(I),I=1,25)
      CALL DGIISC(X,YD,N,A,B,Q,Y,C,ISW,WK,IERR)
      WRITE(6,1200)  IERR
      WRITE(6,1300)  Q
      WRITE(6,1400)  (I,Y(I),I=1,N)
      WRITE(6,1500)
      DO 20  J=1,N-1
      WRITE(6,1510)  J,(C(I,J),I=1,3)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIISC ***',/,/,6X,'** INPUT **',&
       /,/,8X,'N   =',I3,/,/,8X,'ISW =',I3,&
       /,/,8X,'LIMITS OF INTEGRATION    A =',F6.2,&
       /,/,33X,'B =',F6.2,/,/,6X,'COORDINATES (X,YD)',&
       /,/,9X,'I',5X,'X(I)',7X,'YD(I)',/,25(8X,I2,F10.4,F11.4,/))
 1200 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1300 FORMAT(' ',/,8X,'( INTEGRAL )',/,/,9X,'Q=',D24.10)
 1400 FORMAT(' ',/,(9X,'Y(',I2,')=',D24.10))
 1500 FORMAT(' ',/,9X,'C(I,J)',/,27X,'I=1',19X,'I=2',19X,'I=3',/)
 1510 FORMAT(11X,'J=',I2,3D22.10)
      END
```

(d) Output results

```
*** DGIISC ***

 ** INPUT **

   N   = 25

   ISW =  2

   LIMITS OF INTEGRATION    A =  0.20

                            B =  0.50

 COORDINATES (X,YD)

    I     X(I)       YD(I)
    1    0.0000     0.0481
    2    0.0416     0.1472
    3    0.0833     0.4416
    4    0.1250     0.5794
    5    0.1666     0.6412
    6    0.2083     0.7395
    7    0.2500     0.7260
    8    0.2916     1.0909
    9    0.3333     0.8045
   10    0.3750     1.0209
   11    0.4166     0.8506
   12    0.4583     1.0020
   13    0.5000     0.5168
   14    0.5416     0.6915
   15    0.5833     0.3485
   16    0.6250     0.3854
   17    0.6666    -0.1024
   18    0.7083    -0.1352
   19    0.7500    -0.3388
   20    0.7916    -0.5710
   21    0.8333    -0.5254
   22    0.8750    -0.8405
   23    0.9166    -0.7953
   24    0.9583    -1.0837
   25    1.0000    -1.1848


 ** OUTPUT **

   IERR=   0

   ( INTEGRAL )

    Q=        0.2554096170D+00

    Y( 1)=        0.9502274321D-01
    Y( 2)=        0.2445740596D+00
    Y( 3)=        0.3906840363D+00
    Y( 4)=        0.5268362807D+00
    Y( 5)=        0.6479571394D+00
    Y( 6)=        0.7512835941D+00
    Y( 7)=        0.8341775208D+00
    Y( 8)=        0.8925288529D+00
    Y( 9)=        0.9199921903D+00
    Y(10)=        0.9153973040D+00
    Y(11)=        0.8761381156D+00
    Y(12)=        0.8021604666D+00
    Y(13)=        0.6952556081D+00
    Y(14)=        0.5626220076D+00
    Y(15)=        0.4079605341D+00
    Y(16)=        0.2374932368D+00
    Y(17)=        0.5779584896D-01
    Y(18)=       -0.1211775851D+00
    Y(19)=       -0.2940767779D+00
    Y(20)=       -0.4579219747D+00
    Y(21)=       -0.6125125122D+00
    Y(22)=       -0.7613707264D+00
    Y(23)=       -0.9065999263D+00
    Y(24)=       -0.1051376070D+01
    Y(25)=       -0.1195843966D+01

    C(I,J)
                        I=1                  I=2                  I=3

      J= 1      0.3600899832D+01     0.2132481200D-13    -0.5898332782D+01
      J= 2      0.3570203920D+01    -0.7369966811D+00    -0.1813855083D+02
      J= 3      0.3414416040D+01    -0.3003408607D+01    -0.1173825797D+02
      J= 4      0.3103144243D+01    -0.4470103941D+01    -0.5130837195D+01
      J= 5      0.2704082848D+01    -0.5111202048D+01    -0.5980230354D+01
      J= 6      0.2247197597D+01    -0.5858431831D+01    -0.7461464285D+01
      J= 7      0.1720359516D+01    -0.6790741793D+01    -0.2105971054D+02
      J= 8      0.1045092463D+01    -0.9422152626D+01     0.3876151258D+01
      J= 9      0.2803992897D+00    -0.8937827526D+01    -0.1064157140D+02
      J=10     -0.5195022632D+00    -0.1026749187D+02     0.2620441166D+01
      J=11     -0.1361147121D+01    -0.9940067749D+01    -0.5897783170D+00
      J=12     -0.2192224070D+01    -0.1001376055D+02     0.2453066414D+02
      J=13     -0.2898708639D+01    -0.6948654065D+01     0.2098246695D+01
```

J=14	-0.3466611895D+01	-0.6686478141D+01	0.1829860575D+02
J=15	-0.3928366515D+01	-0.4400067353D+01	0.1082423418D+02
J=16	-0.4238560998D+01	-0.3047579292D+01	0.2941656810D+02
J=17	-0.4339335605D+01	0.6280208913D+00	0.9279458398D+01
J=18	-0.4238729609D+01	0.1787489218D+01	0.7516797823D+01
J=19	-0.4050713082D+01	0.2726713106D+01	0.1894951599D+01
J=20	-0.3813716235D+01	0.2963487308D+01	-0.1231930310D+02
J=21	-0.3630969459D+01	0.1424190386D+01	-0.1368993061D+01
J=22	-0.3519458869D+01	0.1253134703D+01	-0.1131578552D+02
J=23	-0.3473961992D+01	-0.1607726983D+00	0.2674956687D+01
J=24	-0.3473433435D+01	0.1734631398D+00	-0.1388260423D+01

6.2.9 DGIIMC, RGIIMC
Least Squares Method Integral Value and Cubic Spline Coefficients

(1) **Function**

DGIIMC or RGIIMC obtains least squares approximation cubic spline coefficients and calculates the value of the integral over the specified range. In addition optimum knot positions are obtained.

(2) **Usage**

Double precision:

CALL DGIIMC (X, YD, N, XK, NXK, ITMX, A, B, Q, S, Y, C, IWK, WK1, WK2, IERR)

Single precision:

CALL RGIIMC (X, YD, N, XK, NXK, ITMX, A, B, Q, S, Y, C, IWK, WK1, WK2, 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}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XK	{D / R}	NXK	Input	Initial estimates of knot positions $XK(i) < XK(i+1), i = 1, \cdots, NXK-1$ $XK(1) \leq X(1)$ $XK(NXK) \geq X(N)$
				Output	Optimum knot positions
5	NXK	I	1	Input	Number of knots
6	ITMX	I	1	Input	Maximum number of iterations (15 is suitable)
				Output	Actual iteration count
7	A	{D / R}	1	Input	Lower limit of the integration range
8	B	{D / R}	1	Input	Upper limit of the integration range
9	Q	{D / R}	1	Output	Integral value of cubic spline function over abscissa interval $[A, B]$

No.	Argument	Type	Size	Input/ Output	Contents
10	S	{D R}	1	Output	Least squares error of cubic spline approximation
11	Y	{D R}	NXK	Output	0th order terms of cubic spline coefficients
12	C	{D R}	See Contents	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t (XK(j) $\leq$ t $<$ XK(j + 1)): f(t) = ((C(3, j) $\times$ D + C(2, j)) $\times$ D + C(1, j)) $\times$ D + Y(j) where: D = t $-$ XK(j) **Size**: 3, (NXK $-$ 1)
13	IWK	I	75	Work	Work area
14	WK1	I	See Contents	Work	Work area **Size**: N $\times$ (NXK + 6)
15	WK2	{D R}	3811	Work	Work area
16	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N $\geq$ 2, 2 $\leq$ NXK $\leq$ 28, and ITMX $\leq$ 20

(b) X(1) $<$ X(2) $<\cdots<$ X(N) (Ascending order)

(c) Sample points are distributed throughout the range determined by the endpoint knots.

(d) XK(1) $<$ XK(2) $<\cdots<$ XK(NXK) (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	A or B was outside the interpolation interval range.	A function extrapolated from the cubic spline coefficients at the endpoints is integrated.
1500	Calculations ended after ITMX iterations without obtaining the minimum least squares error (S). (The data is nearly uncorrelated.)	Interpolation is performed using the current spline coefficients and least squares error.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) Different cubic spline coefficients may be obtained and convergence status may be changed depending on the initial estimates of the knot positions.

(b) Usually values Y(1) and Y(NXK) become extrapolated ones using cubic spline coefficients.

(c) To continue further obtaining interpolation values, derivatives or integrals, you have called this subroutine and then call subroutine 6.2.18 $\left\{\begin{matrix}\text{DGISCX}\\\text{RGISCX}\end{matrix}\right\}$, 6.2.19 $\left\{\begin{matrix}\text{DGIDCY}\\\text{RGIDCY}\end{matrix}\right\}$ or 6.2.20 $\left\{\begin{matrix}\text{DGIICZ}\\\text{RGIICZ}\end{matrix}\right\}$, respectively. In this case, contents of array XK, Y, C and variable NXK must input to corresponding arguments of the succeeding subroutine. Obtaining derivatives, there is no need to pass array Y. This enables you to eliminate unnecessary calculations since you calculate cubic spline coefficients only once.

(7) **Example**

(a) Problem

Use the equation

$$y_i = \begin{cases} 1.0 - x_i & (0.0 \le x_i \le 0.5) \\ x_i & (0.5 < x_i \le 1.0) \\ 2.0 - x_i & (1.0 < x_i \le 2.0) \end{cases}$$

to find y_i at each point $x_i = 0.1 \times (i-1)$ $(i = 1, 2, \cdots, 21)$ and perform least squares approximation cubic spline interpolation with four knots $\{\xi_j\} = \{0.0, 0.33, 1.33, 2.0\}$ using these points as sample points. In addition, obtain the integral value of the cubic spline function over abscissa interval $[0.5, 1.5]$.

(b) Input data

X(i)=x_i, YD(i)=y_i (i = $1, \cdots,$ N), XK(j)=ξ_j (j = $1, \cdots,$ NXK),
A=0.5, B=1.5, N=21, NXK=4 and ITMX=15.

(c) Main program

```
      PROGRAM BGIIMC
! *** EXAMPLE OF DGIIMC ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=21,NXK=4,A=0.5,B=1.5)
      DIMENSION X(N),YD(N),XK(NXK),Y(NXK),C(3,NXK-1),&
                IWK(75),WK1(210),WK2(3811)
      READ(5,*) XK
      READ(5,*) X
      READ(5,*) YD
      ITMX=15
      WRITE(6,1000)   N,NXK,ITMX,A,B,(I,X(I),YD(I),I=1,21)
      WRITE(6,1002)   (K,XK(K),K=1,4)
      CALL DGIIMC(X,YD,N,XK,NXK,ITMX,A,B,Q,S,&
                  Y,C,IWK,WK1,WK2,IERR)
      WRITE(6,1400)  IERR
      WRITE(6,1500)  Q
      WRITE(6,1600)  (I,Y(I),I=1,NXK)
      WRITE(6,1700)
      DO 20  J=1,NXK-1
      WRITE(6,1710)  J,(C(I,J),I=1,3)
   20 CONTINUE
      WRITE(6,1800)  (I,XK(I),I=1,NXK)
      WRITE(6,1900)  ITMX
      WRITE(6,2000)  S
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIIMC ***',/,/,6X,'** INPUT **',&
       /,/,8X,'N    =',I3,/,/,8X,'NXK  =',I3,/,/,8X,'ITMX =',I3,&
       /,/,8X,'LIMITS OF INTEGRATION    A =',F6.2,/,/,33X,'B =',F6.2,/,/,6X,&
       'COORDINATES (X,YD)',/,/,9X,'I    X(I)  YD(I)',/,21(8X,I2,2F7.2,/))
 1002 FORMAT(' ',/,6X,'LOCATIONS OF KNOTS',/,/,9X,'K',4X,&
       'XK(K)',/,4(8X,I2,F8.2,/))
 1400 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=' ,I4)
 1500 FORMAT(' ',/,8X,'( INTEGRAL )',/,/,9X,'Q=',D24.10)
 1600 FORMAT(' ',/,(8X,'Y(',I2,')=',D22.10))
 1700 FORMAT(' ',/,9X,'C(I,J)',/,27X,'I=1',19X,'I=2',19X,'I=3',/)
 1710 FORMAT(11X,'J=',I2,3D22.10)
 1800 FORMAT(' ',/,8X,'( OPTIMAL LOCATIONS OF KNOTS )',/,/,&
             (8X,'XK(',I1,')=',D22.10))
 1900 FORMAT(' ',/,8X,'( NUMBER OF ITERATIONS )',/,/,9X,'ITMX=',I2)
 2000 FORMAT(' ',/,8X,'( LEAST SQUARES ERROR )',/,/,9X,'S =',D24.10)
      END
```

(d) Output results

```
*** DGIIMC ***

 ** INPUT **

   N    = 21

   NXK  =  4

   ITMX = 15

   LIMITS OF INTEGRATION    A =  0.50

                            B =  1.50

 COORDINATES (X,YD)

    I    X(I)  YD(I)
    1   0.00   1.00
    2   0.10   0.90
    3   0.20   0.80
    4   0.30   0.70
    5   0.40   0.60
    6   0.50   0.50
    7   0.60   0.60
    8   0.70   0.70
    9   0.80   0.80
   10   0.90   0.90
   11   1.00   1.00
   12   1.10   0.90
   13   1.20   0.80
   14   1.30   0.70
   15   1.40   0.60
   16   1.50   0.50
   17   1.60   0.40
   18   1.70   0.30
   19   1.80   0.20
   20   1.90   0.10
   21   2.00   0.00


 LOCATIONS OF KNOTS

    K    XK(K)
    1    0.00
    2    0.33
    3    1.33
    4    2.00


 ** OUTPUT **

   IERR=   0

   ( INTEGRAL )

    Q=        0.7551389177D+00

   Y( 1)=      0.9837783573D+00
   Y( 2)=      0.5580871037D+00
   Y( 3)=      0.8013007574D+00
   Y( 4)=      0.4845275096D-01

    C(I,J)
                       I=1                  I=2                  I=3

      J= 1     -0.3454003900D+00    -0.3855853842D+01     0.5529117634D+01
      J= 2      0.6481571877D+00     0.5598937312D+01    -0.1431286281D+02
      J= 3      0.1148362744D+01    -0.3141652192D+01     0.1390040784D+01

   ( OPTIMAL LOCATIONS OF KNOTS )

   XK(1)=      0.0000000000D+00
   XK(2)=      0.5700000000D+00
   XK(3)=      0.7735602432D+00
   XK(4)=      0.2000000000D+01

   ( NUMBER OF ITERATIONS )

    ITMX= 3

   ( LEAST SQUARES ERROR )

    S =        0.2888606848D-01
```

6.2.10 DGICCP, RGICCP
Cubic Spline Coefficients (Endpoint Condition Input Unnecessary)

(1) **Function**

DGICCP or RGICCP obtains cubic spline coefficients for "not-a-knot" endpoint conditions when endpoint conditions need not be entered. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGICCP  (X, Y, N, C, IERR)
```

Single precision:

```
CALL RGICCP  (X, Y, N, C, 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}	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)) for $i = 1, \cdots, N$ ($X(i) < X(i+1), i \neq N$)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
5	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(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**

None

6.2.11 DGICCQ, RGICCQ
Cubic Spline Coefficients (Endpoint Conditions Are Input)

(1) **Function**

DGICCQ or RGICCQ obtains cubic spline coefficients for specified endpoint conditions. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGICCQ  (X, Y, N, END, C, ISW, IERR)
```

Single precision:

```
CALL RGICCQ  (X, Y, N, END, C, ISW, 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}	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)) for $i = 1, \cdots, N$ ($X(i) < X(i+1), i \neq N$)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	END	{D / R}	4	Input	Endpoint conditions The following values must be set to array END correspond to the values of ISW. (See the explanation of the algorithm or the notes.) **Endpoint conditions at initial point** X(1) ISW(1)=1:END(1)=value of y' at initial point ISW(1)=2:END(1)=value of y'' at initial point ISW(1)=3:END(1)=value of y''' at initial point ISW(1)=4:END(1)=λ_1, END(2)=d_1 **Endpoint conditions at final point** X(N) ISW(2)=1:END(3)=value of y' at final point ISW(2)=2:END(3)=value of y'' at final point ISW(2)=3:END(3)=value of y''' at final point ISW(2)=4:END(3)=λ_N, END(4)=d_N

No.	Argument	Type	Size	Input/ Output	Contents
5	C	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	$3, (\mathrm{N}-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): $\mathrm{C}(\mathrm{k},\mathrm{j})$ The value of cubic spline function f(t) at abscissa value t ($\mathrm{X(j)} \leq \mathrm{t} < \mathrm{X(j+1)}$): $\mathrm{f(t)} = ((\mathrm{C}(3,\mathrm{j}) \times \mathrm{D} + \mathrm{C}(2,\mathrm{j})) \times \mathrm{D} + \mathrm{C}(1,\mathrm{j})) \times \mathrm{D} + \mathrm{Y(j)}$ where: $\mathrm{D} = \mathrm{t} - \mathrm{X(j)}$
6	ISW	I	2	Input	Switch for the value set in endpoint condition array END. ISW(1) and ISW(2) correspond to the initial and final point, respectively (See argument END).
7	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq 2$

(b) $\mathrm{X(1)} < \mathrm{X(2)} < \cdots < \mathrm{X(N)}$ (Ascending order)

(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) You must declare the size of the array END to be 4 regardless of whether ISW(1) or ISW(2) equal to 1, 2 or 3. In this case, no initialization is necessary for END(2) and END(4).

(b) When ISW(1)$\geq$ 4 or ISW(2)$\geq$ 4, the values must be set in array END are the following values $\lambda_1, d_1, \mu_\mathrm{N}$ and d_N of coefficient of simultaneous linear equations obtaining cubic spline coefficients. For details, see section 6.1.2Algorithms Used.

$$\begin{bmatrix} 2 & \lambda_1 & & & & 0 \\ \mu_2 & 2 & \lambda_2 & & & \\ & \mu_3 & \ddots & \ddots & & \\ & & \ddots & \ddots & \lambda_{\mathrm{N}-2} & \\ & \mu_{\mathrm{N}-1} & & 2 & & \lambda_{\mathrm{N}-1} \\ 0 & & & \mu_\mathrm{N} & & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ \vdots \\ \vdots \\ M_{\mathrm{N}-1} \\ M_\mathrm{N} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ \vdots \\ d_{\mathrm{N}-1} \\ d_\mathrm{N} \end{bmatrix}$$

where $M_1, \cdots, M_\mathrm{N}$ is second derivatives of the cubic spline function.

6.2.12 DGICCR, RGICCR
Cubic Spline Coefficients (Periodic Spline)

(1) **Function**

DGICCR or RGICCR obtains cubic spline coefficients for periodic endpoint conditions. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGICCR  (X, Y, N, C, WK, IERR)
```

Single precision:

```
CALL RGICCR  (X, Y, N, C, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)) for $i = 1, \cdots, N$ ($X(i) < X(i+1), i \neq N$)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
				Output	0th order terms of cubic spline coefficients (Same as input values)
3	N	I	1	Input	Number of sample points
4	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
5	WK	{D / R}	$5 \times N$	Work	Work area
6	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(c) $Y(1) = Y(N)$ (Periodic endpoint condition)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2000	Restriction (c) was not satisfied.	Processing is performed regarding Y(N) as Y(1).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) The periodic spline is interpolated as if the curved-line pattern on the abscissa interval [X(1), X(N)] were extended continuously. So, only when values such as interpolation values, derivatives, and integrals are within the interpolation interval, the result makes its meaning.

6.2.13 DGICCS, RGICCS
Cubic Spline Coefficients (Automatic Smoothing)

(1) **Function**

DGICCS or RGICCS automatically determines the smoothing control variable and obtains the optimum smoothed cubic spline coefficients. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

```
CALL DGICCS  (X, YD, N, Y, C, S, ISW, WK, IERR)
```

Single precision:

```
CALL RGICCS  (X, YD, N, Y, C, S, ISW, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
5	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): $C(k, j)$ The value of cubic spline function f(t) at abscissa value t $(X(j) \leq t < X(j+1))$: $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
6	S	{D / R}	1	Output	Sum of the squares of the error $(\sum_{i=1}^{N} \{YD(i) - Y(i)\}^2)$
7	ISW	I	1	Input	Processing switch ISW=1: When the abscissa value spacing may not be uniform. When sample points are nearly uncorrelated, limit value N up to 20. ISW=2: When the abscissa value spacing is uniform.
8	WK	{D / R}	See Contents	Work	Work area **Size**: $N \times (2 \times N + 4)$ (when ISW=1) $6 \times N$ (when ISW=2)
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(c) If ISW=2, abscissa values must be uniformly spaced.

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	The minimum cross-validation value could not be found. (The data is uncorrelated.)	

(6) **Notes**

None

6.2.14 DGICCO, RGICCO
Cubic Spline Coefficients (Automatic Smoothing Periodic Conditions)

(1) **Function**

DGICCO or RGICCO automatically determines the smoothing control variable and obtains the optimum smoothed cubic spline coefficients for periodic end point conditions. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

CALL DGICCO (X, YD, N, Y, C, S, WK, IERR)

Single precision:

CALL RGICCO (X, YD, N, Y, C, S, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
5	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): $C(k, j)$ The value of cubic spline function f(t) at abscissa value t $(X(j) \leq t < X(j+1))$: $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
6	S	{D / R}	1	Output	Sum of the squares of the error $(\sum_{i=1}^{N} \{YD(i) - Y(i)\}^2)$
7	WK	{D / R}	See Contents	Work	Work area **Size**: $N \times (2 \times N + 4)$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(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.	
4000	The minimum cross-validation value could not be found. (The data is uncorrelated.)	

