ADVANCED SCIENTIFIC LIBRARY
ASL C INTERFACE
User's Guide
<Basic Functions Vol.2>

PROPRIETARY NOTICE

PREFACE

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

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 2.

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 function related to storage mode conversion of array data.
3	Basic Matrix Algebra	Explanation of algorithms, method of using, and usage example of function related to basic calculations involving matrices.
4	Eigenvalues and Eigenvectors	Explanation of algorithms, method of using, and usage example of function 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 function 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 function 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 function 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 function 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 function 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 function 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 function 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 function 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 function related to sorting and ranking.
4	Roots of Equations	Explanation of algorithms, method of using, and usage example of function 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 function 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 function related to uniform random number tests, and distribution random number tests.
3	Probability Distributions	Explanation and usage examples of function related to continuous distributions and discrete distributions.
4	Basic Statistics	Explanation and usage examples of function related to basic statistics, variance-covariance and correlation.
5	Tests and Estimates	Explanation and usage examples of function related to interval estimates and tests.
6	Analysis of Variance and Design of Experiments	Explanation and usage examples of function 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 function related to tests using χ^2 distribution and tests using other distributions.
8	Multivariate Analysis	Explanation and usage examples of function related to principal component analysis, factor analysis, canonical correlation analysis, discriminant analysis, cluster analysis.
9	Time Series Analysis	Explanation and usage examples of function related to autocorrelation, cross correlation, autocovariance, cross covariance, smoothing and demand forecasting.
10	Regression analysis	Explanation and usage examples of function 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 function 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 function 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 function 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 function 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 function 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 function 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 C interface

Table 1−1 lists correspondences among product categories, functions of ASL and supported hardware platforms. Interfaces of those functions that have the same name and that belong to the same version of ASL 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 C interface

ASL C interface has the following distinctive characteristics.

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

(2) Special-purpose functions 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 functions.

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

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

1.2 KINDS OF LIBRARIES

Numeric storage units of ASL C interface is 4-byte.

Table 1−2 Kinds of libraries providing ASL C interface

Size of variable(byte)		Declaration of arguments	Kind	Kind of library
integer	real			
4	8	int double	32bit integer Double-precision function	32bit integer library (link option: -lasl_sequential)
4	4	int float	32bit integer Single-precision function	
8	8	long double	64bit integer Double-precision function	64bit integer library (link option: -lasl_sequential_i64)
8	4	long float	64bit integer Single-precision function	

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

(∗2) For compiling the program with functions in the 64-bit integer library, the option "`-DASL_LIB_INT64`" must be specified (See the Note (2) in 1.5).

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 functions, techniques employed, algorithms on which the functions are based, and notes.

1.3.2 Organization of Function Description

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

(1) Function

(2) Usage

(3) Arguments and return value

(4) Restrictions

(5) Error indicator (Return Value)

(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 C interface function.

(2) **Usage**

Usage describes the function name and the order of its arguments. In general, arguments are arranged as follows. When an argument is an address-passing variable, & is appended in front of the argument name.

ierr = function-name (input-arguments, input/output-arguments, output-arguments, isw, work);

isw is an input argument for specifying the processing procedure. ierr is a return value. 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 and return value**

Arguments and return value are explained in the order described above in paragraph (2). The explanation format is as follows.

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

(a) Arguments and return value

Arguments and return value 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 function, 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 `int/ long`, 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 function.

1 : Indicates that argument is a variable.

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

(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 function, information when the argument is input is preserved. The user must assign input-time information unless specifically instructed otherwise. When the argument is a variable, the variable value must be passed.

ii. When only “Output” appears

Results calculated within the function are output to the argument. No data is entered at input time. When the argument is a variable, the variable address must be passed.

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

Argument contents change between the time control passes to the function and the time control returns from the function. The user must assign input-time information unless specifically instructed otherwise. When the argument is a variable, the variable address must be passed.

iv. When “Work” appears

Work indicates that the argument is an area used when performing calculations within the function. A work area having the specified size must be reserved in the program calling this function. 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 function (ASL_dbgmlc, ASL_rbgmlc) that obtains the LU decomposition and the condition number of a real matrix is as follows.

  Double precision:
  ierr = ASL_dbgmlc (a, lna, n, ipvt, &cond, w1);
  Single precision:
  ierr = ASL_rbgmlc (a, lna, n, ipvt, &cond, w1);

  The explanation of the arguments and return value is as follows.

Table 1−3   Sample Arguments and Return Value

D:Double precision real   Z:Double precision complex   I: $\begin{Bmatrix}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{Bmatrix}$
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}^{\text{Note}}$	lna×n	Input	Real matrix A(two-dimensional array)
				Output	The matrix A decomposed into the matrix LU where U is a unit upper triangular matrix and L is a lower triangular matrix.
2	lna	I	1	Input	Adjustable dimension size of array a
3	n	I	1	Input	Order n of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th step.
5	cond	$\begin{Bmatrix}\text{D}\\ \text{R}\end{Bmatrix}$	1	Output	Reciprocal of the condition number
6	w1	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

To use this function, arrays a, ipvt and w1 must first be allocated in the calling program so they can be used as arguments. a is a $\begin{Bmatrix}\text{double-precision}\\ \text{single-precision}\end{Bmatrix}^{\text{Note}}$ real array of size [lna × n] , ipvt is an integer array of size n and w1 is a $\begin{Bmatrix}\text{double-precision}\\ \text{single-precision}\end{Bmatrix}$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 `long`, not `int`.

Note The entries enclosed in brace { } mean that the array should be declared double precision type when using function ASL_dbgmlc and real type when using function ASL_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 function is called. The LU decomposition and condition number of the assigned matrix are calculated with in the function, and the results are stored in array a and variable cond. In addition, pivoting information is stored in ipvt for use by subsequent functions.

ierr is a return value 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 function, its contents at input and output time have no special meaning.

(4) Restrictions

Restrictions indicate limiting ranges for function arguments.

(5) Error indicator (Return Value)

Each function has been given an error indicator as a return value. 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 function, a corresponding value is output to ierr. Error indicator values are divided into five levels.

Table 1−4 Classification of Return Values

Level	Return 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 function.

(7) Example

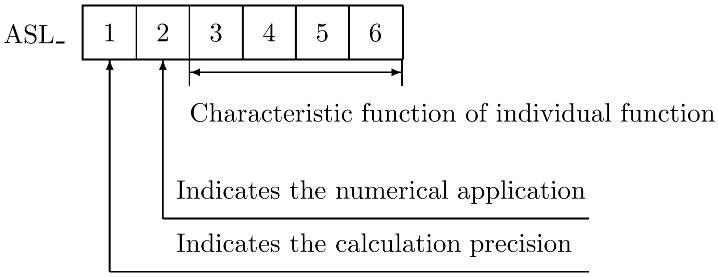
Here gives an example of how to use the function. Note that in some cases, multiple functions are combined in a single example. The output results are given in the 32-bit integer version, and may differ within the range of rounding error if the compiler or intrinsic functions are different.

In addition, when the 64-bit integer version library is used, the `long`-type conversion specification to be given to `printf` or `scanf` must be `%ld`. 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 FUNCTION NAMES

The functions name of ASL C interface basic functions consists of ten characters with a prefix "ASL_" and ⟨six alphanumeric characters⟩.

Figure 1−1 Function Name Components



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

d, w	Double precision real-type calculation
r, v	Single precision real-type calculation
z, j	Double precision complex-type calculation
c, i	Single precision complex-type calculation

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

"2" in Figure 1−1 **:** Currently, the following letters lettererererere are used to indicate the application field in the ASL C interface 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

Letter	Application Field	Volume
s	Sorting and ranking	5, 7
x	Basic matrix algebra	1
	Simultaneous linear equations (iterative method)	7
1	Probability distributions	6
2	Basic statics	6
3	Tests and estimates	6
4	Analysis of variance and design of experiments	6
5	Nonparametric tests	6
6	Multivariate analysis	6

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

1.5 NOTES

(1) To use ASL C interface, the header file `asl.h`must be included.

(2) For compiling the program with functions in ASL C interface 64-bit integer library, the compile option "`-DASL_LIB_INT64`" must be specified. This option will activate the prototype declaration for 64-bit integer functions in the header file `asl.h`, and without the option "`-DASL_LIB_INT64`", those for 32-bit integer functions will be activated.

(3) The name "⟨6 lowercase letters⟩ following ASL_ " is reserved by ASL C interface.

(4) For using 64-bit integer library, you must use "`long`" for integer type declaration. Otherwise, use "`int`" for integer type declaration.

(5) Use the functions 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.

(6) To suppress compiler operation exceptions, ASL C interface functions 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.

(7) 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.

(8) 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.

(9) 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.

(10) 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 function.

(11) 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.

(12) 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

SIMULTANEOUS LINEAR EQUATIONS(DIRECT METHOD)

2.1 INTRODUCTION

This chapter describes functions that solve simultaneous linear equations and obtain the determinant value and inverse matrix of a matrix.
In this library, functions having the following functions are provided individually for each set of matrix characteristics and storage mode.

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

(2) Perform triangular decomposition of coefficient matrix.

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

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

(5) Obtain determinant value and inverse matrix.

You can freely combine the various types of functions (1) through (5) to suit your processing needs. This enables you to perform efficient processing by eliminating unnecessary calculation steps.

In addition, since triangular decomposition of a matrix is performed using the technique most suited to the characteristics of the matrix, the technique used differs for each type of matrix.
In addition, real tridiagonal matrices are classified into two type-real tridiagonal matrix (vector type) and fixed coefficient real tridiagonal matrix (scalar type) according to characteristics of the coefficient matrix. Functions having the following functions are provided for tridiagonal matrices.

(1) Solves simultaneous linear equations (performs reduction operations or Gauss method and solves the equations).

(2) Obtains solutions (only solves the equations after reduction operation).

Users can freely combine the above two functions to suit processing objective. This enables processing to be performed efficiently by eliminating unnecessary computations.

2.1.1 Methods of using functions

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

(1) Simultaneous linear equations

(1) Using $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\} (A, \cdots, \boldsymbol{b}, \cdots);$$

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

(2) Using $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ and $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\} (A, \cdots, \boldsymbol{b}, \cdots);$$

$\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ performs a triangular decomposition of coefficient matrix A, and $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ solves $A\boldsymbol{x} = \boldsymbol{b}$.

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

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\} (A, \cdots, \text{\&cond}, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\} (A, \cdots, \boldsymbol{b}, \cdots);$$

$\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ calculates the condition number and performs a triangular decomposition of coefficient matrix A, and $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ solves $A\boldsymbol{x} = \boldsymbol{b}$.

(2) Determinant and inverse matrix

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmdi}\\ \text{ASL\_rbgmdi}\end{matrix}\right\} (A, \cdots, \text{det}, \cdots);$$

$\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ performs a triangular decomposition of matrix A, and $\left\{\begin{matrix}\text{ASL\_dbgmdi}\\ \text{ASL\_rbgmdi}\end{matrix}\right\}$ obtains the determinant and inverse matrix.

(3) Improving the solution

(1) Using $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$

$A_2 \leftarrow A$

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

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\} (A_2, \cdots, \boldsymbol{b}_2, \cdots);$$

$$\text{ierr} = \begin{Bmatrix} \text{ASL\_dbgmlx} \\ \text{ASL\_rbgmlx} \end{Bmatrix} (A, \cdots, A_2, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_2, \cdots);$$

The function shown above improves the solution obtained by using $\begin{Bmatrix} \text{ASL\_dbgmsl} \\ \text{ASL\_rbgmsl} \end{Bmatrix}$.

(2) Using $\begin{Bmatrix} \text{ASL\_dbgmlu} \\ \text{ASL\_rbgmlu} \end{Bmatrix}$ and $\begin{Bmatrix} \text{ASL\_dbgmls} \\ \text{ASL\_rbgmls} \end{Bmatrix}$

$A_2 \leftarrow A$

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

$$\text{ierr} = \begin{Bmatrix} \text{ASL\_dbgmlu} \\ \text{ASL\_rbgmlu} \end{Bmatrix} (A_2, \cdots);$$

$$\text{ierr} = \begin{Bmatrix} \text{ASL\_dbgmls} \\ \text{ASL\_rbgmls} \end{Bmatrix} (A_2, \cdots, \boldsymbol{b}_2, \cdots);$$

$$\text{ierr} = \begin{Bmatrix} \text{ASL\_dbgmlx} \\ \text{ASL\_rbgmlx} \end{Bmatrix} (A, \cdots, A_2, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_2, \cdots);$$

$\begin{Bmatrix} \text{ASL\_dbgmlu} \\ \text{ASL\_rbgmlu} \end{Bmatrix}$ performs a triangular decomposition of matrix A, $\begin{Bmatrix} \text{ASL\_dbgmls} \\ \text{ASL\_rbgmls} \end{Bmatrix}$ solves $A\boldsymbol{x} = \boldsymbol{b}$, and $\begin{Bmatrix} \text{ASL\_dbgmlx} \\ \text{ASL\_rbgmlx} \end{Bmatrix}$ improves the solution.

2.1.2 Notes

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

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

Example :

To solve the equations:

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

execute either:

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}(A, \cdots, \boldsymbol{b}_1, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}(A, \cdots, \boldsymbol{b}_2, \cdots);$$

or

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}(A, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}(A, \cdots, \boldsymbol{b}_1, \cdots);$$

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}(A, \cdots, \boldsymbol{b}_2, \cdots);$$

$\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ or $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ performs the triangular decomposition of coefficient matrix A, and this result is only referred thereafter without its contents being changed.

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

(4) Although the array type of the input and output data of complex argument type functions is complex type, the array type of the input and output data of all other functions is real type.

(5) Although an iterative method can be used to solve simultaneous linear equations having a sparse matrix as the coefficient matrix, the following points should be carefully considered.

- When solving simultaneous linear equations having a sparse matrix as the coefficient matrix, a solution is obtained by a finite number of operations when using a direct method, regardless of the properties of the coefficient matrix. With an iterative method, however, the solution may quickly converge or no solution may be obtained depending on the properties of the coefficient matrix.
- When the coefficient matrix is positive symmetric or diagonally dominant, a solution generally is obtained faster by using an iterative method function.
- Even if no solution is obtained by using an iterative method, a solution may be obtained by using a direct method.
- When the coefficient matrix is nearly singular, a precise solution may not be obtained regardless of which method is used.
- Two functions are provided for performing triangular decomposition. One obtains the condition number and the other does not. The function that obtains the condition number requires many more calculations just to obtain the condition number.

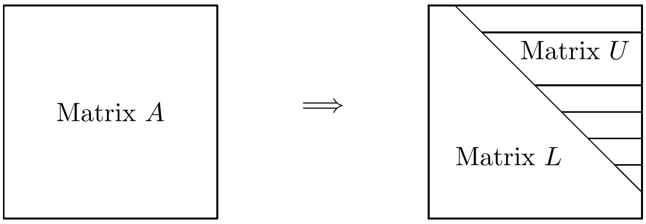
  Therefore, unless you specifically need the condition number, you can save execution time by performing triangular decomposition without obtaining the condition number.

2.1.3 Algorithms Used

2.1.3.1 Crout Method

The Crout method decomposes coefficient matrix A into the product of the lower triangular matrix L and the unit upper triangular matrix U.

$$A = LU$$



Since partial pivoting is performed in this library, this actually becomes $PA = LU$ (where P is the replacement matrix for row exchange).
Assume $A = (a_{ij}), L = (l_{ij})$ and $U = (u_{ij})$ $(i, j = 1, 2, \cdots, N)$. Then, the algorithm is as follows.

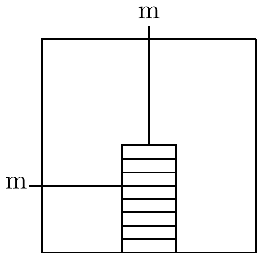
```
    $l_{i1} \leftarrow a_{i1}$ $(i = 1, 2, \cdots, N)$
    $\boxed{\text{Partial pivoting}}$
  $u_{1j} \leftarrow a_{1j}/l_{11}$ $(j = 1, 2, \cdots, N)$
for    $k = 2, 3, \cdots, N$
    $l_{kk} \leftarrow a_{kk} - \sum_{m=1}^{k-1} l_{km}u_{mk}$
    for    $i = k+1, k+2, \cdots, N$
        $l_{ik} \leftarrow a_{ik} - \sum_{m=1}^{k-1} l_{im}u_{mk}$

    $\boxed{\text{Partial pivoting}}$

    for    $j = k+1, k+2, \cdots, N$
        $u_{kj} \leftarrow (a_{kj} - \sum_{m=1}^{k-1} l_{km}u_{mj})/l_{kk}$
```

Partial pivoting is an operation for stable decomposition that exchanges rows so that the pivot is the maximum within the column. The operation at the m-th stage (when $k = m$ in the algorithm shown above) is as follows.

Matrix A during decomposition

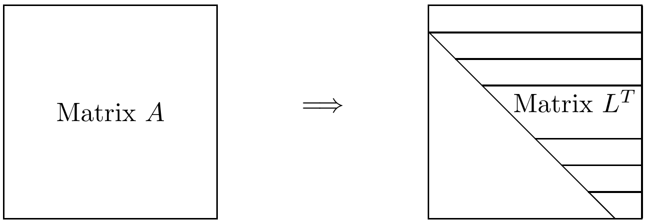


The element having the maximum absolute value within the hatched portion shown in the figure is selected, and the row containing that element is exchanged with the m-th row.

2.1.3.2 Cholesky method

The Cholesky method decomposes coefficient matrix A into the product of the lower triangular matrix L and the upper triangular matrix L^T.

$$A = LL^T$$



Assume $A = (a_{ij}), L = (l_{ij})$ and $L^T = (l'_{ij})$ $(i, j = 1, 2, \cdots, N)$. If the Cholesky method is applied in the column direction to the upper right triangular portion of coefficient matrix A, the algorithm is as follows.

```
for $k = 1, 2, \cdots, N$
    for $i = 1, 2, \cdots, k-1$
        $l'_{ik} \leftarrow (a_{ik} - \sum_{m=1}^{i-1} l'_{mi} l'_{mk}) / l'_{ii}$
    $l'_{kk} \leftarrow \sqrt{a_{kk} - \sum_{m=1}^{k-1} {l'_{mk}}^2}$
```

The calculation efficiency of matrix calculations is increased by generally applying external product calculations rather than inner product calculations and by further employing an unrolling technique to reduce the memory access frequency.

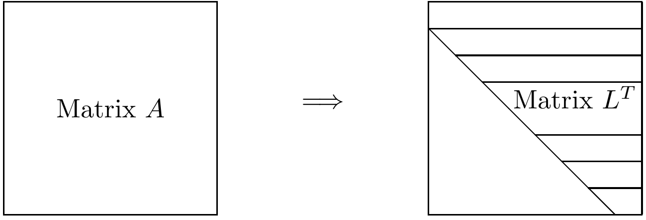
Therefore, the Cholesky method that uses external product calculations is used for simultaneous linear equations having a one-dimensional compressed type coefficient matrix. In addition, the data can be accessed continuously by storing it in row-oriented format.

2.1.3.3 Modified Cholesky method

The modified Cholesky method decomposes coefficient matrix A into the product of the lower triangular matrix L, diagonal matrix D, and upper triangular matrix L^T.

$$A = LDL^T$$

The diagonal matrix D consists of the reciprocals of the diagonal components of the upper triangular matrix L^T.



Assume $A = (a_{ij}), L = (l_{ij}), D = (d_{ij})$ and $L^T = (l'_{ij})$ $(i, j = 1, 2, \cdots, N)$. Then, the algorithm is as follows.

$$l'_{1j} \leftarrow a_{1j} \;\; (j = 1, 2, 3, \cdots, N)$$

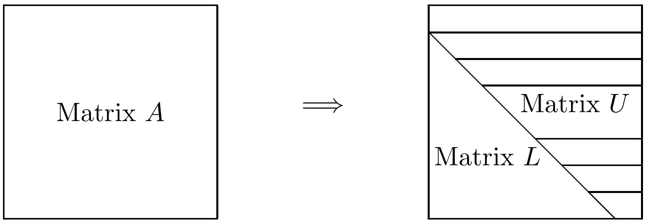
```
for $k = 2, 3, \cdots, N$
    for $i = 1, 2, \cdots, k-1$
        $w_i \leftarrow l'_{ik}/l'_{ii}$
    for $j = k, k+1, \cdots, N$
        $l'_{kj} \leftarrow a_{kj} - \sum_{m=1}^{k-1} w_m l'_{mj}$
```

w indicates a work area, N areas are required.

2.1.3.4 Gauss method

The Gauss method decomposes coefficient matrix A into the product of the unit lower triangular matrix L and the upper triangular matrix U.

$A = LU$

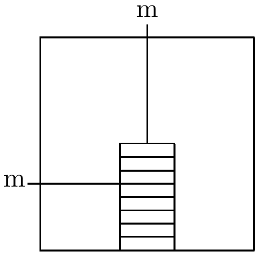


Since partial pivoting is performed in this library, this actually becomes $PA = LU$ (when P is the replacement matrix for row exchange).
Assume $A = (a_{ij}), L = (l_{ij})$ and $U = (u_{ij})$ $(i, j = 1, 2, \cdots, N)$. Then, the algorithm is as follows.

```
for $k = 1, 2, \cdots, N$
    Partial pivoting
    for $i = k+1, k+2, \cdots, N$
        $l_{ik} \leftarrow a_{ik}/u_{kk}$
        for $j = k+1, k+2, \cdots, N$
            $u_{ij} \leftarrow a_{ij} - l_{ik}u_{kj}$
```

Partial pivoting is an operation for stable decomposition that exchanges row so that the pivot is the maximum within the column. The operation at the m-th stage (when $k = m$ in the algorithm shown above) is as follows.

Matrix A during decomposition



The element having the maximum absolute value within the hatched portion shown in the figure is selected, and the m-th through N-th columns of the row containing that element are exchanged with the m-th through N-th columns of the m-th row.

2.1.3.5 Levinson method

When the Toeplitz matrix R is represented by:

$$R = \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\ r_1 & r_0 & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_{-1} \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

the following simultaneous linear equations:

$$\sum_{j=1}^{n} r_{i-j} x_j = b_i \quad (i = 1, \cdots, n)$$

having the Toeplitz matrix as coefficient matrix can be solved as described below by considering the solutions $x_j^{(m)}$ $(j = 1, \cdots, m;\ m = 1, 2, \cdots, n)$ of the following kind of n simultaneous linear equations:

$$\sum_{j=1}^{m} r_{i-j} x_j^{(m)} = b_i \quad (i = 1, \cdots, m;\ m = 1, 2, \cdots, n)$$

(1) Initial solution $(m = 1)$

$$x_1^{(1)} = \frac{b_1}{r_0}$$

$$g_1^{(1)} = \frac{r_{-1}}{r_0}$$

$$h_1^{(1)} = \frac{r_1}{r_0}$$

(2) For $m = 2, 3, \cdots, n$, perform the following sequential iterative calculations.

$$x^{(nu)} = \sum_{j=1}^{m-1} r_{m-j} x_j - b_m$$

$$x^{(de)} = \sum_{j=1}^{m-1} r_{m-j} g_{m-j}^{(m-1)} - r_0$$

$$x_m^{(m)} = \frac{x^{(nu)}}{x^{(de)}}$$

$$x_j^{(m)} = x_j^{(m-1)} - x_m^{(m)} g_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

$$g^{(nu)} = \sum_{j=1}^{m-1} r_{j-m} g_j^{(m-1)} - r_{-m}$$

$$g^{(de)} = \sum_{j=1}^{m-1} r_{j-m} h_{m-j}^{(m-1)} - r_0$$

$$h^{(nu)} = \sum_{j=1}^{m-1} r_{m-j} h_j^{(m-1)} - r_m$$

$$g_m^{(m)} = \frac{g^{(nu)}}{g^{(de)}}$$

$$h_m^{(m)} = \frac{h^{(nu)}}{x^{(de)}}$$

$$g_j^{(m)} = g_j^{(m-1)} - g_m^{(m)} h_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

$$h_j^{(m)} = h_j^{(m-1)} - h_m^{(m)} g_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

The solutions are obtained by letting $x_j = x_j^{(n)}$. Since r_i and r_{-i} are related as follows for a symmetric Toeplitz matrix:

$$r_i = r_{-i} \quad (i = 1, 2, \cdots, n)$$

the following relationship holds:

$$g_j^{(m)} = h_j^{(m)} \quad (j = 1, 2, \cdots, m;\ m = 1, 2, \cdots, n)$$

and the calculations can proceed more efficiently than for the general case. Since this method makes practical use of the properties of the matrix, it is superior to the general Gaussian elimination method in terms of memory usage and calculation efficiency. However, the solution may not be able to be obtained theoretically even if the matrix is regular. For example, a solution clearly cannot be obtained by this method when $r_0 = 0$.