(6) **Notes**

None

6.2.15 DGICCT, RGICCT
Cubic Spline Coefficients (Smoothing by Specifying a Control Variable)

(1) **Function**

DGICCT or RGICCT determines the smoothed (natural) cubic spline coefficients using the specified control variable.

(2) **Usage**

Double precision:

```
CALL DGICCT  (X, YD, W, N, SF, Y, C, WK, IERR)
```

Single precision:

```
CALL RGICCT  (X, YD, W, N, SF, Y, C, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	W	{D / R}	N	Input	Relative weights at each sample points (W(i) > 0) Usually 1.0 is set for non weighted one.
4	N	I	1	Input	Number of sample points
5	SF	{D / R}	1	Input	Control variable that controls degree of smoothing (SF > 0) The cubic spline function f(x) is determined so the following expression is satisfied. $\sum_{i=1}^{N}((f(X(i)) - YD(i))/W(i))^2 \leq SF$ where the equality holds if f(x) is not a linear function.
6	Y	{D / R}	N	Output	0th order terms of cubic spline coefficients
7	C	{D / R}	$3, (N-1)$	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($X(j) \leq t < X(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - X(j)$
8	WK	{D / R}	$7 \times N + 14$	Work	Work area.
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(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) The input SF value is proportional to the sum of the squares of the distances between the input data points and the smoothed curve. For a large value, it is nearly a straight line; for a small value, it is nearly a complete interpolation.

(b) To have the spline coefficients become **a natural cubic spline**, let the second derivatives of the spline function be zero at X(1) and X(N).

6.2.16 DGICCM, RGICCM
Cubic Spline Coefficients (Least Squares Method When Knot Positions are Set Automatically)

(1) **Function**

DGICCM or RGICCM automatically locates the optimum knot positions and obtains the best least squares approximation cubic spline coefficients. In addition optimum knot positions are obtained.

(2) **Usage**

Double precision:

```
CALL DGICCM  (X, YD, N, XK, NXK, ITMX, Y, C, S, IWK, WK1, WK2, IERR)
```

Single precision:

```
CALL RGICCM  (X, YD, N, XK, NXK, ITMX, Y, C, S, IWK, WK1, WK2, 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}	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{D / R}	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XK	{D / R}	NXK	Input	Initial estimates of knot positions $XK(i) < XK(i+1), i = 1, \cdots, NXK-1$ $XK(1) \leq X(1)$ $XK(NXK) \geq X(N)$
				Output	Optimum knot positions
5	NXK	I	1	Input	Number of knots
6	ITMX	I	1	Input	Maximum number of iterations (15 is suitable)
				Output	Actual iteration count
7	Y	{D / R}	NXK	Output	0th order terms of cubic spline coefficients
8	C	{D / R}	See Contents	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): C(k, j) The value of cubic spline function f(t) at abscissa value t ($XK(j) \leq t < XK(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - XK(j)$ **Size**: $3, (NXK-1)$
9	S	{D / R}	1	Output	Cubic spline approximation least squares error
10	IWK	I	75	Work	Work area
11	WK1	{D / R}	See Contents	Work	Work area **Size**: $N \times (NXK+6)$
12	WK2	{D / R}	3811	Work	Work area
13	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$, $2 \leq NXK \leq 28$, and $ITMX \leq 20$

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(c) Sample points are distributed throughout the range determined by the endpoint knots.

(d) XK(1) < XK(2) < ⋯ < XK(NXK) (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1500	Calculations ended after ITMX iterations without obtaining the minimum least squares error (S). (The data is nearly uncorrelated.)	Interpolation is performed using the current cubic spline coefficients and least squares error.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) Different cubic spline coefficients may be obtained and convergence status may be changed depending on the initial estimates of the knot positions.

(b) Usually values Y(1) and Y(NXK) become extrapolated ones using cubic spline coefficients.

6.2.17 DGICCN, RGICCN
Cubic Spline Coefficients (Least Squares Method When Knot Positions are Specified)

(1) **Function**

DGICCN or RGICCN obtains the least squares approximation cubic spline coefficients at the specified knots.

(2) **Usage**

Double precision:

```
CALL DGICCN  (X, YD, N, XK, NXK, Y, C, NKM, S, ISW, IWK, WK1, WK2, IERR)
```

Single precision:

```
CALL RGICCN  (X, YD, N, XK, NXK, Y, C, NKM, S, ISW, IWK, WK1, WK2, 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 }	N	Input	Abscissa values X(i) of sample point $(X(i), YD(i)), i = 1, \cdots, N$ $(X(i) < X(i+1), i \neq N)$
2	YD	{ D / R }	N	Input	Ordinate values of sample points or function values YD(i)
3	N	I	1	Input	Number of sample points
4	XK	{ D / R }	See Contents	Input	Knot position abscissa values ISW=0: Input all knots abscissa values for XK(i) $i = 1, \cdots, NXK$ in the order: left endpoint knot, insertion knots, right endpoint knot. ISW = 1 or 2: Input added knot abscissa values for XK(i) $i = 1, \cdots, NXK$ ISW=3: Input modified knot abscissa value for XK(1).
				Output	All knots abscissa values at that time (when ISW = 4) **Size**: $MAX(1, NXK)$
5	NXK	I	1	Input	Number of knots ISW=0: Set number of all knots ISW = 1 or 2: When you add knots, set number of added knots (positive value), when you delete knots, set sign changed number of deleted knots (negative value). (See Note (d)) ISW=3: NXK=1
				Output	Number of all knots at that time (when ISW = 4)
6	Y	{ D / R }	NKM	Output	0th order terms of cubic spline coefficients

No.	Argument	Type	Size	Input/ Output	Contents
7	C	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	k-th order terms of cubic spline coefficients (k=1, 2, 3): $\mathrm{C(k,j)}$ The value of cubic spline function f(t) at abscissa value t ($\mathrm{XK(j)} \leq \mathrm{t} < \mathrm{XK(j+1)}$): $\mathrm{f(t)} = ((\mathrm{C(3,j)} \times \mathrm{D} + \mathrm{C(2,j)}) \times \mathrm{D} + \mathrm{C(1,j)}) \times \mathrm{D} + \mathrm{Y(j)}$ where: $\mathrm{D} = \mathrm{t} - \mathrm{XK(j)}$ **Size**: $3, (\mathrm{NKM}-1)$
8	NKM	I	1	Input	Maximum number of knots
9	S	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Output	Cubic spline approximation least squares error
10	ISW	I	1	Input	Processing switch ISW=0: Least squares approximation cubic spline coefficients are output for input knots (Initially, processing must be performed with ISW=0). ISW=1: When $\mathrm{NXK} \geq 0$, new cubic spline coefficients are calculated from the previous and added knots. if $\mathrm{NXK} < 0$, cubic spline coefficients are recalculated deleting $-\mathrm{NXK}$ knots form the previously added knots. ISW=2: Same as ISW = 1. It is effective when you are again inserting knots after having deleted several knots. ISW=3: The abscissa value of the knot input last is changed to the value of XK(1). Only the least squares error is output; cubic spline coefficients are not calculated. ISW=4: Number of knots at that time is returned to NXK and the abscissa values of each knot at that time are returned to XK(i) $\mathrm{i} = 1, \cdots, \mathrm{NXK}$.
11	IWK	I	75	Work	Work area
12	WK1	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $\mathrm{N} \times (\mathrm{NKM}+6)$
13	WK2	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	3811	Work	Work area
14	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2, NXK \leq NKM \leq N, NKM \leq 28$, and if $ISW = 0, NXK \geq 2$.

(b) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(c) Knots must not be duplicated.

(d) Specify knots within the range from the left endpoint knot to the right endpoint knot. In addition, assign data within this range.

(e) When ISW=4, then NXK $\geq$ (number of all knots at that time)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) or (d) was not satisfied.	
3030	Restriction (e) was not satisfied.	

(6) **Notes**

(a) You must retain arrays X, YD, IWK, WK1, and WK2 until the optimum cubic spline coefficients are obtained using this subroutine.

(b) Input the left endpoint knot to XK(1) and the right endpoint to XK(NXK) with ISW=0.

(c) Specify ISW=0 to obtain the least squares approximation cubic spline coefficients for the first time. To modify knots thereafter, use ISW= 1, 2 or 3. If you want to know the number of knots and knot positions, specify ISW=4, and they are output in NXK and XK(i) (ascending order), respectively.

(d) When deleting some knots with ISW=1 or ISW=2, knots are deleted in reversed order of knots added and descending order of knot position abscissa values. However endpoint knots are never deleted.
When NXK $< -$(Number of knots $- 2) < 0$, then all knots except endpoint knots are deleted. For example,

i. Using ISW=0, knot position abscissa values {1.0, 2.0, 5.0, 7.0, 9.0} are input

ii. Using ISW=1, knot position abscissa values {1.5, 2.5, 6.0} are added,

iii. Using ISW=1, knot position abscissa values {3.0, 8.0} are added,

then current knot position abscissa values become to be {1.0, 1.5, 2.0, 2.5, 3.0, 5.0, 6.0, 7.0, 8.0, 9.0}. At this time, if 3 knots are deleted by NXK $= -3$, then the knots of abscissa values {8.0, 3.0, 6.0} are deleted. If 6 knots are deleted by NXK $= -6$, then the knots of abscissa values {8.0, 3.0, 6.0, 2.5, 1.5, 7.0} are deleted. If you try to delete 9 knots by NXK $= -9$, 8 knots are deleted and 2 endpoint knots of abscissa values {1.0, 9.0} are remained.

(e) After modifying knot positions with ISW=3, execute this subroutine with ISW= 1 or 2 and NXK=0 to obtain cubic spline coefficients at that time.

6.2.18 DGISCX, RGISCX
Interpolation Values According to Cubic Spline Coefficients

(1) **Function**

DGISCX or RGISCX calculates interpolation values of cubic spline function at specified points using given cubic spline coefficients.

(2) **Usage**

Double precision:

CALL DGISCX (X, Y, N, C, XL, YL, M, IERR)

Single precision:

CALL RGISCX (X, Y, N, C, XL, YL, 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	X	{D / R}	N	Input	Knots abscissa values X(i) of cubic spline function ($X(i) < X(i+1), i \neq N$)
2	Y	{D / R}	N	Input	0th order terms of cubic spline coefficients
3	N	I	1	Input	Number of knots
4	C	{D / R}	$3, (N-1)$	Input	k-th order terms of cubic spline coefficients (k=1, 2, 3): $C(k, j)$ The value of cubic spline function f(t) at abscissa value t ($XK(j) \leq t < XK(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - XK(j)$
5	XL	{D / R}	M	Input	Abscissa values of points where interpolation values are calculated
6	YL	{D / R}	M	Output	Cubic spline interpolation values at XL(i)
7	M	I	1	Input	Number of interpolation values
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$ and $M > 0$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

6.2.19 DGIDCY, RGIDCY
Derivative Values According to Cubic Spline Coefficients

(1) **Function**

DGIDCY or RGIDCY calculates first and second derivatives of cubic spline function at specified points using given cubic spline coefficients.

(2) **Usage**

Double precision:

CALL DGIDCY (X, N, C, XL, DL, DDL, M, IERR)

Single precision:

CALL RGIDCY (X, N, C, XL, DL, DDL, 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	X	{D / R}	N	Input	Knots abscissa values X(i) of cubic spline function ($X(i) < X(i+1), i \neq N$)
2	N	I	1	Input	Number of knots
3	C	{D / R}	$3, (N-1)$	Input	k-th order terms of cubic spline coefficients (k=1, 2, 3): $C(k,j)$ The value of cubic spline function f(t) at abscissa value t ($XK(j) \leq t < XK(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - XK(j)$ and Y(j) are 0th order terms of cubic spline coefficients.
4	XL	{D / R}	M	Input	Abscissa values of points where derivatives of cubic spline function are calculated.
5	DL	{D / R}	M	Output	First derivatives of cubic spline function at XL(i) $DL(i) = (3.0 \times C(3,j) \times D + 2.0 \times C(2,j)) \times D + C(1,j)$ where: $XK(j) \leq XL(i) < XK(j+1)$ $D = XL(i) - XK(j)$
6	DDL	{D / R}	M	Output	Second derivatives of cubic spline function at XL(i) $DDL(i) = 6.0 \times C(3,j) \times D + 2.0 \times C(2,j)$ where: $XK(j) \leq XL(i) < XK(j+1)$ $D = XL(i) - XK(j)$
7	M	I	1	Input	Number of points where derivatives are calculated
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	XL(i) was outside the interpolation interval range (XL(i) < X(1) or XL(i) > X(N)).	The extrapolated value is output using the cubic spline coefficients at the endpoints.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

6.2.20 DGIICZ, RGIICZ
Integral Value According to Cubic Spline Coefficients

(1) **Function**

DGIICZ or RGIICZ calculates the integral of the cubic spline function using given cubic spline coefficients.

(2) **Usage**

Double precision:

CALL DGIICZ (X, Y, N, C, A, B, Q, IERR)

Single precision:

CALL RGIICZ (X, Y, N, C, A, B, Q, 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}	N	Input	Knots abscissa values X(i) of cubic spline function ($X(i) < X(i+1), i \neq N$)
2	Y	{D / R}	N	Input	0th order terms of cubic spline coefficients
3	N	I	1	Input	Number of knots
4	C	{D / R}	$3, (N-1)$	Input	k-th order terms of cubic spline coefficients (k=1, 2, 3): $C(k, j)$ The value of cubic spline function f(t) at abscissa value t ($XK(j) \leq t < XK(j+1)$): $f(t) = ((C(3,j) \times D + C(2,j)) \times D + C(1,j)) \times D + Y(j)$ where: $D = t - XK(j)$
5	A	{D / R}	1	Input	Lower limit of the integration range
6	B	{D / R}	1	Input	Upper limit of the integration range
7	Q	{D / R}	1	Output	Integral value of cubic spline function over abscissa interval $[A, B]$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 2$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	A or B was outside the interpolation interval	A function extrapolated from the cubic spline coefficients at the endpoints is integrated.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

6.3 BICUBIC SPLINE (CURVED SURFACE INTERPOLATION)

6.3.1 DGISXB, RGISXB
Interpolation Values

(1) **Function**

DGISXB or RGISXB performs a bicubic spline interpolation from data on a lattice and obtains interpolation values at new points on the lattice. Endpoint condition is "not-a-knot" and the knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGISXB  (X, NX, Y, NY, Z, XL, NXL, YL, NYL, FL, WK, IERR)
```

Single precision:

```
CALL RGISXB  (X, NX, Y, NY, Z, XL, NXL, YL, NYL, FL, 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	X	{D / R}	NX	Input	X coordinate values of sample points X(i), i = $1, \cdots, NX$ $(X(i) < X(i+1), i \neq NX)$
2	NX	I	1	Input	Number of sample points in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of sample points Y(j), j = $1, \cdots, NY$ $(Y(j) < Y(j+1), j \neq NY)$
4	NY	I	1	Input	Number of sample points in Y direction
5	Z	{D / R}	NX, NY	Input	Array formed from function values z_{ij} at sample point (X(i), Y(j)) $Z(i,j) = z_{ij}$
6	XL	{D / R}	NXL	Input	X coordinate values of lattice points where interpolation values are calculated
7	NXL	I	1	Input	Number of interpolation points in X direction
8	YL	{D / R}	NYL	Input	Y coordinate values of lattice points where interpolation values are calculated
9	NYL	I	1	Input	Number of interpolation points in Y direction
10	FL	{D / R}	See Contents	Output	Array formed from bicubic spline function values f_{ij} at interpolation points (XL(i), YL(j)) $FL(i,j) = f_{ij}$ Note that when NY > NYL, area FL(i, j) $(i = 1, \cdots, NXL; j = NYL+1, \cdots, NY)$ are used as a work area. After calculation the values corresponding to them become indefinite. **Size**: NXL, MAX(NYL, NY)
11	WK	{D / R}	See Contents	Work	Work area **Size**: $MAX\{3 \times (NX-1), 3 \times (NY-1) + NY\}$.
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $NX \geq 2, NY \geq 2$

(b) $X(1) < X(2) < \cdots < X(NX)$ (Ascending order)

(c) $Y(1) < Y(2) < \cdots < Y(NY)$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	Interpolation point (XL(i), YL(j)) was out-side the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) or (c) was not satisfied.	

(6) **Notes**

(a) Bicubic spline coefficients are not output.

(b) This subroutine obtain bicubic spline interpolation value at all lattice point. To obtain the bicubic spline interpolation value at a specific point, it is more effective to call subroutine 6.3.4 $\left\{\begin{matrix}\text{DGICBP}\\\text{RGICBP}\end{matrix}\right\}$, and subroutine 6.3.5 $\left\{\begin{matrix}\text{DGISBX}\\\text{RGISBX}\end{matrix}\right\}$ successively.

(7) **Example**

(a) Problem
When the matrix Z formed from function values z_{ij} at sample point $(x_i, y_j) = (2 \times i - 1, 2 \times j - 1)$ $(i = 1, 2, 3;\ j = 1, 2, 3)$ is given as follows:

$$Z = \begin{bmatrix} 2.0 & 0.5 & 1.0 \\ 4.0 & 1.0 & 2.0 \\ 3.0 & 0.75 & 1.5 \end{bmatrix}$$

perform bicubic spline interpolation using these points as sample points. In addition, obtain the values of the bicubic spline function at the lattice points $(xl_i, yl_j) = (0.5 \times (i+1), 0.5 \times (j+2))$ $(i = 1, 2;\ j = 1, 2)$.

(b) Input data
$X(i) = x_i (i = 1, \cdots, NX)$,
$Y(j) = y_j (j = 1, \cdots, NY)$,
$Z(i, j) = z_{ij}$,
$XL(i) = xl_i (i = 1, \cdots, NXL)$,
$YL(j) = yl_j (j = 1, \cdots, NYL)$,
NX=3, NY=3, NXL=2 and NYL=2.

(c) Main program

```
      PROGRAM  BGISXB
! *** EXAMPLE OF DGISXB ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (NX=3,NY=3,NXL=2,NYL=2)
      DIMENSION X(NX),Y(NX),Z(NX,NY),XL(NXL),YL(NYL),&
                FL(NXL,NY),WK(9)
      READ(5,*) X
      READ(5,*) Y
      READ(5,*) Z
      READ(5,*) XL
      READ(5,*) YL
      WRITE(6,1000)  NX,NY,NXL,NYL,(I,X(I),Y(I),I=1,3),(J,XL(J),YL(J),&
                     J=1,2)
      WRITE(6,1001)  ((Z(I,J),J=1,3),I=1,3)
      CALL DGISXB(X,NX,Y,NY,Z,XL,NXL,YL,NYL,FL,WK,IERR)
      WRITE(6,1500)  IERR
      WRITE(6,1600)
      DO 20  I=1,NXL
```

```
      WRITE(6,1610)  I,(FL(I,J),J=1,NYL)
   20 CONTINUE
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGISXB ***',/,/,6X,'** INPUT **',/,/,8X,'NX  =',&
             I3,/,/,8X,'NY  =',I3,/,/,8X,'NXL =',I3,/,/,8X,'NYL =',I3,&
             /,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',3X,&
             'X(I)',2X,'Y(I)',/,3(8X,I2,2F6.2,/),/,/,6X,&
             'SPECIFIED POINTS',/,/,9X,'J',3X,'XL(J)',&
             2X,'YL(J)',/,2(8X,I2,2F7.2,/))
 1001 FORMAT(' ',/,6X,'FUNCTION VALUES',/,/,17X,'I',F5.2,2F6.2,' I',/,&
             8X,'Z(X,Y) = I',F5.2,2F6.2,' I',/,17X,'I',F5.2,2F6.2,' I')
 1500 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1600 FORMAT(' ',/,8X,'( ESTIMATED FUNCTION VALUES IN (X,Y) )',&
             /,/,9X,'FL(I,J)',/,30X,'J=1',23X,'J=2')
 1610 FORMAT(' ',/,11X,'I=',I1,2D26.10)
      END
```

(d) Output results

```
*** DGISXB ***

 ** INPUT **

   NX  =  3

   NY  =  3

   NXL =  2

   NYL =  2

 COORDINATES (X,Y)

    I   X(I)  Y(I)
    1  1.00  1.00
    2  3.00  3.00
    3  5.00  5.00


 SPECIFIED POINTS

    J   XL(J)  YL(J)
    1   1.00   1.50
    2   1.50   2.00


 FUNCTION VALUES

             I 2.00  0.50  1.00 I
   Z(X,Y) = I 4.00  1.00  2.00 I
             I 3.00  0.75  1.50 I

 ** OUTPUT **

   IERR=   0

   ( ESTIMATED FUNCTION VALUES IN (X,Y) )

    FL(I,J)
                              J=1                       J=2

      I=1          0.1437500000D+01          0.1000000000D+01

      I=2          0.1999023438D+01          0.1390625000D+01
```

6.3.2 DGIDYB, RGIDYB
Mixed Partial Derivative Values and Bicubic Spline Coefficients

(1) **Function**

DGIDYB or RGIDYB obtains bicubic spline coefficients from lattice data and calculates mixed partial derivatives of the bicubic spline function at arbitrary points. Endpoint condition is "not-a-knot" and the knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGIDYB  (X, NX, Y, NY, Z, XL, YL, DL, C, WK, IERR)
```

Single precision:

```
CALL RGIDYB  (X, NX, Y, NY, Z, XL, YL, DL, C, 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	X	{D / R}	NX	Input	X coordinate values of sample points X(i), i = $1, \cdots,$ NX (X(i) < X(i + 1), i ≠ NX)
2	NX	I	1	Input	Number of sample points in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of sample points Y(j), j = $1, \cdots,$ NY (Y(j) < Y(j + 1), j ≠ NY)
4	NY	I	1	Input	Number of sample points in Y direction
5	Z	{D / R}	NX, NY	Input	Array formed from function values z_{ij} at sample point (X(i), Y(j)) Z(i, j) = z_{ij}
6	XL	{D / R}	1	Input	X coordinate values of lattice points where mixed partial derivatives are calculated
7	YL	{D / R}	1	Input	Y coordinate values of lattice points where mixed partial derivatives are calculated
8	DL	{D / R}	6	Output	Obtained value of bicubic spline function $f(x, y)$ at point (x, y)=(XL, YL) and its mixed partial derivative DL(1)=$f(x, y)$, DL(2)=$\partial f/\partial x$, DL(3)=$\partial f/\partial y$, DL(4)=$\partial^2 f/(\partial x \partial y)$ DL(5)=$\partial^2 f/(\partial x)^2$ DL(6)=$\partial^2 f/(\partial y)^2$
9	C	{D / R}	4, NX, NY	Output	Spline coefficients (See Note (a))
10	WK	{D / R}	See Contents	Work	Work area **Size**: 2 × NX × NY + 2 × MAX(NX, NY).
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX ≥ 4, NY ≥ 4

(b) X(1) < X(2) < ⋯ < X(NX) (Ascending order)

(c) Y(1) < Y(2) < ⋯ < Y(NY) (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The point where the partial derivative was to be calculated was outside the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) or (c) was not satisfied.	

(6) **Notes**

(a) This subroutine obtain the values z(i, j), $z_x(i,j)$, $z_y(i,j)$, $z_{xy}(i,j)$ as bicubic spline coefficients instead of the values $\alpha_{e,r}^{i,j}$ (See Section 6.1.2).

(b) To continue further obtaining interpolation values, partial derivatives or double integrals, you have called this subroutine and then call subroutine 6.3.5 $\begin{Bmatrix}\text{DGISBX}\\ \text{RGISBX}\end{Bmatrix}$, 6.3.6 $\begin{Bmatrix}\text{DGIDBY}\\ \text{RGIDBY}\end{Bmatrix}$ or 6.3.7 $\begin{Bmatrix}\text{DGIIBZ}\\ \text{RGIIBZ}\end{Bmatrix}$, respectively. In this case, contents of array X, Y, C and variable NX and NY must input to corresponding arguments of the succeeding subroutine. This enables you to eliminate unnecessary calculations since you calculate bicubic spline coefficients only once.

(7) **Example**

(a) Problem
When the matrix Z formed from function values z_{ij} at sample point $(x_i, y_j) = (2 \times i - 1, 2 \times j - 1)(i = 1, 2, 3, 4;\ j = 1, 2, 3, 4)$ is given as follows:

$$Z = \begin{bmatrix} 2.0 & 0.5 & 1.0 & 3.5 \\ 4.0 & 1.0 & 2.0 & 7.0 \\ 3.0 & 0.75 & 1.5 & 5.25 \\ -1.0 & -0.25 & -0.5 & -1.75 \end{bmatrix}$$

perform bicubic spline interpolation using these points as sample points. In addition, obtain the partial derivatives of the bicubic spline function at the specified points.

(b) Input data
$X(i) = x_i(i = 1, \cdots, NX)$, $Y(j) = y_j(j = 1, \cdots, NY)$, $Z(i,j) = z_{ij}$, NX=4, NY=4, XL=1.5 and YL=1.5.

(c) Main program

```
      PROGRAM BGIDYB
! *** EXAMPLEOF DGIDYB ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (NX=4,NY=4)
      DIMENSION X(NX),Y(NY),Z(NX,NY),DL(6),C(2,NX,2,NY),WK(40)
      READ(5,*) X
      READ(5,*) Y
      READ(5,*) Z
      XL=1.5
      YL=1.5
      WRITE(6,1000)  NX,NY,XL,YL,(I,X(I),Y(I),I=1,4)
      WRITE(6,1001)  ((Z(I,J),J=1,4),I=1,4)
      CALL DGIDYB(X,NX,Y,NY,Z,XL,YL,DL,C,WK,IERR)
      WRITE(6,1300)  IERR
      WRITE(6,1400)  (I,DL(I),I=1,6)
      WRITE(6,1480)
      WRITE(6,1500) ((((I,J,K,L,C(I,J,K,L),I=1,2),J=1,4),K=1,2),L=1,4)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIDYB ***',/,/,6X,'** INPUT **',/,/,8X,'NX  =',&
             I3,/,/,8X,'NY  =',I3,/,/,8X,'SPECIFIED POINTS      XL =',F8.4,&
             /,/,29X,'YL =',F8.4,/,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',3X,&
             'X(I)',2X,'Y(I)',/,4(8X,I2,2F6.2,/))
 1001 FORMAT(' ',/,6X,'FUNCTION VALUES',/,/,17X,'I',4F6.2,' I',/,&
```

```
           8X,'Z(X,Y) = I',4F6.2,' I',/,2(17X,'I',4F6.2,' I',/))
1300 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4,&
           /,/,/,8X,'( THE PARTIAL DERIVATIVES )')
1400 FORMAT(' ',/,(9X,'DL(',I1,')=',D24.10),/)
1480 FORMAT(9X,'( SPLINE COEFFICIENTS )',/,/)
1500 FORMAT(2(6X,'C(',3(I1,','),I1,') =',D22.10),/)
     END
```

(d) Output results

```
*** DGIDYB ***

 ** INPUT **

   NX  =  4

   NY  =  4

   SPECIFIED POINTS     XL =  1.5000

                        YL =  1.5000

 COORDINATES (X,Y)

    I   X(I)  Y(I)
    1  1.00  1.00
    2  3.00  3.00
    3  5.00  5.00
    4  7.00  7.00

 FUNCTION VALUES

            I  2.00  0.50  1.00  3.50 I
   Z(X,Y) = I  4.00  1.00  2.00  7.00 I
            I  3.00  0.75  1.50  5.25 I
            I -1.00 -0.25 -0.50 -1.75 I

 ** OUTPUT **

   IERR=   0

   ( THE PARTIAL DERIVATIVES )

    DL(1)=        0.1999023438D+01

    DL(2)=        0.9882812500D+00

    DL(3)=       -0.1390625000D+01

    DL(4)=       -0.6875000000D+00

    DL(5)=       -0.5390625000D+00

    DL(6)=        0.6953125000D+00

    ( SPLINE COEFFICIENTS )

C(1,1,1,1) =      0.2000000000D+01      C(2,1,1,1) =      0.1750000000D+01

C(1,2,1,1) =      0.4000000000D+01      C(2,2,1,1) =      0.2500000000D+00

C(1,3,1,1) =      0.3000000000D+01      C(2,3,1,1) =     -0.1250000000D+01

C(1,4,1,1) =     -0.1000000000D+01      C(2,4,1,1) =     -0.2750000000D+01

C(1,1,2,1) =     -0.1250000000D+01      C(2,1,2,1) =     -0.1093750000D+01

C(1,2,2,1) =     -0.2500000000D+01      C(2,2,2,1) =     -0.1562500000D+00

C(1,3,2,1) =     -0.1875000000D+01      C(2,3,2,1) =      0.7812500000D+00

C(1,4,2,1) =      0.6250000000D+00      C(2,4,2,1) =      0.1718750000D+01

C(1,1,1,2) =      0.5000000000D+00      C(2,1,1,2) =      0.4375000000D+00

C(1,2,1,2) =      0.1000000000D+01      C(2,2,1,2) =      0.6250000000D-01

C(1,3,1,2) =      0.7500000000D+00      C(2,3,1,2) =     -0.3125000000D+00

C(1,4,1,2) =     -0.2500000000D+00      C(2,4,1,2) =     -0.6875000000D+00

C(1,1,2,2) =     -0.2500000000D+00      C(2,1,2,2) =     -0.2187500000D+00

C(1,2,2,2) =     -0.5000000000D+00      C(2,2,2,2) =     -0.3125000000D-01

C(1,3,2,2) =     -0.3750000000D+00      C(2,3,2,2) =      0.1562500000D+00

C(1,4,2,2) =      0.1250000000D+00      C(2,4,2,2) =      0.3437500000D+00

C(1,1,1,3) =      0.1000000000D+01      C(2,1,1,3) =      0.8750000000D+00
```

```
C(1,2,1,3) =      0.2000000000D+01      C(2,2,1,3) =      0.1250000000D+00
C(1,3,1,3) =      0.1500000000D+01      C(2,3,1,3) =     -0.6250000000D+00
C(1,4,1,3) =     -0.5000000000D+00      C(2,4,1,3) =     -0.1375000000D+01
C(1,1,2,3) =      0.7500000000D+00      C(2,1,2,3) =      0.6562500000D+00
C(1,2,2,3) =      0.1500000000D+01      C(2,2,2,3) =      0.9375000000D-01
C(1,3,2,3) =      0.1125000000D+01      C(2,3,2,3) =     -0.4687500000D+00
C(1,4,2,3) =     -0.3750000000D+00      C(2,4,2,3) =     -0.1031250000D+01
C(1,1,1,4) =      0.3500000000D+01      C(2,1,1,4) =      0.3062500000D+01
C(1,2,1,4) =      0.7000000000D+01      C(2,2,1,4) =      0.4375000000D+00
C(1,3,1,4) =      0.5250000000D+01      C(2,3,1,4) =     -0.2187500000D+01
C(1,4,1,4) =     -0.1750000000D+01      C(2,4,1,4) =     -0.4812500000D+01
C(1,1,2,4) =      0.1750000000D+01      C(2,1,2,4) =      0.1531250000D+01
C(1,2,2,4) =      0.3500000000D+01      C(2,2,2,4) =      0.2187500000D+00
C(1,3,2,4) =      0.2625000000D+01      C(2,3,2,4) =     -0.1093750000D+01
C(1,4,2,4) =     -0.8750000000D+00      C(2,4,2,4) =     -0.2406250000D+01
```

6.3.3 DGIIZB, RGIIZB
Double Integral Value

(1) **Function**

DGIIZB or RGIIZB calculates double integral of the bicubic spline function obtained from data on lattice. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGIIZB  (X, NX, Y, NY, Z, AX, BX, CY, DY, Q, WK, IERR)
```

Single precision:

```
CALL RGIIZB  (X, NX, Y, NY, Z, AX, BX, CY, DY, Q, 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	X	{D / R}	NX	Input	X coordinate values of sample points X(i), i = $1, \cdots,$ NX (X(i) < X(i + 1), i ≠ NX)
2	NX	I	1	Input	Number of sample points in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of sample points Y(j), j = $1, \cdots,$ NY (Y(j) < Y(j + 1), j ≠ NY)
4	NY	I	1	Input	Number of sample points in Y direction
5	Z	{D / R}	NX, NY	Input	Array formed from function values z_{ij} at sample point (X(i), Y(j)) Z(i, j) = z_{ij}
6	AX	{D / R}	1	Input	Lower limit of the integration region in the X direction
7	BX	{D / R}	1	Input	Upper limit of the integration region in the X direction
8	CY	{D / R}	1	Input	Lower limit of the integration region in the Y direction
9	DY	{D / R}	1	Input	Upper limit of the integration region in the Y direction

No.	Argument	Type	Size	Input/ Output	Contents
10	Q	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	1	Output	Value of the double integral over the rectangular region $[\text{AX}, \text{BX}] \times [\text{CY}, \text{DY}]$
11	WK	$\left\{\begin{matrix}\text{D}\\\text{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $(\text{NY}+5)\times\text{NX}+\text{NY}-1+\text{MAX}(5\times\text{NX}-4, 5\times\text{NY}-2)$
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\text{NX} \geq 2, \text{NY} \geq 2$

(b) $\text{X}(1) < \text{X}(2) < \cdots < \text{X}(\text{NX})$ (Ascending order)

(c) $\text{Y}(1) < \text{Y}(2) < \cdots < \text{Y}(\text{NY})$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The rectangular region $[\text{AX}, \text{BX}] \times [\text{CY}, \text{DY}]$ went into outside the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) or (c) was not satisfied.	

(6) **Notes**

(a) Bicubic spline coefficients are not calculated. (See Section 6.1.2)

(b) The calculation method differs from that used by subroutine 6.3.7 $\left\{\begin{matrix}\text{DGIIBZ}\\\text{RGIIBZ}\end{matrix}\right\}$. Therefore, the results may not match completely.

(7) **Example**

(a) Problem

When the matrix Z formed from function values z_{ij} at sample points $(\text{x}_\text{i}, \text{y}_\text{j}) = (\text{i}-1, \text{j}-1)$, $(\text{i} = 1, 2, 3, 4;\ \text{j} = 1, 2, 3, 4)$ is given as follows:

$$Z = \begin{bmatrix} 8.0 & 7.0 & 6.0 & 5.0 \\ 2.0 & 3.0 & 4.0 & 5.0 \\ -4.0 & -1.0 & 2.0 & 5.0 \\ -10.0 & -5.0 & 0.0 & 5.0 \end{bmatrix}$$

perform bicubic spline interpolation using these points as sample points. In addition, obtain the double integral of the bicubic spline function over a rectangular region $[0.5, 2.5] \times [0.5, 2.5]$.

(b) Input data

$\text{X}(\text{i}) = \text{x}_\text{i} (\text{i} = 1, \cdots, \text{NX})$,
$\text{Y}(\text{j}) = \text{y}_\text{j} (\text{j} = 1, \cdots, \text{NY})$,
$\text{Z}(\text{i}, \text{j}) = \text{z}_{\text{ij}}$,
NX=4, NY=4, AX=0.5, BX=2.5, CY=0.5 and DY=2.5.

(c) Main program

```
      PROGRAM BGIIZB
!  *** EXAMPLEOF DGIIZB ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER(NX=4,NY=4,AX=0.5,BX=2.5,CY=0.5,DY=2.5)
      DIMENSION X(NX),Y(NY),Z(NX,NY),WK(57)
      READ(5,*) X
      READ(5,*) Y
      READ(5,*) Z
      WRITE(6,1000)  NX,NY,AX,BX,CY,DY,(I,X(I),Y(I),I=1,4)
      WRITE(6,1001)  ((Z(I,J),J=1,4),I=1,4)
      CALL DGIIZB(X,NX,Y,NY,Z,AX,BX,CY,DY,Q,WK,IERR)
      WRITE(6,1300)  IERR
      WRITE(6,1400)  Q
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGIIZB ***',/,/,6X,'** INPUT **',/,/,8X,'NX  =',&
             I3,/,/,8X,'NY  =',I3,/,/,8X,'LIMITS OF INTEGRATION    AX =',&
             F8.4,/,/,33X,'BX =',F8.4,/,/,33X,'CX =',F8.4,/,/,33X,'DX =',F8.4,&
             /,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',3X,&
             'X(I)',2X,'Y(I)',/,4(8X,I2,2F6.2,/))
 1001 FORMAT(' ',/,6X,'FUNCTION VALUES',/,/,17X,'I',4F6.2,' I',/,&
             8X,'Z(X,Y) = I',4F6.2,' I',/,2(17X,'I',4F6.2,' I',/))
 1300 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 1400 FORMAT(' ',/,8X,'( DOUBLE INTEGRAL )',/,/,9X,'Q=',D23.10)
      END
```

(d) Output results

```
   *** DGIIZB ***

    ** INPUT **

      NX  =  4

      NY  =  4

      LIMITS OF INTEGRATION    AX =  0.5000

                               BX =  2.5000

                               CX =  0.5000

                               DX =  2.5000

    COORDINATES (X,Y)

       I   X(I)  Y(I)
       1  0.00  0.00
       2  1.00  1.00
       3  2.00  2.00
       4  3.00  3.00


    FUNCTION VALUES

                I  8.00  7.00  6.00  5.00 I
      Z(X,Y) = I  2.00  3.00  4.00  5.00 I
               I -4.00 -1.00  2.00  5.00 I
               I-10.00 -5.00  0.00  5.00 I


    ** OUTPUT **

      IERR=   0

      ( DOUBLE INTEGRAL )

       Q=       0.8000000000D+01
```

6.3.4 DGICBP, RGICBP
Bicubic Spline Coefficients

(1) **Function**

DGICBP or RGICBP obtains bicubic spline coefficients for "not-a-knot" endpoint conditions. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGICBP  (X, NX, Y, NY, Z, C, WK, IERR)
```

Single precision:

```
CALL RGICBP  (X, NX, Y, NY, Z, C, 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	X	{D / R}	NX	Input	X coordinate values of sample points X(i), i = $1, \cdots, NX$ $(X(i) < X(i+1), i \neq NX)$
2	NX	I	1	Input	Number of sample points in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of sample points Y(j), j = $1, \cdots, NY$ $(Y(j) < Y(j+1), j \neq NY)$
4	NY	I	1	Input	Number of sample points in Y direction
5	Z	{D / R}	NX, NY	Input	Array formed from function values z_{ij} at sample point (X(i), Y(j)) $Z(i,j) = z_{ij}$
6	C	{D / R}	4, NX, NY	Output	Bicubic spline coefficients (See Notes (a))
7	WK	{D / R}	See Contents	Work	Work area **Size**: $2 \times NX \times NY + 2 \times MAX(NX, NY)$
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $NX \geq 4, NY \geq 4$

(b) $X(1) < X(2) < \cdots < X(NX)$ (Ascending order)

(c) $Y(1) < Y(2) < \cdots < Y(NY)$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) or (c) was not satisfied.	

(6) **Notes**

(a) This subroutine obtain the values z(i, j), $z_x(i,j)$, $z_y(i,j)$, $z_{xy}(i,j)$ as bicubic spline coefficients instead of the values $\alpha_{e,r}^{i,j}$ (See Section 6.1.2).

6.3.5 DGISBX, RGISBX
Interpolation Values According to Bicubic Spline Coefficients

(1) **Function**

DGISBX or RGISBX calculates an interpolation value of bicubic spline function using given bicubic spline coefficients.

(2) **Usage**

Double precision:

```
CALL DGISBX  (X, NX, Y, NY, C, XL, YL, FL, IERR)
```

Single precision:

```
CALL RGISBX  (X, NX, Y, NY, C, XL, YL, FL, 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}	NX	Input	X coordinate values of knots X(i), $i = 1, \cdots, NX$ $(X(i) < X(i+1), i \neq NX)$
2	NX	I	1	Input	Number of knots in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of knots Y(j), $j = 1, \cdots, NY$ $(Y(j) < Y(j+1), j \neq NY)$
4	NY	I	1	Input	Number of knots in Y direction
5	C	{D / R}	$4, NX, NY$	Input	Bicubic spline coefficients (See Notes (a))
6	XL	{D / R}	1	Input	X coordinate value of point where interpolation value is calculated.
7	YL	{D / R}	1	Input	Y coordinate value of point where interpolation value is calculated.
8	FL	{D / R}	1	Output	Bicubic spline function value at point (XL, YL)
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $NX \geq 4, NY \geq 4$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The interpolation point (XL, YL) was outside the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The bicubic spline coefficients for this subroutine input are z(i, j), z_x(i, j), z_y(i, j), z_{xy}(i, j), not $\alpha_{e,r}^{i,j}$ (See Section 6.1.2).

(b) This subroutine obtain bicubic spline interpolation value at a specific point. To obtain the bicubic spline interpolation value at all lattice point, it is more effective to call subroutine 6.3.1 $\left\{\begin{matrix}\text{DGISXB}\\\text{RGISXB}\end{matrix}\right\}$.

6.3.6 DGIDBY, RGIDBY
Mixed Partial Derivative Values According to Bicubic Spline Coefficients

(1) **Function**

DGIDBY or RGIDBY calculates mixed partial derivatives of bicubic spline function using given bicubic spline coefficients.

(2) **Usage**

Double precision:

CALL DGIDBY (X, NX, Y, NY, C, XL, YL, DL, IERR)

Single precision:

CALL RGIDBY (X, NX, Y, NY, C, XL, YL, DL, 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}	NX	Input	X coordinate values of knots X(i), i = 1, ⋯, NX (X(i) < X(i + 1), i ≠ NX)
2	NX	I	1	Input	Number of knots in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of knots Y(j), j = 1, ⋯, NY (Y(j) < Y(j + 1), j ≠ NY)
4	NY	I	1	Input	Number of knots in Y direction
5	C	{D / R}	4, NX, NY	Input	Bicubic spline coefficients (See Notes (a))
6	XL	{D / R}	1	Input	X coordinate value of point where mixed partial derivative is calculated.
7	YL	{D / R}	1	Input	Y coordinate value of point where mixed partial derivative is calculated.
8	DL	{D / R}	6	Output	Bicubic spline function value $f(x, y)$ and its mixed partial derivatives at point (x, y)=(XL, YL): DL(1)=f, DL(2)=$\partial f/\partial x$, DL(3)=$\partial f/\partial y$, DL(4)=$\partial^2 f/(\partial x \partial y)$ DL(5)=$\partial^2 f/(\partial x)^2$ DL(6)=$\partial^2 f/(\partial y)^2$
9	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) NX ≥ 4, NY ≥ 4

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The point (XL, YL) was outside the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The bicubic spline coefficients for this subroutine input are z(i, j), z_x(i, j), z_y(i, j), z_{xy}(i, j), not $\alpha_{e,r}^{i,j}$ (See Section 6.1.2).