2.1.3.6 Vandermonde matrix

The Vandermonde matrix V of order n consisting of n different elements v_k $(k = 1, 2, \cdots, n)$ is represented as follows.

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & \cdots & v_1^{n-2} & v_1^{n-1} \\ 1 & v_2 & v_2^2 & \cdots & v_2^{n-2} & v_2^{n-1} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 1 & v_{n-1} & v_{n-1}^2 & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\ 1 & v_n & v_n^2 & \cdots & v_n^{n-2} & v_n^{n-1} \end{bmatrix}$$

Let's solve the simultaneous linear equations $V\boldsymbol{x} = \boldsymbol{b}$ having the Vandermonde matrix V as coefficient matrix, which are represented as follows.

$$\sum_{j=1}^{n} v_i^{j-1} x_j = b_i \quad (i = 1, \cdots, n)$$

If the polynomial $P_i^{(n)}(x)$ of degree $n-1$ is defined as follows:

$$P_i^{(n)}(x) = \prod_{\substack{k=1 \\ (k \neq i)}}^{n} \frac{x - v_k}{v_i - v_k} = \sum_{j=1}^{n} u_{i,j} x^{j-1}$$

the relationship $P_i^{(n)}(v_k) = \delta_{ik}$ (where δ_{ik} is the Kronecker delta) holds. Therefore, if the matrix consisting of the coefficients of the x^{j-1} terms of this polynomial is represented by $U = \{u_{i,j}\}$, the relationship $UV^T = E$ (where E is the unit matrix), that is, $V^{-1} = U^T$ holds. Consequently, the solution $\boldsymbol{x}$ of the simultaneous linear equations $V\boldsymbol{x} = \boldsymbol{b}$ is obtained by calculating:

$$\boldsymbol{x} = U^T\boldsymbol{b}$$

Now, to calculate the various coefficients of U, consider the master polynomial $P^{(n)}(x)$ defined by the following equation.

$$P^{(n)}(x) = \prod_{k=1}^{n}(x - v_k)$$

Let the coefficient of the x^{j-1} term of the master polynomial $P^{(n)}(x)$ be $w_{n-j+1}^{(n)}$, and the master polynomial can be represented as follows.

$$P^{(n)}(x) = x^n + w_1^{(n)}x^{n-1} + \cdots + w_{n-1}^{(n)}x + w_n^{(n)}$$

From the relationship $P^{(i)}(x) = (x - v_i)P^{(i-1)}(x)$ $(i = 2, 3, \cdots, n)$, the following relationships are obtained by comparing the coefficients for x^{j-1}:

$$w_1^{(i)} = w_1^{(i-1)} - v_i \quad (i = 2, \cdots, n)$$

$$w_j^{(i)} = w_j^{(i-1)} - v_i w_{j-1}^{(i-1)} \quad (j = i, i-1, \cdots, 2;\ i = 2, 3, \cdots, n)$$

where, the following equations hold.

$$w_1^{(1)} = -v_1$$

$$w_j^{(j-1)} = 0 \quad (j = 2, 3, \cdots, n)$$

The various coefficients of the master polynomial can be calculated from the above. On the other hand, the following relationship holds:

$$\frac{dP^{(n)}(x)}{dx}|_{x=v_i} = \prod_{\substack{k=1 \\ (k\neq i)}}^{n}(v_i - v_k)$$

and this value can be calculated from the following:

$$\frac{dP^{(n)}(x)}{dx}|_{x=v_i} = n{v_i}^{n-1} + (n-1)w_1^{(n)}{v_i}^{n-2} + \cdots + w_{n-1}^{(n)}$$

Also, since:

$$P_i^{(n)}(x) = \frac{P^{(n)}(x)}{(x - v_i)\frac{dP^{(n)}(x)}{dx}|_{x=v_i}}$$

the coefficients $u_{i,j}$ of the x^{j-1} terms of this polynomial can be obtained by using synthetic division to calculate the coefficients of the x^{j-1} terms of $\dfrac{P^{(n)}(x)}{(x - v_i)}$. The simultaneous linear equations having the Vandermonde matrix as the coefficient matrix essentially are ill-conditioned, and it is difficult to obtain a solution with good precision except when n is extremely small.

2.1.3.7 Cyclic Reduction Method

(1) Cyclic reduction method

The cyclic reduction method is used to solve the simultaneous linear equations:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.1}$$

having the real tridiagonal matrix A as the coefficient matrix.
If we assume that A, $\boldsymbol{x}$, and $\boldsymbol{b}$ are as follows:

$$A = \begin{bmatrix} d_1 & u_1 & & & 0 \\ \ell_2 & d_2 & u_2 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & u_{n-1} \\ & 0 & & \ell_n & d_n \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

then:

$$\ell_i x_{i-1} + d_i x_i + u_i x_{i+1} = b_i \tag{2.2}$$

This algorithm repeatedly performs a reduction operation $\lfloor \log_2(n) \rfloor$ times. The reduction operation creates a set of simultaneous linear equations having a coefficient matrix with one-half the order of the coefficient matrix before the reduction operation. Ultimately, a single linear equation is created from which a single solution is obtained.

$$\begin{aligned} \boldsymbol{dx} &= \boldsymbol{b} \\ \boldsymbol{x} &= \boldsymbol{b}/\boldsymbol{d} \end{aligned} \tag{2.3}$$

All of the solutions then are obtained by repeatedly performing back substitution based on this solution.
In this section, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

The reduction operation and back substitution of the cyclic reduction method are described below.

(a) Reduction operation

First, let's assume $n = 2^m - 1$.

We will eliminate x_{i-1} and x_{i+1} from three rows of (2.1) consisting of an even numbered row and the rows before and after it. That is, we will obtain the following equation:

$$\ell'_i x_{i-2} + d'_i x_i + u'_i x_{i+2} = b'_i \tag{2.4}$$

$$\begin{cases} \ell'_i &= d_{i+1}\ell_{i-1}\ell_i \\ u'_i &= d_{i-1}u_i u_{i+1} \\ d'_i &= \ell_i d_{i+1} u_{i-1} + \ell_{i+1} d_{i-1} u_i - d_{i-1} d_i d_{i+1} \\ b'_i &= \ell_i d_{i+1} b_{i-1} + d_{i-1} u_i b_{i+1} - d_{i-1} d_{i+1} b_i \end{cases}$$

from the three rows:

$$\begin{cases} \ell_{i-1}x_{i-2} + d_{i-1}x_{i-1} + u_{i-1}x_i & = b_{i-1} \\ \qquad\qquad \ell_i x_{i-1} + d_i x_i + u_i x_{i+1} & = b_i \\ \qquad\qquad\qquad\qquad \ell_{i+1}x_i + d_{i+1}x_{i+1} + u_{i+1}x_{i+2} & = b_{i+1} \end{cases}$$

where, i is an even number.

By applying (2.4) to all even numbered rows contained in (2.1) ($x_0 = x_{n+1} = 0$), we obtain a set of simultaneous linear equations having a real tridiagonal coefficient matrix of order $\lfloor n/2 \rfloor$ as the coefficient matrix.

Next, let's consider $n = 2^m$. Although we could apply (2.4) to all even numbered rows when $n = 2^m - 1$, we cannot apply (2.4) to row $n-1$ and row n when $n = 2^m$ since row $n-1$ is an odd numbered row. Therefore, we will apply the following equation:

$$\ell'_n x_{n-2} + d'_n x_n = b'_n \tag{2.5}$$

$$\begin{cases} \ell'_n &= \ell_{n-1}\ell_n \\ d'_n &= \ell_n u_{n-1} - d_n d_{n-1} \\ b'_n &= \ell_n b_{n-1} - d_{n-1} d_n \end{cases}$$

which was obtained by eliminating x_{n-1} from the two rows:

$$\begin{cases} \ell_{n-1}x_{n-2} & + & d_{n-1}x_{n-1} & + & u_{n-1}x_n & = b_{n-1} \\ & & \ell_n x_{n-1} & + & d_n x_n & = b_n \end{cases}$$

Consequently, we can reduce the set of simultaneous linear equations to a set having a real tridiagonal matrix of order $\lfloor n/2 \rfloor$ as the coefficient matrix regardless of the value of n.

(b) Back substitution

We can obtain the other solutions based on the solution (2.3), which we obtained by using the reduction method. To obtain these additional solutions, we substitute previously obtained solutions back into the various sets of simultaneous linear equations produced by the reduction method, proceeding in the reverse order as when applying the reduction method.

If the solution has been obtained for an even numbered row, the solution for an odd numbered row is obtained by using the following equation:

$$x_{i-1} = (b_{i-1} - \ell_{i-1}x_{i-2} - u_{i-1}x_i)/d_{i-1}, \qquad i = 2, 4, 6, \cdots, n+1$$

(2) Increasing the speed of the cyclic reduction method

Since the cyclic reduction method is not a successive elimination method such as the Gauss method, the calculations are independent of one another. Although this essentially allows vectorization to be performed, the following kind of vectorization also is carried out.

(a) Increasing the speed of the fixed coefficient type cyclic reduction method

If the fixed coefficient type cyclic reduction method, a modified version of the cyclic reduction method, is used, the processing speed can be increased for the coefficient matrix that appears when discretizing the Dirichlet or Neumann boundary value problem. The fixed coefficient type cyclic reduction method is described below.

First, consider the following coefficient matrices:

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & s & d \end{bmatrix} \quad , \quad d \neq 0, \quad s \neq 0 \tag{2.6}$$

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & 2\cdot s & d \end{bmatrix} \quad , \quad d \neq 0, \quad s \neq 0 \tag{2.7}$$

If we compare (2.6) with the matrix obtained by normalizing the last row of (2.7) by 2, we see that only the last rows of these two matrices differ, and all other rows are identical. Therefore, we can replace (2.6) and (2.7) by the following matrix (2.8).

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & d & s \\ 0 & & & s & e \end{bmatrix} \quad , \quad d \neq 0, \quad s \neq 0, \quad e \neq 0 \tag{2.8}$$

Now, let's first assume $n = 2^m - 1$.

We will eliminate x_{i-1} and x_{i+1} from three rows of (2.7) consisting of an even numbered row and the rows before and after it. That is, we will obtain the following equation:

$$s'x_{i-2} + d'x_i + s'x_{i+2} = b'_i \tag{2.9}$$

$$\begin{cases} s' & = s^2 \\ d' & = 2 \cdot s^2 - d^2 \\ b' & = s(b_{i-1} + b_{i+1}) - db_i \end{cases}$$

from the three rows:

$$\begin{cases} sx_{i-2} + dx_{i-1} + sx_i & = b_{i-1} \\ \quad sx_{i-1} + dx_i + sx_{i+1} & = b_i \\ \quad\quad sx_i + dx_{i+1} + sx_{i+2} & = b_{i+1} \end{cases}$$

where, i is an even number.

By applying (2.9) to all even numbered rows contained in (2.8) ($x_0 = x_{n+1} = 0$), we obtain a set of simultaneous linear equations having a real tridiagonal coefficient matrix of order $\lfloor n/2 \rfloor$ as the coefficient matrix. However, for row $n-1$, we have:

$$s'x_{n-3} + e'x_{n-1} = b'_{n-1} \tag{2.10}$$

$$\begin{cases} s' & = e \cdot s^2 \\ e' & = e \cdot s^2 - e \cdot d^2 + d \cdot s^2 \\ b'_{n-1} & = e \cdot s \cdot b_{n-2} - e \cdot d \cdot b_{n-1} + d \cdot s \cdot b_n \end{cases}$$

Next, let's consider $n = 2^m$. Since row $n-1$ is an odd numbered row when $n = 2^m$, we will apply the following equation:

$$s'x_{n-2} + e'x_n = b_{n-1} \tag{2.11}$$

$$\begin{cases} s' & = s^2 \\ d' & = s^2 - d \cdot e \\ b'_{n-1} & = s \cdot b_{n-1} - d \cdot b_n \end{cases}$$

which was obtained by eliminating x_{n-1} from the two rows:

$$\begin{cases} sx_{n-2} + dx_{n-1} + sx_n & = b_{n-1} \\ \quad sx_{n-1} + ex_n & = b_n \end{cases}$$

Consequently, we can reduce the set of simultaneous linear equations to a set having a real tridiagonal matrix of order $\lfloor n/2 \rfloor$ as the coefficient matrix regardless of the value of n. These operations are repeatedly performed $\lfloor \log_2(n) \rfloor$ times until, ultimately, a single linear equation is created from which a single solution is obtained.

$$\begin{aligned} \boldsymbol{dx} &= \boldsymbol{b} \\ \boldsymbol{x} &= \boldsymbol{b}/\boldsymbol{d} \end{aligned}$$

All of the solutions then are obtained by repeatedly performing back substitution based on this solution. If the solutions for even numbered rows have been obtained, then the solutions for odd numbered rows are obtained from the following equation:

$$x_{i-1} = (b_{i-1} - s \cdot x_{i-2} - s \cdot x_i)/d, \qquad i = 2, 4, 6, \cdots, n+1$$

Next, consider the following coefficient matrices:

$$\begin{bmatrix} d & 2 \cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & s & d \end{bmatrix}, \quad d \neq 0, \quad s \neq 0 \tag{2.12}$$

$$\begin{bmatrix} d & 2 \cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & 2 \cdot s & d \end{bmatrix}, \quad d \neq 0, \quad s \neq 0 \tag{2.13}$$

If we compare (2.12) with the matrix obtained by normalizing the last row of (2.13) by 2, we see that only the last rows of these two matrices differ, and all other rows are identical. Therefore, we can replace (2.12) and (2.13) by the following matrix (2.14).

$$\begin{bmatrix} d & 2 \cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & d & s \\ 0 & & & s & e \end{bmatrix}, \quad d \neq 0, \quad s \neq 0, \quad e \neq 0 \tag{2.14}$$

This time, let's consider the operations based on odd numbered rows instead of even numbered rows. First, we will eliminate x_2 from the first and second rows. That is, we will obtain the following equation:

$$(d^2 - 2 \cdot s^2)x_1 - 2 \cdot s^2 x_3 = db_1 - 2sb_2 \tag{2.15}$$

from the two rows:

$$\begin{cases} dx_1 + 2 \cdot sx_2 = b_1 \\ sx_1 + dx_2 + sx_3 = b_2 \end{cases}$$

Next, we will eliminate x_{2i} and x_{2i+2} from the three rows of (2.14) consisting of row $2i$, row $2i+1$, and row $2i+2$. That is, we will obtain the following equation:

$$s' x_{2i-1} + d' x_{2i+1} + s' x_{2i+3} = b'_{2i+1} \tag{2.16}$$

$$\begin{cases} s' &= s^2 \\ d' &= 2 \cdot s^2 - d^2 \\ b'_{2i+1} &= s \cdot b_{2i} - d \cdot b_{2i+1} + s \cdot b_{2i+2} \end{cases}$$

from the three rows:

$$\left\{\begin{array}{lllllllll} sx_{2i-1} & + & dx_{2i} & + & sx_{2i+1} & & & & = b_{2i} \\ & & sx_{2i} & + & dx_{2i+1} & + & sx_{2i+2} & & = b_{2i+1} \\ & & & & sd_{2i+1} & + & sx_{2i+2} & + \; sx_{2i+3} & = b_{2i+2} \end{array}\right.$$

This is performed for each of the values $i = 1,\ 2,\ 3,\ \cdots,\ m$, where m is the maximum value of i that satisfies the relationship $2i + 1 \leq n - 2$.

Finally, for $n = 2^m$, we obtain (2.17) by eliminating x_{n-2} and x_n from the three rows consisting of row $n-2$, $n-1$, and row n. Also, for $n = 2^m - 1$, since row $n-1$ is an even numbered row, we obtain (2.18) by eliminating x_{n-1} from the two rows consisting of row $n-1$ and row n.

That is, for $n = 2^m$, we obtain the following equation:

$$s'x_{n-3} + e'x_{n-1} = b'_{n-1} \tag{2.17}$$

$$\begin{cases} s' & = e \cdot s^2 \\ e' & = e \cdot s^2 - e \cdot d^2 + d \cdot s^2 \\ b'_{n-1} & = s \cdot e \cdot b_{n-2} - d \cdot e \cdot b_{n-1} + d \cdot s \cdot b_n \end{cases}$$

from the three rows:

$$\left\{\begin{array}{lllllll} sx_{n-3} & + & dx_{n-2} & + & sx_{n-1} & & = b_{n-2} \\ & & sx_{n-2} & + & dx_{n-1} & + \; sx_n & = b_{n-1} \\ & & & & sx_{n-1} & + \; ex_n & = b_n \end{array}\right.$$

and for $n = 2^m - 1$, we obtain the following equation:

$$s'x_{n-2} + e'x_n = b'_n \tag{2.18}$$

$$\begin{cases} s' & = & s^2 \\ e' & = & s^2 - e \cdot d \\ b'_n & = & s \cdot b_{n-1} - d \cdot b_n \end{cases}$$

from the two rows:

$$\left\{\begin{array}{lllll} sx_{n-2} & + & dx_{n-1} & + \; sx_n & = b_{n-1} \\ & & sx_{n-1} & + \; ex_n & = b_n \end{array}\right.$$

Consequently, we can reduce the set of simultaneous linear equations to a set having a real tridiagonal matrix of order $\lfloor (n-1)/2 \rfloor + 1$ as the coefficient matrix regardless of the value of n. These operations are repeatedly performed $\lfloor \log_2(n-1) \rfloor$ times until, ultimately, a set of equations having the following coefficient matrix is obtained:

$$\begin{bmatrix} d^{(\mathrm{m})} & 2 \\ 1 & e^{(\mathrm{m})} \end{bmatrix}, \quad m = \lfloor (n-1)/2 \rfloor + 1$$

All of the solutions then are obtained by repeatedly performing back substitution based on this solution.

(b) Reduction operation truncation

As the reduction operation is repeated, the magnitude of the diagonal elements may be increased based on a certain assumption (sufficient but not necessary condition) and the ratio of the diagonal element and subdiagonal element may become larger than 1/ep (ep: Units for determining error) at an intermediate stage of the reduction operation.

Consider the following as one such assumption:

$$\mid l_i^{(k)} \mid, \mid u_i^{(k)} \mid < \mid d_i^{(k)}/2 \mid, \quad 1 \leq i \leq n \tag{2.19}$$

Here, $l_i^{(k)}$, $d_i^{(k)}$ and $u_i^{(k)}$ are the lower subdiagonal element, the diagonal element and the upper subdiagonal element, respectively, in the i-th row of the coefficient matrix after the k-th reduction operation. If this assumption holds, and the coefficient matrix is normalized to:

$$(\cdots, l_i{}^{(k)}/d_i{}^{(k)}, 1, u_i{}^{(k)}/d_i{}^{(k)}, \cdots) \tag{2.20}$$

then the subdiagonal elements may become as small as ep, and the constant vector $\boldsymbol{b}^{(k)}$(k: Reduction frequency) will converge to several solutions before the reduction operation is completed.

Therefore, if the reduction frequency when convergence occurs is known before performing the reduction operation, the reduction operation need not be performed all the way to completion. If the reduction operation is halted before completion and the calculations switch to back substitution, efficiency can be increased because the calculation time will be reduced. This is called truncation of the cycling reduction operation.

To obtain the value of the reduction frequency up to truncation, we will check the lower limit for convergence when (2.20) is satisfied.

First, let's obtain $\boldsymbol{e} = \max_i(l_i{}^{(k)}/d_i{}^{(k)}, u_i{}^{(k)}/d_i{}^{(k)})$ and consider the matrix $(\cdots, \boldsymbol{e}, 1, \boldsymbol{e}, \cdots)$ obtained by replacing all $l_i(k)$ and $u_i(k)$ of (2.20) by $\boldsymbol{e}$. If also would be sufficient to consider a coefficient matrix such as $(\cdots, 1, \boldsymbol{d}, 1, \cdots)$. To determine the convergence rate, we define:

$$\boldsymbol{\varepsilon}^{(k)} = \mid \boldsymbol{d}^{(k)} \mid -2 > 0$$

where, $\boldsymbol{d}^{(k)}$ is the diagonal element computed during the k-th iteration. Let's try to measure whether $\mid \boldsymbol{d}^{(k)} \mid$ increases towards 1/ep as a function of k. If we take the absolute value of:

$$\boldsymbol{d}^{(k+1)} = 2 - \left[\boldsymbol{d}^{(k)}\right]^2$$

then from (2.9) we get:

$$\mid \boldsymbol{d}^{(k+1)} \mid = \mid 2 - \left[2 + \boldsymbol{\varepsilon}^{(k)}\right]^2 \mid \geq 2 + 4\boldsymbol{\varepsilon}^{(k)} + \left[\boldsymbol{\varepsilon}^{(k)}\right]^2$$

and it follows that:

$$\boldsymbol{\varepsilon}^{(k+1)} > 4\boldsymbol{\varepsilon}^{(k)} + \left[\boldsymbol{\varepsilon}^{(k)}\right]^2 \tag{2.21}$$

From (2.21), we have:

- If $\boldsymbol{\varepsilon}^{(k)} < 1$, then $\boldsymbol{\varepsilon}^{(k+1)} > 4\boldsymbol{\varepsilon}^{(k)}$
  and the rate of increase is said to be at least of first order speed.
- If $\boldsymbol{\varepsilon}^{(k)} > 1$, then $\boldsymbol{\varepsilon}^{(k+1)} > \left[\boldsymbol{\varepsilon}^{(k)}\right]^2$
  and the rate of increase is said to be at least of second order speed.

Consequently, the minimum integer k for which the following relationship holds for the value of $\boldsymbol{\varepsilon}^{(k)}$ obtained from (2.21):

$$\boldsymbol{\varepsilon}^{(k)} \geq 1/\mathrm{ep}$$

is assumed to be the reduction frequency up to truncation. Moreover, truncation will not occur if $k \geq \lfloor \log_2(n) \rfloor$.

(3) **Supplementary item**

- Affect on calculation time

  For simultaneous linear equations having a real tridiagonal coefficient matrix that does not satisfy condition (2.19) (that is, the magnitude of the diagonal elements is not strong), the calculation process must determine whether or not the coefficient matrix is singular. Therefore, more calculation time is required than for a coefficient matrix for which the magnitude of the diagonal elements is strong.

2.1.3.8 Calculating the inverse matrix

Triangular decomposition is used to calculate the inverse matrix of matrix A.
If A is decomposed into $A = LU$, then L^{-1} or U^{-1} is obtained as the first step by the sweeping out method. Next, that result is transformed as the second step to calculate $A^{-1} = U^{-1}L^{-1}$.
For example, since L^T can be obtained by the Cholesky method as $L^{-1}A = L^T$, A^{-1} is obtained by multiplying the transformation matrix for triangular decomposition L^{-1} by $(L^T)^{-1}$ from the right side.
Whether L^{-1} or U^{-1} is calculated as the first step differs according to the triangular decomposition method.