6.3.7 DGIIBZ, RGIIBZ
Double Integral Value According to Bicubic Spline Coefficients

(1) **Function**

DGIIBZ or RGIIBZ obtains the value of an integral of bicubic spline function over a rectangular region using given bicubic spline coefficients.

(2) **Usage**

Double precision:

CALL DGIIBZ (X, NX, Y, NY, C, AX, BX, CY, DY, Q, IERR)

Single precision:

CALL RGIIBZ (X, NX, Y, NY, C, AX, BX, CY, DY, Q, 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}	NX	Input	X coordinate values of knots X(i), $i = 1, \cdots, NX$ $(X(i) < X(i+1), i \neq NX)$
2	NX	I	1	Input	Number of knots in X direction
3	Y	{D / R}	NY	Input	Y coordinate values of knots Y(j), $j = 1, \cdots, NY$ $(Y(j) < Y(j+1), j \neq NY)$
4	NY	I	1	Input	Number of knots in Y direction
5	C	{D / R}	4, NX, NY	Input	Bicubic spline coefficients (See Notes (a))
6	AX	{D / R}	1	Input	Lower limit of the integration region in the X direction
7	BX	{D / R}	1	Input	Upper limit of the integration region in the X direction
8	CY	{D / R}	1	Input	Lower limit of the integration region in the Y direction
9	DY	{D / R}	1	Input	Upper limit of the integration region in the Y direction
10	Q	{D / R}	1	Output	Value of the double integral over the rectangular region $[AX, BX] \times [CY, DY]$
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NX} \geq 4, \mathrm{NY} \geq 4$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	The rectangular region $[\mathrm{AX}, \mathrm{BX}] \times [\mathrm{CY}, \mathrm{DY}]$ went into outside the interpolation region.	A value extrapolated by using the bicubic spline coefficients on the boundary is output.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The bicubic spline coefficients for this subroutine input are z(i, j), $\mathrm{z_x}$(i, j), $\mathrm{z_y}$(i, j), $\mathrm{z_{xy}}$(i, j), not $\alpha_{e,r}^{i,j}$ (See Section 6.1.2).

(b) The calculation method differs from that used by subroutine 6.3.3 $\left\{\begin{matrix}\mathrm{DGIIZB} \\ \mathrm{RGIIZB}\end{matrix}\right\}$. Therefore, the results may not match completely.

6.4 PLANE DATA INTERPOLATION

6.4.1 DGISPO, RGISPO
Open Curve Interpolation

(1) **Function**

DGISPO or RGISPO performs a plane data interpolation when the string of input data points takes the shape of an open curve. The knots are set equal to sample points.

(2) **Usage**

Double precision:

```
CALL DGISPO  (X, Y, N, IS, IE, M, XO, YO, WK, IERR)
```

Single precision:

```
CALL RGISPO  (X, Y, N, IS, IE, M, XO, YO, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), Y(i)), i = 1, \cdots, N$ (Input them in the order they are to be interpolated)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
3	N	I	1	Input	Number of sample points
4	IS	I	1	Input	Sample point element number of interpolation starting point Interpolation starting point is $(X(IS), Y(IS))$
5	IE	I	1	Input	Sample point element number of interpolation end point Interpolation end point is $(X(IE), Y(IE))$
6	M	I	1	Input	Number of interpolation data values (=Number of subdivisions+1)
7	XO	{D / R}	M	Output	Abscissa value XO(i) of interpolated point
8	YO	{D / R}	M	Output	Ordinate value YO(i) of interpolated point

No.	Argument	Type	Size	Input/ Output	Contents
9	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area **Size**: $4 \times \mathrm{N} + \mathrm{M} - 3$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{N} \geq 2$

(b) $\mathrm{M} \geq 2$

(c) $1 \leq \mathrm{IS} \leq \mathrm{N}$

(d) $1 \leq \mathrm{IE} \leq \mathrm{N}$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	X coordinate became to be out of interpolation interval.	The extrapolated value is output for points out of interpolation interval.
3000	Restriction (a), (b), (c) or (d) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) Generally, obtained interpolation points are not equally spaced.

(7) **Example**

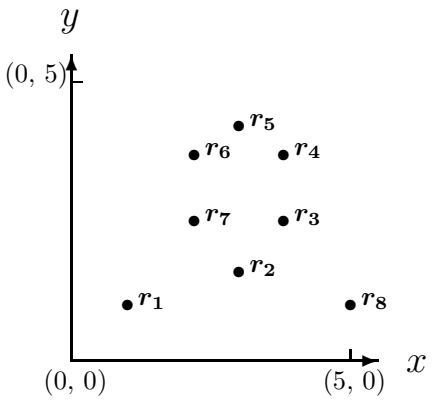
(a) Problem

If data points are given as shown in the figure, then interpolate points in the following order with 20 points.

Order: $\boldsymbol{r_1} \to \boldsymbol{r_2} \to \boldsymbol{r_3} \to \boldsymbol{r_4} \to \boldsymbol{r_5} \to \boldsymbol{r_6} \to \boldsymbol{r_7} \to \boldsymbol{r_2} \to \boldsymbol{r_8}$

where:

$\boldsymbol{r_1} = (1.0, 1.0)$, $\boldsymbol{r_2} = (3.0, 1.6)$, $\boldsymbol{r_3} = (3.8, 2.5)$, $\boldsymbol{r_4} = (3.8, 3.7)$,
$\boldsymbol{r_5} = (3.0, 4.2)$, $\boldsymbol{r_6} = (2.2, 3.7)$, $\boldsymbol{r_7} = (2.2, 2.5)$, $\boldsymbol{r_8} = (5.0, 1.0)$.



(b) Input data

Array X and Y, N = 9, IS = 1, IE = 9 and M = 20.

(c) Main program

```
      PROGRAM BGISPO
! *** EXAMPLEOF DGISPO ***
      IMPLICIT REAL(8) (D)
      PARAMETER (N=9,M=20)
      DIMENSION DX(9),DY(9),DXO(20),DYO(20),DWK(53)
      READ(5,*) DX
      READ(5,*) DY
      IS = 1
      IE = 9
      WRITE(6,1000)   N,IS,IE,M,(I,DX(I),DY(I),I=1,9)
      CALL DGISPO(DX,DY,N,IS,IE,M,DXO,DYO,DWK,IERR)
      WRITE(6,2000)   IERR
      WRITE(6,2001)   (I,DXO(I),DYO(I),I=1,M)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGISPO ***',/,/,6X,'** INPUT **',/,/,8X,'N  =',&
             I3,/,/,8X,'IS =',I3,/,/,8X,'IE =',&
             I3,/,/,8X,'M  =',I3,/,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',5X,&
             'X(I)',9X,'Y(I)',/,9(8X,I2,F12.6,F13.6,/))
 2000 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 2001 FORMAT(' ',/,9X,'I',5X,'XO(I)',8X,'YO(I)',/,20(8X,I2,F12.6,F13.6,/))
      END
```

(d) Output results

```
*** DGISPO ***

 ** INPUT **

   N  =  9

   IS =  1

   IE =  9

   M  = 20

  COORDINATES (X,Y)

     I     X(I)         Y(I)
     1    1.000000     1.000000
     2    3.000000     1.600000
     3    3.800000     2.500000
     4    3.800000     3.700000
     5    3.000000     4.200000
     6    2.200000     3.700000
     7    2.200000     2.500000
     8    3.000000     1.600000
     9    5.000000     1.000000


  ** OUTPUT **

    IERR=   0

     I     XO(I)        YO(I)
     1    1.000000     1.000000
     2    1.539007     1.061616
     3    2.107448     1.196011
     4    2.664189     1.411954
     5    3.168095     1.718216
     6    3.578033     2.123565
     7    3.852763     2.636448
     8    3.937679     3.224415
     9    3.752102     3.774691
    10    3.285278     4.144166
    11    2.714722     4.144166
    12    2.247898     3.774691
    13    2.062321     3.224415
    14    2.147237     2.636448
    15    2.421967     2.123565
    16    2.831905     1.718216
    17    3.335811     1.411954
    18    3.892552     1.196011
    19    4.460993     1.061616
    20    5.000000     1.000000
```

6.4.2 DGISPR, RGISPR
Closed Curve Interpolation

(1) **Function**

DGISPR or RGISPR performs a plane data interpolation when the string of input data points takes the shape of a closed curve. The knots are set equal to sample points.

(2) **Usage**

Double precision:

CALL DGISPR (X, Y, N, IS, IE, M, XO, YO, WK, IERR)

Single precision:

CALL RGISPR (X, Y, N, IS, IE, M, XO, YO, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), Y(i)), i = 1, \cdots, N$ (Input them in the order they are to be interpolated)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
3	N	I	1	Input	Number of sample points
4	IS	I	1	Input	Sample point element number of interpolation starting point Interpolation starting point is $(X(IS), Y(IS))$
5	IE	I	1	Input	Sample point element number of interpolation end point Interpolation end point is $(X(IE), Y(IE))$
6	M	I	1	Input	Number of interpolation data values (=Number of subdivisions+1)
7	XO	{D / R}	M	Output	Abscissa value XO(i) of interpolated point
8	YO	{D / R}	M	Output	Ordinate value YO(i) of interpolated point
9	WK	{D / R}	See Contents	Work	Work area **Size**: $9 \times N + M - 3$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $M \geq 2$

(c) $1 \leq IS \leq N$

(d) $1 \leq IE \leq N$

(e) $X(1) = X(N)$, $Y(1) = Y(N)$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	X coordinate became to be out of interpolation interval.	The extrapolated value is output for points out of interpolation interval.
2000	Restriction (e) was not satisfied.	Processing is performed regarding $(X(N), Y(N))$ as $(X(1), Y(1))$.
3000	Restriction (a), (b), (c) or (d) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) Generally, obtained interpolation points are not equally spaced.

(7) **Example**

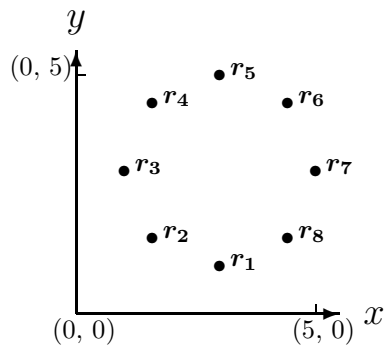
(a) Problem

If data points are given as shown in the figure, then interpolate points in the following order with 20 points.

Order: $\boldsymbol{r_1} \to \boldsymbol{r_2} \to \boldsymbol{r_3} \to \boldsymbol{r_4} \to \boldsymbol{r_5} \to \boldsymbol{r_6} \to \boldsymbol{r_7} \to \boldsymbol{r_8} \to \boldsymbol{r_1}$

where:

$\boldsymbol{r_1} = (3.0, 1.0)$, $\boldsymbol{r_2} = (1.5858, 1.5858)$, $\boldsymbol{r_3} = (1.0, 3.0)$, $\boldsymbol{r_4} = (1.5858, 4.4142)$,
$\boldsymbol{r_5} = (3.0, 5.0)$, $\boldsymbol{r_6} = (4.4142, 4.4142)$, $\boldsymbol{r_7} = (5.0, 3.0)$, $\boldsymbol{r_8} = (4.4142, 1.5858)$.



(b) Input data

Array X and Y, N = 9, IS = 1, IE = 9 and M = 20.

(c) Main program

```
      PROGRAM BGISPR
!  *** EXAMPLEOF DGISPR ***
      IMPLICIT REAL(8) (D)
      PARAMETER (N=9,M=20)
      DIMENSION DX(9),DY(9),DXO(20),DYO(20),DWK(98)
      READ(5,*) DX
      READ(5,*) DY
      IS = 1
      IE = 9
      WRITE(6,1000)   N,IS,IE,M,(I,DX(I),DY(I),I=1,9)
      CALL DGISPR(DX,DY,N,IS,IE,M,DXO,DYO,DWK,IERR)
      WRITE(6,2000)   IERR
      WRITE(6,2001)   (I,DXO(I),DYO(I),I=1,M)
      STOP
 1000 FORMAT(' ',/,/,5X,'*** DGISPR ***',/,/,6X,'** INPUT **',/,/,8X,'N  =',&
             I3,/,/,8X,'IS =',I3,/,/,8X,'IE =',&
             I3,/,/,8X,'M  =',I3,/,/,6X,'COORDINATES (X,Y)',/,/,9X,'I',5X,&
             'X(I)',9X,'Y(I)',/,9(8X,I2,F12.6,F13.6,/))
 2000 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 2001 FORMAT(' ',/,9X,'I',5X,'XO(I)',8X,'YO(I)',/,20(8X,I2,F12.6,F13.6,/))
      END
```

(d) Output results

```
   *** DGISPR ***

    ** INPUT **

      N  =  9

      IS =  1

      IE =  9

      M  = 20

    COORDINATES (X,Y)

       I     X(I)         Y(I)
       1    3.000000     1.000000
       2    1.585800     1.585800
       3    1.000000     3.000000
       4    1.585800     4.414200
       5    3.000000     5.000000
       6    4.414200     4.414200
       7    5.000000     3.000000
       8    4.414200     1.585800
       9    3.000000     1.000000


    ** OUTPUT **

      IERR=   0

       I     XO(I)        YO(I)
       1    3.000000     1.000000
       2    2.351547     1.110361
       3    1.771620     1.422514
       4    1.327111     1.906430
       5    1.062744     2.509913
       6    1.007119     3.164790
       7    1.170590     3.802536
       8    1.528754     4.354625
       9    2.048735     4.757012
      10    2.671491     4.971818
      11    3.328509     4.971818
      12    3.951265     4.757012
      13    4.471246     4.354625
      14    4.829410     3.802536
      15    4.992881     3.164790
      16    4.937256     2.509913
      17    4.672889     1.906430
      18    4.228380     1.422514
      19    3.648453     1.110361
      20    3.000000     1.000000
```

6.4.3 DGISSO, RGISSO
Open Curve Smoothed Interpolation

(1) **Function**

DGISSO or RGISSO performs a plane data smoothed interpolation when the string of input data points takes the shape of an open curve. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

CALL DGISSO (X, Y, N, IS, IE, M, XO, YO, WK, IERR)

Single precision:

CALL RGISSO (X, Y, N, IS, IE, M, XO, YO, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point $(X(i), Y(i)), i = 1, \cdots, N$ (Input them in the order they are to be interpolated)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
3	N	I	1	Input	Number of sample points
4	IS	I	1	Input	Sample point element number of interpolation starting point Interpolation starting point is $(X(IS), Y(IS))$
5	IE	I	1	Input	Sample point element number of interpolation end point Interpolation end point is $(X(IE), Y(IE))$
6	M	I	1	Input	Number of interpolation data values (=Number of subdivisions+1)
7	XO	{D / R}	M	Output	Abscissa value XO(i) of interpolated point
8	YO	{D / R}	M	Output	Ordinate value YO(i) of interpolated point
9	WK	{D / R}	See Contents	Work	Work area **Size**: $2 \times N \times N + 9 \times N + M - 3$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $M \geq 2$

(c) $1 \leq IS \leq N$

(d) $1 \leq IE \leq N$

(e) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	X coordinate became to be out of interpolation interval.	The extrapolated value is output for points out of interpolation interval.
3000	Restriction (a), (b), (c) or (d) was not satisfied.	Processing is aborted.
3010	Restriction (e) was not satisfied.	
4000	The cross validation minimum value was not found (data is uncorrelated).	

(6) **Notes**

(a) Generally, obtained interpolation points are not equally spaced.

(7) **Example**

(a) Problem

$$\begin{cases} x_i = 7.5S_i^3 - 11.5S_i^2 + 5S_i + e_{xi} \\ y_i = -0.67S_i^3 - S_i^2 + 2S_i + e_{yi} \end{cases} \quad (i = 1, \cdots, 25)$$

$$S_i = 0.04 \times i$$

Assume that e_{xi} and e_{yi} are normal distribution random numbers having mean 0 and standard deviation 0.05.

(b) Input data
N=25, IS=1, IE=25 and M=20.

(c) Main program

```
      PROGRAM BGISSO
! *** EXAMPLEOF DGISSO ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=25,M=20)
      DIMENSION RN(50),X(25),Y(25),XO(20),YO(20),WK(1492)
      IX = 1
      IY = 1
      AM = 0.0D0
      SG = 0.05D0
      CALL DJDBNO(50,AM,SG,IX,IY,RN,IERR)
      SI = 0.04D0
      ESI = 0.04D0
      DO 10 I=1,25
         X(I) = ((7.5D0*SI-11.5D0)*SI+5.D0)*SI+RN(2*I-1)
         Y(I) = ((-.67D0*SI-1.0D0)*SI+2.D0)*SI+RN(2*I)
         SI = SI+ESI
  10  CONTINUE
      IS = 1
      IE = 25
      WRITE(6,1000) N,IS,IE,M,(I,X(I),Y(I),I=1,25)
!
```

```
      CALL DGISSO(X,Y,N,IS,IE,M,XO,YO,WK,IERR)
!
      WRITE(6,2000) IERR
      WRITE(6,2001) (I,XO(I),YO(I),I=1,M)
      STOP
 1000 FORMAT(' ',/,/,&
             5X,'*** DGISSO ***',/,/,&
             6X,'** INPUT **',/,/,&
             8X,'N  =',I3,/,/,&
             8X,'IS =',I3,/,/,&
             8X,'IE =',I3,/,/,&
             8X,'M  =',I3,/,/,&
             6X,'COORDINATES (X,Y)',/,/,&
             9X,'I',5X,'X(I)',9X,'Y(I)',/,&
          25(8X,I2,F12.6,F13.6,/))
 2000 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 2001 FORMAT(' ',/,9X,'I',5X,'XO(I)',8X,'YO(I)',/,20(8X,I2,F12.6,F13.6,/))
      END
```

(d) Output results

```
 *** DGISSO ***

  ** INPUT **

    N  = 25

    IS =  1

    IE = 25

    M  = 20

  COORDINATES (X,Y)

     I     X(I)         Y(I)
     1    0.086229     0.092834
     2    0.399696     0.146783
     3    0.393525     0.236404
     4    0.555558     0.310744
     5    0.506236     0.318227
     6    0.636209     0.354766
     7    0.684160     0.544700
     8    0.712913     0.561848
     9    0.609777     0.636983
    10    0.609383     0.632210
    11    0.670828     0.565832
    12    0.492495     0.523301
    13    0.522747     0.669854
    14    0.543484     0.808033
    15    0.473625     0.633059
    16    0.510784     0.622908
    17    0.425944     0.730020
    18    0.406879     0.704866
    19    0.497104     0.656456
    20    0.477562     0.578033
    21    0.563355     0.501534
    22    0.594122     0.571620
    23    0.737901     0.570101
    24    0.856588     0.402942
    25    0.989914     0.393659


  ** OUTPUT **

    IERR=   0

     I     XO(I)        YO(I)
     1    0.098636     0.059718
     2    0.233760     0.126601
     3    0.356571     0.195354
     4    0.459831     0.267424
     5    0.549465     0.342219
     6    0.623363     0.417992
     7    0.662661     0.490677
     8    0.651722     0.553686
     9    0.609959     0.602901
    10    0.559036     0.639896
    11    0.527104     0.666685
    12    0.505890     0.680685
    13    0.479247     0.679536
    14    0.455391     0.665304
    15    0.466187     0.638085
    16    0.523418     0.600084
    17    0.617333     0.556038
    18    0.732031     0.507563
    19    0.852980     0.455483
    20    0.980219     0.402289
```

6.4.4 DGISSR, RGISSR
Closed Curve Smoothed Interpolation

(1) **Function**

DGISSR or RGISSR performs a plane data smoothed interpolation when the string of input data points takes the shape of a closed curve. Abscissa values of knots are set equal to abscissa values of sample points.

(2) **Usage**

Double precision:

```
CALL DGISSR  (X, Y, N, IS, IE, M, XO, YO, WK, IERR)
```

Single precision:

```
CALL RGISSR  (X, Y, N, IS, IE, M, XO, YO, 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	X	{D / R}	N	Input	Abscissa values X(i) of sample point (X(i), Y(i)), i = 1, ···, N (Input them in the order they are to be interpolated)
2	Y	{D / R}	N	Input	Ordinate values of sample points or function values Y(i)
3	N	I	1	Input	Number of sample points
4	IS	I	1	Input	Sample point element number of interpolation starting point Interpolation starting point is (X(IS), Y(IS))
5	IE	I	1	Input	Sample point element number of interpolation end point Interpolation end point is (X(IE), Y(IE))
6	M	I	1	Input	Number of interpolation data values (=Number of subdivisions+1)
7	XO	{D / R}	M	Output	Abscissa value XO(i) of interpolated point
8	YO	{D / R}	M	Output	Ordinate value YO(i) of interpolated point
9	WK	{D / R}	See Contents	Work	Work area **Size:** $2 \times N \times N + 9 \times N + M - 3$
10	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $N \geq 4$

(b) $M \geq 2$

(c) $1 \leq IS \leq N$

(d) $1 \leq IE \leq N$

(e) $X(1) < X(2) < \cdots < X(N)$ (Ascending order)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
1000	X coordinate became to be out of interpolation interval.	The extrapolated value is output for points out of interpolation interval.
3000	Restriction (a), (b), (c) or (d) was not satisfied.	Processing is aborted.
3010	Restriction (e) was not satisfied.	
4000	The cross validation minimum value was not found (data is uncorrelated).	

(6) **Notes**

(a) Generally, obtained interpolation points are not equally spaced.

(b) The point (X(1), Y(1)) must be almost identical with (X(N), Y(N)).

(7) **Example**

(a) Problem

$$\begin{cases} x_i = \sin(S_i) + e_{xi} \\ y_i = \cos(S_i) + e_{yi} \end{cases} \quad (i = 1, \cdots, 20)$$

$$S_i = 0.330694 \times (i - 1)$$

Assume that e_{xi} and e_{yi} are normal distribution random numbers having mean 0 and standard deviation 0.05.