2.1.3.9 Calculating the determinant

The determinant is obtained as follows.
If A has been decomposed into $A = LU$, then

$$det(A) = det(L)det(U) = \prod_{i=1}^{n} l_{ii} \prod_{i=1}^{n} u_{ii}$$

where, $L = (l_{ij})$ and $U = (u_{ij})$.

2.1.3.10 Improving the solution

Consider improving the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$. Let $\boldsymbol{x}^{(1)}$ be the initially obtained solution, and assume that $A\boldsymbol{x}^{(1)} \neq \boldsymbol{b}$ due to computational error. The following algorithm is used to improve $\boldsymbol{x}^{(1)}$.

(1) $\boldsymbol{r}^{(k)} = \boldsymbol{b} - A\boldsymbol{x}^{(k)}$

(2) $A\boldsymbol{y}^{(k)} = \boldsymbol{r}^{(k)}$

(3) $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)}$ $(k = 1, 2, \cdots)$

This iterative procedure generates a rounding error in (2). Therefore, the formula in (2) actually becomes:

$$(A + E)\boldsymbol{y}^{(k)} = \boldsymbol{r}^{(k)}$$

Using this equation together with (1) and (3) yields:

$$\begin{array}{rcl} \boldsymbol{x}^{(k+1)} - \boldsymbol{x} & = & [\ I - (A+E)^{-1}A\ ]^k(\boldsymbol{x}^{(1)} - \boldsymbol{x}) \\ \boldsymbol{r}^{(k+1)} & = & [\ I - A(A+E)^{-1}\ ]\boldsymbol{r}^{(k)} \end{array}$$

Therefore, if $\|E\|\|A^{-1}\| < \dfrac{1}{2}$, then

$$\begin{array}{l} \boldsymbol{x}^{(k+1)} \to \boldsymbol{x} \\ \boldsymbol{r}^{(k+1)} \to 0 \end{array} \qquad (k \to \infty)$$

Moreover, if

$$\frac{\|\boldsymbol{y}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k+1)}\|_\infty} > \frac{1}{2}\frac{\|\boldsymbol{y}^{(k-1)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$$

the solution does not converge.
(See reference bibliography (6).)

2.1.3.11 Precise estimate of the approximate solution

For the approximate solution $\boldsymbol{x}^{(k)}$,

$$\boldsymbol{y}^{(k)} = (A+E)^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{(k)}) = (I + A^{-1}E)^{-1}(\boldsymbol{x} - \boldsymbol{x}^{(k)})$$

The relative error of the solution $\dfrac{\|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$ can be replaced by $\dfrac{\|\boldsymbol{y}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$ if the solution converges sufficiently and matrix A is well conditioned.

2.1.3.12 Condition Number

(1) Condition numbers and their use The condition number $\kappa(A)$ of matrix A is a numeric value that indicates the degree of influence the error included in coefficient matrix A or constant vector $\boldsymbol{b}$ exerts on the solution when solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$. The condition number is given by the following formula:

$$\kappa(A) = \|A\|\|A^{-1}\|$$

If error E is contained in coefficient matrix A, the relative error between the derived solution $\boldsymbol{y}$ and real solution $\boldsymbol{x}$ is in the range:

$$\frac{\|\boldsymbol{y} - \boldsymbol{x}\|}{\|\boldsymbol{y}\|} \le \kappa(A)\varepsilon$$

where:

$$\varepsilon = \frac{\|E\|}{\|A\|}.$$

If error $\boldsymbol{e}$ is contained in constant vector $\boldsymbol{b}$, the relative error is in the range:

$$\frac{\|\boldsymbol{y} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \kappa(A)\varepsilon$$

where:

$$\varepsilon = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{b}\|}.$$

Therefore, if the condition number is on the order of 10^{α}, the precision of the derived solution may be approximately α digits less than the precision of the original data.
This library obtain the reciprocal of the condition number and store it in the variable cond. Note that even if solution is obtained for simultaneous linear equations having a coefficient matrix for which the cond value is extremely small, the precision will be extremely poor. In particular, if the following decision formula holds, the matrix is computationally singular, and the solution is unreliable.
(Singular matrix decision formula):

$$1.0 + \text{cond} = 1.0$$

(2) Calculating the condition number Although the condition number

$$\kappa(A) = \|A\|\|A^{-1}\|$$

this library approximate $\|A^{-1}\|$ without obtaining A^{-1} and then multiply that value by $\|A\|$.
Let $A = U\Sigma V^T$ be a singular decomposition of A where

$$\begin{aligned} U, V \quad &: \quad \text{Orthogonal matrices} \\ \Sigma \quad &= \begin{bmatrix} \sigma_1 & & & & 0 \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ 0 & & & & \sigma_n \end{bmatrix} \\ & \sigma_i : \text{Singular value} \\ & \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \end{aligned}$$

Consider the system of equations $A\boldsymbol{x} = \boldsymbol{y}$. If $\boldsymbol{y}$ is represented as:

$$\boldsymbol{y} = \|\boldsymbol{y}\| \sum_{i=1}^{n} \alpha_i u_i \quad (\sum_i {\alpha_i}^2 = 1)$$

where u_i (a column vector of U) is a basis, then, the following relationship holds:

$$\|A^{-1}\| \geq \frac{\|\boldsymbol{x}\|}{\|\boldsymbol{y}\|} = \left[\sum_{i=1}^{n} (\frac{\alpha_i}{\sigma_i})^2 \right]^{\frac{1}{2}}$$

As long as α_n is not particularly small, the size of the right side is on the order of ${\sigma_n}^{-1}(= \|A^{-1}\|)$ for any type of vector $\boldsymbol{y}$.
This library select $\boldsymbol{y}$ so that approximate solutions get successively better.
The inequality shown above holds when $\boldsymbol{y} = \boldsymbol{u}_n(\alpha_n = 1, \alpha_i = 0; i = 1, 2, \cdots, n-1)$. Therefore, $\boldsymbol{y}$ should be determined so that it has $\boldsymbol{u}_n$ as its principle elements. Actually, for:

$$\boldsymbol{z} = \begin{pmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{pmatrix}$$

$\boldsymbol{y}$ should be obtained in $A^T\boldsymbol{y} = \boldsymbol{z}$ by determining the sign of each element of $\boldsymbol{z}$ so that $\dfrac{\|\boldsymbol{y}\|}{\|\boldsymbol{z}\|}$ is maximized.
Using this $\boldsymbol{y}$ to solve $A\boldsymbol{x} = \boldsymbol{y}$, $\dfrac{\|\boldsymbol{x}\|}{\|\boldsymbol{y}\|}$ is the approximate value of $\|A^{-1}\|$.
The actual procedure for obtaining the condition number is as follows.

(a) Obtain $\|A\|$.

(b) Perform a triangular decomposition of A into $A = LU$.

(c) Obtain $\boldsymbol{w}$ by determining $\boldsymbol{z}$ in $U^T\boldsymbol{w} = \boldsymbol{z}$ so that $\dfrac{\|\boldsymbol{w}\|}{\|\boldsymbol{z}\|}$ is maximized.

(d) Obtain $\boldsymbol{y}$ by solving $L^T\boldsymbol{y} = \boldsymbol{w}$.

(e) Obtain $\boldsymbol{x}$ by solving $LU\boldsymbol{x} = \boldsymbol{y}$.

(f) Obtain $\dfrac{\|\boldsymbol{y}\|}{\|\boldsymbol{x}\|\|A\|}$ (reciprocal of the condition number) and store this value in the argument cond.

(See reference bibliography (3).)

2.1.4 Reference Bibliography

(1) Wilkinson, J. H. and Reinsch, C. , “Handbook for Automatic Computation, vol II, Linear Algebra”, Springer–Verlag, (1971).

(2) Dahlquist, G. and Björck, Å., “Numerical Methods”, translated by Anderson, N. Prentice–Hall, Inc., (1974).

(3) Cline, A. K. , Moler, C. B. , Stewart, G. W. and Wilkinson, J. H. , “An estimate for the condition number of a matrix”, SIAM Numerical Analysis Vol. 16, pp. 368–375, (1979).

(4) Dongarra, J. J., Moler, C. B., Bunch, J. R. and Stewart, G. W., “LINPACK Users’ Guide”, SIAM, (1979).

(5) Forsythe, G. E. and Moler, C. B. , “Computer Solution of Linear Algebraic Systems”, Prentice-Hall, Inc., (1967).

(6) Wilkinson, J. H. , “Rounding Errors in Algebraic Processes, Notes on Applied Science No. 32”, Prentice-Hall, Inc., (1963).

(7) Robert, Y. and Sguazzero, P. “The LU decomposition algorithm and its efficient FORTRAN implementation on IBM3090 Vector Multiprocessor”, IBM Tech. Rep. , ICE-0006, (1987).

(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 Computers”

2.2 REAL MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

2.2.1 ASL_dbgmsm, ASL_rbgmsm

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

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_dbgmsm (ab, lna, n, m, ipvt);

Single precision:

ierr = ASL_rbgmsm (ab, lna, n, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real   Z:Double precision complex   I: { int as for 32bit Integer
R:Single precision real   C:Single precision complex   long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ab	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	See Contents	Input	Matrix (real matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **Size**: $(\mathrm{lna} \times (\mathrm{n} + \mathrm{m}))$
				Output	Matrix (real matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array ab
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	Number of right-hand side vectors, m
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (a))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(5) **Error indicator (Return Value)**

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

(6) **Notes**

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

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

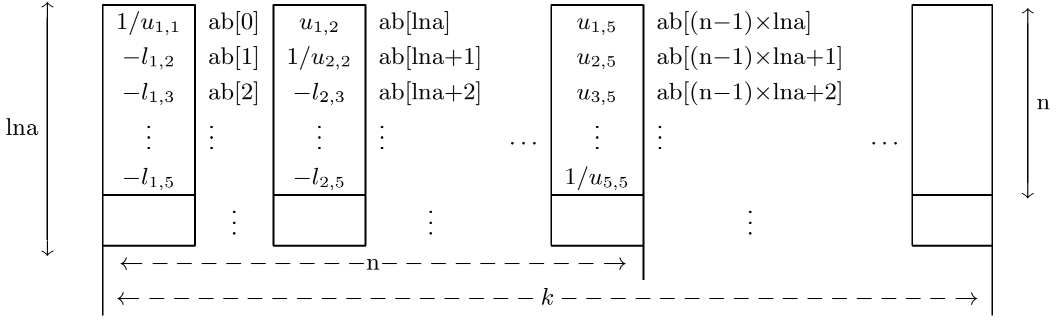
Figure 2−1 Storage Status of Matrices L and U

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

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

$$\Downarrow$$

Storage status of array ab[lna×k]



Remarks

a. lna $\geq$ n and n+m $\leq$ k must be hold.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Array ab in which coefficient matrix A and constant vectors $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are stored, lna=11, n=4 and m=2.

(c) Main program

```
/*      C interface example for ASL_dbgmsm */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ab;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbgmsm.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbgmsm ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    printf( "\t n = %6d m = %6d\n", n, m );

    ab = ( double * )malloc((size_t)( sizeof(double) * (lna*(lna+lma)) ));
    if( ab == NULL )
    {
        printf( "no enough memory for array ab\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ab[i+lna*j] );
            printf( "%8.3g  ", ab[i+lna*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            fscanf( fp, "%lf", &ab[i+lna*(n+j)] );
            printf( "%8.3g  ", ab[i+lna*(n+j)] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_dbgmsm(ab, lna, n, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "%8.3g  ", ab[i+lna*(n+j)] );
        }
        printf( "\n" );
    }

    free( ab );
    free( ipvt );

    return 0;
}
```

(d) Output results

```
*** ASL_dbgmsm ***

 ** Input **

 n =      4 m =      2

Coefficient Matrix

       2         4        -1         6
      -1        -5         4         2
       1         2         3         1
       3         5        -1        -3

Constant Vectors

      36        11
      15         0
      22         7
      -6         4

 ** Output **

ierr =      0
```

```
     Solution

             1           1
             2           1
             4           1
             5           1
```

2.2.2 ASL_dbgmsl, ASL_rbgmsl Simultaneous Linear Equations (Real Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_dbgmsl (a, lna, n, b, ipvt);
Single precision:
  ierr = ASL_rbgmsl (a, lna, n, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (real matrix, two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (b) and (c))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

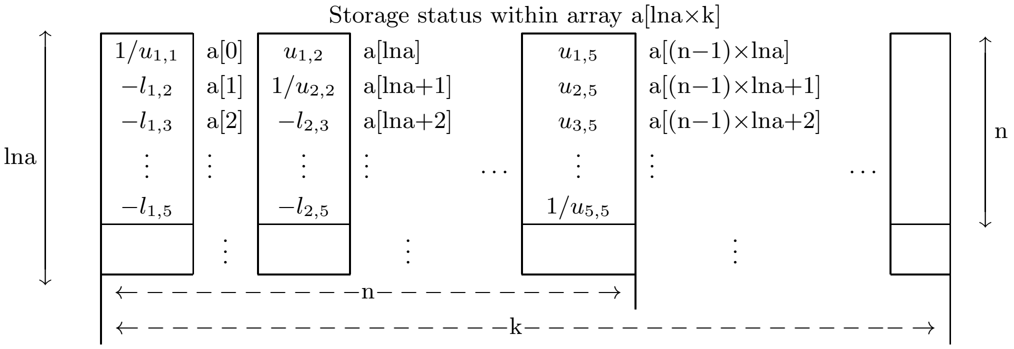
(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.2.1 $\left\{\begin{array}{l}\text{ASL\_dbgmsm}\\ \text{ASL\_rbgmsm}\end{array}\right\}$ to perform the calculations. However, when 2.2.1 $\left\{\begin{array}{l}\text{ASL\_dbgmsm}\\ \text{ASL\_rbgmsm}\end{array}\right\}$ cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this function only once and then call function 2.2.5 $\left\{\begin{array}{l}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LU decomposition of matrix *A* only once.

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

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

$$\begin{array}{cc} \text{Matrix } L & \text{Matrix } U \\ \begin{bmatrix} 1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0 \end{bmatrix} & \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\ 0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\ 0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & u_{5,5} \end{bmatrix} \end{array}$$

$$\Downarrow$$



Remarks

a. lna ≥ n and n ≤ k must hold.

Figure 2−2 Storage Status of Matrices L and U

(7) Example

(a) Problem

Solve the following simultaneous linear equations and obtain the condition number.

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

(b) Input data

Coefficient matrix a, $\mathrm{lna} = 11, \mathrm{n} = 4$, and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbgmsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int lna;
    int n;
    double *b;
    int *ipvt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbgmsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbgmsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &lna );
```

```
    fscanf( fp, "%d", &n );

    a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+lna*j] );
            printf( "%8.3g  ", a[i+lna*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector\n\n");
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );

    ierr = ASL_dbgmsl(a, lna, n, b, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
    }

    free( a );
    free( b );
    free( ipvt );


    return 0;
}
```

(d) Output results

```
*** ASL_dbgmsl ***

 ** Input **

 n =      4

Coefficient Matrix

       2         4        -1         6
      -1        -5         4         2
       1         2         3         1
       3         5        -1        -3

Constant Vector

      36
      15
      22
      -6

 ** Output **

ierr =      0

Solution
```

```
x[     0] =         1
x[     1] =         2
x[     2] =         4
x[     3] =         5
```

2.2.3 ASL_dbgmlu, ASL_rbgmlu
LU Decomposition of a Real Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbgmlu (a, lna, n, ipvt);
Single precision:
ierr = ASL_rbgmlu (a, lna, n, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real Matrix A (two-dimensional array type)
				Output	Unit upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The unit lower triangular matrix *L* is stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix *U* is stored in the upper triangular portion. However, since the diagonal components of matrix *L* always are 1.0, they are not stored in array a. In addition, the reciprocals of the diagonal components of *U* are stored. (See Fig. 2−2 in Section 2.2.2)

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

2.2.4 ASL_dbgmlc, ASL_rbgmlc
LU Decomposition and Condition Number of a Real Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbgmlc (a, lna, n, ipvt, &cond, w1);
Single precision:
ierr = ASL_rbgmlc (a, lna, n, ipvt, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	lna×n	Input	Real Matrix A (two-dimensional array type)
				Output	Unit upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
5	cond	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
6	w1	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of matrix L always are 1.0, they are not stored in array a. In addition, the reciprocals of the diagonal components of U are stored. (See Fig. 2−2 in Section 2.2.2)

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

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

2.2.5 ASL_dbgmls, ASL_rbgmls Simultaneous Linear Equations (LU-Decomposed Real Matrix)

(1) **Function**

ASL_dbgmls or ASL_rbgmls solves the simultaneous linear equations $LU\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbgmls (a, lna, n, b, ipvt);
Single precision:
ierr = ASL_rbgmls (a, lna, n, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LU decomposition (real matrix, two-dimensional array type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
5	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n is equal to 1	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using this function. Normally, you should decompose matrix A by calling the 2.2.3 $\left\{\begin{array}{c}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.2.4 $\left\{\begin{array}{c}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{array}\right\}$.
In addition, if you have already used 2.2.2 $\left\{\begin{array}{c}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.2.6 $\left\{\begin{array}{c}\text{ASL\_dbgmms}\\ \text{ASL\_rbgmms}\end{array}\right\}$ to perform the calculations.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of matrix L always are 1.0, they should not be stored in array a. In addition, the reciprocals of the diagonal components of U must be stored. (See Fig. 2−2 in Section 2.2.2.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.2.3 $\left\{\begin{array}{c}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{array}\right\}$, 2.2.4 $\left\{\begin{array}{c}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{array}\right\}$, and 2.2.2 $\left\{\begin{array}{c}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ functions which perform LU decomposition of matrix A.

2.2.6 ASL_dbgmms, ASL_rbgmms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Real Matrix)

(1) **Function**

ASL_dbgmms or ASL_rbgmms solves the simultaneous linear equations $LU\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
  ierr = ASL_dbgmms (a, lna, n, b, lnb, m, ipvt);
Single precision:
  ierr = ASL_rbgmms (a, lna, n, b, lnb, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D*}\\ \text{R*}\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LU decomposition (real matrix, two-dimensional array type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}\text{D*}\\ \text{R*}\end{array}\right\}$	lna×n	Input	Matrix consisting of constant vector $\boldsymbol{b_i}$ $[A', \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Matrix consisting of Solution vector $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Number of right-hand side vectors, m
7	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step (See Note (c)).
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0 < n ≤ lna

(b) 0 < m

(c) 0 < ipvt[i − 1] ≤ n (i = 1, . . . , n)

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n is equal to 1	$\mathrm{b}[\mathrm{lna}*(\mathrm{i}-1)] \leftarrow \mathrm{b}[\mathrm{lna}*(\mathrm{i}-1)]/\mathrm{a}[0]$ $(i=1,2,\cdots,\mathrm{m})$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	

(6) Notes

(a) The coefficient matrix A must be LU decomposed before using this function. Normally, you should decompose matrix A by calling the 2.2.3 $\left\{\begin{array}{c}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.2.4 $\left\{\begin{array}{c}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{array}\right\}$.

In addition, if you have already used 2.2.2 $\left\{\begin{array}{c}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of matrix L always are 1.0, they should not be stored in array a. In addition, the reciprocals of the diagonal components of U must be stored. (See Fig. 2−2 in Section 2.2.2.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.2.3 $\left\{\begin{array}{c}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{array}\right\}$, 2.2.4 $\left\{\begin{array}{c}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{array}\right\}$, and 2.2.2 $\left\{\begin{array}{c}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ functions which perform LU decomposition of matrix A.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix a, $\mathrm{lna} = 10, \mathrm{n} = 4$, matrix consisting of constant vector B, lnb=B and m=2.

(c) Main program

```
/* C interface example for ASL_dbgmms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int lna=11;
    int n=4;
```

```
double *b;
int lnb=11;
int m=2;
int *ipvt;
int ierr_lu,ierr_ms;
int i,j;
FILE *fp;

fp = fopen( "dbgmms.dat", "r" );

if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_dbgmms ***\n" );
printf( "\n     ** Input **\n\n" );

a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
if( a == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}

b = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}

ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
if( ipvt == NULL )
{
    printf( "no enough memory for array ipvt\n" );
    return -1;
}

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", m );

printf( "\n\tCoefficient Matrix a\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+lna*j] );
        printf( "%8.3g", a[i+lna*j] );
    }
    printf( "\n" );
}

ierr_lu = ASL_dbgmlu(a, lna, n, ipvt);

if( ierr_lu != 0 ) {
    printf( "\tierr ( ASL_dbgmlu ) = %6d\n", ierr_lu );
    return 0;
}

printf( "\n\tConstant Vectors b\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        fscanf( fp, "%lf", &b[i+lnb*j] );
        printf( "%8.3g", b[i+lnb*j] );
    }
    printf( "\n" );
}

fclose( fp );

ierr_ms = ASL_dbgmms(a, lna, n, b, lnb, m, ipvt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr ( ASL_dbgmlu ) = %6d\n\n", ierr_lu );
printf( "\tierr ( ASL_dbgmms ) = %6d\n",   ierr_ms );

printf( "\n\tSolution\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "%8.3g", b[i+lnb*j] );
    }
```

```
            printf( "\n" );
        }

        free(    a );
        free(    b );
        free( ipvt );

        return 0;
    }
```

(d) Output results

```
*** ASL_dbgmms ***

 ** Input **

n =      4
m =      2

Coefficient Matrix a

       2       4      -1       6
      -1      -5       4       2
       1       2       3       1
       3       5      -1      -3

Constant Vectors b

      36      11
      15       0
      22       7
      -6       4

 ** Output **

ierr ( ASL_dbgmlu ) =      0

ierr ( ASL_dbgmms ) =      0

Solution

       1       1
       2       1
       4       1
       5       1
```

2.2.7 ASL_dbgmdi, ASL_rbgmdi Determinant and Inverse Matrix of a Real Matrix

(1) **Function**

ASL_dbgmdi or ASL_rbgmdi obtains the determinant and inverse matrix of the real matrix A (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method.

(2) **Usage**

Double precision:
ierr = ASL_dbgmdi (a, lna, n, ipvt, det, isw, w1);
Single precision:
ierr = ASL_rbgmdi (a, lna, n, ipvt, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×n	Input	Real matrix A (two-dimensional array type) after LU decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
5	det	{D* / R*}	2	Output	Determinant of matrix A (See Note (d))
6	isw	I	1	Input	Processing switch isw > 0: Obtain determinant. isw = 0: Obtain determinant and inverse matrix. isw < 0: Obtain inverse matrix.
7	w1	{D* / R*}	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	det[0] ← a[0] (See Note (d)) det[1] ← 0.0 a[0] ← 1.0/a[0].
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using this function. Use any of the 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ functions to perform the decomposition.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of matrix L always are 1.0, they should not be stored in array a. In addition, the reciprocals of the diagonal components of U must be stored. (See 2.2.2 Figure 2−2).

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the function that performs the LU decomposition of matrix A.

(d) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.2.8 ASL_dbgmlx, ASL_rbgmlx Improving the Solution of Simultaneous Linear Equations (Real Matrix)

(1) **Function**

ASL_dbgmlx or ASL_rbgmlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbgmlx (a, lna, n, alu, b, x, &itol, nit, ipvt, w1);
Single precision:
ierr = ASL_rbgmlx (a, lna, n, alu, b, x, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
$\text{I:} \left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Coefficient matrix A (real matrix, two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of arrays a and alu
3	n	I	1	Input	Order of matrix A
4	alu	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Coefficient matrix A after LU decomposition (See Note (a))
5	b	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	ipvt	I*	n	Input	Pivoting information (See Note (a))
10	w1	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ or 2.2.5 $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$, 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ or 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ function and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\mathrm{itol} \leq 0$$

or

$$\mathrm{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations and improve the solution.

$$\begin{bmatrix} 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 8 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ 4 \\ 4 \\ 4 \\ 3 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 10$ and constant vector $\boldsymbol{b}$.

(c) Main Program