(b) Input data
N=20, IS=1, IE=20 and M=16.

(c) Main program

```
      PROGRAM BGISSR
! *** EXAMPLEOF DGISSR ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER (N=20,M=16)
      DIMENSION RN(40),X(20),Y(20),XO(16),YO(16),WK(993)
      IX = 1
      IY = 1
      AM = 0.0D0
      SG = 0.05D0
      CALL DJDBNO(40,AM,SG,IX,IY,RN,IERR)
      SI = 0.0D0
      ESI = 0.330694D0
      DO 10 I=1,20
         X(I) = SIN(SI)+RN(2*I-1)
         Y(I) = COS(SI)+RN(2*I)
         SI = SI+ESI
   10 CONTINUE
      IS = 1
      IE = 20
```

```
      WRITE(6,1000)   N,IS,IE,M,(I,X(I),Y(I),I=1,20)
!
      CALL DGISSR(X,Y,N,IS,IE,M,XO,YO,WK,IERR)
!
      WRITE(6,2000)   IERR
      WRITE(6,2001)   (I,XO(I),YO(I),I=1,M)
      STOP
 1000 FORMAT(' ',/,/,&
             5X,'*** DGISSR ***',/,/,&
             6X,'** INPUT **',/,/,&
             8X,'N  =',I3,/,/,&
             8X,'IS =',I3,/,/,&
             8X,'IE =',I3,/,/,&
             8X,'M  =',I3,/,/,&
             6X,'COORDINATES (X,Y)',/,/,&
             9X,'I',5X,'X(I)',9X,'Y(I)',/,&
          20(8X,I2,F12.6,F13.6,/))
 2000 FORMAT(' ',/,6X,'** OUTPUT **',/,/,8X,'IERR=',I4)
 2001 FORMAT(' ',/,9X,'I',5X,'XO(I)',8X,'YO(I)',/,16(8X,I2,F12.6,F13.6,/))
      END
```

(d) Output results

```
 *** DGISSR ***

  ** INPUT **

    N  = 20

    IS =  1

    IE = 20

    M  = 16

  COORDINATES (X,Y)

     I     X(I)          Y(I)
     1   -0.095851     1.014477
     2    0.394156     0.939343
     3    0.560377     0.801102
     4    0.856405     0.566036
     5    0.875636     0.209072
     6    0.991513    -0.140951
     7    0.936893    -0.323888
     8    0.780477    -0.631079
     9    0.426205    -0.801631
    10    0.133977    -0.951271
    11   -0.106247    -1.049856
    12   -0.563293    -1.011676
    13   -0.757937    -0.682820
    14   -0.883009    -0.282399
    15   -1.002959    -0.144800
    16   -0.914296     0.173631
    17   -0.851862     0.590038
    18   -0.645093     0.822483
    19   -0.277515     0.953988
    20   -0.002437     0.961073


  ** OUTPUT **

    IERR=   0

     I     XO(I)         YO(I)
     1   -0.049372     0.987775
     2    0.336049     0.966068
     3    0.681186     0.710185
     4    0.893141     0.363916
     5    0.965113    -0.065260
     6    0.862252    -0.480692
     7    0.569078    -0.749396
     8    0.170205    -0.933629
     9   -0.248497    -1.075322
    10   -0.605679    -0.937948
    11   -0.836793    -0.516854
    12   -0.952318    -0.149708
    13   -0.930026     0.274095
    14   -0.764474     0.683135
    15   -0.431771     0.918466
    16   -0.049372     0.987775
```

6.5 B–SPLINE

6.5.1 DGICBS, RGICBS
B-Spline Calculation

(1) **Function**

DGICBS or RGICBS calculates m nonzero B–spline values for x in the range $\xi_0 \leq x \leq \xi_{n+1}$ when the knots $\xi_{1-m}, \xi_{2-m}, \cdots, \xi_{n+m}$ and the B–spline order (degree+1) m are given.

(2) **Usage**

Double precision:

```
CALL DGICBS  (XD, XK, N, M, ARYN, KNOT, WK, IERR)
```

Single precision:

```
CALL RGICBS  (XD, XK, N, M, ARYN, KNOT, 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	XD	{D / R}	1	Input	Sample points x.
2	XK	{D / R}	N + 2 × M	Input	Knots (including additional knots) ξ_i.
3	N	I	1	Input	Number of internal knots n.
4	M	I	1	Input	B–spline order m.
5	ARYN	{D / R}	M	Output	Normalized B–spline $N_{mi}(x)$.
6	KNOT	I	1	Output	Nonzero B–spline position.
7	WK	{D / R}	M+1	Work	m–order B–spline $M_{mi}(x)$.
8	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) N ≥ 1

(b) M ≥ 1

(c) XK(M) ≤ XD ≤ XK(M + N + 1)

(d) XK(1) ≤ XK(2) ≤ ··· ≤ XK(N + 2 × M)

(e) XK(i) < XK(i + M) (i = 1, 2, ···, N + M)

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3400	Restriction (e) was not satisfied.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(7) **Example**

(a) Problem

Given the knots $\xi_i = \{0, 0, 0, 0, 1, 2, 3, 5, 6, 7, 7, 7, 7\}$ $(i = -3, -2, \cdots, 9)$ and the B–spline order $m = 4$, obtain 4 nonzero B–spline values for $x = 4$.

(b) Input data

XD=4, knots XK, N=5 and M=4.

(c) Main Program

```
      PROGRAM BGICBS
! *** EXAMPLE OF DGICBS ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( N = 5, M = 4 )
      DIMENSION XK(N+2*M)
      DIMENSION ARYN(M),ARYM(M+1)
!
      READ(5,*) XD
      DO 100 I=1,N+2*M
         READ(5,*) XK(I)
  100 CONTINUE
!
      WRITE(6,6000) N,M,XD
      WRITE(6,6010)
      DO 110 I=1,N+2*M
         WRITE(6,6020) I,XK(I)
  110 CONTINUE
!
      CALL DGICBS&
      (XD,XK,N,M,ARYN,KNOT,ARYM,IERR)
!
      WRITE(6,6030) IERR
      IF( IERR .LT. 3000 ) THEN
         WRITE(6,6040)
         DO 120 I=1,M
            WRITE(6,6050) I,ARYN(I)
  120    CONTINUE
      ENDIF
      STOP
 6000 FORMAT(' ',/,&
             /,5X,'*** DGICBS ***',/,&
             /,6X,'** INPUT **',/,/,&
              8X,'N     = ',I6,/,&
              8X,'M     = ',I6,/,&
              8X,'XD = ',F8.3)
 6010 FORMAT(/,8X,'KNOTS ****',/)
 6020 FORMAT( 8X,'XK(',I2,') =',F8.3)
 6030 FORMAT(/,6X,'** OUTPUT **',/,/,&
              8X,'IERR = ',I6)
 6040 FORMAT(/,8X,'VALUES OF NORMALIZED B-SPLINES ****',/)
 6050 FORMAT( 8X,'ARYN(',I2,') =',D15.8)
      END
```

(d) Output results

```
 *** DGICBS ***

  ** INPUT **

    N     =      5
```

```
  M     =      4
  XD =    0.400

  KNOTS ****

  XK( 1) =   0.000
  XK( 2) =   0.000
  XK( 3) =   0.000
  XK( 4) =   0.000
  XK( 5) =   0.100
  XK( 6) =   0.200
  XK( 7) =   0.300
  XK( 8) =   0.500
  XK( 9) =   0.600
  XK(10) =   0.700
  XK(11) =   0.700
  XK(12) =   0.700
  XK(13) =   0.700

** OUTPUT **

  IERR =      0

  VALUES OF NORMALIZED B-SPLINES ****

  ARYN( 1) = 0.41666667D-01
  ARYN( 2) = 0.45833333D+00
  ARYN( 3) = 0.45833333D+00
  ARYN( 4) = 0.41666667D-01
```

6.5.2 DGISI1, RGISI1
Interpolation Using a B-Spline (One-Dimensional Data)

(1) **Function**

DGISI1 or RGISI1 obtains the following spline function of degree $(m-1)$ for interpolating the data f_i $(i = 1, 2, \cdots, N)$ at the sample points x_i $(i = 1, 2, \cdots, N)$:

$$S(x) = \sum_{i=1}^{n+m} c_i N_{mi}(x)$$

(2) **Usage**

Double precision:

CALL DGISI1 (XD, ND, FD, XK, M, XX, NN, S, WK, IWK, IERR)

Single precision:

CALL RGISI1 (XD, ND, FD, XK, M, XX, NN, S, WK, 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	XD	{D / R}	ND	Input	Sample points x_i.
2	ND	I	1	Input	Number of sample points N.
3	FD	{D / R}	ND	Input	Data f_i given at sample points x_i.
4	XK	{D / R}	ND + M	Input	Knots (including additional knots) ξ_i.
5	M	I	1	Input	B–spline order m.
6	XX	{D / R}	NN	Input	x coordinates for calculating interpolation values.
7	NN	I	1	Input	Number of points for calculating interpolation values.
8	S	{D / R}	NN	Output	Approximation function value $S(x)$ at $x = \mathrm{XX(i)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NN})$.
9	WK	{D / R}	See Contents	Work	Work area. **Size**:$\mathrm{ND}^2 + \mathrm{ND} + 2 \times \mathrm{M} + 1$
10	IWK	I	ND	Work	Work area.
11	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{ND} \geq 1$

(b) $\mathrm{ND} - \mathrm{M} \geq 1, \mathrm{M} \geq 1$

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

(d) $\mathrm{XD}(1) < \mathrm{XD}(2) < \cdots < \mathrm{XD}(\mathrm{ND})$

(e) $\mathrm{XK}(\mathrm{M}) \leq \mathrm{XD}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{ND}+1)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$

(f) $\mathrm{XK}(1) \leq \mathrm{XK}(2) \leq \cdots \leq \mathrm{XK}(\mathrm{ND}+\mathrm{M})$

(g) $\mathrm{XK}(\mathrm{i}) < \mathrm{XK}(\mathrm{i}+\mathrm{M})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$

(h) The sample values $\mathrm{XD}(\mathrm{i})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(i) $\mathrm{XK}(\mathrm{M}) \leq \mathrm{XX}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{ND}+1)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NN})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2100	When the normalized equation is solved, there existed the diagonal element which was close to zero in the *LU* decomposition. The precise solutions may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3400	Restriction (d) was not satisfied.	
3500	Restriction (e) was not satisfied.	
3600	Restriction (f) was not satisfied.	
3700	Restriction (g) was not satisfied.	
3800	Restriction (h) was not satisfied.	
3900	Restriction (i) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is $\mathrm{ND} - \mathrm{M}$.

(7) **Example**

(a) Problem

Obtain interpolation values for $x : 0.05, 0.10, \cdots, 0.95$ by interpolating the following data:

$$f_i = \frac{1}{0.01 + (x_i - 0.3)^2} \quad (x_i = 0.0, 0.1, \cdots, 1.0)$$

(b) Input data

Sample points (XD, FD), ND=11, knots XK, M=4, x coordinates for calculating interpolation values XX and NN=19.

(c) Main Program

```
      PROGRAM BGISI1
! *** EXAMPLE OF DGISI1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( ND = 11, M = 4, NN = 19 )
      DIMENSION IWK(ND)
      DIMENSION XD(ND)
      DIMENSION FD(ND)
      DIMENSION XK(ND+M)
      DIMENSION XX(NN)
      DIMENSION S(NN)
      DIMENSION WK(ND*ND+ND+2*M+1)
!
      DO 100 I=1,ND
         XD(I) = 0.1D0 * DBLE(I-1)
         FD(I) = 1.0D0 / ( 0.01D0 + ( XD(I) - 0.3D0 )**2 )
  100 CONTINUE
      DO 110 I=1,ND+M
         READ(5,*) XK(I)
  110 CONTINUE
      DO 120 I=1,NN
         XX(I) = 0.05D0 * I
  120 CONTINUE
!
      WRITE(6,6000) ND,M,NN
      WRITE(6,6010)
      DO 130 I=1,ND
         WRITE(6,6020) I,XD(I)
  130 CONTINUE
      WRITE(6,6030)
      DO 140 I=1,ND+M
         WRITE(6,6040) I,XK(I)
  140 CONTINUE
!
      CALL DGISI1&
      (XD,ND,FD,XK,M,XX,NN,S,WK,IWK,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .LT. 3000 ) THEN
         WRITE(6,6060)
         DO 150 I=1,NN
            WRITE(6,6070) XX(I),S(I)
  150    CONTINUE
      ENDIF
      STOP
 6000 FORMAT(' ',/,&
             /,5X,'*** DGISI1 ***',/,&
             /,6X,'** INPUT **',/,/,&
              8X,'ND = ',I6,/,&
              8X,'M     = ',I6,/,&
              8X,'NN    = ',I6)
 6010 FORMAT(/,8X,'XD ****',/)
 6020 FORMAT( 8X,'XD(',I2,') =',F8.3)
 6030 FORMAT(/,8X,'KNOTS ****',/)
 6040 FORMAT( 8X,'XK(',I2,') =',F8.3)
 6050 FORMAT(/,6X,'** OUTPUT **',/,/,&
              8X,'IERR = ',I6)
 6060 FORMAT(/,8X,&
      'VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****',&
             /,&
             /,13X,'XX',14X,'S',/)
 6070 FORMAT( 8X,F8.3,6X,D15.8)
      END
```

(d) Output results

```
   *** DGISI1 ***

    ** INPUT **

      ND =     11
      M     =      4
      NN    =     19
```

```
  XD ****

  XD( 1) =   0.000
  XD( 2) =   0.100
  XD( 3) =   0.200
  XD( 4) =   0.300
  XD( 5) =   0.400
  XD( 6) =   0.500
  XD( 7) =   0.600
  XD( 8) =   0.700
  XD( 9) =   0.800
  XD(10) =   0.900
  XD(11) =   1.000

  KNOTS ****

  XK( 1) =   0.000
  XK( 2) =   0.000
  XK( 3) =   0.000
  XK( 4) =   0.000
  XK( 5) =   0.200
  XK( 6) =   0.300
  XK( 7) =   0.400
  XK( 8) =   0.500
  XK( 9) =   0.600
  XK(10) =   0.700
  XK(11) =   0.800
  XK(12) =   1.000
  XK(13) =   1.000
  XK(14) =   1.000
  XK(15) =   1.000

** OUTPUT **

  IERR =      0

  VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****

       XX              S

     0.050       0.15725472D+02
     0.100       0.20000000D+02
     0.150       0.29274528D+02
     0.200       0.50000000D+02
     0.250       0.82176415D+02
     0.300       0.10000000D+03
     0.350       0.82019813D+02
     0.400       0.50000000D+02
     0.450       0.29744335D+02
     0.500       0.20000000D+02
     0.550       0.14002848D+02
     0.600       0.10000000D+02
     0.650       0.74795690D+01
     0.700       0.58823529D+01
     0.750       0.47214103D+01
     0.800       0.38461538D+01
     0.850       0.31920846D+01
     0.900       0.27027027D+01
     0.950       0.23235079D+01
```

6.5.3 DGISI2, RGISI2
Interpolation Using a B-Spline (Two-Dimensional Data)

(1) **Function**

DGISI2 or RGISI2 obtains the following spline function for interpolating the data f_{ij} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ at the sample points (x_i, y_j) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$:

$$S(x, y) = \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y)$$

and calculates the interpolation values at the specified grid points.

(2) **Usage**

Double precision:

CALL DGISI2 (XD, NXD, YD, NYD, FD, XK, MX, YK, MY, XX, NNX, YY, NNY, S, WK, IWK, IERR)

Single precision:

CALL RGISI2 (XD, NXD, YD, NYD, FD, XK, MX, YK, MY, XX, NNX, YY, NNY, S, WK, IWK, 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	XD	{D / R}	NXD	Input	Sample points in x direction x_i.
2	NXD	I	1	Input	Number of sample points in x direction N_x.
3	YD	{D / R}	NYD	Input	Sample points in y direction y_j.
4	NYD	I	1	Input	Number of sample points in y direction N_y.
5	FD	{D / R}	NXD, NYD	Input	Data f_{ij} given at sample points (x_i, y_j).
6	XK	{D / R}	NXD + MX	Input	Knots (including additional knots) in x direction ξ_i.
7	MX	I	1	Input	x–direction B–spline order m.
8	YK	{D / R}	NYD + MY	Input	Knots (including additional knots) in y direction ζ_j.
9	MY	I	1	Input	y–direction B–spline order n.
10	XX	{D / R}	NNX	Input	x coordinates for calculating interpolation values.
11	NNX	I	1	Input	Number of x–direction points for calculating interpolation values.
12	YY	{D / R}	NNY	Input	y coordinates for calculating interpolation values.
13	NNY	I	1	Input	Number of y–direction points for calculating interpolation values.
14	S	{D / R}	NNX, NNY	Output	Approximation function value $S(x, y)$ at $(x, y) = (\mathrm{XX(i)}, \mathrm{YY(j)})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX}\ ;\ \mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$.
15	WK	{D / R}	See Contents	Work	Work area. **Size**: $\mathrm{NXD}^2 \times \mathrm{NYD}^2 + \mathrm{NXD} \times (\mathrm{NYD} + \mathrm{MX}) + 2 \times \mathrm{MX} + 2 \times \mathrm{MY} + 2$
16	IWK	I	See Contents	Work	Work area. **Size**: $\mathrm{NXD} \times \mathrm{NYD} + \mathrm{NXD} + \mathrm{NYD}$
17	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NXD} \geq 1, \mathrm{NYD} \geq 1$

(b) $\mathrm{NXD} - \mathrm{MX} \geq 1, \mathrm{NYD} - \mathrm{MY} \geq 1, \mathrm{MX} \geq 1, \mathrm{MY} \geq 1$

(c) $\mathrm{NNX} \geq 1, \mathrm{NNY} \geq 1$

(d) $\mathrm{XD}(1) < \mathrm{XD}(2) < \cdots < \mathrm{XD}(\mathrm{NXD})$

(e) $\mathrm{YD}(1) < \mathrm{YD}(2) < \cdots < \mathrm{YD}(\mathrm{NYD})$

(f) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XD}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(g) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YD}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(h) $\mathrm{XK}(1) \leq \mathrm{XK}(2) \leq \cdots \leq \mathrm{XK}(\mathrm{NXD}+\mathrm{MX})$

(i) $\mathrm{YK}(1) \leq \mathrm{YK}(2) \leq \cdots \leq \mathrm{YK}(\mathrm{NYD}+\mathrm{MY})$

(j) $\mathrm{XK}(\mathrm{i}) < \mathrm{XK}(\mathrm{i}+\mathrm{MX}) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(k) $\mathrm{YK}(\mathrm{j}) < \mathrm{YK}(\mathrm{j}+\mathrm{MY}) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(l) The sample values $\mathrm{XD}(\mathrm{i}) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(m) The sample values $\mathrm{YD}(\mathrm{j}) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(n) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XX}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NNX})$

(o) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YY}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2100	When the normalized equation is solved, there existed the diagonal element which was close to zero in the LU decomposition. The precise solutions may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3400	Restriction (d) was not satisfied.	
3410	Restriction (e) was not satisfied.	
3500	Restriction (f) was not satisfied.	
3510	Restriction (g) was not satisfied.	
3600	Restriction (h) was not satisfied.	
3610	Restriction (i) was not satisfied.	
3700	Restriction (j) was not satisfied.	
3710	Restriction (k) was not satisfied.	
3800	Restriction (l) was not satisfied.	
3810	Restriction (m) was not satisfied.	
3900	Restriction (n) was not satisfied.	
3910	Restriction (o) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is $(\mathrm{NXD} - \mathrm{MX}, \mathrm{NYD} - \mathrm{MY})$.

(7) **Example**

(a) Problem

Obtain interpolation values for $x : 0.0, 0.1, \cdots, 0.9; y : 0.0, 0.1, \cdots, 0.9$ by interpolating the following data:

$$f_{ij} = \frac{1}{0.01 + (x_i - 0.3)^2} + \frac{1}{0.02 + (y_j - 0.4)^2}$$
$$(x_i = 0.0, 0.1, \cdots, 1.0; y_j = 0.0, 0.1, \cdots, 1.0)$$

(b) Input data

Sample points (XD, YD, FD), NXD=11, NYD=11, knots in x direction XK, MX=4, knots in y direction YK, MY=4, x coordinates for calculating interpolation values XX, NNX=10, y coordinates for calculating interpolation values YY and NNY=10.

(c) Main Program

```
      PROGRAM BGISI2
! *** EXAMPLE OF DGISI2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( NXD = 11, MX = 4, NNX = 10 )
      PARAMETER( NYD = 11, MY = 4, NNY = 10 )
      DIMENSION IWK(NXD*NYD+NXD+NYD)
      DIMENSION XD(NXD)
      DIMENSION YD(NYD)
      DIMENSION FD(NXD,NYD)
      DIMENSION XK(NXD+MX)
      DIMENSION YK(NYD+MY)
      DIMENSION XX(NNX)
      DIMENSION YY(NNY)
      DIMENSION S(NNX,NNY)
      DIMENSION WK(NXD**2*NYD**2+NXD*(NYD+MX)+MX*2+MY*2+2)
!
      DO 100 I=1,NXD
         XD(I) = 0.1D0 * DBLE( I - 1 )
  100 CONTINUE
      DO 110 J=1,NYD
         YD(J) = 0.1D0 * DBLE( J - 1 )
  110 CONTINUE
      DO 120 J=1,NYD
      DO 121 I=1,NXD
         FD(I,J) = 1.0D0 / ( 0.01D0 + ( XD(I) - 0.3D0 )**2 )&
                   + 1.0D0 / ( 0.02D0 + ( YD(J) - 0.4D0 )**2 )
  121 CONTINUE
  120 CONTINUE
      DO 130 I=1,NXD+MX
         READ(5,*) XK(I)
  130 CONTINUE
      DO 140 J=1,NYD+MY
         READ(5,*) YK(J)
  140 CONTINUE
      DO 150 I=1,NNX
         XX(I) = 0.1D0 * DBLE( I - 1 )
  150 CONTINUE
      DO 160 J=1,NNY
         YY(J) = 0.1D0 * DBLE( J - 1 )
  160 CONTINUE
!
      WRITE(6,6000) NXD,NYD,MX,MY,NNX,NNY
      WRITE(6,6010)
      DO 170 I=1,NXD
         WRITE(6,6020) I,XD(I)
  170 CONTINUE
      WRITE(6,6030)
      DO 180 J=1,NYD
         WRITE(6,6040) J,YD(J)
  180 CONTINUE
      WRITE(6,6050)
      DO 190 I=1,NXD+MX
         WRITE(6,6060) I,XK(I)
  190 CONTINUE
      WRITE(6,6070)
      DO 200 J=1,NYD+MY
         WRITE(6,6080) J,YK(J)
  200 CONTINUE
!
      CALL DGISI2&
      (XD,NXD,YD,NYD,FD,XK,MX,YK,MY,&
       XX,NNX,YY,NNY,S,WK,IWK,IERR)
!
      WRITE(6,6090) IERR
      IF( IERR .LT. 3000 ) THEN
         WRITE(6,6100)
         DO 210 I=1,NNX
```

```
        DO 211 J=1,NNY
           WRITE(6,6110) XX(I),YY(J),S(I,J)
 211    CONTINUE
 210    CONTINUE
     ENDIF
     STOP
6000 FORMAT(' ',/,&
           /,5X,'*** DGISI2 ***',/,&
           /,6X,'** INPUT **',/,/,&
            8X,'NXD = ',I6,5X,'NYD = ',I6,/,&
            8X,'MX     = ',I6,5X,'MY     = ',I6,/,&
            8X,'NNX    = ',I6,5X,'NNY    = ',I6)
6010 FORMAT(/,8X,'XD ****',/)
6020 FORMAT( 8X,'XD(',I2,') =',F8.3)
6030 FORMAT(/,8X,'YD ****',/)
6040 FORMAT( 8X,'YD(',I2,') =',F8.3)
6050 FORMAT(/,8X,'XKS ****',/)
6060 FORMAT( 8X,'XK(',I2,') =',F8.3)
6070 FORMAT(/,8X,'YKS ****',/)
6080 FORMAT( 8X,'YK(',I2,') =',F8.3)
6120 FORMAT(/,8X,'XX ****',/)
6130 FORMAT( 8X,'XX(',I2,') =',F8.3)
6140 FORMAT(/,8X,'YY ****',/)
6150 FORMAT( 8X,'YY(',I2,') =',F8.3)
6090 FORMAT(/,6X,'** OUTPUT **',/,/,&
            8X,'IERR = ',I6)
6100 FORMAT(/,8X,&
     'VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****',&
           /,&
           /,13X,'XX',8X,'YY',14X,'S',/)
6110 FORMAT( 8X,F8.3,2X,F8.3,6X,D15.8)
     END
```

(d) Output results

```
*** DGISI2 ***

 ** INPUT **

   NXD =     11     NYD =     11
   MX     =      4     MY     =      4
   NNX    =     10     NNY    =     10

   XD ****

   XD( 1) =   0.000
   XD( 2) =   0.100
   XD( 3) =   0.200
   XD( 4) =   0.300
   XD( 5) =   0.400
   XD( 6) =   0.500
   XD( 7) =   0.600
   XD( 8) =   0.700
   XD( 9) =   0.800
   XD(10) =   0.900
   XD(11) =   1.000

   YD ****

   YD( 1) =   0.000
   YD( 2) =   0.100
   YD( 3) =   0.200
   YD( 4) =   0.300
   YD( 5) =   0.400
   YD( 6) =   0.500
   YD( 7) =   0.600
   YD( 8) =   0.700
   YD( 9) =   0.800
   YD(10) =   0.900
   YD(11) =   1.000

   XKS ****

   XK( 1) =   0.000
   XK( 2) =   0.000
   XK( 3) =   0.000
   XK( 4) =   0.000
   XK( 5) =   0.200
   XK( 6) =   0.300
   XK( 7) =   0.400
   XK( 8) =   0.500
   XK( 9) =   0.600
   XK(10) =   0.700
   XK(11) =   0.800
   XK(12) =   1.000
   XK(13) =   1.000
   XK(14) =   1.000
   XK(15) =   1.000

   YKS ****

   YK( 1) =   0.000
   YK( 2) =   0.000
   YK( 3) =   0.000
```

```
  YK( 4) =   0.000
  YK( 5) =   0.200
  YK( 6) =   0.300
  YK( 7) =   0.400
  YK( 8) =   0.500
  YK( 9) =   0.600
  YK(10) =   0.700
  YK(11) =   0.800
  YK(12) =   1.000
  YK(13) =   1.000
  YK(14) =   1.000
  YK(15) =   1.000

** OUTPUT **

  IERR =      0

  VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****

       XX         YY              S

     0.000      0.000       0.15555556D+02
     0.000      0.100       0.19090909D+02
     0.000      0.200       0.26666667D+02
     0.000      0.300       0.43333333D+02
     0.000      0.400       0.60000000D+02
     0.000      0.500       0.43333333D+02
     0.000      0.600       0.26666667D+02
     0.000      0.700       0.19090909D+02
     0.000      0.800       0.15555556D+02
     0.000      0.900       0.13703704D+02
     0.100      0.000       0.25555556D+02
     0.100      0.100       0.29090909D+02
     0.100      0.200       0.36666667D+02
     0.100      0.300       0.53333333D+02
     0.100      0.400       0.70000000D+02
     0.100      0.500       0.53333333D+02
     0.100      0.600       0.36666667D+02
     0.100      0.700       0.29090909D+02
     0.100      0.800       0.25555556D+02
     0.100      0.900       0.23703704D+02
     0.200      0.000       0.55555556D+02
     0.200      0.100       0.59090909D+02
     0.200      0.200       0.66666667D+02
     0.200      0.300       0.83333333D+02
     0.200      0.400       0.10000000D+03
     0.200      0.500       0.83333333D+02
     0.200      0.600       0.66666667D+02
     0.200      0.700       0.59090909D+02
     0.200      0.800       0.55555556D+02
     0.200      0.900       0.53703704D+02
     0.300      0.000       0.10555556D+03
     0.300      0.100       0.10909091D+03
     0.300      0.200       0.11666667D+03
     0.300      0.300       0.13333333D+03
     0.300      0.400       0.15000000D+03
     0.300      0.500       0.13333333D+03
     0.300      0.600       0.11666667D+03
     0.300      0.700       0.10909091D+03
     0.300      0.800       0.10555556D+03
     0.300      0.900       0.10370370D+03
     0.400      0.000       0.55555556D+02
     0.400      0.100       0.59090909D+02
     0.400      0.200       0.66666667D+02
     0.400      0.300       0.83333333D+02
     0.400      0.400       0.10000000D+03
     0.400      0.500       0.83333333D+02
     0.400      0.600       0.66666667D+02
     0.400      0.700       0.59090909D+02
     0.400      0.800       0.55555556D+02
     0.400      0.900       0.53703704D+02
     0.500      0.000       0.25555556D+02
     0.500      0.100       0.29090909D+02
     0.500      0.200       0.36666667D+02
     0.500      0.300       0.53333333D+02
     0.500      0.400       0.70000000D+02
     0.500      0.500       0.53333333D+02
     0.500      0.600       0.36666667D+02
     0.500      0.700       0.29090909D+02
     0.500      0.800       0.25555556D+02
     0.500      0.900       0.23703704D+02
     0.600      0.000       0.15555556D+02
     0.600      0.100       0.19090909D+02
     0.600      0.200       0.26666667D+02
     0.600      0.300       0.43333333D+02
     0.600      0.400       0.60000000D+02
     0.600      0.500       0.43333333D+02
     0.600      0.600       0.26666667D+02
     0.600      0.700       0.19090909D+02
     0.600      0.800       0.15555556D+02
     0.600      0.900       0.13703704D+02
     0.700      0.000       0.11437908D+02
     0.700      0.100       0.14973262D+02
     0.700      0.200       0.22549020D+02
     0.700      0.300       0.39215686D+02
```

```
0.700     0.400     0.55882353D+02
0.700     0.500     0.39215686D+02
0.700     0.600     0.22549020D+02
0.700     0.700     0.14973262D+02
0.700     0.800     0.11437908D+02
0.700     0.900     0.95860566D+01
0.800     0.000     0.94017094D+01
0.800     0.100     0.12937063D+02
0.800     0.200     0.20512821D+02
0.800     0.300     0.37179487D+02
0.800     0.400     0.53846154D+02
0.800     0.500     0.37179487D+02
0.800     0.600     0.20512821D+02
0.800     0.700     0.12937063D+02
0.800     0.800     0.94017094D+01
0.800     0.900     0.75498575D+01
0.900     0.000     0.82582583D+01
0.900     0.100     0.11793612D+02
0.900     0.200     0.19369369D+02
0.900     0.300     0.36036036D+02
0.900     0.400     0.52702703D+02
0.900     0.500     0.36036036D+02
0.900     0.600     0.19369369D+02
0.900     0.700     0.11793612D+02
0.900     0.800     0.82582583D+01
0.900     0.900     0.64064064D+01
```

6.5.4 DGISI3, RGISI3
Interpolation Using a B-Spline (Three-Dimensional Data)

(1) **Function**

DGISI3 or RGISI3 obtains the following spline function for interpolating the data f_{ijk} $(i = 1, 2, \cdots, N_x;\ j = 1, 2, \cdots, N_y;\ k = 1, 2, \cdots, N_z)$ at the sample points (x_i, y_j, z_k) $(i = 1, 2, \cdots, N_x;\ j = 1, 2, \cdots, N_y;\ k = 1, 2, \cdots, N_z)$:

$$S(x, y, z) = \sum_{i=1}^{n_x+m_x} \sum_{j=1}^{n_y+m_y} \sum_{k=1}^{n_z+m_z} c_{ijk} N_{m_x i}(x) N_{m_y j}(y) N_{m_z k}(z)$$

and calculates the interpolation values at the specified grid points.

(2) **Usage**

Double precision:

CALL DGISI3 (XD, NXD, YD, NYD, ZD, NZD, FD, XK, MX, YK, MY, ZK, MZ, XX, NNX, YY, NNY, ZZ, NNZ, S, WK, IWK, IERR)

Single precision:

CALL RGISI3 (XD, NXD, YD, NYD, ZD, NZD, FD, XK, MX, YK, MY, ZK, MZ, XX, NNX, YY, NNY, ZZ, NNZ, S, WK, 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	XD	{ D / R }	NXD	Input	Sample points in x direction x_i.
2	NXD	I	1	Input	Number of sample points in x direction N_x.
3	YD	{ D / R }	NYD	Input	Sample points in y direction y_j.
4	NYD	I	1	Input	Number of sample points in y direction N_y.
5	ZD	{ D / R }	NZD	Input	Sample points in z direction z_k.
6	NZD	I	1	Input	Number of sample points in z direction N_z.
7	FD	{ D / R }	See Contents	Input	Data f_{ijk} given at sample points (x_i, y_j, z_k). **Size**: NXD, NYD, NZD

No.	Argument	Type	Size	Input/ Output	Contents
8	XK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NXD + MX	Input	Knots (including additional knots) in x direction ξ_i.
9	MX	I	1	Input	x–direction B–spline order m_x.
10	YK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NYD + MY	Input	Knots (including additional knots) in y direction ζ_j.
11	MY	I	1	Input	y–direction B–spline order m_y.
12	ZK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NZD + MZ	Input	Knots (including additional knots) in z direction η_k.
13	MZ	I	1	Input	z–direction B–spline order m_z.
14	XX	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNX	Input	x coordinates for calculating interpolation values.
15	NNX	I	1	Input	Number of x–direction points for calculating interpolation values.
16	YY	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNY	Input	y coordinates for calculating interpolation values.
17	NNY	I	1	Input	Number of y–direction points for calculating interpolation values.
18	ZZ	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNZ	Input	z coordinates for calculating interpolation values.
19	NNZ	I	1	Input	Number of z–direction points for calculating interpolation value.
20	S	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Approximation function value $S(x, y, z)$ at $(x, y, z) = (\mathrm{XX(i)}, \mathrm{YY(j)}, \mathrm{ZZ(i)})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX}\ ;\ \mathrm{j} = 1, 2, \cdots, \mathrm{NNY};$ $\mathrm{k} = 1, 2, \cdots, \mathrm{NNZ})$. **Size**: NNX, NNY, NNZ
21	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area. **Size**: $\mathrm{NXD}^2 \times \mathrm{NYD}^2 \times \mathrm{NZD}^2 + \mathrm{NXD} \times \mathrm{NYD} \times \mathrm{NZD} + \mathrm{MX} \times (\mathrm{NXD} + 2) + \mathrm{MY} \times (\mathrm{NYD} + 2) + 2 \times \mathrm{MZ} + 3$
22	IWK	I	See Contents	Work	Work area. **Size**: NXD×NYD×NZD+NXD+NYD+NZD
23	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NXD} \geq 1, \mathrm{NYD} \geq 1, \mathrm{NZD} \geq 1$

(b) $\mathrm{NXD} - \mathrm{MX} \geq 1, \mathrm{NYD} - \mathrm{MY} \geq 1, \mathrm{NZD} - \mathrm{MZ} \geq 1, \mathrm{MX} \geq 1, \mathrm{MY} \geq 1, \mathrm{MZ} \geq 1$

(c) $\mathrm{NNX} \geq 1, \mathrm{NNY} \geq 1, \mathrm{NNZ} \geq 1$

(d) $\mathrm{XD(1)} < \mathrm{XD(2)} < \cdots < \mathrm{XD(NXD)}$

(e) $\mathrm{YD(1)} < \mathrm{YD(2)} < \cdots < \mathrm{YD(NYD)}$

(f) $\mathrm{ZD}(1) < \mathrm{ZD}(2) < \cdots < \mathrm{ZD}(\mathrm{NZD})$

(g) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XD}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(h) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YD}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1)$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(i) $\mathrm{ZK}(\mathrm{MZ}) \leq \mathrm{ZD}(\mathrm{k}) \leq \mathrm{ZK}(\mathrm{NZD}+1)$ $(\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$

(j) $\mathrm{XK}(1) \leq \mathrm{XK}(2) \leq \cdots \leq \mathrm{XK}(\mathrm{NXD}+\mathrm{MX})$

(k) $\mathrm{YK}(1) \leq \mathrm{YK}(2) \leq \cdots \leq \mathrm{YK}(\mathrm{NYD}+\mathrm{MY})$

(l) $\mathrm{ZK}(1) \leq \mathrm{ZK}(2) \leq \cdots \leq \mathrm{ZK}(\mathrm{NZD}+\mathrm{MZ})$

(m) $\mathrm{XK}(\mathrm{i}) < \mathrm{XK}(\mathrm{i}+\mathrm{MX})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(n) $\mathrm{YK}(\mathrm{j}) < \mathrm{YK}(\mathrm{j}+\mathrm{MY})$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(o) $\mathrm{ZK}(\mathrm{k}) < \mathrm{ZK}(\mathrm{k}+\mathrm{MZ})$ $(\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$

(p) The sample values $\mathrm{XD}(\mathrm{i})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(q) The sample values $\mathrm{YD}(\mathrm{j})$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(r) The sample values $\mathrm{ZD}(\mathrm{k})$ $(\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(s) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XX}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1)$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX})$

(t) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YY}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1)$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$

(u) $\mathrm{ZK}(\mathrm{MZ}) \leq \mathrm{ZZ}(\mathrm{k}) \leq \mathrm{ZK}(\mathrm{NZD}+1)$ $(\mathrm{k} = 1, 2, \cdots, \mathrm{NNZ})$

(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.	
3400	Restriction (d) was not satisfied.	
3410	Restriction (e) was not satisfied.	
3420	Restriction (f) was not satisfied.	
3500	Restriction (g) was not satisfied.	
3510	Restriction (h) was not satisfied.	
3520	Restriction (i) was not satisfied.	
3600	Restriction (j) was not satisfied.	
3610	Restriction (k) was not satisfied.	
3620	Restriction (l) was not satisfied.	
3700	Restriction (m) was not satisfied.	
3710	Restriction (n) was not satisfied.	
3720	Restriction (o) was not satisfied.	
3800	Restriction (p) was not satisfied.	
3810	Restriction (q) was not satisfied.	
3820	Restriction (r) was not satisfied.	
3900	Restriction (s) was not satisfied.	

IERR value	Meaning	Processing
3910	Restriction (t) was not satisfied.	Processing is aborted.
3920	Restriction (u) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is $(\mathrm{NXD}-\mathrm{MX}, \mathrm{NYD}-\mathrm{MY}, \mathrm{NZD}-\mathrm{MZ})$.

(7) **Example**

(a) Problem

Obtain interpolation values for $x : 0.0, 0.1, \cdots, 0.8; y : 0.0, 0.1, \cdots, 0.8; z : 0.0, 0.1, \cdots, 0.8$ by interpolating the following data:

$$f_{ijk} = 10 + \frac{1}{0.03 + (x_i - 0.5)^2} + \frac{1}{0.04 + (y_j - 0.2)^2} + \frac{1}{0.05 + (z_k - 0.6)^2}$$
$$(x_i = 0.0, 0.1, \cdots, 0.8; y_j = 0.0, 0.1, \cdots, 0.8; z_k = 0.0, 0.1, \cdots, 0.8)$$

(b) Input data

Sample points (XD, YD, ZD, FD), NXD=9, NYD=9, NZD=9, knots in x direction XK, MX=4, knots in y direction YK, MY=4, knots in z direction ZK, MZ=4, x coordinates for calculating interpolation values XX, NNX=9, y coordinates for calculating interpolation values YY, NNY=9, z coordinates for calculating interpolation values ZZ and NNZ=9.

(c) Main Program

```
      PROGRAM BGISI3
! *** EXAMPLE OF DGISI3 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( NXD = 9, MX = 4, NNX = 9 )
      PARAMETER( NYD = 9, MY = 4, NNY = 9 )
      PARAMETER( NZD = 9, MZ = 4, NNZ = 9 )
      DIMENSION IWK(NXD*NYD*NZD+NXD+NYD+NZD)
      DIMENSION XD(NXD),YD(NYD),ZD(NZD),FD(NXD,NYD,NZD)
      DIMENSION XK(NXD+MX),YK(NYD+MY),ZK(NZD+MZ)
      DIMENSION XX(NNX),YY(NNY),ZZ(NNZ),S(NNX,NNY,NNZ)
      DIMENSION WK(NXD**2*NYD**2*NZD**2+NXD*NYD*NZD&
                  +MX*(NXD+2)+MY*(NYD+2)+2*MZ+3)
!
      DO 100 I=1,NXD
         XD(I) = 0.1D0 * DBLE( I - 1 )
  100 CONTINUE
      DO 110 J=1,NYD
         YD(J) = 0.1D0 * DBLE( J - 1 )
  110 CONTINUE
      DO 120 K=1,NZD
         ZD(K) = 0.1D0 * DBLE( K - 1 )
  120 CONTINUE
      DO 130 K=1,NZD
      DO 131 J=1,NYD
      DO 132 I=1,NXD
         FD(I,J,K) = 10.0D0&
                      + 1.0D0 / ( 0.03D0 + ( XD(I) - 0.5D0 )**2 )&
                      + 1.0D0 / ( 0.04D0 + ( YD(J) - 0.2D0 )**2 )&
                      + 1.0D0 / ( 0.05D0 + ( ZD(K) - 0.6D0 )**2 )
  132 CONTINUE
  131 CONTINUE
  130 CONTINUE
      DO 140 I=1,NXD+MX
         READ(5,*) XK(I)
  140 CONTINUE
      DO 150 J=1,NYD+MY
         READ(5,*) YK(J)
  150 CONTINUE
      DO 160 K=1,NZD+MZ
         READ(5,*) ZK(K)
  160 CONTINUE
      DO 170 I=1,NNX
         XX(I) = 0.1D0 * DBLE( I - 1 )
  170 CONTINUE
```

```
      DO 180 J=1,NNY
         YY(J) = 0.1D0 * DBLE( J - 1 )
  180 CONTINUE
      DO 190 K=1,NNZ
         ZZ(K) = 0.1D0 * DBLE( K - 1 )
  190 CONTINUE
!
      WRITE(6,6000) NXD,NYD,NZD,MX,MY,MZ,NNX,NNY,NNZ
!
      CALL DGISI3&
      (XD,NXD,YD,NYD,ZD,NZD,FD,&
       XK,MX,YK,MY,ZK,MZ,&
       XX,NNX,YY,NNY,ZZ,NNZ,S,WK,IWK,IERR)
!
      WRITE(6,6010) IERR
      IF( IERR .LT. 3000 ) THEN
         SUMT2 = 0.0D0
         SUMD2 = 0.0D0
         DO 200 I=1,NNX
         DO 201 J=1,NNY
         DO 202 K=1,NNZ
            FF = 10.0D0&
               + 1.0D0 / ( 0.03D0 + ( XX(I) - 0.5D0 )**2 )&
               + 1.0D0 / ( 0.04D0 + ( YY(J) - 0.2D0 )**2 )&
               + 1.0D0 / ( 0.05D0 + ( ZZ(K) - 0.6D0 )**2 )
            SUMT2 = SUMT2 + FF ** 2
            SUMD2 = SUMD2 + ( FF - S(I,J,K) ) ** 2
  202    CONTINUE
  201    CONTINUE
  200    CONTINUE
         SUMT2 = SQRT(SUMT2)
         SUMD2 = SQRT(SUMD2)
         FIT = 100.0D0
         IF( SUMT2.NE.0.0D0 .OR. SUMD2.NE.0.0D0 ) THEN
            FIT = SUMT2 / ( SUMT2 + SUMD2 ) * 100.0D0
         ENDIF
         WRITE(6,6020) FIT
      ENDIF
      STOP
 6000 FORMAT(' ',/,&
            /,5X,'*** DGISI3 ***',/,&
             /,6X,'** INPUT **',/,/,&
              8X,'NXD = ',I6,5X,'NYD = ',I6,5X,'NZD = ',I6,/,&
              8X,'MX  = ',I6,5X,'MY  = ',I6,5X,'MZ  = ',I6,/,&
              8X,'NNX = ',I6,5X,'NNY = ',I6,5X,'NNZ = ',I6)
 6010 FORMAT(/,6X,'** OUTPUT **',/,/,&
              8X,'IERR = ',I6)
 6020 FORMAT(/,8X,'FITTING RATE = ',F8.3)
      END
```

(d) Output results