```
/*      C interface example for ASL_dbgmlx */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*sa;
    int lna;
    int n;
    double *b,*sb;
    int itol=0;
    int nnit=0;
    int *ipvt;
    double *wk;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbgmlx.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbgmlx ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &lna );
    fscanf( fp, "%d", &n );

    a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    sa = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( sa == NULL )
    {
        printf( "no enough memory for array sa\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    sb = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( sb == NULL )
    {
        printf( "no enough memory for array sb\n" );
```

```
            return -1;
        }

        wk = ( double * )malloc((size_t)( sizeof(double) * n ));
        if( wk == NULL )
        {
            printf( "no enough memory for array wk\n" );
            return -1;
        }

        ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
        if( ipvt == NULL )
        {
            printf( "no enough memory for array ipvt\n" );
            return -1;
        }

        printf( "\t n = %6d\n", n );
        printf( "\n\tCoefficient Matrix\n\n");

        for( i=0 ; i<n ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<n ; j++ )
            {
                fscanf( fp, "%lf", &a[i+lna*j] );
                sa[i+lna*j] = a[i+lna*j];
                printf( "%8.3g  ", a[i+lna*j] );
            }
            printf( "\n" );
        }

        printf( "\n\n\tConstant Vector\n\n");
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            sb[i] = b[i];
            printf( "\t%8.3g\n", b[i] );
        }


        fclose( fp );


        ierr = ASL_dbgmsl(a, lna, n, b, ipvt);

        printf( "\n\tOriginal Solution\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
        }


        ierr = ASL_dbgmlx(sa, lna, n, a, sb, b, &itol, nnit, ipvt, wk);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n\n", ierr );
        printf( "\tImproved Solution\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
        }

        free( a );
        free( b );
        free( sa );
        free( sb );
        free( wk );
        free( ipvt );

        return 0;
    }
```

(d) Output results

```
*** ASL_dbgmlx ***

 ** Input **

 n =     10

Coefficient Matrix

      10         9         8         7         6         5         4         3         2         1  
       9         9         8         7         6         5         4         3         2         1  
       8         8         8         7         6         5         4         3         2         1  
       7         7         7         7         6         5         4         3         2         1  
       6         6         6         6         6         5         4         3         2         1  
       5         5         5         5         5         5         4         3         2         1  
```

```
              4         4         4         4         4         4         4         3         2         1
              3         3         3         3         3         3         3         3         2         1
              2         2         2         2         2         2         2         2         2         1
              1         1         1         1         1         1         1         1         1         1

        Constant Vector

              6
              5
              4
              4
              4
              3
              2
              2
              2
              1

        Original Solution

          x[     0] =         1
          x[     1] = -1.23e-16
          x[     2] =        -1
          x[     3] = -2.54e-16
          x[     4] =         1
          x[     5] = 7.99e-16
          x[     6] =        -1
          x[     7] = -7.4e-17
          x[     8] =         1
          x[     9] =         0

         ** Output **

        ierr =      0

        Improved Solution

          x[     0] =         1
          x[     1] = -4.68e-31
          x[     2] =        -1
          x[     3] = -1.33e-30
          x[     4] =         1
          x[     5] = -1.97e-31
          x[     6] =        -1
          x[     7] = -9.86e-32
          x[     8] =         1
          x[     9] =         0
```

2.3 COMPLEX MATRIX (TWO DIMENSIONAL ARRAY TYPE) (REAL ARGUMENT TYPE)

2.3.1 ASL_zbgmsm, ASL_cbgmsm

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

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_zbgmsm (abr, abi, lna, n, m, ipvt, w1);
Single precision:
  ierr = ASL_cbgmsm (abr, abi, lna, n, m, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	abr	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	See Contents	Input	Real part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **Size**: (lna×(n + m))
				Output	Real part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b)).
2	abi	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	See Contents	Input	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **Size**: (lna×(n + m))
				Output	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays abr and abi

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
4	n	I	1	Input	Order of matrix A
5	m	I	1	Input	Number of right-hand side vectors, m
6	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step (See Note (a)).
7	w1	$\begin{Bmatrix} \text{D}* \\ \text{R}* \end{Bmatrix}$	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(b) $0 < \text{m}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	abr[lna∗(n+i-1)] ← (abr[lna∗(n+i−1)] × abr[0] + abi[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2), abi[lna∗(n+i−1)] ← (abi[lna∗(n+i−1)] × abr[0] − abr[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2) (i=1, 2, · · ·, m)
2100	There existed the diagonal element which was close to zero in the *LU* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
$4000 + i$	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

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

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

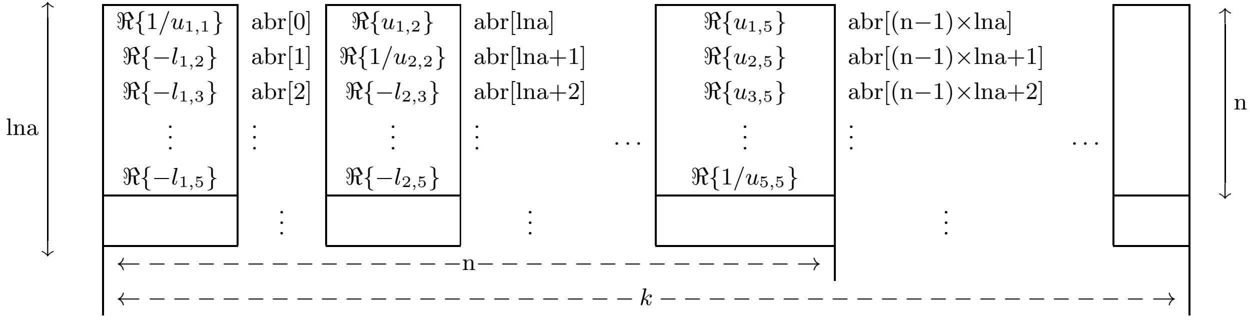
Figure 2−3 Storage Status of Matrices L and U

Matrix L Matrix U

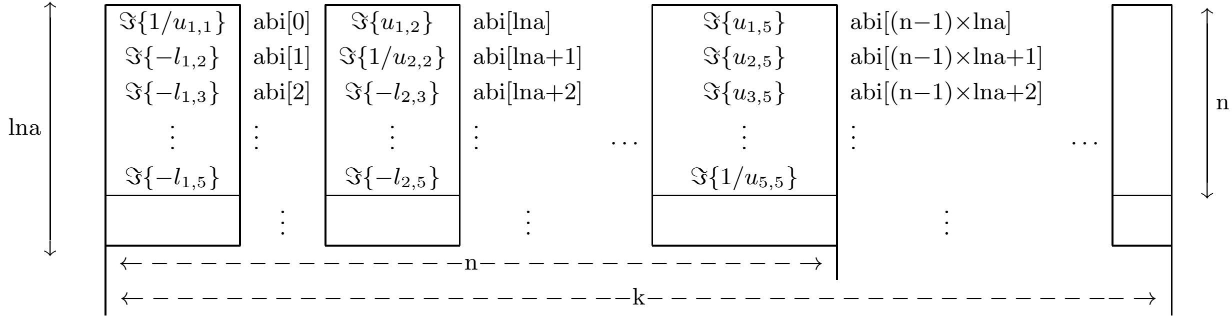
$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0 \end{bmatrix} \quad \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5}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

$$\Downarrow$$

Storage status within array abr[lna×k]



Storage status within array abi[lna×k]



Remarks

a. lna $\geq$ n and n+m $\leq$ k must be hold.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Array abr and abi in which coefficient matrix A, constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, lna=11, n=4 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbgmsm */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *abr;
    double *abi;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    double *w1;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgmsm.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgmsm ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    printf( "\t n = %6d m = %6d\n", n, m );

    abr = ( double * )malloc((size_t)( sizeof(double) * (lna*(lna+lma)) ));
    if( abr == NULL )
    {
        printf( "no enough memory for array abr\n" );
        return -1;
    }

    abi = ( double * )malloc((size_t)( sizeof(double) * (lna*(lna+lma)) ));
    if( abi == NULL )
    {
        printf( "no enough memory for array abi\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    w1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &abr[i+lna*j] );
            fscanf( fp, "%lf", &abi[i+lna*j] );
            printf( "(%8.3g,%8.3g)", abr[i+lna*j], abi[i+lna*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            fscanf( fp, "%lf", &abr[i+lna*(n+j)] );
            fscanf( fp, "%lf", &abi[i+lna*(n+j)] );
            printf( "(%8.3g,%8.3g)",abr[i+lna*(n+j)],abi[i+lna*(n+j)] );
        }
        printf( "\n" );
    }
```

```
    fclose( fp );

    ierr = ASL_zbgmsm(abr, abi, lna, n, m, ipvt, w1);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g)", abr[i+lna*(n+j)],abi[i+lna*(n+j)] );
        }
        printf( "\n" );
    }

    free( abr );
    free( abi );
    free( ipvt );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_zbgmsm ***

 ** Input **

 n =      4 m =      4

Coefficient Matrix

(       4,       2)(       3,       9)(       4,       1)(       7,       9)
(       6,       7)(       0,       4)(       4,       7)(       2,       5)
(       9,       3)(       6,       2)(       9,       5)(       8,       5)
(       1,       5)(       7,       9)(       3,       5)(       2,       4)

Constant Vectors

(       1,       0)(       0,       0)(       0,       0)(       0,       0)
(       0,       0)(       1,       0)(       0,       0)(       0,       0)
(       0,       0)(       0,       0)(       1,       0)(       0,       0)
(       0,       0)(       0,       0)(       0,       0)(       1,       0)

 ** Output **

ierr =      0

Solution

(  0.0133,  -0.073)(   0.181,  -0.247)(  -0.184,   0.178)(  -0.104,  -0.056)
( -0.0178, -0.0189)(  -0.068, -0.0696)( -0.0128,     0.1)(  0.0415, -0.0657)
( -0.0353,   0.138)( -0.0585,    0.17)(   0.133,  -0.241)(   0.131,  0.0191)
(  0.0494, -0.0686)(-0.00961,    0.13)(  0.0885, -0.0709)( -0.0462,  0.0662)
```

2.3.2 ASL_zbgmsl, ASL_cbgmsl Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbgmsl (ar, ai, lna, n, br, bi, ipvt, w1);
Single precision:
ierr = ASL_cbgmsl (ar, ai, lna, n, br, bi, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of coefficient matrix A (complex matrix, two-dimensional array type)
				Output	Real parts of unit upper triangular matrix U and low triangular matrix L when A is decomposed into $A = LU$ (See Notes (b) and (c))
2	ai	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of coefficient matrix (complex matrix, two-dimensional array type)
				Output	Imaginary parts of unit upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (b) and (c))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
8	w1	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	br[0] $\leftarrow \{\text{br}[0] \times \text{ar}[0] + \text{bi}[0] \times \text{ai}[0]\}$ $/ \ \{\text{ar}[0]^2 + \text{ai}[0]^2\}$ bi[0] $\leftarrow \{\text{bi}[0] \times \text{ar}[0] - \text{br}[0] \times \text{ai}[0]\}$ $/ \ \{\text{ar}[0]^2 + \text{ai}[0]^2\}$
2100	There existed the diagonal element which was close to zero in the *LU* decomposition of the coefficient matrix *A*. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the *i*-th processing step of the LU decomposition of coefficient matrix *A*. *A* is nearly singular.	

(6) **Notes**

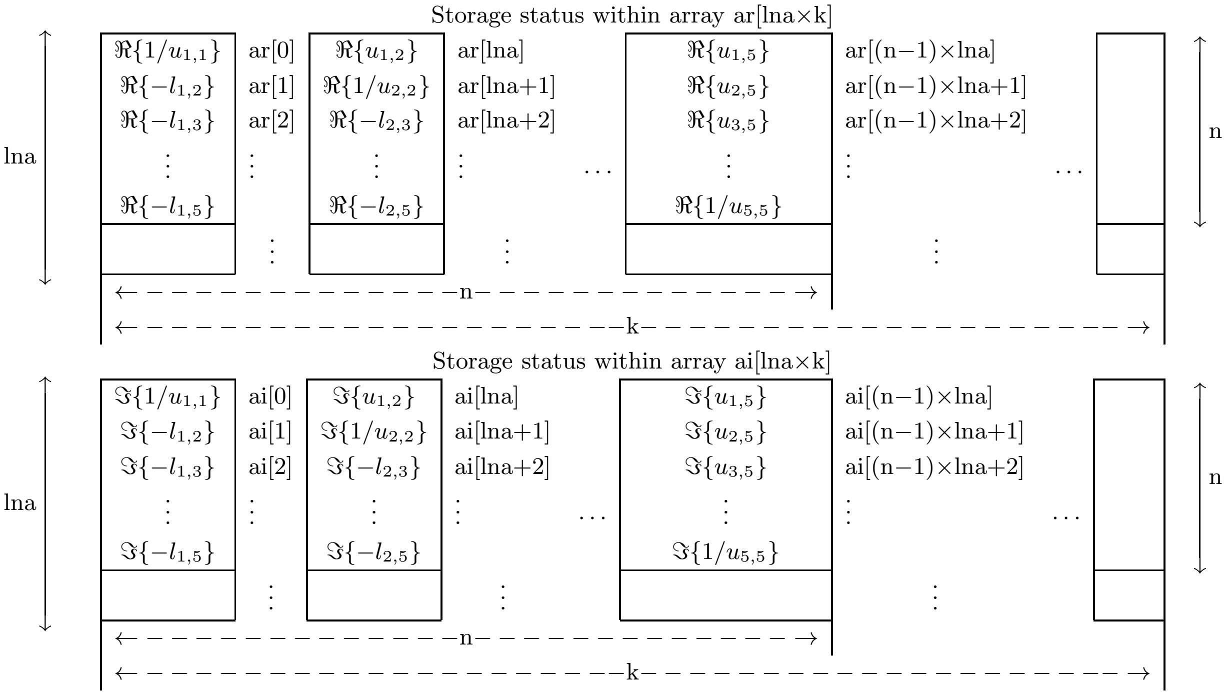
(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.3.1 $\left\{\begin{array}{l}\text{ASL\_zbgmsm}\\ \text{ASL\_cbgmsm}\end{array}\right\}$ to perform the calculations. However, when 2.3.1 $\left\{\begin{array}{l}\text{ASL\_zbgmsm}\\ \text{ASL\_cbgmsm}\end{array}\right\}$ cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this function only once and then call function 2.3.5 $\left\{\begin{array}{l}\text{ASL\_zbgmls}\\ \text{ASL\_cbgmls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LU decomposition of matrix *A* only once.

(b) This function performs partial pivoting when obtaining the LU decomposition of coefficient matrix *A*=(ar, ai). If the pivot row in the i-th step is row j ($\text{i} \leq \text{j}$), then j is stored in ipvt[i − 1]. In addition, among the column elements corresponding to row i and row j of matrix *A*, elements from column 1 to column n actually are exchanged at this time.

(c) The unit lower triangular matrix *L* is stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix *U* is stored in the upper triangular portion. However, since the diagonal components of *L* always are 1.0, they are not stored in array ar and ai. In addition, the reciprocals of the diagonal components of *U* are stored. In Fig. 2−4, $\Re\{z\}$ and $\Im\{z\}$ denote a real part and an imaginary part of a complex number z, respectively.

$$
\begin{array}{cc}
\text{Matrix } L & \text{Matrix } U \\
\begin{bmatrix}
1.0 & 0.0 & 0.0 & \cdots & 0.0 \\
l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\
l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0
\end{bmatrix}
&
\begin{bmatrix}
u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\
0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\
0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & u_{5,5}
\end{bmatrix}
\end{array}
$$

$$\Downarrow$$



Remarks

a. lna ≥ n, n ≤ k must hold.

Figure 2−4 Storage Status of Matrices L and U

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix real part ar and imaginary part ai, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_zbgmsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    int *kpvt;
    double *w;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgmsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgmsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );

    ar = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n" );
        return -1;
    }

    ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n" );
        return -1;
    }

    br = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( br == NULL )
    {
        printf( "no enough memory for array br\n" );
        return -1;
    }

    bi = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n" );
        return -1;
    }

    w = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }

    kpvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( kpvt == NULL )
```

```
    {
        printf( "no enough memory for array kpvt\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ar[i+na*j] );
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ai[i+na*j] );
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", ar[i+na*j],ai[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &br[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &bi[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g) \n", br[i], bi[i] );
    }


    fclose( fp );

    ierr = ASL_zbgmsl(ar, ai, na, n, br, bi, kpvt, w);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = (%8.3g , %8.3g) \n", i, br[i], bi[i] );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( w );
    free( kpvt );

    return 0;
}
```

(d) Output results

```
*** ASL_zbgmsl ***

 ** Input **

 n =      4

Coefficient Matrix (Real, Imaginary)

(       5 ,        8) (       7 ,        1) (       6 ,        3) (       1 ,        2)
(       1 ,        1) (       9 ,        5) (       4 ,        1) (       5 ,        0)
(       0 ,        4) (       3 ,        3) (       4 ,        2) (       6 ,        9)
(       7 ,        8) (       6 ,        0) (       7 ,        6) (      10 ,        4)

Constant Vector (Real, Imaginary)

(       3 ,       20)
(      -6 ,        7)
(       0 ,       -6)
(       0 ,       13)
```

```
 ** Output **

ierr =      0

Solution (Real, Imaginary)

  x[     0] = (        1 ,         1)
  x[     1] = (-2.22e-16 ,         1)
  x[     2] = (        1 ,    -5e-16)
  x[     3] = (       -1 ,        -1)
```

2.3.3 ASL_zbgmlu, ASL_cbgmlu
LU Decomposition of a Complex Matrix

(1) **Function**

ASL_zbgmlu or ASL_cbgmlu uses the Gauss method or the Crout method to perform an LU decomposition of the complex matrix A=(ar, ai) (two-dimensional array type).

(2) **Usage**

Double precision:
  ierr = ASL_zbgmlu (ar, ai, lna, n, ipvt, w1);
Single precision:
  ierr = ASL_cbgmlu (ar, ai, lna, n, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of coefficient matrix A (two-dimensional array type)
				Output	Real parts of unit upper triangular matrix U and low triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	ai	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of coefficient matrix (two-dimensional array type)
				Output	Imaginary parts of unit upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
6	w1	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array ar and ai. In addition, the reciprocals of the diagonal components of U are stored. (See 2.3.2 Figure 2−4.)

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

2.3.4 ASL_zbgmlc, ASL_cbgmlc
LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

ASL_zbgmlc or ASL_cbgmlc uses the Gauss method or the Crout method to perform an LU decomposition and obtain the condition number of the complex matrix A=(ar, ai) (two-dimensional array type).

(2) **Usage**

Double precision:
ierr = ASL_zbgmlc (ar, ai, lna, n, ipvt, &cond, w1);
Single precision:
ierr = ASL_cbgmlc (ar, ai, lna, n, ipvt, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of coefficient matrix A (two-dimensional array type)
				Output	Real parts of unit upper triangular matrix U and low triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	ai	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of coefficient matrix (two-dimensional array type)
				Output	Imaginary parts of unit upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
6	cond	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	1	Output	Reciprocal of the condition number

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
7	w1	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	$2 \times$ n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 <$ n $\leq$ lna

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array ar and ai are not changed and cond $\leftarrow$ 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the *LU* decomposition of the coefficient matrix *A*. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the i-th processing step. *A* is nearly singular.	

(6) **Notes**

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array ar and ai. In addition, the reciprocals of the diagonal components of U are stored. (See 2.3.2 Figure 2−4.)

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

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

2.3.5 ASL_zbgmls, ASL_cbgmls
Simultaneous Linear Equations (LU-Decomposed Complex Matrix)

(1) **Function**

ASL_zbgmls or ASL_cbgmls solves the simultaneous linear equations $LU\boldsymbol{x} = \boldsymbol{b}$ having the complex matrix A=(ar, ai) (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbgmls (ar, ai, lna, n, br, bi, ipvt);
Single precision:
ierr = ASL_cbgmls (ar, ai, lna, n, br, bi, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
2	ai	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \le \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	br[0] ← $\{\mathrm{br}[0] \times \mathrm{ar}[0] + \mathrm{bi}[0] \times \mathrm{ai}[0]\} / \{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$ bi[1] ← $\{\mathrm{bi}[0] \times \mathrm{ar}[0] - \mathrm{br}[0] \times \mathrm{ai}[0]\} / \{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A=(ar, ai) must be LU decomposed before using this function. Normally, you should decompose matrix A by calling the 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$.
In addition, if you have already used 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.3.6 $\left\{\begin{matrix}\text{ASL\_zbgmms}\\ \text{ASL\_cbgmms}\end{matrix}\right\}$ to perform the calculations.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they should not be stored in array ar and ai. In addition, the reciprocals of the diagonal components of U must be stored. (See 2.3.2 Figure 2−4.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$ functions which perform LU decomposition of matrix A.

2.3.6 ASL_zbgmms, ASL_cbgmms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Complex Matrix)

(1) Function

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

(2) Usage

Double precision:
ierr = ASL_zbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt);
Single precision:
ierr = ASL_cbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{matrix}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{matrix}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Real parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
2	ai	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Imaginary parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lnb×m	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lnb×m	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	lnb	I	1	Input	Adjustable dimension of arrays ar and ai
8	m	I	1	Input	Order of matrix B
9	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
10	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \le \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(c) $0 < \mathrm{ipvt[i-1]} \le \mathrm{n} \quad (\mathrm{i} = 1, \ldots, \mathrm{n})$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	br[lnb×i]← { br[lnb×i]×ar[0] +bi[lnb×i]×ai[0] } / { ar[0]^2+ ai[0]^2 } bi[lnb×i]← { bi[lnb×i]×ar[0] −br[lnb×i]×ai[0] } / { ar[0]^2+ ai[0]^2 } (i =0,1,···,m-1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	

(6) Notes

(a) The coefficient matrix A=(ar, ai) must be LU decomposed before using this function. Normally, you should decompose matrix A by calling the 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\\text{ASL\_cbgmlu}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\\text{ASL\_cbgmlc}\end{matrix}\right\}$.
In addition, if you have already used 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\\text{ASL\_cbgmsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they should not be stored in array ar and ai. In addition, the reciprocals of the diagonal components of U must be stored. (See 2.3.2 Figure 2−4.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\\text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\\text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\\text{ASL\_cbgmsl}\end{matrix}\right\}$ functions which perform LU decomposition of matrix A.

(7) Example

(a) ProblemSolve the following simultaneous linear equations.

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

(b) Input data
Array abr and abi in which coefficient matrix A, constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, lna=11, lnb=11, n=4 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbgmms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar;
    double *ai;
    int lna=11;
    int lnb=11;
    int n;
    int m;
    double *br;
    double *bi;
    int *ipvt;
    double *w;
    int ierr,kerr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgmms.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgmms ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );

    ar = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n" );
        return -1;
    }

    ai = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n" );
        return -1;
    }

    br = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
    if( br == NULL )
    {
        printf( "no enough memory for array br\n" );
        return -1;
    }

    bi = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n" );
        return -1;
    }

    w = ( double * )malloc((size_t)( sizeof(double) * (2*n) ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\tn = %6d\tm = %6d\n", n,m );

    printf( "\n\tCoefficient Matrix (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ar[i+lna*j] );
            fscanf( fp, "%lf", &ai[i+lna*j] );
            printf( "(%6.3g,%6.3g) ", ar[i+lna*j],ai[i+lna*j] );
        }
        printf( "\n" );
```

```
    }

    printf( "\n\tConstant Vectors (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            fscanf( fp, "%lf", &br[i+lnb*j] );
            fscanf( fp, "%lf", &bi[i+lnb*j] );
            printf( "(%6.3g,%6.3g) ", br[i+lnb*j],bi[i+lnb*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_zbgmlu(ar, ai, lna, n, ipvt, w);
    kerr = ASL_zbgmms(ar, ai, lna, n, br, bi, lnb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tzbgmlu ierr = %6d\n", ierr );
    printf( "\tzbgmms ierr = %6d\n", kerr );

    printf( "\n\tSolution (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g) ", br[i+lnb*j],bi[i+lnb*j] );
        }
        printf( "\n" );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( w );
    free( ipvt );

    return 0;
}
```

(d) Output results

```
    *** ASL_zbgmms ***

     ** Input **

    n =      4  m =      4

    Coefficient Matrix (Real,Imaginary)

    (     4,     2) (     3,     9) (     4,     1) (     7,     9)
    (     6,     7) (     0,     4) (     4,     7) (     2,     5)
    (     9,     3) (     6,     2) (     9,     5) (     8,     5)
    (     1,     5) (     7,     9) (     3,     5) (     2,     4)

    Constant Vectors (Real,Imaginary)

    (     1,     0) (     0,     0) (     0,     0) (     0,     0)
    (     0,     0) (     1,     0) (     0,     0) (     0,     0)
    (     0,     0) (     0,     0) (     1,     0) (     0,     0)
    (     0,     0) (     0,     0) (     0,     0) (     1,     0)

     ** Output **

    zbgmlu ierr =      0
    zbgmms ierr =      0

    Solution (Real,Imaginary)

(  0.0133,  -0.073) (   0.181,  -0.247) (  -0.184,   0.178) (  -0.104,  -0.056)
( -0.0178, -0.0189) (  -0.068, -0.0696) ( -0.0128,     0.1) (  0.0415, -0.0657)
( -0.0353,   0.138) ( -0.0585,    0.17) (   0.133,  -0.241) (   0.131,  0.0191)
(  0.0494, -0.0686) (-0.00961,    0.13) (  0.0885, -0.0709) ( -0.0462,  0.0662)
```

2.3.7 ASL_zbgmdi, ASL_cbgmdi
Determinant and Inverse Matrix of a Complex Matrix

(1) **Function**

ASL_zbgmdi or ASL_cbgmdi obtains the determinant and inverse matrix of the complex matrix A=(ar, ai) (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method.

(2) **Usage**

Double precision:
  ierr = ASL_zbgmdi (ar, ai, lna, n, ipvt, det, isw, w1);
Single precision:
  ierr = ASL_cbgmdi (ar, ai, lna, n, ipvt, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Real parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
				Output	Real parts of inverse matrix of matrix A
2	ai	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Imaginary parts of coefficient matrix A after LU decomposition (See Notes (a) and (b))
				Output	Imaginary parts inverse matrix of matrix A
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
6	det	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	3	Output	Determinant of matrix A (See Note (d))
7	isw	I	1	Input	Processing switch isw>0: Obtain determinant. isw=0: Obtain determinant and inverse matrix. isw<0: Obtain inverse matrix.
8	w1	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	2 × n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	$\mathrm{det}[0] \leftarrow \mathrm{ar}[0]$ $\mathrm{det}[1] \leftarrow \mathrm{ai}[0]$ $\mathrm{det}[2] \leftarrow 0.0$ $\mathrm{ar}[0] \leftarrow \ \mathrm{ar}[0]/\{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$ $\mathrm{ai}[0] \leftarrow \ -\mathrm{ai}[0]/\{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using this function. Use any of the 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\\text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\\text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\\text{ASL\_cbgmsl}\end{matrix}\right\}$ functions to perform the decomposition.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array ar and ai with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they should not be stored in array ar and ai. In addition, the reciprocals of the diagonal components of U must be stored. (See 2.3.2 Figure 2−4).

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt[i − 1]. This information is given by the function that performs the LU decomposition of matrix A.

(d) The determinant is given by the following expression: $\Re\{det(A)\} = \mathrm{det}[0] \times 10^{\mathrm{det}[2]}$
$\Im\{det(A)\} = \mathrm{det}[1] \times 10^{\mathrm{det}[2]}$
Scaling is performed at this time so that:

$$1.0 \leq |\mathrm{det}[0]| + |\mathrm{det}[1]| < 10.0$$

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.3.8 ASL_zbgmlx, ASL_cbgmlx Improving the Solution of Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

ASL_zbgmlx or ASL_cbgmlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the complex matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbgmlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, ipvt, w1);
Single precision:
ierr = ASL_cbgmlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer, long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* R*}	lna×n	Input	Real parts of coefficient matrix A (two-dimensional array type)
2	ai	{D* R*}	lna×n	Input	Imaginary parts of coefficient matrix A (two-dimensional array type)
3	lna	I	1	Input	Adjustable dimension of array ar, ai, alr, and ali
4	n	I	1	Input	Order of matrix A
5	alr	{D* R*}	lna×n	Input	Real parts of coefficient matrix A after LU decomposition (See Note (a))
6	ali	{D* R*}	lna×n	Input	Imaginary parts of coefficient matrix A after LU decomposition (See Note (a))
7	br	{D* R*}	n	Input	Real part of constant vector $\boldsymbol{b}$
8	bi	{D* R*}	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
9	xr	{D* R*}	n	Input	Real part of approximate solution $\boldsymbol{x}$
				Output	Real part of iteratively improved solution $\boldsymbol{x}$
10	xi	{D* R*}	n	Input	Imaginary part of approximate solution $\boldsymbol{x}$
				Output	Imaginary part of iteratively improved solution $\boldsymbol{x}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
11	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
12	nit	I	1	Input	Maximum number of iterations (See Note (d))
13	ipvt	I*	n	Input	Pivoting information (See Note (a))
14	w1	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	$3 \times$ n	Work	Work area
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculation the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$ or 2.3.5 $\left\{\begin{matrix}\text{ASL\_zbgmls}\\ \text{ASL\_cbgmls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after being decomposed by 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$, 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$ function and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.4.1 ASL_zbgnsm, ASL_cbgnsm

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

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_zbgnsm (ab, lna, n, m, ipvt);

Single precision:

ierr = ASL_cbgnsm (ab, lna, n, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: { int as for 32bit Integer
R:Single precision real	C:Single precision complex	long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ab	{Z* C*}	See Contents	Input	Matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **Size**: (lna×(n + m))
				Output	Matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array ab
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	Number of right-hand side vectors, m
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step (See Note (a)).
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0 < n ≤ lna

(b) 0 < m

(5) **Error indicator (Return Value)**

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

(6) **Notes**

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

(b) The unit lower triangular matrix L is stored in the lower triangular portion of array ab with the sign changed, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array ab. In addition, the reciprocals of the diagonal components of U are stored. (See Figure 2−1 in Section 2.2.1).

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Array ab in which coefficient matrix A, constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, lna=11, n=4 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbgnsm */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
```

```
{
    double _Complex *ab;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgnsm.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgnsm ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    printf( "\t n = %6d m = %6d\n", n, m );

    ab = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*(lna+lma)) ));
    if( ab == NULL )
    {
        printf( "no enough memory for array ab\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            ab[i+lna*j] = tmp_re + tmp_im * _Complex_I;
            printf( "(%8.3g,%8.3g)", creal(ab[i+lna*j]), cimag(ab[i+lna*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            ab[i+lna*(n+j)] = tmp_re + tmp_im * _Complex_I;
            printf( "(%8.3g,%8.3g)",creal(ab[i+lna*(n+j)]),cimag(ab[i+lna*(n+j)]) );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_zbgnsm(ab, lna, n, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g)", creal(ab[i+lna*(n+j)]),cimag(ab[i+lna*(n+j)]) );
        }
        printf( "\n" );
    }

    free( ab );
    free( ipvt );
```

```
    return 0;
}
```

(d) Output results

```
*** ASL_zbgnsm ***

 ** Input **

 n =     4 m =     4

Coefficient Matrix

(       4,       2)(       3,       9)(       4,       1)(       7,       9)
(       6,       7)(       0,       4)(       4,       7)(       2,       5)
(       9,       3)(       6,       2)(       9,       5)(       8,       5)
(       1,       5)(       7,       9)(       3,       5)(       2,       4)

Constant Vectors

(       1,       0)(       0,       0)(       0,       0)(       0,       0)
(       0,       0)(       1,       0)(       0,       0)(       0,       0)
(       0,       0)(       0,       0)(       1,       0)(       0,       0)
(       0,       0)(       0,       0)(       0,       0)(       1,       0)

 ** Output **

ierr =      0

Solution

(  0.0133,  -0.073)(   0.181,  -0.247)(  -0.184,   0.178)(  -0.104,  -0.056)
( -0.0178, -0.0189)(  -0.068, -0.0696)( -0.0128,     0.1)(  0.0415, -0.0657)
( -0.0353,   0.138)( -0.0585,    0.17)(   0.133,  -0.241)(   0.131,  0.0191)
(  0.0494, -0.0686)(-0.00961,    0.13)(  0.0885, -0.0709)( -0.0462,  0.0662)
```

2.4.2 ASL_zbgnsl, ASL_cbgnsl Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbgnsl (a, lna, n, b, ipvt);
Single precision:
ierr = ASL_cbgnsl (a, lna, n, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix (complex matrix, two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (b) and (c))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step. (See Note (b))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \le \text{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.4.1 $\left\{\begin{array}{l}\text{ASL\_zbgnsm}\\ \text{ASL\_cbgnsm}\end{array}\right\}$ to perform the calculations. However, when 2.4.1 $\left\{\begin{array}{l}\text{ASL\_zbgnsm}\\ \text{ASL\_cbgnsm}\end{array}\right\}$ cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this function only once and then call function 2.4.5 $\left\{\begin{array}{l}\text{ASL\_zbgnls}\\ \text{ASL\_cbgnls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LU decomposition of matrix A only once.

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

(c) The unit lower triangular matrix L is stored in the lower triangular portion of array a with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array a. Also, reciprocals are stored for the diagonal components of U. (See Figure 2−2 in Section 2.2.2).

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\text{lna} = 11, \text{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_zbgnsl */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a;
    int lna;
    int n;
    double _Complex *b;
    int *ipvt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgnsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgnsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &lna );
    fscanf( fp, "%d", &n );

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = (  double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix   (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            double tmp_re;
            fscanf( fp, "%lf", &tmp_re );
            a[i+lna*j] = tmp_re;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            double tmp_im;
            fscanf( fp, "%lf", &tmp_im );
            a[i+lna*j] = a[i+lna*j] + tmp_im * _Complex_I;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+lna*j]),cimag(a[i+lna*j]) );
        }
        printf( "\n" );
    }

    for( i=0 ; i<n ; i++ )
    {
        double tmp_re;
        fscanf( fp, "%lf", &tmp_re );
        b[i] = tmp_re;
```

```
        }
        for( i=0 ; i<n ; i++ )
        {
            double tmp_im;
            fscanf( fp, "%lf", &tmp_im );
            b[i] = b[i] + tmp_im * _Complex_I;
        }

        printf( "\n\tConstant Vector  (Real, Imaginary)\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
        }

        fclose( fp );

        ierr = ASL_zbgnsl(a, lna, n, b, ipvt);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution   (Real, Imaginary)\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] =(%8.3g , %8.3g)\n", i,creal(b[i]), cimag(b[i]) );
        }

        free( a );
        free( b );
        free( ipvt );


        return 0;
    }
```

(d) Output results

```
        *** ASL_zbgnsl ***

         ** Input **

         n =      4

        Coefficient Matrix   (Real, Imaginary)

        (       5 ,        8) (       7 ,        1) (       6 ,        3) (       1 ,        2)
        (       1 ,        1) (       9 ,        5) (       4 ,        1) (       5 ,        0)
        (       0 ,        4) (       3 ,        3) (       4 ,        2) (       6 ,        9)
        (       7 ,        8) (       6 ,        0) (       7 ,        6) (      10 ,        4)

        Constant Vector  (Real, Imaginary)

        (       3 ,       20)
        (      -6 ,        7)
        (       0 ,       -6)
        (       0 ,       13)

         ** Output **

        ierr =      0

        Solution   (Real, Imaginary)

          x[     0] =(       1 ,        1)
          x[     1] =(-1.67e-16 ,        1)
          x[     2] =(       1 , -2.78e-16)
          x[     3] =(      -1 ,       -1)
```

2.4.3 ASL_zbgnlu, ASL_cbgnlu
LU Decomposition of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbgnlu (a, lna, n, ipvt);
Single precision:
ierr = ASL_cbgnlu (a, lna, n, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{Z}*\\ \text{C}*\end{array}\right\}$	lna×n	Input	Complex matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step. (See Note (b))
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array a with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array a. Also, reciprocals are stored for the diagonal components of U. (See Fig. 2−2 in Section 2.2.2.)

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

2.4.4 ASL_zbgnlc, ASL_cbgnlc LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_zbgnlc (a, lna, n, ipvt, &cond, w1);
Single precision:
  ierr = ASL_cbgnlc (a, lna, n, ipvt, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}Z*\\ C*\end{array}\right\}$	lna×n	Input	Complex matrix (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step. (See Note (b))
5	cond	$\left\{\begin{array}{l}D*\\ R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
6	w1	$\left\{\begin{array}{l}Z*\\ C*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The unit lower triangular matrix L is stored in the lower triangular portion of array a with a minus sign added to each element, and the upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of L always are 1.0, they are not stored in array a. Also, reciprocals are stored for the diagonal components of U. (See Fig. 2−2 in Section 2.2.2.)

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

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

2.4.5 ASL_zbgnls, ASL_cbgnls
Simultaneous Linear Equations (LU-Decomposed Complex Matrix)

(1) **Function**

ASL_zbgnls or ASL_cbgnls solves the simultaneous linear equations $LU\boldsymbol{x} = \boldsymbol{b}$ having the complex matrix A (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbgnls (a, lna, n, b, ipvt);
Single precision:
ierr = ASL_cbgnls (a, lna, n, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix after LU decomposition (complex matrix, two-dimensional array type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step. (See Note (c))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using the ASL_zbgnls or ASL_cbgnls function. Normally, you should decompose matrix A by calling the 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$. In addition, if you have already used 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.4.6 $\left\{\begin{matrix}\text{ASL\_zbgnms}\\ \text{ASL\_cbgnms}\end{matrix}\right\}$ to perform the calculations.

(b) The unit lower triangular matrix L is stored in the lower triangular portions of array a with a minus sign added to each element, and the unit upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of U always are 1.0, they are not stored in array a. Also, reciprocals must be stored for the diagonal components of U. (See Fig. 2−2 in Section 2.2.2.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ functions which perform LU decomposition of matrix A.

2.4.6 ASL_zbgnms, ASL_cbgnms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Complex Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbgnms (a, lna, n, b, lnb, m, ipvt);
Single precision:
ierr = ASL_cbgnms (a, lna, n, b, lnb, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real	C:Single precision complex	

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix after LU decomposition (complex matrix, two-dimensional array type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lnb×m	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Order of matrix B
7	ipvt	I*	n	Input	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step. (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(c) $0 < \mathrm{ipvt}[\mathrm{i}-1] \leq \mathrm{n} \quad (\mathrm{i} = 1, \ldots, \mathrm{n})$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	b[lnb×i]← b[lnb×i]/a[0] (i =0,1,$\cdots$,m−1) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using the ASL_zbgnls or ASL_cbgnls function. Normally, you should decompose matrix A by calling the 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\\text{ASL\_cbgnlu}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\\text{ASL\_cbgnlc}\end{matrix}\right\}$. In addition, if you have already used 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\\text{ASL\_cbgnsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output.

(b) The unit lower triangular matrix L is stored in the lower triangular portions of array a with a minus sign added to each element, and the unit upper triangular matrix U is stored in the upper triangular portion. However, since the diagonal components of U always are 1.0, they are not stored in array a. Also, reciprocals must be stored for the diagonal components of U. (See Fig. 2−2 in Section 2.2.2.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\\text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\\text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\\text{ASL\_cbgnsl}\end{matrix}\right\}$ functions which perform LU decomposition of matrix A.

(7) **Example**

(a) ProblemSolve the following simultaneous linear equations.

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

(b) Input data
Coefficient matrix A, constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, lna=11, lnb=11, n=4 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbgnms */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a;
    int lna=11;
    int lnb=11;
    int n;
    int m;
    double _Complex *b;
    int *ipvt;
    int ierr,kerr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgnms.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgnms ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lnb*m) ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    printf( "\tn = %6d\tm = %6d\n", n,m );

    printf( "\n\tCoefficient Matrix (Real,Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            a[i+lna*j] = tmp_re + tmp_im * _Complex_I;
            printf( "(%6.3g,%6.3g) ", creal(a[i+lna*j]),cimag(a[i+lna*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            b[i+lnb*j] = tmp_re + tmp_im * _Complex_I;
            printf( "(%6.3g,%6.3g) ", creal(b[i+lnb*j]),cimag(b[i+lnb*j]) );
        }
        printf( "\n" );
    }

    fclose( fp );
```

```
    ierr = ASL_zbgnlu(a, lna, n, ipvt);
    kerr = ASL_zbgnms(a, lna, n, b, lnb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tzbgnlu ierr = %6d\n", ierr );
    printf( "\tzbgnms ierr = %6d\n", kerr );

    printf( "\n\tSolution (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g) ", creal(b[i+lnb*j]),cimag(b[i+lnb*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( ipvt );

    return 0;
}
```

(d) Output results

```
    *** ASL_zbgnms ***

     ** Input **

    n =      4  m =      4

    Coefficient Matrix (Real,Imaginary)

    (     4,     2) (     3,     9) (     4,     1) (     7,     9)
    (     6,     7) (     0,     4) (     4,     7) (     2,     5)
    (     9,     3) (     6,     2) (     9,     5) (     8,     5)
    (     1,     5) (     7,     9) (     3,     5) (     2,     4)

    Constant Vectors (Real,Imaginary)

    (     1,     0) (     0,     0) (     0,     0) (     0,     0)
    (     0,     0) (     1,     0) (     0,     0) (     0,     0)
    (     0,     0) (     0,     0) (     1,     0) (     0,     0)
    (     0,     0) (     0,     0) (     0,     0) (     1,     0)

     ** Output **

    zbgnlu ierr =      0
    zbgnms ierr =      0

    Solution (Real,Imaginary)

(  0.0133,  -0.073) (   0.181,  -0.247) (  -0.184,   0.178) (  -0.104,  -0.056)
( -0.0178, -0.0189) (  -0.068, -0.0696) ( -0.0128,     0.1) (  0.0415, -0.0657)
( -0.0353,   0.138) ( -0.0585,    0.17) (   0.133,  -0.241) (   0.131,  0.0191)
(  0.0494, -0.0686) (-0.00961,    0.13) (  0.0885, -0.0709) ( -0.0462,  0.0662)
```

2.4.7 ASL_zbgndi, ASL_cbgndi Determinant and Inverse Matrix of a Complex Matrix

(1) **Function**

ASL_zbgndi or ASL_cbgndi obtains the determinant and inverse matrix of the complex matrix A (two-dimensional array type) which has been LU decomposed by the Gauss method or the Crout method.

(2) **Usage**

Double precision:
ierr = ASL_zbgndi (a, lna, n, ipvt, &cdet, &det, isw, w1);
Single precision:
ierr = ASL_cbgndi (a, lna, n, ipvt, &cdet, &det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{Z}*\\ \text{C}*\end{array}\right\}$	lna×n	Input	Complex matrix A (two-dimensional array type) after LU decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Input	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th processing step of the LU decomposition. (See Note (c))
5	cdet	$\left\{\begin{array}{l}\text{Z}*\\ \text{C}*\end{array}\right\}$	1	Output	Determinant of matrix A (See Note (d))
6	det	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	1	Output	Determinant of matrix A (See Note (d))
7	isw	I	1	Input	Processing switch isw > 0: Obtain determinant. isw = 0: Obtain determinant and inverse matrix. isw < 0: Obtain inverse matrix.
8	w1	$\left\{\begin{array}{l}\text{Z}*\\ \text{C}*\end{array}\right\}$	n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	cdet ← a[0] det ← 0.0 (See Note (d)) and a[0] ← 1.0/a[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be LU decomposed before using the ASL_zbgndi or ASL_cbgndi function. Use any of the functions 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ to perform the decomposition.

(b) The unit lower triangular matrix L must be stored in the lower triangular portion of array a with the sign changed, and the upper triangular matrix U must be stored in the upper triangular portion. However, since the diagonal components of matrix L always are 1.0, they should not be stored in array a. In addition, the reciprocals of the diagonal components of U must be stored. (See 2.2.2 Figure 2−2).

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ functions which perform LU decomposition of matrix A.

(d) The determinant is given by the following expression:

$$\det(A) = \text{cdet} \times 10^{\text{det}}$$

Scaling is performed at this time so that:

$$1.0 \le |\Re\{\text{cdet}\}| + |\Im\{\text{cdet}\}| < 10.0$$

where, the notation $\Re$ and $\Im$ mean that the real and imaginary parts of the complex number are to be taken, respectively.

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.4.8 ASL_zbgnlx, ASL_cbgnlx
Improving the Solution of Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

ASL_zbgnlx or ASL_cbgnlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the complex matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbgnlx (a, lna, n, alu, b, x, &itol, nit, ipvt, w1);
Single precision:
ierr = ASL_cbgnlx (a, lna, n, alu, b, x, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (complex matrix, two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of arrays a and alu
3	n	I	1	Input	Order of matrix A
4	alu	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LU decomposition (See Note (a))
5	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Approximate number of digits to which solution was improved (See Note (d))
				Output	Approximate number of digits to which solution was improved. (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	ipvt	I*	n	Input	Pivoting information (See Note (a))
10	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) Notes

(a) This function improves the solution obtained by the 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ or 2.4.5 $\left\{\begin{matrix}\text{ASL\_zbgnls}\\ \text{ASL\_cbgnls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after being decomposed by 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$, 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, or 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$ function and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\mathrm{itol} \leq 0$$

or

$$\mathrm{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

2.5 POSITIVE SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)

2.5.1 ASL_dbpdsl, ASL_rbpdsl
Simultaneous Linear Equations (Positive Symmetric Matrix)

(1) **Function**

ASL_dbpdsl or ASL_rbpdsl uses the Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbpdsl (a, lna, n, b);
Single precision:
ierr = ASL_rbpdsl (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{ \begin{array}{l} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{ \begin{array}{l} \text{D}* \\ \text{R}* \end{array} \right\}$	lna×n	Input	Coefficient matrix A (positive symmetric matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{ \begin{array}{l} \text{D}* \\ \text{R}* \end{array} \right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	a[0] ← $\sqrt{\text{a[0]}}$ and b[0] ← b[0]/a[0]
2100	There existed the diagonal element which was close to zero in the LL^T decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

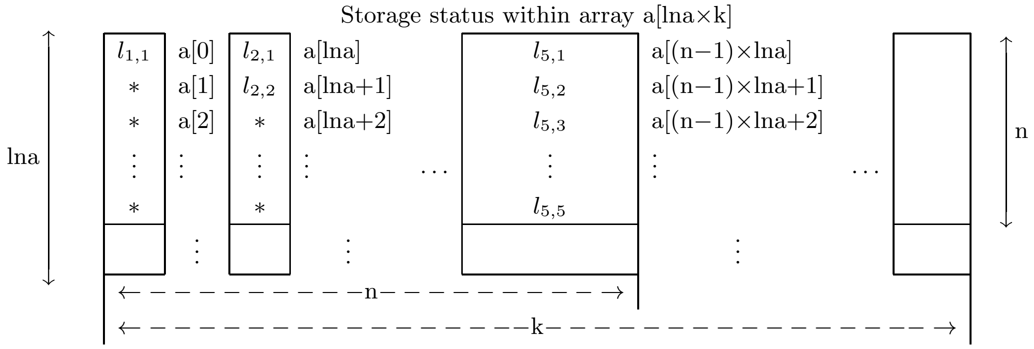
(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function 2.5.4 $\left\{\begin{array}{c}\text{ASL\_dbpdls}\\ \text{ASL\_rbpdls}\end{array}\right\}$ required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the $\mathrm{LL^T}$ decomposition of matrix A only once.

(b) The upper triangular matrix L^T is stored in the upper triangular portion of array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a. this function uses only the upper triangular portion of array a.

Matrix L^T

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

$$\Downarrow$$



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−5 Storage status of Matrix L^T

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$, and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbpdsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbpdsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbpdsl ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
        fscanf( fp, "%d", &na );
        fscanf( fp, "%d", &n );

        a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
        if( a == NULL )
        {
            printf( "no enough memory for array a\n" );
            return -1;
        }
        b = ( double * )malloc((size_t)( sizeof(double) * n ));
        if( b == NULL )
        {
            printf( "no enough memory for array b\n" );
            return -1;
        }

        printf( "\t n = %6d\n\n", n );
        printf( "\tCoefficient Matrix \n\n");
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<n ; j++ )
            {
                fscanf( fp, "%lf", &a[i+na*j] );
                printf( "%8.3g  ", a[i+na*j] );
            }
            printf( "\n" );
        }

        printf( "\n\tConstant Vector\n\n");
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "\t%8.3g\n", b[i] );
        }

        fclose( fp );

        ierr = ASL_dbpdsl(a, na, n, b);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution\n\n" );
        for( i=0 ; i<n ; i++ )
        {
                printf( "\t  x[%6d] = %8.3g\n",i, b[i] );
        }


        free( a );
        free( b );

        return 0;
    }
```

(d) Output results

```
        *** ASL_dbpdsl ***

         ** Input **

         n =      4

        Coefficient Matrix

               5         7         6         5
               7        10         8         7
               6         8        10         9
               5         7         9        10

        Constant Vector

              23
              32
              33
              31

         ** Output **

        ierr =      0

        Solution

          x[     0] =        1
          x[     1] =        1
          x[     2] =        1
          x[     3] =        1
```

2.5.2 ASL_dbpduu, ASL_rbpduu
LL^{T} Decomposition of a Positive Symmetric Matrix

(1) **Function**

ASL_dbpduu or ASL_rbpduu uses the Cholesky method to perform an LL^T decomposition of the positive symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
  ierr = ASL_dbpduu (a, lna, n);
Single precision:
  ierr = ASL_rbpduu (a, lna, n);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Positive symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	a[0] ← $\sqrt{\mathrm{a}[0]}$ is performed.
2100	There existed the diagonal element which was close to zero in the LL^T decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step. A is nearly singular.	