```
 *** DGISI3 ***

  ** INPUT **

    NXD =      9     NYD =      9     NZD =      9
    MX  =      4     MY  =      4     MZ  =      4
    NNX =      9     NNY =      9     NNZ =      9

  ** OUTPUT **

    IERR =      0

    FITTING RATE =  100.000
```

6.5.5 DGISS1, RGISS1
B-Spline Smoothing (One-Dimensional Data)

(1) **Function**

DGISS1 or RGISS1 obtains the following spline function of degree $(m-1)$ for smoothing the data f_i $(i = 1, 2, \cdots, N)$ at the sample points x_i $(i = 1, 2, \cdots, N)$:

$$S(x) \quad = \quad \sum_{i=1}^{n+m} c_i N_{mi}(x)$$

and calculates the approximation function values at the specified points.

(2) **Usage**

Double precision:

CALL DGISS1 (XD, ND, FD, XK, M, XX, NN, S, RS, AIC, WK, IERR)

Single precision:

CALL RGISS1 (XD, ND, FD, XK, M, XX, NN, S, RS, AIC, 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	XD	{D / R}	ND	Input	Sample points x_i.
2	ND	I	1	Input	Number of sample points N.
3	FD	{D / R}	ND	Input	Data f_i given at sample points x_i.
4	XK	{D / R}	ND + M	Input	Knots (including additional knots) ξ_i.
5	M	I	1	Input	B–spline order m.
6	XX	{D / R}	NN	Input	x coordinates for calculating approximation function values.
7	NN	I	1	Input	Number of points for calculating approximation function values.
8	S	{D / R}	NN	Output	Approximation function value $S(x)$ at $x =$ XX(i) (i = 1, 2, ⋯, NN).
9	RS	{D / R}	ND	Output	Residual $S(x_i) - f_i$.

No.	Argument	Type	Size	Input/ Output	Contents
10	AIC	{D R}	1	Output	Akaike's Information Criterion AIC.
11	WK	{D R}	See Contents	Work	Work area. **Size**:$\mathrm{ND}^2 + \mathrm{ND} + 2 \times \mathrm{M} + 1$
12	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{ND} \geq 1$

(b) $\mathrm{ND} - \mathrm{M} \geq 1, \mathrm{M} \geq 1$

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

(d) $\mathrm{XD(1)} < \mathrm{XD(2)} < \cdots < \mathrm{XD(ND)}$

(e) $\mathrm{XK(M)} \leq \mathrm{XD(i)} \leq \mathrm{XK(ND+1)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$

(f) $\mathrm{XK(1)} \leq \mathrm{XK(2)} \leq \cdots \leq \mathrm{XK(ND+M)}$

(g) $\mathrm{XK(i)} < \mathrm{XK(i+M)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$

(h) The sample values $\mathrm{XD(i)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{ND})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(i) $\mathrm{XK(M)} \leq \mathrm{XX(i)} \leq \mathrm{XK(ND+1)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NN})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2100	When the normalized equation is solved, there existed the diagonal element which was close to zero in the LU decomposition. The precise solutions may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3400	Restriction (d) was not satisfied.	
3500	Restriction (e) was not satisfied.	
3600	Restriction (f) was not satisfied.	
3700	Restriction (g) was not satisfied.	
3800	Restriction (h) was not satisfied.	
3900	Restriction (i) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is ND − M.

(7) **Example**

(a) Problem

Perform the fitting to the following data f_i using a cubic spline function.

$$f_i = \frac{1}{0.01 + (x_i - 0.3)^2} + e_i \quad (x_i = 0.0, 0.1, \cdots, 1.0)$$

Here, e_i are mutually independent errors that obey a normal distribution having mean 0 and variance 1.0.

(b) Input data

Sample points (XD, FD), ND=11, knots XK, M=4, x coordinates for calculating interpolation values XX and NN=19.

(c) Main Program

```
      PROGRAM BGISS1
! *** EXAMPLE OF DGISS1 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( ND = 11, M = 4, NN = 19 )
      DIMENSION XD(ND)
      DIMENSION FD(ND)
      DIMENSION XK(ND+M)
      DIMENSION XX(NN)
      DIMENSION S(NN),RS(ND)
      DIMENSION E(ND)
      DIMENSION WK(ND**2+ND+M*2+1)
!
      DO 100 I=1,ND
         XD(I) = 0.1D0 * DBLE( I - 1 )
  100 CONTINUE
      IX = 1
      IY = 1
      AM = 0.0D0
      SG = 1.0D0
      CALL DJDBNO&
      (ND,AM,SG,IX,IY,E,KERR)
      DO 110 I=1,ND
         FD(I) = 1.0D0 / ( 0.01D0 + ( XD(I) - 0.3D0 )**2 ) + E(I)
  110 CONTINUE
      DO 120 I=1,ND+M
         READ(5,*) XK(I)
  120 CONTINUE
      DO 130 I=1,NN
         XX(I) = 0.05D0 * DBLE( I )
  130 CONTINUE
!
      WRITE(6,6000) ND,M,NN
      WRITE(6,6010)
      DO 140 I=1,ND
         WRITE(6,6020) I,XD(I)
  140 CONTINUE
      WRITE(6,6030)
      DO 150 I=1,ND+M
         WRITE(6,6040) I,XK(I)
  150 CONTINUE
!
      CALL DGISS1&
      (XD,ND,FD,XK,M,XX,NN,S,RS,AIC,WK,IERR)
!
      WRITE(6,6050) IERR
      IF( IERR .LT. 3000 ) THEN
         WRITE(6,6060) AIC
         WRITE(6,6070)
         DO 160 I=1,NN
            WRITE(6,6080) XX(I),S(I)
  160    CONTINUE
      ENDIF
      STOP
 6000 FORMAT(' ',/,&
             /,5X,'*** DGISS1 ***',/,&
             /,6X,'** INPUT **',/,/,&
              8X,'ND = ',I6,/,&
              8X,'M     = ',I6,/,&
              8X,'NN    = ',I6)
 6010 FORMAT(/,8X,'XD ****',/)
 6020 FORMAT( 8X,'XD(',I2,') =',F8.3)
```

```
6030 FORMAT(/,8X,'XKS ****',/)
6040 FORMAT( 8X,'XK(',I2,') =',F8.3)
6050 FORMAT(/,6X,'** OUTPUT **',/,/,&
            8X,'IERR = ',I6)
6060 FORMAT(/,8X,'AIC = ',F13.3)
6070 FORMAT(/,8X,&
     'VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****',&
           /,&
           /,13X,'XX',14X,'S',/)
6080 FORMAT( 8X,F8.3,6X,D15.8)
     END
```

(d) Output results

```
*** DGISS1 ***

 ** INPUT **

   ND =     11
   M  =      4
   NN =     19

   XD ****

   XD( 1) =   0.000
   XD( 2) =   0.100
   XD( 3) =   0.200
   XD( 4) =   0.300
   XD( 5) =   0.400
   XD( 6) =   0.500
   XD( 7) =   0.600
   XD( 8) =   0.700
   XD( 9) =   0.800
   XD(10) =   0.900
   XD(11) =   1.000

   XKS ****

   XK( 1) =   0.000
   XK( 2) =   0.000
   XK( 3) =   0.000
   XK( 4) =   0.000
   XK( 5) =   0.200
   XK( 6) =   0.300
   XK( 7) =   0.400
   XK( 8) =   0.500
   XK( 9) =   0.600
   XK(10) =   0.700
   XK(11) =   0.800
   XK(12) =   1.000
   XK(13) =   1.000
   XK(14) =   1.000
   XK(15) =   1.000

 ** OUTPUT **

   IERR =      0

   AIC =      -677.075

   VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****

        XX              S

      0.050       0.14878308D+02
      0.100       0.20289540D+02
      0.150       0.30424026D+02
      0.200       0.51389125D+02
      0.250       0.82995937D+02
      0.300       0.99870525D+02
      0.350       0.81115324D+02
      0.400       0.48923294D+02
      0.450       0.29278541D+02
      0.500       0.20239232D+02
      0.550       0.14394686D+02
      0.600       0.10384769D+02
      0.650       0.80851442D+01
      0.700       0.62641130D+01
      0.750       0.39065356D+01
      0.800       0.19708646D+01
      0.850       0.15091744D+01
      0.900       0.19744345D+01
      0.950       0.24198384D+01
```

6.5.6 DGISS2, RGISS2
B-Spline Smoothing (Two-Dimensional Data)

(1) **Function**

DGISS2 or RGISS2 obtains the following spline function for smoothing the data f_{ij} $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$ at the sample points (x_i, y_j) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y)$:

$$S(x, y) = \sum_{i=1}^{h+m} \sum_{j=1}^{k+n} c_{ij} N_{mi}(x) N_{nj}(y)$$

and calculates the approximate function values at the specified grid points.

(2) **Usage**

Double precision:

CALL DGISS2 (XD, NXD, YD, NYD, FD, XK, MX, YK, MY, XX, NNX, YY, NNY, S, RS, AIC, WK, IWK, IERR)

Single precision:

CALL RGISS2 (XD, NXD, YD, NYD, FD, XK, MX, YK, MY, XX, NNX, YY, NNY, S, RS, AIC, WK, 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	XD	{D / R}	NXD	Input	Sample points in x direction x_i.
2	NXD	I	1	Input	Number of sample points in x direction N_x.
3	YD	{D / R}	NYD	Input	Sample points in y direction y_j.
4	NYD	I	1	Input	Number of sample points in y direction N_y.
5	FD	{D / R}	NXD, NYD	Input	Data f_{ij} given at sample points (x_i, y_j).
6	XK	{D / R}	NXD + MX	Input	Knots (including additional knots) in x direction ξ_i.
7	MX	I	1	Input	x–direction B–spline order m.
8	YK	{D / R}	NYD + MY	Input	Knots (including additional knots) in y direction ζ_j.
9	MY	I	1	Input	y–direction B–spline order n.

No.	Argument	Type	Size	Input/ Output	Contents
10	XX	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNX	Input	x coordinates for calculating approximate function values.
11	NNX	I	1	Input	Number of x–direction points for calculating approximate function values.
12	YY	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNY	Input	y coordinates for calculating approximate function values.
13	NNY	I	1	Input	Number of y–direction points for calculating approximate function values.
14	S	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNX, NNY	Output	Approximation function value $S(x, y)$ at $(x, y) = (\mathrm{XX(i)}, \mathrm{YY(j)})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX}\ ;\ \mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$.
15	RS	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NXD, NYD	Output	Residual $S(x_i, y_j) - f_{ij}$.
16	AIC	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Output	Akaike's Information Criterion AIC.
17	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area. **Size**: $\mathrm{NXD}^2 \times (\mathrm{NYD}^2 + 1) + \mathrm{NXD} \times (\mathrm{NYD} + \mathrm{MX}) + 2 \times \mathrm{MX} + 2 \times \mathrm{MY} + 2$
18	IWK	I	NXD+NYD	Work	Work area.
19	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NXD} \geq 1, \mathrm{NYD} \geq 1$

(b) $\mathrm{NXD} - \mathrm{MX} \geq 1, \mathrm{NYD} - \mathrm{MY} \geq 1, \mathrm{MX} \geq 1, \mathrm{MY} \geq 1$

(c) $\mathrm{NNX} \geq 1, \mathrm{NNY} \geq 1$

(d) $\mathrm{XD(1)} < \mathrm{XD(2)} < \cdots < \mathrm{XD(NXD)}$

(e) $\mathrm{YD(1)} < \mathrm{YD(2)} < \cdots < \mathrm{YD(NYD)}$

(f) $\mathrm{XK(MX)} \leq \mathrm{XD(i)} \leq \mathrm{XK(NXD+1)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(g) $\mathrm{YK(MY)} \leq \mathrm{YD(j)} \leq \mathrm{YK(NYD+1)}$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(h) $\mathrm{XK(1)} \leq \mathrm{XK(2)} \leq \cdots \leq \mathrm{XK(NXD+MX)}$

(i) $\mathrm{YK(1)} \leq \mathrm{YK(2)} \leq \cdots \leq \mathrm{YK(NYD+MY)}$

(j) $\mathrm{XK(i)} < \mathrm{XK(i+MX)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(k) $\mathrm{YK(j)} < \mathrm{YK(j+MY)}$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(l) The sample values $\mathrm{XD(i)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(m) The sample values $\mathrm{YD(j)}$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(n) $\mathrm{XK(MX)} \leq \mathrm{XX(i)} \leq \mathrm{XK(NXD+1)}$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX})$

(o) $\mathrm{YK(MY)} \leq \mathrm{YY(j)} \leq \mathrm{YK(NYD+1)}$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2100	When the normalized equation is solved, there existed the diagonal element which was close to zero in the LU decomposition. The precise solutions may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3400	Restriction (d) was not satisfied.	
3410	Restriction (e) was not satisfied.	
3500	Restriction (f) was not satisfied.	
3510	Restriction (g) was not satisfied.	
3600	Restriction (h) was not satisfied.	
3610	Restriction (i) was not satisfied.	
3700	Restriction (j) was not satisfied.	
3710	Restriction (k) was not satisfied.	
3800	Restriction (l) was not satisfied.	
3810	Restriction (m) was not satisfied.	
3900	Restriction (n) was not satisfied.	
3910	Restriction (o) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is $(\mathrm{NXD}-\mathrm{MX}, \mathrm{NYD}-\mathrm{MY})$.

(7) **Example**

(a) Problem

Perform the fitting to the following data f_{ij} using a cubic spline function.

$$f_{ij} = \frac{1}{0.01+(x_i-0.3)^2} + \frac{1}{0.02+(y_j-0.4)^2} + e_{ij}$$
$$(x_i = 0.0, 0.1, \cdots, 1.0; y_j = 0.0, 0.1, \cdots, 1.0)$$

Here, e_{ij} are mutually independent errors that obey a normal distribution having mean 0 and variance 1.0.

(b) Input data

Sample points (XD, YD, FD), NXD=11, NYD=11,
knot XK, MX=4, knot YK, MY=4,
x coordinates for calculating interpolation values XX, NNX=11,
y coordinates for calculating interpolation values YY and NNY=11.

(c) Main Program

```
      PROGRAM BGISS2
! *** EXAMPLE OF DGISS2 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( NXD = 11, MX = 4, NNX = 11 )
      PARAMETER( NYD = 11, MY = 4, NNY = 11 )
      DIMENSION IWK(NXD+NYD)
      DIMENSION XD(NXD)
      DIMENSION YD(NYD)
      DIMENSION FD(NXD,NYD)
      DIMENSION XK(NXD+MX)
      DIMENSION YK(NYD+MY)
      DIMENSION XX(NNX)
      DIMENSION YY(NNY)
      DIMENSION S(NNX,NNY),RS(NXD,NYD)
      DIMENSION E(NXD*NYD)
      DIMENSION WK(NXD**2*(NYD**2+1)+NXD*(NYD+MX)&
                  +MX*2+MY*2+2)
!
      DO 100 I=1,NXD
         XD(I) = 0.1D0 * DBLE( I - 1 )
  100 CONTINUE
      DO 110 J=1,NYD
         YD(J) = 0.1D0 * DBLE( J - 1 )
  110 CONTINUE
      IX = 1
      IY = 1
      AM = 0.0D0
      SG = 1.0D0
      CALL DJDBNO&
      (NXD*NYD,AM,SG,IX,IY,E,KERR)
      DO 120 J=1,NYD
      DO 121 I=1,NXD
         FD(I,J) = 1.0D0 / ( 0.01D0 + ( XD(I) - 0.3D0 )**2 )&
                   + 1.0D0 / ( 0.02D0 + ( YD(J) - 0.4D0 )**2 )&
                   + E((J-1)*NXD+I)
  121 CONTINUE
  120 CONTINUE
      DO 130 I=1,NXD+MX
         READ(5,*) XK(I)
  130 CONTINUE
      DO 140 J=1,NYD+MY
         READ(5,*) YK(J)
  140 CONTINUE
      DO 150 I=1,NNX
         XX(I) = 0.1D0 * DBLE( I - 1 )
  150 CONTINUE
      DO 160 J=1,NNY
         YY(J) = 0.1D0 * DBLE( J - 1 )
  160 CONTINUE
!
      WRITE(6,6000) NXD,NYD,MX,MY,NNX,NNY
      WRITE(6,6010)
      DO 170 I=1,NXD
         WRITE(6,6020) I,XD(I)
  170 CONTINUE
      WRITE(6,6030)
      DO 180 J=1,NYD
         WRITE(6,6040) J,YD(J)
  180 CONTINUE
      WRITE(6,6050)
      DO 190 I=1,NXD+MX
         WRITE(6,6060) I,XK(I)
  190 CONTINUE
      WRITE(6,6070)
      DO 200 J=1,NYD+MY
         WRITE(6,6080) J,YK(J)
  200 CONTINUE
!
      CALL DGISS2&
      (XD,NXD,YD,NYD,FD,XK,MX,YK,MY,&
       XX,NNX,YY,NNY,S,RS,AIC,WK,IWK,IERR)
!
      WRITE(6,6090) IERR
      IF( IERR .LT. 3000 ) THEN
         WRITE(6,6100) AIC
         WRITE(6,6110)
         DO 210 I=1,NNX
         DO 211 J=1,NNY
            WRITE(6,6120) XX(I),YY(J),S(I,J)
  211    CONTINUE
  210    CONTINUE
      ENDIF
      STOP
 6000 FORMAT(' ',/,&
             /,5X,'*** DGISS2 ***',/,&
             /,6X,'** INPUT **',/,/,&
              8X,'NXD = ',I6,5X,'NYD = ',I6,/,&
              8X,'MX     = ',I6,5X,'MY     = ',I6,/,&
              8X,'NNX    = ',I6,5X,'NNY    = ',I6)
 6010 FORMAT(/,8X,'XD ****',/)
 6020 FORMAT( 8X,'XD(',I2,') =',F8.3)
 6030 FORMAT(/,8X,'YD ****',/)
```

```
6040 FORMAT( 8X,'YD(',I2,') =',F8.3)
6050 FORMAT(/,8X,'XKS ****',/)
6060 FORMAT( 8X,'XK(',I2,') =',F8.3)
6070 FORMAT(/,8X,'YKS ****',/)
6080 FORMAT( 8X,'YK(',I2,') =',F8.3)
6090 FORMAT(/,6X,'** OUTPUT **',/,/,&
            8X,'IERR = ',I6)
6100 FORMAT(/,8X,'AIC = ',F13.3)
6110 FORMAT(/,8X,&
     'VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****',&
           /,&
           /,13X,'XX',8X,'YY',14X,'S',/)
6120 FORMAT( 8X,F8.3,2X,F8.3,6X,D15.8)
     END
```

(d) Output results

```
*** DGISS2 ***

 ** INPUT **

   NXD =     11     NYD =     11
   MX      =      4     MY      =      4
   NNX     =     11     NNY     =     11

   XD ****

   XD( 1) =   0.000
   XD( 2) =   0.100
   XD( 3) =   0.200
   XD( 4) =   0.300
   XD( 5) =   0.400
   XD( 6) =   0.500
   XD( 7) =   0.600
   XD( 8) =   0.700
   XD( 9) =   0.800
   XD(10) =   0.900
   XD(11) =   1.000

   YD ****

   YD( 1) =   0.000
   YD( 2) =   0.100
   YD( 3) =   0.200
   YD( 4) =   0.300
   YD( 5) =   0.400
   YD( 6) =   0.500
   YD( 7) =   0.600
   YD( 8) =   0.700
   YD( 9) =   0.800
   YD(10) =   0.900
   YD(11) =   1.000

   XKS ****

   XK( 1) =   0.000
   XK( 2) =   0.000
   XK( 3) =   0.000
   XK( 4) =   0.000
   XK( 5) =   0.200
   XK( 6) =   0.300
   XK( 7) =   0.400
   XK( 8) =   0.500
   XK( 9) =   0.600
   XK(10) =   0.700
   XK(11) =   0.800
   XK(12) =   1.000
   XK(13) =   1.000
   XK(14) =   1.000
   XK(15) =   1.000

   YKS ****

   YK( 1) =   0.000
   YK( 2) =   0.000
   YK( 3) =   0.000
   YK( 4) =   0.000
   YK( 5) =   0.200
   YK( 6) =   0.300
   YK( 7) =   0.400
   YK( 8) =   0.500
   YK( 9) =   0.600
   YK(10) =   0.700
   YK(11) =   0.800
   YK(12) =   1.000
   YK(13) =   1.000
   YK(14) =   1.000
   YK(15) =   1.000