(6) Notes

(a) The upper triangular matrix L^T is stored in the upper triangular portion of array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.5.1, Figure 2−5)

2.5.3 ASL_dbpduc, ASL_rbpduc
$\mathrm{LL^T}$ Decomposition and Condition Number of a Positive Symmetric Matrix

(1) **Function**

ASL_dbpduc or ASL_rbpduc uses the Cholesky method to perform an LL^T decomposition and obtain the condition number of the positive symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
ierr = ASL_dbpduc (a, lna, n, &cond, w1);
Single precision:
ierr = ASL_rbpduc (a, lna, n, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Positive symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	cond	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	1	Output	Reciprocal of the condition number
5	w1	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	a[0] ← $\sqrt{\text{a[0]}}$ and cond ← 1.0 are performed.
2100	There existed the diagonal element which was close to zero in the LL^T decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step. A is nearly singular.	Processing is aborted. The condition number is not obtained.

(6) Notes

(a) The upper triangular matrix L^T is stored in the upper triangular portion of array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.5.1, Figure 2−5).

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

2.5.4 ASL_dbpdls, ASL_rbpdls Simultaneous Linear Equations (LL$^{\mathrm{T}}$-Decomposed Positive Symmetric Matrix)

(1) Function

ASL_dbpdls or ASL_rbpdls solves the simultaneous linear equations $LL^T\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric matrix A (two-dimensional array type) (upper triangular type) which has been LL^T decomposed by the Cholesky method as coefficient matrix.

(2) Usage

Double precision:
  ierr = ASL_dbpdls (a, lna, n, b);
Single precision:
  ierr = ASL_rbpdls (a, lna, n, b);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LL^T decomposition (positive symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0]^2 is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LL^T decomposed before using this function. Normally, you should decompose matrix A by calling the 2.5.2 $\left\{\begin{array}{c}\text{ASL\_dbpduu}\\ \text{ASL\_rbpduu}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.5.3 $\left\{\begin{array}{c}\text{ASL\_dbpduc}\\ \text{ASL\_rbpduc}\end{array}\right\}$. In addition, if you have already used 2.5.1 $\left\{\begin{array}{c}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LL^T decomposition obtained as part of its output.

(b) The upper triangular matrix L^T must be stored in the upper triangular portion of array a. Since the lower triangular matrix L is calculated from L^T, it need not be stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.5.1, Figure 2−5).

2.5.5 ASL_dbpddi, ASL_rbpddi Determinant and Inverse Matrix of a Positive Symmetric Matrix

(1) Function

ASL_dbpddi or ASL_rbpddi obtains the determinant and inverse matrix of the positive symmetric matrix A (two-dimensional array type) (upper triangular type) which has been LL^T decomposed by the Cholesky method.

(2) Usage

Double precision:
ierr = ASL_dbpddi (a, lna, n, det, isw);
Single precision:
ierr = ASL_rbpddi (a, lna, n, det, isw);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	lna×n	Input	Positive symmetric matrix A (two-dimensional array type) (upper triangular type) after LL^T decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	det	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
5	isw	I	1	Input	Processing switch isw $>$ 0: Obtain determinant. isw $=$ 0: Obtain determinant and inverse matrix. isw $<$ 0: Obtain inverse matrix.
6	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \le \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	$\mathrm{det}[0] \leftarrow \mathrm{a}[0]^2$ $\mathrm{det}[1] \leftarrow 1.0$ $\mathrm{a}[0] \leftarrow 1.0/\mathrm{a}[0]^2$
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be LL^T decomposed before using this function. Use any of the 2.5.2 $\left\{\begin{array}{l}\text{ASL\_dbpduu}\\ \text{ASL\_rbpduu}\end{array}\right\}$, 2.5.3 $\left\{\begin{array}{l}\text{ASL\_dbpduc}\\ \text{ASL\_rbpduc}\end{array}\right\}$, 2.5.1 $\left\{\begin{array}{l}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{array}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^T must be stored in the upper triangular portion of array a. Since the lower triangular matrix L is calculated from L^T, it need not be stored in array a. Since the inverse matrix A^{-1} is a symmetric matrix, only its upper triangular portion is stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.5.1, Figure 2−5).

(c) The determinant is given by the following expression:

$$\mathrm{det}(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$$

Scaling is performed at this time so that:

$$1.0 \leq |\mathrm{det}[0]| < 10.0$$

(d) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.5.6 ASL_dbpdlx, ASL_rbpdlx Improving the Solution of Simultaneous Linear Equations (Positive Symmetric Matrix)

(1) **Function**

ASL_dbpdlx or ASL_rbpdlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_dbpdlx (a, lna, n, all, b, x, &itol, nit, w1);
Single precision:
  ierr = ASL_rbpdlx (a, lna, n, all, b, x, &itol, nit, w1);

(3) **Arguments and Return Value**

D:Double precision real   Z:Double precision complex   I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (positive symmetric matrix, two-dimensional array type, upper triangular type)
2	lna	I	1	Input	Adjustable dimension of array a and all
3	n	I	1	Input	Order of matrix A
4	all	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LL^T decomposition (See Note (a))
5	b	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved. (See Note (b))
				Output	Approximate number of digits to which solution was improved. (See Note (c))
8	nit	I	1	Input	Maximum number of iterations. (See Note (d))
9	w1	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
10	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating itol output value.
6000	The solution could not be improved.	

(6) Notes

(a) This function improves the solution obtained by the 2.5.1 $\left\{\begin{matrix}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{matrix}\right\}$ or 2.5.4 $\left\{\begin{matrix}\text{ASL\_dbpdls}\\ \text{ASL\_rbpdls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed 2.5.1 $\left\{\begin{matrix}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{matrix}\right\}$, 2.5.2 $\left\{\begin{matrix}\text{ASL\_dbpduu}\\ \text{ASL\_rbpduu}\end{matrix}\right\}$, or 2.5.3 $\left\{\begin{matrix}\text{ASL\_dbpduc}\\ \text{ASL\_rbpduc}\end{matrix}\right\}$ function must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}\ (2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.6.1 ASL_dbspsl, ASL_rbspsl

Simultaneous Linear Equations (Real Symmetric Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbspsl (a, lna, n, b, ipvt, wk);
Single precision:
ierr = ASL_rbspsl (a, lna, n, b, ipvt, wk);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	wk	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Work	Work Area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
2100	There existed the diagonal element which was close to zero in the *LU* decomposition of the coefficient matrix *A*. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became equal to 0.0 in the i-th processing step of the LDL^T decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

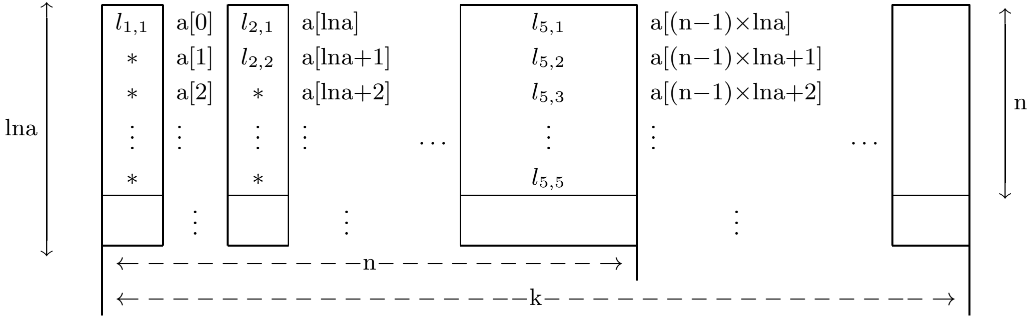
(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this function only once and then call function 2.6.4 $\left\{\begin{array}{c}\text{ASL\_dbspls}\\ \text{ASL\_rbspls}\end{array}\right\}$ you to eliminate unnecessary calculations by performing the $\mathrm{LDL}^{\mathrm{T}}$ decomposition of matrix A only once.

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

$$\text{Matrix } L^T \quad \begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix} \qquad \text{Matrix } D \quad \begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix}$$

$$\Downarrow$$

Storage status within array a[lna×k]



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−6 Storage Status of Matrix L^T and Contents of Matrix D

(c) This function performs partial pivoting when obtaining the LDL$^{\mathrm{T}}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbspsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int *ipvt;
    double *wk;
    int ierr;
```

```
    int i,j;

    FILE *fp;

    fp = fopen( "dbspsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbspsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+na*j] );
            printf( "%8.3g  ", a[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }


    fclose( fp );

    ierr = ASL_dbspsl(a, na, n, b, ipvt, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );


    printf( "\tSolution \n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i, b[i] );
    }

    free( a );
    free( b );
    free( ipvt );
    free( wk );


    return 0;
}
```

(d) Output results

```
*** ASL_dbspsl ***

 ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

 ** Output **

ierr =      0

Solution

  x[     0] =        1
  x[     1] =       -1
  x[     2] =        2
  x[     3] =       -2
```

2.6.2 ASL_dbspud, ASL_rbspud
LDL$^{\mathrm{T}}$ Decomposition of a Real Symmetric Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbspud (a, lna, n, ipvt, wk);
Single precision:
ierr = ASL_rbspud (a, lna, n, ipvt, wk);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{ \begin{array}{l} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{ \begin{array}{c} D* \\ R* \end{array} \right\}$	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	wk	$\left\{ \begin{array}{c} D* \\ R* \end{array} \right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.6.1, Figure 2−6.)

(b) This function performs partial pivoting when obtaining the LDL$^{\mathrm{T}}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

2.6.3 ASL_dbspuc, ASL_rbspuc
LDL^T Decomposition and Condition Number of a Real Symmetric Matrix

(1) **Function**

ASL_dbspuc or ASL_rbspuc uses the modified Cholesky method to perform an LDL^T decomposition and obtain the condition number of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
ierr = ASL_dbspuc (a, lna, n, ipvt, &cond, wk);
Single precision:
ierr = ASL_rbspuc (a, lna, n, ipvt, &cond, wk);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: { int as for 32bit Integer }
R:Single precision real	C:Single precision complex	{ long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	cond	{D* R*}	1	Output	Reciprocal of the condition number
6	wk	{D* R*}	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.6.1, Figure 2−6.)

(b) This function performs partial pivoting when obtaining the LDL$^{\mathrm{T}}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

2.6.4 ASL_dbspls, ASL_rbspls Simultaneous Linear Equations (LDL^T-Decomposed Real Symmetric Matrix)

(1) Function

ASL_dbspls or ASL_rbspls solves the simultaneous linear equations having the real symmetric matrix A (two-dimensional array type) which has been LDL^T decomposed by the modified Cholesky method as coefficient matrix.

(2) Usage

Double precision:
  ierr = ASL_dbspls (a, lna, n, b, ipvt);
Single precision:
  ierr = ASL_rbspls (a, lna, n, b, ipvt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lna×n	Input	Coefficient matrix A after LDL^T decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension af array a
3	n	I	1	Input	Order of matrix A
4	b	{D* R*}	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be $\mathrm{LDL^T}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.6.2 $\left\{\begin{array}{c}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.6.3 $\left\{\begin{array}{c}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{array}\right\}$ function. In addition, if you have already used 2.6.1 $\left\{\begin{array}{c}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the $\mathrm{LDL^T}$ decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.6.5 $\left\{\begin{array}{c}\text{ASL\_dbspms}\\ \text{ASL\_rbspms}\end{array}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^T must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they need not be stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.6.1, Figure 2−6.)

(c) This function performs partial pivoting when obtaining the $\mathrm{LDL^T}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

2.6.5 ASL_dbspms, ASL_rbspms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL$^{\mathrm{T}}$ decomposed Real Matrix)

(1) **Function**

ASL_dbspms or ASL_rbspms solves the simultaneous linear equations $LDL^T\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) which has been LDL$^{\mathrm{T}}$ decomposed by the Gauss method or the Crout method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
  ierr = ASL_dbspms (a, lna, n, b, lnb, m, ipvt);
Single precision:
  ierr = ASL_rbspms (a, lna, n, b, lnb, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL$^{\mathrm{T}}$ decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Matrix consisting of constant vector $\boldsymbol{b_i}$ $[A', \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Matrix consisting of Solution vector $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Number of right-hand side vectors, m
7	ipvt	I*	n	Input	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(c) $0 < \mathrm{ipvt[i-1]} \leq \mathrm{n} \quad (\mathrm{i} = 1, \ldots, \mathrm{n})$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n is equal to 1	$\mathrm{b[lna*(i-1)]} \leftarrow \mathrm{b[lna*(i-1)]/a[0]}$ $(i = 1, 2, \cdots, \mathrm{m})$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) The coefficient matrix A must be $\mathrm{LDL^T}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.6.2 $\left\{\begin{array}{c}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.6.3 $\left\{\begin{array}{c}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{array}\right\}$.
In addition, if you have already used 2.6.1 $\left\{\begin{array}{c}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the $\mathrm{LDL^T}$ decomposition obtained as part of its output.

(b) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.6.1, Figure 2−6.)

(c) Information about partial pivoting performed during $\mathrm{LDL^T}$ decomposition must be stored in ipvt. This information is given by the 2.6.2 $\left\{\begin{array}{c}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{array}\right\}$, 2.6.3 $\left\{\begin{array}{c}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{array}\right\}$, and 2.6.1 $\left\{\begin{array}{c}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{array}\right\}$ functions which perform $\mathrm{LDL^T}$ decomposition of matrix A.

(7) **Example**

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13 \end{bmatrix}$$

(b) Input data
Coefficient matrix a, $\mathrm{lna} = 10, \mathrm{n} = 4$, matrix consisting of constant vector B, lnb=B and m=2.

(c) Main program

```
/* C interface example for ASL_dbspms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int lna=11;
    int n=4;
    double *b;
    int lnb=11;
    int m=2;
    double *wk;
    int *ipvt;
    int ierr_ud,ierr_ms;
```

```
int i,j;
FILE *fp;

fp = fopen( "dbspms.dat", "r" );

if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_dbspms ***\n" );
printf( "\n     ** Input **\n\n" );

a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
if( a == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}

b = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}

ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
if( ipvt == NULL )
{
    printf( "no enough memory for array ipvt\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (n) ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", m );

printf( "\n\tCoefficient Matrix a\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+lna*j] );
        printf( "%8.3g", a[i+lna*j] );
    }
    printf( "\n" );
}

ierr_ud = ASL_dbspud(a, lna, n, ipvt, wk);

if( ierr_ud != 0 ) {
    printf( "\tierr ( ASL_dbspud ) = %6d\n", ierr_ud );
    return 0;
}

printf( "\n\tConstant Vectors b\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        fscanf( fp, "%lf", &b[i+lnb*j] );
        printf( "%8.3g", b[i+lnb*j] );
    }
    printf( "\n" );
}

fclose( fp );

ierr_ms = ASL_dbspms(a, lna, n, b, lnb, m, ipvt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr ( ASL_dbspud ) = %6d\n\n", ierr_ud );
printf( "\tierr ( ASL_dbspms ) = %6d\n",   ierr_ms );

printf( "\n\tSolution\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
```

```
                printf( "%8.3g", b[i+lnb*j] );
            }
            printf( "\n" );
        }

        free(    a );
        free(    b );
        free(   wk );
        free( ipvt );

        return 0;
    }
```

(d) Output results

```
*** ASL_dbspms ***

 ** Input **

n =      4
m =      2

Coefficient Matrix a

       5       4       1       1
       4       5       1       1
       1       1       4       2
       1       1       2       4

Constant Vectors b

       1      -2
      -1       1
       4       9
      -4      13

 ** Output **

ierr ( ASL_dbspud ) =      0

ierr ( ASL_dbspms ) =      0

Solution

       1      -2
      -1       1
       2       1
      -2       3
```

2.6.6 ASL_dbspdi, ASL_rbspdi Determinant and Inverse Matrix of a Real Symmetric Matrix

(1) Function

ASL_dbspdi or ASL_rbspdi obtains the determinant and inverse matrix of the real symmetric matrix A (two-dimensional array type) (upper triangular type) which has been LDL^T decomposed by the modified Cholesky method.

(2) Usage

Double precision:
ierr = ASL_dbspdi (a, lna, n, ipvt, det, isw, wk);
Single precision:
ierr = ASL_rbspdi (a, lna, n, ipvt, det, isw, wk);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type) after LDL^T decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A (See Note (b))
2	lna	I	1	Input	Adjustable dimensional pf array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
5	det	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
6	isw	I	1	Input	Processing switch isw > 0: Obtain determinant. isw = 0: Obtain determinant and inverse matrix. isw < 0: Obtain inverse matrix.
7	wk	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0], det[1] ← 0.0 a[0] ← 1.0/a[0] are performed. (See Note (c))
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be $\mathrm{LDL^T}$ decomposed before using this function. Use any of the 2.6.2 $\left\{\begin{array}{l}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{array}\right\}$, 2.6.3 $\left\{\begin{array}{l}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{array}\right\}$, 2.6.1 $\left\{\begin{array}{l}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{array}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^T must be stored in array a at input time. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they need not be stored in array a. Since the inverse matrix A^{-1} is a symmetric matrix, only its upper triangular portion is stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.6.1, Figure 2−6.)

(c) This function performs partial pivoting when obtaining the $\mathrm{LDL^T}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(d) The determinant is given by the following expression:

$$\det(A) = \det[0] \times 10^{\det[1]}$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.6.7 ASL_dbsplx, ASL_rbsplx Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix)

(1) **Function**

ASL_dbsplx or ASL_rbsplx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real symmetric Matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_dbsplx (a, lna, n, ald, b, x, &itol, nit, ipvt, wk);
Single precision:
  ierr = ASL_rbsplx (a, lna, n, ald, b, x, &itol, nit, ipvt, wk);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, upper triangular type)
2	lna	I	1	Input	Adjustable dimension of array a and ald
3	n	I	1	Input	Order of matrix A
4	ald	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL^T decomposition (See Note (a))
5	b	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	ipvt	I*	n	Output	Pivoting information. (See Note (a))
10	wk	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	n	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculation the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.6.1 $\left\{\begin{matrix}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{matrix}\right\}$ or 2.6.4 $\left\{\begin{matrix}\text{ASL\_dbspls}\\ \text{ASL\_rbspls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.6.1 $\left\{\begin{matrix}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{matrix}\right\}$, 2.6.2 $\left\{\begin{matrix}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{matrix}\right\}$, or 2.6.3 $\left\{\begin{matrix}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{matrix}\right\}$ function and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

2.7 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)(NO PIVOTING)

2.7.1 ASL_dbsmsl, ASL_rbsmsl
Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbsmsl (a, lna, n, b, w1);
Single precision:
ierr = ASL_rbsmsl (a, lna, n, b, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{ \begin{array}{l} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c} D* \\ R* \end{array}\right\}$	lna×n	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c} D* \\ R* \end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	w1	$\left\{\begin{array}{c} D* \\ R* \end{array}\right\}$	n	Work	Work Area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

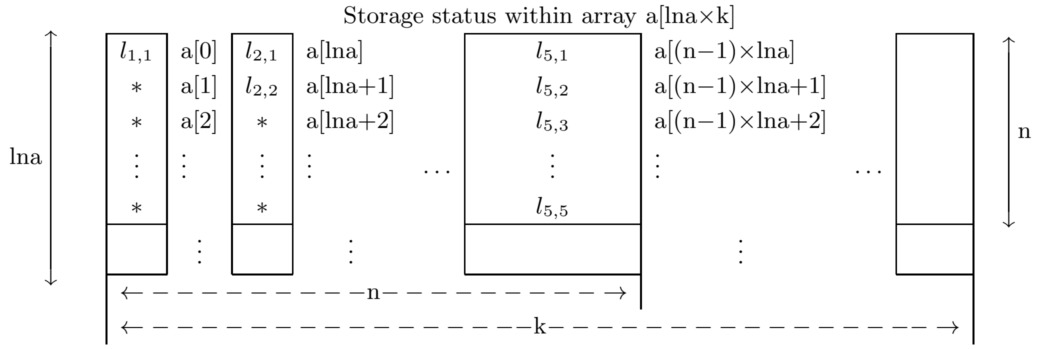
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this function only once and then call function 2.7.4 $\left\{\begin{matrix}\text{ASL\_dbsmls}\\ \text{ASL\_rbsmls}\end{matrix}\right\}$ you to eliminate unnecessary calculations by performing the LDL$^{\mathrm{T}}$ decomposition of matrix A only once.

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

$$\begin{array}{cc}
\text{Matrix } L^T & \text{Matrix } D \\
\begin{bmatrix}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{bmatrix}
&
\begin{bmatrix}
1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5}
\end{bmatrix}
\end{array}$$

$$\Downarrow$$



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−7 Storage Status of Matrix L^T and Contents of Matrix D

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbsmsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    double *wk;
    int ierr;

    int i,j;

    FILE *fp;

    fp = fopen( "dbsmsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbsmsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+na*j] );
            printf( "%8.3g  ", a[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
```

```
            fscanf( fp, "%lf", &b[i] );
            printf( "\t%8.3g\n", b[i] );
        }

        fclose( fp );

        ierr = ASL_dbsmsl(a, na, n, b, wk);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n\n", ierr );

        printf( "\tSolution \n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i, b[i] );
        }

        free( a );
        free( b );
        free( wk );

        return 0;
    }
```

(d) Output results

```
*** ASL_dbsmsl ***

 ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

 ** Output **

ierr =      0

Solution

  x[     0] =        1
  x[     1] =       -1
  x[     2] =        2
  x[     3] =       -2
```

2.7.2 ASL_dbsmud, ASL_rbsmud
$\mathrm{LDL^T}$ Decomposition of a Real Symmetric Matrix (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbsmud (a, lna, n, w1);
Single precision:
ierr = ASL_rbsmud (a, lna, n, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{D*}\\ \text{R*}\end{array}\right\}$	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	w1	$\left\{\begin{array}{l}\text{D*}\\ \text{R*}\end{array}\right\}$	n	Work	Work area
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.7.1, Figure 2−7.)

2.7.3 ASL_dbsmuc, ASL_rbsmuc
LDL$^{\mathrm{T}}$ Decomposition and Condition Number of a Real Symmetric Matrix (No Pivoting)

(1) **Function**

ASL_dbsmuc or ASL_rbsmuc uses the modified Cholesky method to perform an LDL$^{\mathrm{T}}$ decomposition and obtain the condition number of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
ierr = ASL_dbsmuc (a, lna, n, &cond, w1);
Single precision:
ierr = ASL_rbsmuc (a, lna, n, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	cond	{D* / R*}	1	Output	Reciprocal of the condition number
5	w1	{D* / R*}	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array a are not changed. cond ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LDL$^{\mathrm{T}}$ decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	Processing is aborted. The condition number is not obtained.

(6) Notes

(a) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.7.1, Figure 2−7.)

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

2.7.4 ASL_dbsmls, ASL_rbsmls Simultaneous Linear Equations (LDL^T-Decomposed Real Symmetric Matrix) (No Pivoting)

(1) **Function**

ASL_dbsmls or ASL_rbsmls solves the simultaneous linear equations having the real symmetric matrix A (two-dimensional array type) which has been LDL^T decomposed by the modified Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_dbsmls (a, lna, n, b);
Single precision:
  ierr = ASL_rbsmls (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer; long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lna×n	Input	Coefficient matrix A after LDL^T decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension af array a
3	n	I	1	Input	Order of matrix A
4	b	{D* R*}	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL$^{\mathrm{T}}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.7.2 $\left\{\begin{array}{l}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.7.3 $\left\{\begin{array}{l}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{array}\right\}$ function. In addition, if you have already used 2.7.1 $\left\{\begin{array}{l}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL$^{\mathrm{T}}$ decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.7.5 $\left\{\begin{array}{l}\text{ASL\_dbsmms}\\ \text{ASL\_rbsmms}\end{array}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^T must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they need not be stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.7.1, Figure 2−7.)

2.7.5 ASL_dbsmms, ASL_rbsmms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL^T-Decomposed Real Matrix) (No Pivoting)

(1) **Function**

ASL_dbsmms or ASL_rbsmms solves the simultaneous linear equations $LDL^T\boldsymbol{x} = \boldsymbol{b}$ having the real matrix A (two-dimensional array type) which has been LDL$^{\mathrm{T}}$ decomposed by the Gauss method or the Crout method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
ierr = ASL_dbsmms (a, lna, n, b, lnb, m);
Single precision:
ierr = ASL_rbsmms (a, lna, n, b, lnb, m);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL$^{\mathrm{T}}$ decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Matrix consisting of constant vector $\boldsymbol{b_i}$ $[A', \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Matrix consisting of Solution vector $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Number of right-hand side vectors, m
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \le \mathrm{lna}$

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

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n is equal to 1	b[lna ∗ (i − 1)] ← b[lna ∗ (i − 1)]/a[0] $(i = 1, 2, \cdots, \mathrm{m})$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) Notes

(a) The coefficient matrix A must be $\mathrm{LDL^T}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.7.2 $\left\{\begin{array}{c}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.7.3 $\left\{\begin{array}{c}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{array}\right\}$.
In addition, if you have already used 2.7.1 $\left\{\begin{array}{c}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the $\mathrm{LDL^T}$ decomposition obtained as part of its output.

(b) The upper triangular matrix L^T is stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they are not stored in array a. (See Section 2.7.1, Figure 2−7.)

(7) Example

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13 \end{bmatrix}$$

(b) Input data
Coefficient matrix a, $\mathrm{lna} = 10, \mathrm{n} = 4$, matrix consisting of constant vector B, lnb=B and m=2.

(c) Main program

```
/* C interface example for ASL_dbsmms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int lna=11;
    int n=4;
    double *b;
    int lnb=11;
    int m=2;
    double *wk;
    int ierr_ud,ierr_ms;
    int i,j;
    FILE *fp;

    fp = fopen( "dbsmms.dat", "r" );

    if( fp == NULL )
    {
```

```
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbsmms ***\n" );
    printf( "\n     ** Input **\n\n" );

    a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (n) ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", m );

    printf( "\n\tCoefficient Matrix a\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+lna*j] );
            printf( "%8.3g", a[i+lna*j] );
        }
        printf( "\n" );
    }

    ierr_ud = ASL_dbsmud(a, lna, n, wk);

    if( ierr_ud != 0 ) {
        printf( "\tierr ( ASL_dbsmud )= %6d\n", ierr_ud );
        return 0;
    }

    printf( "\n\tConstant Vectors b\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            fscanf( fp, "%lf", &b[i+lnb*j] );
            printf( "%8.3g", b[i+lnb*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr_ms = ASL_dbsmms(a, lna, n, b, lnb, m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr ( ASL_dbsmud )= %6d\n\n", ierr_ud );
    printf( "\tierr ( ASL_dbsmms )= %6d\n",   ierr_ms );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "%8.3g", b[i+lnb*j] );
        }
        printf( "\n" );
    }

    free(  a );
    free(  b );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_dbsmms ***

 ** Input **

n =      4
m =      2

Coefficient Matrix a

       5       4       1       1
       4       5       1       1
       1       1       4       2
       1       1       2       4

Constant Vectors b

       1      -2
      -1       1
       4       9
      -4      13

 ** Output **

ierr ( ASL_dbsmud )=      0

ierr ( ASL_dbsmms )=      0

Solution

       1      -2
      -1       1
       2       1
      -2       3
```

2.7.6 ASL_dbsmdi, ASL_rbsmdi
Determinant and Inverse Matrix of a Real Symmetric Matrix (No Pivoting)

(1) **Function**

ASL_dbsmdi or ASL_rbsmdi obtains the determinant and inverse matrix of the real symmetric matrix A (two-dimensional array type) (upper triangular type) which has been $\mathrm{LDL}^{\mathrm{T}}$ decomposed by the modified Cholesky method.

(2) **Usage**

Double precision:
ierr = ASL_dbsmdi (a, lna, n, det, isw, w1);
Single precision:
ierr = ASL_rbsmdi (a, lna, n, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type) after $\mathrm{LDL}^{\mathrm{T}}$ decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A (See Note (b))
2	lna	I	1	Input	Adjustable dimensional pf array a
3	n	I	1	Input	Order of matrix A
4	det	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
5	isw	I	1	Input	Processing switch isw $>$ 0: Obtain determinant. isw $=$ 0: Obtain determinant and inverse matrix. isw $<$ 0: Obtain inverse matrix.
6	w1	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0], det[1] ← 0.0 a[0] ← 1.0/a[0] are performed. (See Note (c))
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be LDL$^{\mathrm{T}}$ decomposed before using this function. Use any of the 2.7.2 $\left\{\begin{array}{c}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{array}\right\}$, 2.7.3 $\left\{\begin{array}{c}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{array}\right\}$, 2.7.1 $\left\{\begin{array}{c}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{array}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^T must be stored in array a at input time. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^T, they need not be stored in array a. Since the inverse matrix A^{-1} is a symmetric matrix, only its upper triangular portion is stored in array a. This function uses only the upper triangular portion of array a. (See Section 2.7.1, Figure 2−7.)

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times 10^{\det[1]}$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.7.7 ASL_dbsmlx, ASL_rbsmlx
Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

(1) Function

ASL_dbsmlx or ASL_rbsmlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real symmetric Matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_dbsmlx (a, lna, n, ald, b, x, &itol, nit, w1);
Single precision:
ierr = ASL_rbsmlx (a, lna, n, ald, b, x, &itol, nit, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, upper triangular type)
2	lna	I	1	Input	Adjustable dimension of array a and ald
3	n	I	1	Input	Order of matrix A
4	ald	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL^{T} decomposition (See Note (a))
5	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	w1	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Work	Work area
10	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculation the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.7.1 $\left\{\begin{array}{l}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{array}\right\}$ or 2.7.4 $\left\{\begin{array}{l}\text{ASL\_dbsmls}\\ \text{ASL\_rbsmls}\end{array}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.7.1 $\left\{\begin{array}{l}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{array}\right\}$, 2.7.2 $\left\{\begin{array}{l}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{array}\right\}$, or 2.7.3 $\left\{\begin{array}{l}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{array}\right\}$ function must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.8.1 ASL_dbsnsl, ASL_rbsnsl

Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_dbsnsl (a, lna, n, b);
Single precision:
ierr = ASL_rbsnsl (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	lna×n	Input	Coefficient matrix A (real symmetric matrix, two-dimensional array type, lower triangular type)
				Output	lower triangular matrix U^T when A is decomposed into $A = U^TDU$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
2100	There existed the diagonal element which was close to zero in the $\mathrm{U^TDU}$ decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became equal to 0.0 in the i-th processing step of the $\mathrm{U^TDU}$ decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

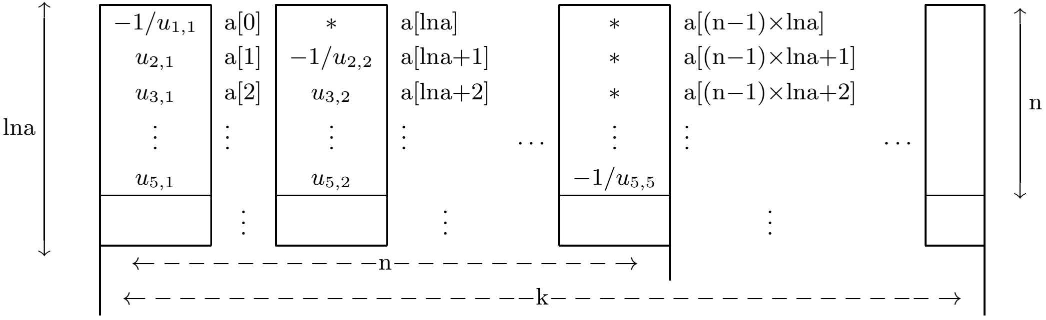
(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this function only once and then call function 2.8.3 $\left\{\begin{matrix}\text{ASL\_dbsnls}\\ \text{ASL\_rbsnls}\end{matrix}\right\}$ you to eliminate unnecessary calculations by performing the $\mathrm{U^TDU}$ decomposition of matrix A only once.

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

$$\text{Matrix } U^T \quad \begin{bmatrix} u_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ u_{2,1} & u_{2,2} & 0.0 & \cdots & 0.0 \\ u_{3,1} & u_{3,2} & u_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{5,1} & u_{5,2} & u_{5,3} & \cdots & u_{5,5} \end{bmatrix} \qquad \text{Matrix } D \quad \begin{bmatrix} 1/u_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/u_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/u_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/u_{5,5} \end{bmatrix}$$

$$\Downarrow$$

Storage status within array a[lna×k]



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−8 Storage Status of Matrix U^T and Contents of Matrix D

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$, and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*    C interface example for ASL_dbsnsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int ierr;

    int i,j;

    FILE *fp;

    fp = fopen( "dbsnsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "       *** ASL_dbsnsl ***\n" );
    printf( "\n      ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+na*j] );
            printf( "%8.3g  ", a[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }


    fclose( fp );

    ierr = ASL_dbsnsl(a, na, n, b);
```

```
        printf( "\n      ** Output **\n\n" );
        printf( "\tierr = %6d\n\n", ierr );

        printf( "\tSolution \n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i, b[i] );
        }

        free( a );
        free( b );

        return 0;
    }
```

(d) Output results

```
     *** ASL_dbsnsl ***

    ** Input **

  n =      4

 Coefficient Matrix

        5         4         1         1
        4         5         1         1
        1         1         4         2
        1         1         2         4

 Constant Vector

        1
       -1
        4
       -4

    ** Output **

 ierr =      0

 Solution

   x[     0] =        1
   x[     1] =       -1
   x[     2] =        2
   x[     3] =       -2
```

2.8.2 ASL_dbsnud, ASL_rbsnud
$\mathrm{U^T DU}$ Decomposition of a Real Symmetric Matrix (No Pivoting)

(1) Function

ASL_dbsnud or ASL_rbsnud uses the modified Cholesky method to perform an $\mathrm{U^T DU}$ decomposition of the real symmetric matrix A (two-dimensional array type) (lower triangular type).

(2) Usage

Double precision:
ierr = ASL_dbsnud (a, lna, n);
Single precision:
ierr = ASL_rbsnud (a, lna, n);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\mathrm{D}*\\ \mathrm{R}*\end{array}\right\}$	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (lower triangular type)
				Output	Lower triangular matrix U^T when A is decomposed into $A = U^T DU$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The lower triangular matrix U^T is stored in array a. For the diagonal components of U^T, their reciprocals are stored in array a with the sign changed. Since the diagonal matrix D and the upper triangular matrix U are calculated from U^T, they are not stored in array a. (See Section 2.8.1, Figure 2−8.)

2.8.3 ASL_dbsnls, ASL_rbsnls Simultaneous Linear Equations ($\mathrm{U^TDU}$-Decomposed Real Symmetric Matrix) (No Pivoting)

(1) **Function**

ASL_dbsnls or ASL_rbsnls solves the simultaneous linear equations having the real symmetric matrix A (two-dimensional array type, lower triangular type) which has been $\mathrm{U^TDU}$ decomposed by the modified Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbsnls (a, lna, n, b);
Single precision:
ierr = ASL_rbsnls (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	D* R*	lna×n	Input	Coefficient matrix A after $\mathrm{U^TDU}$ decomposition (real symmetric matrix, two-dimensional array type, lower triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension af array a
3	n	I	1	Input	Order of matrix A
4	b	D* R*	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be $\mathrm{U^TDU}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.8.2 $\left\{\begin{array}{c}\text{ASL\_dbsnud}\\ \text{ASL\_rbsnud}\end{array}\right\}$ function. In addition, if you have already used 2.8.1 $\left\{\begin{array}{c}\text{ASL\_dbsnsl}\\ \text{ASL\_rbsnsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the $\mathrm{U^TDU}$ decomposition obtained as part of its output.

(b) The lower triangular matrix U^T must be stored in array a. Since the diagonal matrix D and the upper triangular matrix U are calculated from U^T, they need not be stored in array a. This function uses only the lower triangular portion of array a. (See Section 2.8.1, Figure 2−8.)

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

2.9.1 ASL_zbhpsl, ASL_cbhpsl
Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhpsl (ar, ai, lna, n, br, bi, ipvt, w1);
Single precision:
ierr = ASL_cbhpsl (ar, ai, lna, n, br, bi, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	ai	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
7	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
8	w1	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

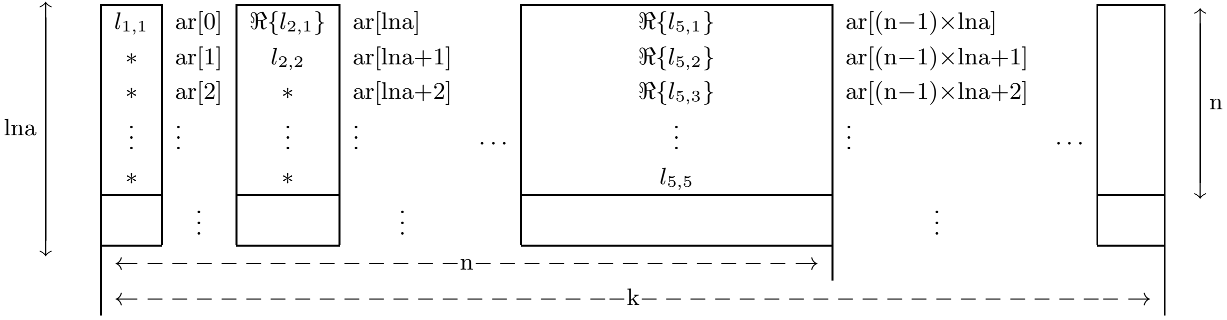
(a) To solve multiple sets of simultaneous linear equations where only the constant vector ***b*** differs, call this function only once and then call function 2.9.4 $\begin{Bmatrix}\text{ASL\_zbhpls}\\ \text{ASL\_cbhpls}\end{Bmatrix}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LDL* decomposition of matrix *A* only once.

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

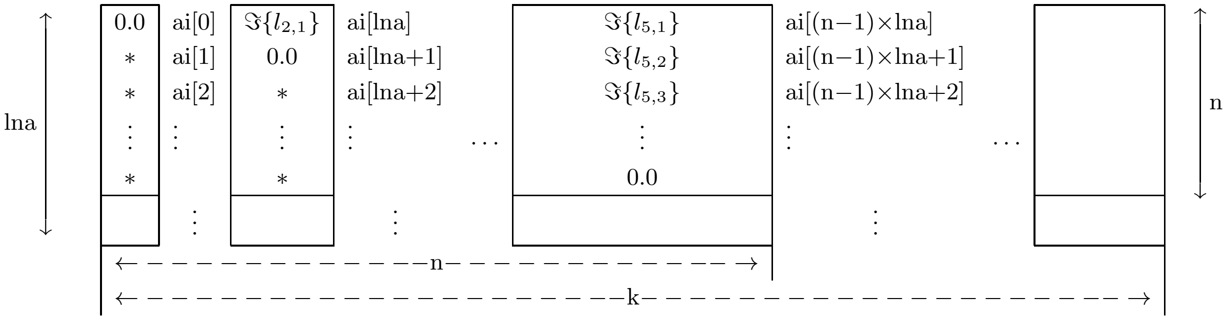
$$
\begin{array}{cc}
\text{Matrix } L^* & \text{Matrix } D \\
\begin{bmatrix}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{bmatrix}
&
\begin{bmatrix}
1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5}
\end{bmatrix}
\end{array}
$$

$$\Downarrow$$

Storage status within array ar[lna×k]



Storage status within array ai[lna×k]



Remarks

a. lna $\geq$ n and n $\leq$ k must hold.

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

Figure 2−9 Storage Status of Matrix L^* and Contents of Matrix D

(c) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix real part ar and Imaginary part ai, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_zbhpsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    int *ipvt;
    double *w1;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbhpsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbhpsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );

    ar = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n" );
        return -1;
    }

    ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n" );
        return -1;
    }

    br = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( br == NULL )
    {
        printf( "no enough memory for array br\n" );
        return -1;
    }

    bi = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

    printf( "\t n = %6d\n\n", n );
    printf( "\tCoefficient Matrix  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ar[i+na*j] );
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ai[i+na*j] );
        }
    }
```

```
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                      " );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", ar[i+na*j],ai[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &br[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &bi[i] );
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
    }

    fclose( fp );

    ierr = ASL_zbhpsl(ar, ai, na, n, br, bi, ipvt, w1);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution   (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = (%8.3g , %8.3g)\n", i, br[i],bi[i] );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( ipvt );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_zbhpsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector  (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 , 8.88e-17)
  x[     2] = (-4.97e-17 ,        1)
  x[     3] = (-4.17e-17 ,        1)
```

2.9.2 ASL_zbhpud, ASL_cbhpud
LDL* Decomposition of a Hermitian Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhpud (ar, ai, lna, n, ipvt, w1);
Single precision:
ierr = ASL_cbhpud (ar, ai, lna, n, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
6	w1	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	2 × n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

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

(6) Notes

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai. (See Sections 2.9.1 Figure 2−9.)

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

2.9.3 ASL_zbhpuc, ASL_cbhpuc
LDL* Decomposition and Condition Number of a Hermitian Matrix

(1) Function

ASL_zbhpuc or ASL_cbhpuc uses the modified Cholesky method to perform an LDL* decomposition and obtain the condition number of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) Usage

Double precision:

ierr = ASL_zbhpuc (ar, ai, lna, n, ipvt, &cond, w1);

Single precision:

ierr = ASL_cbhpuc (ar, ai, lna, n, ipvt, &cond, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
6	cond	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	1	Output	Reciprocal of the condition number
7	w1	$\left\{\begin{array}{c}\text{D}*\\\text{R}*\end{array}\right\}$	$2 \times n$	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of arrays ar and ai are not changed. cond ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	Processing is aborted. The condition number is not obtained.

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai. (See 2.9.1 Figure 2−9.)

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

2.9.4 ASL_zbhpls, ASL_cbhpls
Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix)

(1) **Function**

ASL_zbhpls or ASL_cbhpls solves the simultaneous linear equations $LDL^*\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbhpls (ar, ai, lna, n, br, bi, ipvt);
Single precision:
ierr = ASL_cbhpls (ar, ai, lna, n, br, bi, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	lna×n	Input	Real part of coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	ai	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	lna×n	Input	Imaginary part of coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/ar[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\ \text{ASL\_cbhpud}\end{matrix}\right\}$. However, if you also want to obtain the condition number, you should use 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\ \text{ASL\_cbhpuc}\end{matrix}\right\}$. In addition, if you have already used 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\ \text{ASL\_cbhpsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.9.5 $\left\{\begin{matrix}\text{ASL\_zbhpms}\\ \text{ASL\_cbhpms}\end{matrix}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^* must be stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai. (See 2.9.1 Figure 2−9.)

(c) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions which perform LDL* decomposition of matrix A.