 ** OUTPUT **

   IERR =      0

   AIC =     -6693.477
```

VALUES OF AN APPROXIMATING SPLINE FUNCTION ON SAMPLE POINTS ****

XX	YY	S
0.000	0.000	0.13638529D+02
0.000	0.100	0.17923477D+02
0.000	0.200	0.24919763D+02
0.000	0.300	0.44195126D+02
0.000	0.400	0.60626824D+02
0.000	0.500	0.43363139D+02
0.000	0.600	0.26116367D+02
0.000	0.700	0.18406707D+02
0.000	0.800	0.15654789D+02
0.000	0.900	0.15358872D+02
0.000	1.000	0.12964439D+02
0.100	0.000	0.25845095D+02
0.100	0.100	0.29513312D+02
0.100	0.200	0.34022614D+02
0.100	0.300	0.52715720D+02
0.100	0.400	0.71964436D+02
0.100	0.500	0.53283156D+02
0.100	0.600	0.35126088D+02
0.100	0.700	0.28834765D+02
0.100	0.800	0.25870747D+02
0.100	0.900	0.24155870D+02
0.100	1.000	0.21481949D+02
0.200	0.000	0.56944680D+02
0.200	0.100	0.60647070D+02
0.200	0.200	0.66222402D+02
0.200	0.300	0.84000179D+02
0.200	0.400	0.10038936D+03
0.200	0.500	0.84892497D+02
0.200	0.600	0.64876867D+02
0.200	0.700	0.59468241D+02
0.200	0.800	0.55086077D+02
0.200	0.900	0.53512599D+02
0.200	1.000	0.51640611D+02
0.300	0.000	0.10542608D+03
0.300	0.100	0.10998597D+03
0.300	0.200	0.11655589D+03
0.300	0.300	0.13427701D+03
0.300	0.400	0.14994631D+03
0.300	0.500	0.13382629D+03
0.300	0.600	0.11524890D+03
0.300	0.700	0.10746238D+03
0.300	0.800	0.10820203D+03
0.300	0.900	0.10294064D+03
0.300	1.000	0.10364633D+03
0.400	0.000	0.54478850D+02
0.400	0.100	0.60014956D+02
0.400	0.200	0.67321948D+02
0.400	0.300	0.83496740D+02
0.400	0.400	0.99798275D+02
0.400	0.500	0.83190280D+02
0.400	0.600	0.64635343D+02
0.400	0.700	0.58433942D+02
0.400	0.800	0.56325893D+02
0.400	0.900	0.53923646D+02
0.400	1.000	0.54000754D+02
0.500	0.000	0.25794787D+02
0.500	0.100	0.28096058D+02
0.500	0.200	0.39052580D+02
0.500	0.300	0.53284576D+02
0.500	0.400	0.71273186D+02
0.500	0.500	0.52471839D+02
0.500	0.600	0.35788154D+02
0.500	0.700	0.28004866D+02
0.500	0.800	0.25298256D+02
0.500	0.900	0.23107380D+02
0.500	1.000	0.21662899D+02
0.600	0.000	0.15940325D+02
0.600	0.100	0.20647757D+02
0.600	0.200	0.26539169D+02
0.600	0.300	0.42554795D+02
0.600	0.400	0.59858030D+02
0.600	0.500	0.43207974D+02
0.600	0.600	0.26330621D+02
0.600	0.700	0.17592775D+02
0.600	0.800	0.16710882D+02
0.600	0.900	0.12725397D+02
0.600	1.000	0.11843629D+02
0.700	0.000	0.11819669D+02
0.700	0.100	0.14360914D+02
0.700	0.200	0.21304596D+02
0.700	0.300	0.39865180D+02
0.700	0.400	0.55387732D+02
0.700	0.500	0.38721332D+02
0.700	0.600	0.21902381D+02
0.700	0.700	0.15159240D+02
0.700	0.800	0.11748982D+02
0.700	0.900	0.89471750D+01
0.700	1.000	0.90357521D+01
0.800	0.000	0.75264201D+01
0.800	0.100	0.13638866D+02

```
0.800     0.200       0.21614906D+02
0.800     0.300       0.35664398D+02
0.800     0.400       0.53605308D+02
0.800     0.500       0.36948173D+02
0.800     0.600       0.21545747D+02
0.800     0.700       0.11644115D+02
0.800     0.800       0.82197703D+01
0.800     0.900       0.81162415D+01
0.800     1.000       0.62005204D+01
0.900     0.000       0.75299901D+01
0.900     0.100       0.12960576D+02
0.900     0.200       0.17932268D+02
0.900     0.300       0.35809671D+02
0.900     0.400       0.52520195D+02
0.900     0.500       0.35080977D+02
0.900     0.600       0.19159762D+02
0.900     0.700       0.11179611D+02
0.900     0.800       0.84038832D+01
0.900     0.900       0.46386709D+01
0.900     1.000       0.49328642D+01
1.000     0.000       0.74541350D+01
1.000     0.100       0.98210060D+01
1.000     0.200       0.18372741D+02
1.000     0.300       0.36185457D+02
1.000     0.400       0.52447660D+02
1.000     0.500       0.35798487D+02
1.000     0.600       0.18177661D+02
1.000     0.700       0.12152331D+02
1.000     0.800       0.75901379D+01
1.000     0.900       0.49307790D+01
1.000     1.000       0.58456048D+01
```

6.5.7 DGISS3, RGISS3
B-Spline Smoothing (Three-Dimensional Data)

(1) **Function**

DGISS3 or RGISS3 obtains the following spline function for smoothing the data f_{ijk} $(i = 1, 2, \cdots, N_x;$ $j = 1, 2, \cdots, N_y;$ $k = 1, 2, \cdots, N_z)$ at the sample points (x_i, y_j, z_k) $(i = 1, 2, \cdots, N_x; j = 1, 2, \cdots, N_y; k = 1, 2, \cdots, N_z)$:

$$S(x, y, z) = \sum_{i=1}^{n_x+m_x} \sum_{j=1}^{n_y+m_y} \sum_{k=1}^{n_z+m_z} c_{ijk} N_{m_x i}(x) N_{m_y j}(y) N_{m_z k}(z)$$

and calculates the approximate function values at the specified grid points.

(2) **Usage**

Double precision:

```
CALL DGISS3 (XD, NXD, YD, NYD, ZD, NZD, FD, XK, MX, YK, MY, ZK, MZ, XX,
             NNX, YY, NNY, ZZ, NNZ, S, RS, AIC, WK, IWK, IERR)
```

Single precision:

```
CALL RGISS3 (XD, NXD, YD, NYD, ZD, NZD, FD, XK, MX, YK, MY, ZK, MZ, XX,
             NNX, YY, NNY, ZZ, NNZ, S, RS, AIC, WK, IWK, 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	XD	{D / R}	NXD	Input	Sample points in x direction x_i.
2	NXD	I	1	Input	Number of sample points in x direction N_x.
3	YD	{D / R}	NYD	Input	Sample points in y direction y_j.
4	NYD	I	1	Input	Number of sample points in y direction N_y.
5	ZD	{D / R}	NZD	Input	Sample points in z direction z_k.
6	NZD	I	1	Input	Number of sample points in z direction N_z.
7	FD	{D / R}	See Contents	Input	Data f_{ijk} given at sample points (x_i, y_j, z_k). **Size**: NXD, NYD, NZD

No.	Argument	Type	Size	Input/ Output	Contents
8	XK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NXD + MX	Input	Knots (including additional knots) in x direction ξ_i.
9	MX	I	1	Input	x–direction B–spline order m_x.
10	YK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NYD + MY	Input	Knots (including additional knots) in y direction ζ_j.
11	MY	I	1	Input	y–direction B–spline order m_y.
12	ZK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NZD + MZ	Input	Knots (including additional knots) in z direction η_k.
13	MZ	I	1	Input	z–direction B–spline order m_z.
14	XX	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNX	Input	x coordinates for calculating approximate function values.
15	NNX	I	1	Input	Number of x–direction points for calculating approximate function values.
16	YY	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNY	Input	y coordinates for calculating approximate function values.
17	NNY	I	1	Input	Number of y–direction points for calculating approximate function values.
18	ZZ	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	NNZ	Input	z coordinates for calculating approximate function values.
19	NNZ	I	1	Input	Number of z–direction points for calculating approximate function values.
20	S	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Approximation function value $S(x, y, z)$ at $(x, y) = (\mathrm{XX(i)}, \mathrm{YY(j)}, \mathrm{ZZ(k)})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{NNX}\ ;\ \mathrm{j} = 1, 2, \cdots, \mathrm{NNY}\ ;$ $\mathrm{k} = 1, 2, \cdots, \mathrm{NNZ})$. **Size**: NNX, NNY, NNZ
21	RS	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Output	Residual $S(x_i, y_j, z_k) - f_{ijk}$. **Size**: NXD, NYD, NZD
22	AIC	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	Output	Akaike's Information Criterion AIC.
23	WK	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	See Contents	Work	Work area. **Size**: $\mathrm{NXD}^2 \times \mathrm{NYD}^2 \times \mathrm{NZD}^2 + \mathrm{NXD} \times \mathrm{NYD} \times \mathrm{NZD} + \mathrm{NXD}^2 \times \mathrm{NYD}^2 + \mathrm{NXD}^2 + \mathrm{MX} \times (\mathrm{NXD} + 2) + \mathrm{MY} \times (\mathrm{NYD} + 2) + 2 \times \mathrm{MZ} + 3$
24	IWK	I	See Contents	Work	Work area. **Size**: NXD × NYD × NZD + NXD + NYD + NZD
25	IERR	I	1	Output	Error indicator

(4) **Restrictions**

(a) $\mathrm{NXD} \geq 1, \mathrm{NYD} \geq 1, \mathrm{NZD} \geq 1$

(b) $\mathrm{NXD} - \mathrm{MX} \geq 1, \mathrm{NYD} - \mathrm{MY} \geq 1, \mathrm{NZD} - \mathrm{MZ} \geq 1, \mathrm{MX} \geq 1, \mathrm{MY} \geq 1, \mathrm{MZ} \geq 1$

(c) $\mathrm{NNX} \geq 1, \mathrm{NNY} \geq 1, \mathrm{NNZ} \geq 1$

(d) $\mathrm{XD}(1) < \mathrm{XD}(2) < \cdots < \mathrm{XD}(\mathrm{NXD})$

(e) $\mathrm{YD}(1) < \mathrm{YD}(2) < \cdots < \mathrm{YD}(\mathrm{NYD})$

(f) $\mathrm{ZD}(1) < \mathrm{ZD}(2) < \cdots < \mathrm{ZD}(\mathrm{NZD})$

(g) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XD}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(h) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YD}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(i) $\mathrm{ZK}(\mathrm{MZ}) \leq \mathrm{ZD}(\mathrm{k}) \leq \mathrm{ZK}(\mathrm{NZD}+1) \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$

(j) $\mathrm{XK}(1) \leq \mathrm{XK}(2) \leq \cdots \leq \mathrm{XK}(\mathrm{NXD}+\mathrm{MX})$

(k) $\mathrm{YK}(1) \leq \mathrm{YK}(2) \leq \cdots \leq \mathrm{YK}(\mathrm{NYD}+\mathrm{MY})$

(l) $\mathrm{ZK}(1) \leq \mathrm{ZK}(2) \leq \cdots \leq \mathrm{ZK}(\mathrm{NZD}+\mathrm{MZ})$

(m) $\mathrm{XK}(\mathrm{i}) < \mathrm{XK}(\mathrm{i}+\mathrm{MX}) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$

(n) $\mathrm{YK}(\mathrm{j}) < \mathrm{YK}(\mathrm{j}+\mathrm{MY}) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$

(o) $\mathrm{ZK}(\mathrm{k}) < \mathrm{ZK}(\mathrm{k}+\mathrm{MZ}) \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$

(p) The sample values $\mathrm{XD}(\mathrm{i}) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NXD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(q) The sample values $\mathrm{YD}(\mathrm{j}) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NYD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(r) The sample values $\mathrm{ZD}(\mathrm{k}) \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{NZD})$ satisfy the Schoenberg–Whitney conditions (see Section 6.1.2).

(s) $\mathrm{XK}(\mathrm{MX}) \leq \mathrm{XX}(\mathrm{i}) \leq \mathrm{XK}(\mathrm{NXD}+1) \quad (\mathrm{i} = 1, 2, \cdots, \mathrm{NNX})$

(t) $\mathrm{YK}(\mathrm{MY}) \leq \mathrm{YY}(\mathrm{j}) \leq \mathrm{YK}(\mathrm{NYD}+1) \quad (\mathrm{j} = 1, 2, \cdots, \mathrm{NNY})$

(u) $\mathrm{ZK}(\mathrm{MZ}) \leq \mathrm{ZZ}(\mathrm{k}) \leq \mathrm{ZK}(\mathrm{NZD}+1) \quad (\mathrm{k} = 1, 2, \cdots, \mathrm{NNZ})$

(5) **Error indicator**

IERR value	Meaning	Processing
0	Normal termination.	
2100	When the normalized equation is solved, there existed the diagonal element which was close to zero in the *LU* decomposition. The precise solutions may not be obtained.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3400	Restriction (d) was not satisfied.	
3410	Restriction (e) was not satisfied.	
3420	Restriction (f) was not satisfied.	
3500	Restriction (g) was not satisfied.	
3510	Restriction (h) was not satisfied.	
3520	Restriction (i) was not satisfied.	
3600	Restriction (j) was not satisfied.	
3610	Restriction (k) was not satisfied.	
3620	Restriction (l) was not satisfied.	
3700	Restriction (m) was not satisfied.	
3710	Restriction (n) was not satisfied.	
3720	Restriction (o) was not satisfied.	
3800	Restriction (p) was not satisfied.	
3810	Restriction (q) was not satisfied.	
3820	Restriction (r) was not satisfied.	
3900	Restriction (s) was not satisfied.	
3910	Restriction (t) was not satisfied.	
3920	Restriction (u) was not satisfied.	
4000	The normalized equation could not be solved.	

(6) **Notes**

(a) Different B–splines may be obtained according to the knot values.

(b) Number of internal knots is $(\mathrm{NXD}-\mathrm{MX}, \mathrm{NYD}-\mathrm{MY}, \mathrm{NZD}-\mathrm{MZ})$.

(c) This library provides both single–precision and double–precision subroutines, the double–precision subroutines should be used for higher precision of solutions.

(7) **Example**

(a) Problem
Perform the fitting to the following data f_{ijk} using a cubic spline function.

$$f_{ijk} = 10 + \frac{1}{0.03+(x_i-0.5)^2} + \frac{1}{0.04+(y_j-0.2)^2} + \frac{1}{0.05+(z_k-0.6)^2} + e_{ijk}$$
$$(x_i = 0.0, 0.1, \cdots, 0.8; y_j = 0.0, 0.1, \cdots, 0.8; z_k = 0.0, 0.1, \cdots, 0.8)$$

Here, e_{ijk} are mutually independent errors that obey a normal distribution having mean 0 and variance 0.01.

(b) Input data

Sample points (XD, YD, ZD, FD), NXD=9, NYD=9, NZD=9,

knots in x direction XK, MX=4,

knots in y direction YK, MY=4,

knots in z direction ZK, MZ=4,

x coordinates for calculating interpolation values XX, NNX=9,

y coordinates for calculating interpolation values YY, NNY=9,

z coordinates for calculating interpolation values ZZ and NNZ=9.

(c) Main Program

```
      PROGRAM BGISS3
! *** EXAMPLE OF DGISS3 ***
      IMPLICIT REAL(8) (A-H,O-Z)
      PARAMETER( NXD = 9, MX = 4, NNX = 9 )
      PARAMETER( NYD = 9, MY = 4, NNY = 9 )
      PARAMETER( NZD = 9, MZ = 4, NNZ = 9 )
      DIMENSION IWK(NXD+NYD+NZD+NXD*NYD*NZD)
      DIMENSION XD(NXD),YD(NYD),ZD(NZD),FD(NXD,NYD,NZD)
      DIMENSION XK(NXD+MX),YK(NYD+MY),ZK(NZD+MZ)
      DIMENSION XX(NNX),YY(NNY),ZZ(NNZ),S(NNX,NNY,NNZ)
      DIMENSION RS(NXD,NYD,NZD)
      DIMENSION E(NXD*NYD*NZD)
      DIMENSION WK(NXD**2*NYD**2*NZD**2&
                  +NXD*NYD*NZD&
                  +NXD**2*NYD**2+NXD**2&
                  +MX*(2+NXD)+MY*(2+NYD)+2*MZ+3)
!
      DO 100 I=1,NXD
         XD(I) = 0.1D0 * DBLE( I - 1 )
  100 CONTINUE
      DO 110 J=1,NYD
         YD(J) = 0.1D0 * DBLE( J - 1 )
  110 CONTINUE
      DO 120 K=1,NZD
         ZD(K) = 0.1D0 * DBLE( K - 1 )
  120 CONTINUE
      IX = 1
      IY = 1
      AM = 0.0D0
      SG = 0.1D0
      CALL DJDBNO&
      (NXD*NYD*NZD,AM,SG,IX,IY,E,KERR)
      DO 130 K=1,NZD
      DO 131 J=1,NYD
      DO 132 I=1,NXD
         FD(I,J,K) = 10.0D0&
                     + 1.0D0 / ( 0.03D0 + ( XD(I) - 0.5D0 )**2 )&
                     + 1.0D0 / ( 0.04D0 + ( YD(J) - 0.2D0 )**2 )&
                     + 1.0D0 / ( 0.05D0 + ( ZD(K) - 0.6D0 )**2 )&
                     + E((K-1)*NXD*NYD+(J-1)*NXD+I)
  132 CONTINUE
  131 CONTINUE
  130 CONTINUE
      DO 140 I=1,NXD+MX
         READ(5,*) XK(I)
  140 CONTINUE
      DO 150 J=1,NYD+MY
         READ(5,*) YK(J)
  150 CONTINUE
      DO 160 K=1,NZD+MZ
         READ(5,*) ZK(K)
  160 CONTINUE
      DO 170 I=1,NNX
         XX(I) = 0.1D0 * DBLE( I - 1 )
  170 CONTINUE
      DO 180 J=1,NNY
         YY(J) = 0.1D0 * DBLE( J - 1 )
  180 CONTINUE
      DO 190 K=1,NNZ
         ZZ(K) = 0.1D0 * DBLE( K - 1 )
  190 CONTINUE
!
      WRITE(6,6000) NXD,NYD,NZD,MX,MY,MZ,NNX,NNY,NNZ
!
      CALL DGISS3&
      (XD,NXD,YD,NYD,ZD,NZD,FD,&
       XK,MX,YK,MY,ZK,MZ,&
       XX,NNX,YY,NNY,ZZ,NNZ,S,RS,AIC,WK,IWK,IERR)
!
      WRITE(6,6010) IERR
      IF( IERR .LT. 3000 ) THEN
```

```
        WRITE(6,6020) AIC
        SUMT2 = 0.0D0
        SUMD2 = 0.0D0
        DO 200 I=1,NNX
        DO 201 J=1,NNY
        DO 202 K=1,NNZ
           FF = 10.0D0&
              + 1.0D0 / ( 0.03D0 + ( XX(I) - 0.5D0 )**2 )&
              + 1.0D0 / ( 0.04D0 + ( YY(J) - 0.2D0 )**2 )&
              + 1.0D0 / ( 0.05D0 + ( ZZ(K) - 0.6D0 )**2 )
           SUMT2 = SUMT2 + FF ** 2
           SUMD2 = SUMD2 + ( FF - S(I,J,K) ) ** 2
 202    CONTINUE
 201    CONTINUE
 200    CONTINUE
        SUMT2 = SQRT(SUMT2)
        SUMD2 = SQRT(SUMD2)
        FIT = 100.0D0
        IF( SUMT2.NE.0.0D0 .OR. SUMD2.NE.0.0D0 ) THEN
           FIT = SUMT2 / ( SUMT2 + SUMD2 ) * 100.0D0
        ENDIF
        WRITE(6,6030) FIT
     ENDIF
     STOP
6000 FORMAT(' ',/,&
           /,5X,'*** DGISS3 ***',/,&
           /,6X,'** INPUT **',/,/,&
            8X,'NXD = ',I6,5X,'NYD = ',I6,5X,'NZD = ',I6,/,&
            8X,'MX  = ',I6,5X,'MY  = ',I6,5X,'MZ  = ',I6,/,&
            8X,'NNX = ',I6,5X,'NNY = ',I6,5X,'NNZ = ',I6)
6010 FORMAT(/,6X,'** OUTPUT **',/,/,&
            8X,'IERR = ',I6)
6020 FORMAT(/,8X,'AIC = ',F13.3)
6030 FORMAT(/,8X,'FITTING RATE = ',F8.3)
     END
```

(d) Output results

```
 *** DGISS3 ***

  ** INPUT **

    NXD =      9     NYD =      9     NZD =      9
    MX  =      4     MY  =      4     MZ  =      4
    NNX =      9     NNY =      9     NNZ =      9

  ** OUTPUT **

    IERR =      0

    AIC =     -3982.473

    FITTING RATE =   99.797
```

Appendix A

MACHINE CONSTANTS USED IN ASL

A.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 A−1 Units for Determining Error

Single-precision	Double-precision
$2^{-23}(\simeq 1.19 \times 10^{-7})$	$2^{-52}(\simeq 2.22 \times 10^{-16})$

Remark: The unit for determining error ε, which is also called the machine ε, is usually defined as the smallest positive constant for which the calculation result of $1+\varepsilon$ differs from 1 in the corresponding floating point mode. Therefore, seeing the unit for determining error enables you to know the maximum number of significant digits of an operation (on the mantissa) in that floating point mode.

A.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 A−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