2.9.5 ASL_zbhpms, ASL_cbhpms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Hermitian Matrix)

(1) **Function**

ASL_zbhpms or ASL_cbhpms solves the simultaneous linear equations $LDL^*\boldsymbol{x_i} = \boldsymbol{b_i}$ $(i = 1, 2, \cdots, m)$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
ierr = ASL_zbhpms (ar, ai, lna, n, br, bi, lnb, m, ipvt);
Single precision:
ierr = ASL_cbhpms (ar, ai, lna, n, br, bi, lnb, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* R*}	lna×n	Input	Real part of Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	ai	{D* R*}	lna×n	Input	Imaginary part of Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	br	{D* R*}	lnb×m	Input	Real part of Constant vector $\boldsymbol{b_i}$ $(i = 1, 2, \cdots, m)$
				Output	Real part of Solution $\boldsymbol{x_i}(i = 1, 2, \cdots, m)$
6	bi	{D* R*}	lnb×m	Input	Imaginary part of Constant vector $\boldsymbol{b_i}$ $(i = 1, 2, \cdots, m)$
				Output	Imaginary part of Solution $\boldsymbol{x_i}$ $(i = 1, 2, \cdots, m)$
7	lnb	I	1	Input	Adjustable dimension of array br and bi
8	m	I	1	Input	Number of right-hand side vectors, m
9	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row (column) i in the i-th processing step. (See Note (c))
10	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \le \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	$\mathrm{br}[\mathrm{lnb} * (\mathrm{i}-1)] \leftarrow \mathrm{br}[\mathrm{lnb} * (\mathrm{i}-1)]/\mathrm{ar}[0]$, $\mathrm{bi}[\mathrm{lnb} * (\mathrm{i}-1)] \leftarrow \mathrm{bi}[\mathrm{lnb} * (\mathrm{i}-1)]/\mathrm{ar}[0]$ (i$= 1, 2, \cdots, m$) are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\ \text{ASL\_cbhpud}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\ \text{ASL\_cbhpuc}\end{matrix}\right\}$. In addition, if you have already used 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\ \text{ASL\_cbhpsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. (See Fig. 2−9 in Section 2.9.1)

(c) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions which perform LDL* decomposition of matrix A.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix A which has been LDL* decomposed by the modified Cholesky method, $\mathrm{lna} = 11, \mathrm{n} = 4$, constant vectors $\boldsymbol{b_i} (i = 1, 2, \cdots, m)$, lnb=11 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbhpms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *ai, *br, *bi;
    double *wk;
    int *ipvt;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

    fp = fopen( "zbhpms.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbhpms ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &nb );
    fscanf( fp, "%d", &m );

    ar = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n" );
        return -1;
    }

    ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n" );
        return -1;
    }

    br = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
    if( br == NULL )
    {
        printf( "no enough memory for array br\n" );
        return -1;
    }

    bi = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * (n) ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", n );

    printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        for( j=i ; j<n ; j++ )
        {
            fscanf( fp, "%lf %lf", &ar[i+na*j], &ai[i+na*j] );
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                      " );
```

```
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", ar[i+na*j],ai[i+na*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf %lf", &br[i+nb*j], &bi[i+nb*j]);
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", br[i+nb*j],bi[i+nb*j] );
        }
        printf( "\n" );
    }
    fclose( fp );

    ierr = ASL_zbhpud(ar, ai, na, n, ipvt,wk);
    ierr = ASL_zbhpms(ar, ai, na, n, br, bi, nb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", br[i+nb*j],bi[i+nb*j] );
        }
        printf( "\n" );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( wk );
    return 0;
}
```

(d) Output results

```
*** ASL_zbhpms ***

 ** Input **

n =      4
m =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-4.28e-16 ,         1) ( 5.35e-17 ,         1) (        1 ,         0)
(        1 ,  8.88e-17) (        1 , -1.78e-16) (-1.78e-16 ,         1) (        0 ,         1)
(-4.97e-17 ,         1) (        1 , -1.33e-16) (        1 ,         0) (        0 ,         1)
(-4.17e-17 ,         1) ( 8.34e-17 ,         1) (        1 , -8.34e-17) (        1 , -8.34e-17)
```

2.9.6 ASL_zbhpdi, ASL_cbhpdi Determinant and Inverse Matrix of a Hermitian Matrix

(1) **Function**

ASL_zbhpdi or ASL_cbhpdi obtains the determinant and inverse matrix of the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method.

(2) **Usage**

Double precision:
ierr = ASL_zbhpdi (ar, ai, lna, n, ipvt, det, isw, w1);
Single precision:
ierr = ASL_cbhpdi (ar, ai, lna, n, ipvt, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{ D* R* }	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Real part of the Inverse matrix of matrix A (See Note (b))
2	ai	{ D* R* }	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Imaginary part of the Inverse matrix of matrix A (See Note (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (d))
6	det	{ D* R* }	2	Output	Determinant of matrix A (See Note (c))
7	isw	I	1	Input	Processing switch isw>0:Obtain determinant. isw=0:Obtain determinant and inverse matrix. isw<0:Obtain inverse matrix.

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
8	w1	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 ar[0] ← 1.0/ar[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Use any of the 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\ \text{ASL\_cbhpud}\end{matrix}\right\}$, 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\ \text{ASL\_cbhpuc}\end{matrix}\right\}$, 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\ \text{ASL\_cbhpsl}\end{matrix}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^* must be stored in arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they should not be stored in arrays ar and ai. Since the inverse matrix A^{-1} is a Hermitian matrix, only its upper triangular portion is stored in A. This function uses only the upper triangular portions of arrays ar and ai. (See 2.9.1 Figure 2−9.)

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions which perform LDL* decomposition of matrix A.

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.9.7 ASL_zbhplx, ASL_cbhplx Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

ASL_zbhplx or ASL_cbhplx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbhplx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, ipvt, w1);
Single precision:
ierr = ASL_cbhplx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{matrix}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{matrix}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
2	ai	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
3	lna	I	1	Input	Adjustable dimension of arrays ar, ai, alr and ali
4	n	I	1	Input	Order of matrix A
5	alr	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Real part of coefficient matrix A after LDL* decomposition (See Note (a))
6	ali	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	lna×n	Input	Imaginary part of coefficient matrix A after LDL* decomposition (See Note (a))
7	br	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
8	bi	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
9	xr	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	n	Input	Real part of approximate solution $\boldsymbol{x}$
				Output	Real part of iteratively improved solution $\boldsymbol{x}$
10	xi	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	n	Input	Imaginary part of approximate solution $\boldsymbol{x}$
				Output	Imaginary part of iteratively improved solution $\boldsymbol{x}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
11	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
12	nit	I	1	Input	Maximum number of iterations (See Note (d))
13	ipvt	I*	n	Output	Pivoting information. (See Note (a))
14	w1	$\left\{\begin{matrix}\text{D}*\\\text{R}*\end{matrix}\right\}$	3 × n	Work	Work area
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\\text{ASL\_cbhpsl}\end{matrix}\right\}$ or 2.9.4 $\left\{\begin{matrix}\text{ASL\_zbhpls}\\\text{ASL\_cbhpls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\\text{ASL\_cbhpsl}\end{matrix}\right\}$, 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\\text{ASL\_cbhpud}\end{matrix}\right\}$, or 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\\text{ASL\_cbhpuc}\end{matrix}\right\}$ functions and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0 \text{ or itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.10.1 ASL_zbhrsl, ASL_cbhrsl

Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhrsl (ar, ai, lna, n, br, bi, w1);
Single precision:
ierr = ASL_cbhrsl (ar, ai, lna, n, br, bi, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	ai	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
7	w1	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	2 × n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

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

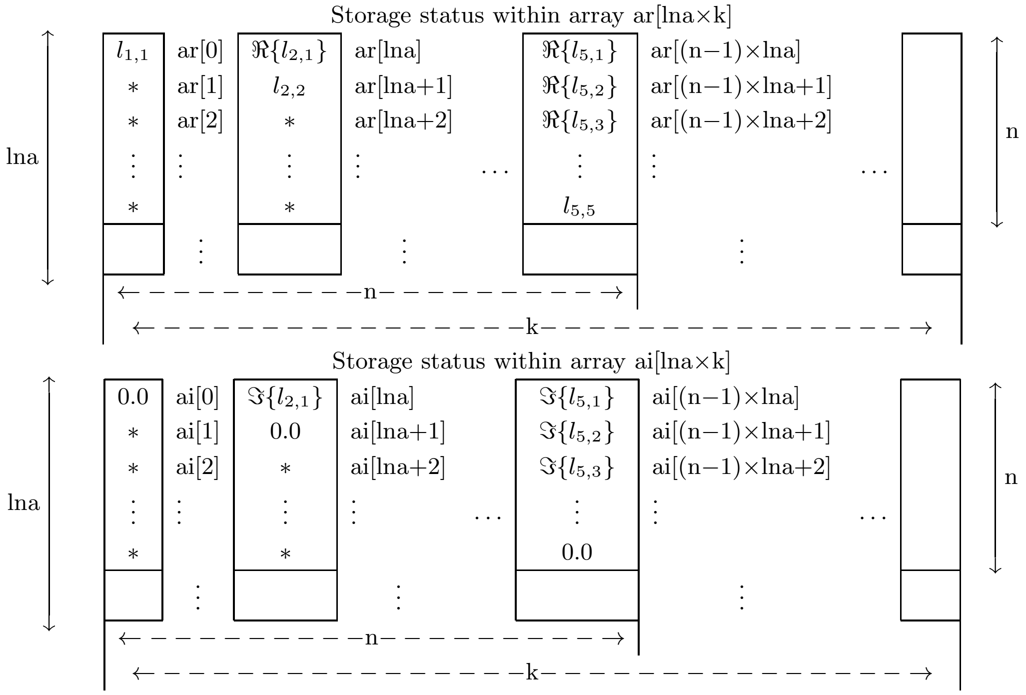
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function 2.10.4 $\left\{\begin{array}{l}\text{ASL\_zbhrls}\\\text{ASL\_cbhrls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LDL* decomposition of matrix A only once.

(b) The upper triangular matrix L^* is stored in the upper triangular portions of arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays ar and ai. The matrix L is the adjoint matrix of the matrix L^*, and the matrix D is a diagonal matrix having the reciprocals of the diagonal elements of the matrix L^* as its components. This function uses only the upper triangular portions of arrays ar and ai (See Fig. 2−10).

$$\begin{array}{cc} \text{Matrix } L^* & \text{Matrix } D \\ \begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix} & \begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix} \end{array}$$

$$\Downarrow$$



Remarks

a. lna $\geq$ n and n $\leq$ k must hold.

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

Figure 2−10 Storage Status of Matrix L^* and Contents of Matrix D

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix real part ar and Imaginary part ai, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_zbhrsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
```

```
double *ar;
double *ai;
int na;
int n;
double *br;
double *bi;
double *w1;
int ierr;
int i,j;
FILE *fp;

fp = fopen( "zbhrsl.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_zbhrsl ***\n" );
printf( "\n     ** Input **\n\n" );
fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );

ar = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( ar == NULL )
{
    printf( "no enough memory for array ar\n" );
    return -1;
}

ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( ai == NULL )
{
    printf( "no enough memory for array ai\n" );
    return -1;
}

br = ( double * )malloc((size_t)( sizeof(double) * n ));
if( br == NULL )
{
    printf( "no enough memory for array br\n" );
    return -1;
}

bi = ( double * )malloc((size_t)( sizeof(double) * n ));
if( bi == NULL )
{
    printf( "no enough memory for array bi\n" );
    return -1;
}

w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\t n = %6d\n\n", n );
printf( "\tCoefficient Matrix  (Real, Imaginary)\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+na*j] );
    }
}
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ai[i+na*j] );
    }
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "                      " );
    }
    for( j=i ; j<n ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", ar[i+na*j],ai[i+na*j] );
    }
    printf( "\n" );
}

printf( "\n\tConstant Vector  (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
```

```
            fscanf( fp, "%lf", &br[i] );
        }
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &bi[i] );
        }

        for( i=0 ; i<n ; i++ )
        {
            printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
        }

        fclose( fp );

        ierr = ASL_zbhrsl(ar, ai, na, n, br, bi, w1);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution   (Real, Imaginary)\n\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  x[%6d] = (%8.3g , %8.3g)\n", i, br[i],bi[i] );
        }

        free( ar );
        free( ai );
        free( br );
        free( bi );
        free( w1 );


        return 0;
    }
```

(d) Output results

```
*** ASL_zbhrsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector  (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 , 5.46e-17)
  x[     2] = (-1.02e-16 ,        1)
  x[     3] = (-4.17e-17 ,        1)
```

2.10.2 ASL_zbhrud, ASL_cbhrud
LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhrud (ar, ai, lna, n, w1);
Single precision:
ierr = ASL_cbhrud (ar, ai, lna, n, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	w1	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	2 × n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai (See Fig. 2−10 in Section 2.10.1).

2.10.3 ASL_zbhruc, ASL_cbhruc

LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting)

(1) Function

ASL_zbhruc or ASL_cbhruc uses the modified Cholesky method to perform an LDL* decomposition and obtain the condition number of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) Usage

Double precision:
ierr = ASL_zbhruc (ar, ai, lna, n, &cond, w1);
Single precision:
ierr = ASL_cbhruc (ar, ai, lna, n, &cond, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	cond	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	1	Output	Reciprocal of the condition number
6	w1	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	2 × n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of arrays ar and ai are not changed. cond $\leftarrow$ 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the LDL* decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became equal to 0.0 in the i-th processing step. A is nearly singular.	Processing is aborted. The condition number is not obtained.

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portions of arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai (See Fig. 2−10 in Section 2.10.1).

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

2.10.4 ASL_zbhrls, ASL_cbhrls Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

(1) **Function**

ASL_zbhrls or ASL_cbhrls solves the simultaneous linear equations $LDL^*\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_zbhrls (ar, ai, lna, n, br, bi);
Single precision:
  ierr = ASL_cbhrls (ar, ai, lna, n, br, bi);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real part of coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	ai	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Imaginary part of coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	br	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/ar[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.10.2 $\left\{\begin{matrix}\text{ASL\_zbhrud}\\ \text{ASL\_cbhrud}\end{matrix}\right\}$. However, if you also want to obtain the condition number, you should use 2.10.3 $\left\{\begin{matrix}\text{ASL\_zbhruc}\\ \text{ASL\_cbhruc}\end{matrix}\right\}$. In addition, if you have already used 2.10.1 $\left\{\begin{matrix}\text{ASL\_zbhrsl}\\ \text{ASL\_cbhrsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.10.5 $\left\{\begin{matrix}\text{ASL\_zbhrms}\\ \text{ASL\_cbhrms}\end{matrix}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^* must be stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in arrays ar and ai. This function uses only the upper triangular portions of arrays ar and ai (See Fig. 2−10 in Section 2.10.1).

2.10.5 ASL_zbhrms, ASL_cbhrms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

(1) **Function**

ASL_zbhrms or ASL_cbhrms solves the simultaneous linear equations $LDL^* \boldsymbol{x_i} = \boldsymbol{b_i} (i = 1, 2, \cdots, m$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
ierr = ASL_zbhrms (ar, ai, lna, n, br, bi, lnb, m);
Single precision:
ierr = ASL_cbhrms (ar, ai, lna, n, br, bi, lnb, m);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* R*}	lna×n	Input	Real part of Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	ai	{D* R*}	lna×n	Input	Imaginary part of Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	br	{D* R*}	lnb×m	Input	Real part of Constant vector $\boldsymbol{b_i}$ $(i = 1, 2, \cdots, m)$
				Output	Real part of Solution $\boldsymbol{x_i}$ $(i = 1, 2, \cdots, m)$
6	bi	{D* R*}	lnb×m	Input	Imaginary part of Constant vector $\boldsymbol{b_i}$ $(i = 1, 2, \cdots, m)$
				Output	Imaginary part of Solution $\boldsymbol{x_i}$ $(i = 1, 2, \cdots, m)$
7	lnb	I	1	Input	Adjustable dimension of array br and bi
8	m	I	1	Input	Number of right-hand side vectors, m
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	br[lnb ∗ (i − 1)] ← br[lnb ∗ (i − 1)]/ar[0], bi[lnb ∗ (i − 1)] ← bi[lnb ∗ (i − 1)]/ar[0] (i= $1, 2, \cdots, m$) are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.10.2 $\left\{\begin{matrix}\text{ASL\_zbhrud}\\ \text{ASL\_cbhrud}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.10.3 $\left\{\begin{matrix}\text{ASL\_zbhruc}\\ \text{ASL\_cbhruc}\end{matrix}\right\}$. In addition, if you have already used 2.10.1 $\left\{\begin{matrix}\text{ASL\_zbhrsl}\\ \text{ASL\_cbhrsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a (See Fig. 2−10 in Section 2.10.1).

(7) **Example**

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

(b) Input data
Coefficient matrix A which has been LDL* decomposed by the modified Cholesky method, $\mathrm{lna} = 11, \mathrm{n} = 4$, constant vectors $\boldsymbol{b_i} (i = 1, 2, \cdots, m)$, lnb=11 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbhrms */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *ai, *br, *bi;
    double *wk;
    int na, nb, n, m, ierr, i, j;
```

```
FILE *fp;

fp = fopen( "zbhrms.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_zbhrms ***\n" );
printf( "\n     ** Input **\n\n" );

fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );
fscanf( fp, "%d", &nb );
fscanf( fp, "%d", &m );

ar = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( ar == NULL )
{
    printf( "no enough memory for array ar\n" );
    return -1;
}

ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( ai == NULL )
{
    printf( "no enough memory for array ai\n" );
    return -1;
}

br = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
if( br == NULL )
{
    printf( "no enough memory for array br\n" );
    return -1;
}

bi = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
if( bi == NULL )
{
    printf( "no enough memory for array bi\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (2*n) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", n );

printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf %lf", &ar[i+na*j], &ai[i+na*j] );
    }
}

for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "                       " );
    }
    for( j=i ; j<n ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", ar[i+na*j],ai[i+na*j] );
    }
    printf( "\n" );
}

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( j=0 ; j<m ; j++ )
{
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf", &br[i+nb*j], &bi[i+nb*j]);
    }
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
```

```
        {
            printf( "(%8.3g , %8.3g) ", br[i+nb*j],bi[i+nb*j] );
        }
        printf( "\n" );
    }
    fclose( fp );

    ierr = ASL_zbhrud(ar, ai, na, n, wk);
    ierr = ASL_zbhrms(ar, ai, na, n, br, bi, nb, m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", br[i+nb*j],bi[i+nb*j] );
        }
        printf( "\n" );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( wk );
    return 0;
}
```

(d) Output results

```
*** ASL_zbhrms ***

 ** Input **

n =      4
m =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-9.87e-16 ,         1) ( 9.87e-17 ,         1) (        1 ,  3.95e-16)
(        1 ,  6.25e-17) (        1 , -5.62e-16) (-4.68e-16 ,         1) (  2.5e-16 ,         1)
(-5.11e-17 ,         1) (        1 , -6.13e-16) (        1 ,  1.02e-16) (-2.04e-16 ,         1)
(-4.17e-17 ,         1) ( 5.01e-16 ,         1) (        1 , -2.09e-16) (        1 ,         0)
```

2.10.6 ASL_zbhrdi, ASL_cbhrdi
Determinant and Inverse Matrix of a Hermitian Matrix (No Pivoting)

(1) **Function**

ASL_zbhrdi or ASL_cbhrdi obtains the determinant and inverse matrix of the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method.

(2) **Usage**

Double precision:
ierr = ASL_zbhrdi (ar, ai, lna, n, det, isw, w1);
Single precision:
ierr = ASL_cbhrdi (ar, ai, lna, n, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Real part of the Inverse matrix of matrix A (See Note (b))
2	ai	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Imaginary part of the Inverse matrix of matrix A (See Note (b))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	det	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
6	isw	I	1	Input	Processing switch isw>0:Obtain determinant. isw=0:Obtain determinant and inverse matrix. isw<0:Obtain inverse matrix.
7	w1	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 ar[0] ← 1.0/ar[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Use any of the 2.10.2 $\left\{\begin{array}{l}\text{ASL\_zbhrud}\\ \text{ASL\_cbhrud}\end{array}\right\}$, 2.10.3 $\left\{\begin{array}{l}\text{ASL\_zbhruc}\\ \text{ASL\_cbhruc}\end{array}\right\}$, 2.10.1 $\left\{\begin{array}{l}\text{ASL\_zbhrsl}\\ \text{ASL\_cbhrsl}\end{array}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^* must be stored in arrays ar and ai. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they should not be stored in arrays ar and ai. Since the inverse matrix A^{-1} is a Hermitian matrix, only its upper triangular portion is stored in A. This function uses only the upper triangular portions of arrays ar and ai (See Fig. 2−10 in Section 2.10.1).

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.10.7 ASL_zbhrlx, ASL_cbhrlx
Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) Function

ASL_zbhrlx or ASL_cbhrlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_zbhrlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, w1);
Single precision:
ierr = ASL_cbhrlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, &itol, nit, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
2	ai	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
3	lna	I	1	Input	Adjustable dimension of arrays ar, ai, alr and ali
4	n	I	1	Input	Order of matrix A
5	alr	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Real part of coefficient matrix A after LDL* decomposition (See Note (a))
6	ali	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	lna×n	Input	Imaginary part of coefficient matrix A after LDL* decomposition (See Note (a))
7	br	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Real part of constant vector $\boldsymbol{b}$
8	bi	$\begin{Bmatrix}\text{D}*\\ \text{R}*\end{Bmatrix}$	n	Input	Imaginary part of constant vector $\boldsymbol{b}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
9	xr	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Real part of approximate solution $\boldsymbol{x}$
				Output	Real part of iteratively improved solution $\boldsymbol{x}$
10	xi	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Imaginary part of approximate solution $\boldsymbol{x}$
				Output	Imaginary part of iteratively improved solution $\boldsymbol{x}$
11	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
12	nit	I	1	Input	Maximum number of iterations (See Note (d))
13	w1	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	3 × n	Work	Work area
14	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.10.1 $\begin{Bmatrix}\text{ASL\_zbhrsl}\\\text{ASL\_cbhrsl}\end{Bmatrix}$ or 2.10.4 $\begin{Bmatrix}\text{ASL\_zbhrls}\\\text{ASL\_cbhrls}\end{Bmatrix}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.10.1 $\begin{Bmatrix}\text{ASL\_zbhrsl}\\\text{ASL\_cbhrsl}\end{Bmatrix}$, 2.10.2 $\begin{Bmatrix}\text{ASL\_zbhrud}\\\text{ASL\_cbhrud}\end{Bmatrix}$, or 2.10.3 $\begin{Bmatrix}\text{ASL\_zbhruc}\\\text{ASL\_cbhruc}\end{Bmatrix}$ functions must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0 \text{ or itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.11.1 ASL_zbhfsl, ASL_cbhfsl
Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhfsl (a, lna, n, b, ipvt, w1);
Single precision:
ierr = ASL_cbhfsl (a, lna, n, b, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	w1	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

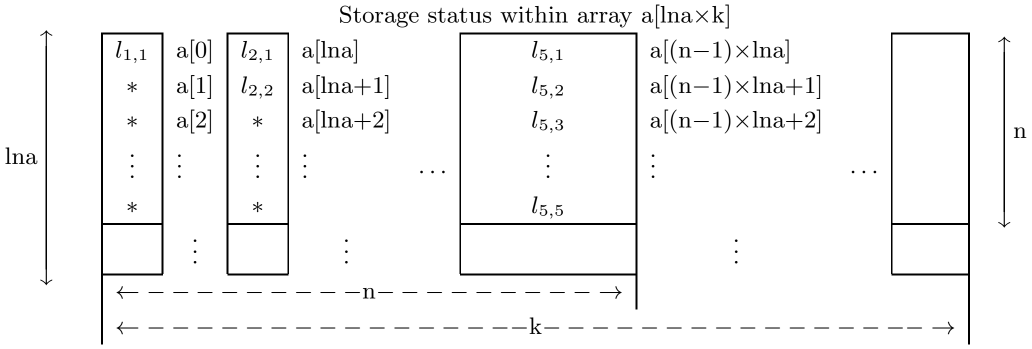
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function 2.11.4 $\left\{\begin{matrix}\text{ASL\_zbhfls}\\\text{ASL\_cbhfls}\end{matrix}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the LDL* decomposition of matrix A only once.

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

$$
\begin{array}{cc}
\text{Matrix } L^* & \text{Matrix } D \\
\begin{bmatrix}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{bmatrix}
&
\begin{bmatrix}
1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5}
\end{bmatrix}
\end{array}
$$

$$\Downarrow$$



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−11 Storage Status of Matrix L^* and Contents of Matrix D

(c) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_zbhfsl */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a;
    int na;
    int n;
    double _Complex *b;
    int *ipvt;
    double *w1;
    int ierr;
```

```
int i,j;
FILE *fp;

fp = fopen( "zbhfsl.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_zbhfsl ***\n" );
printf( "\n     ** Input **\n\n" );

fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );

a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
if( a == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}

b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}

ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
if( ipvt == NULL )
{
    printf( "no enough memory for array ipvt\n" );
    return -1;
}

w1 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\t n = %6d\n", n );
printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        double tmp_re;
        fscanf( fp, "%lf", &tmp_re );
        a[i+na*j] = tmp_re;
    }
}
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        double tmp_im;
        fscanf( fp, "%lf", &tmp_im );
        a[i+na*j] = a[i+na*j] + tmp_im * _Complex_I;
    }
}

for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "                      " );
    }
    for( j=i ; j<n ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
    }
    printf( "\n" );
}

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    double tmp_re;
    fscanf( fp, "%lf", &tmp_re );
    b[i] = tmp_re;
}
for( i=0 ; i<n ; i++ )
{
    double tmp_im;
```

```
        fscanf( fp, "%lf", &tmp_im );
        b[i] = b[i] + tmp_im * _Complex_I;
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
    }
    fclose( fp );

    ierr = ASL_zbhfsl(a, na, n, b, ipvt, w1);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = (%8.3g , %8.3g)\n", i,creal(b[i]),cimag(b[i]));
    }

    free( a );
    free( b );
    free( ipvt );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_zbhfsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 , 8.88e-17)
  x[     2] = (-4.97e-17 ,        1)
  x[     3] = (-4.17e-17 ,        1)
```

2.11.2 ASL_zbhfud, ASL_cbhfud
LDL* Decomposition of a Hermitian Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhfud (a, lna, n, ipvt, w1);
Single precision:
ierr = ASL_cbhfud (a, lna, n, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z∗ C∗}	lna×n	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	w1	{D∗ R∗}	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array a. This function uses only the upper triangular portion of array a. (See Fig. 2−11 in Section 2.11.1)

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

2.11.3 ASL_zbhfuc, ASL_cbhfuc
LDL* Decomposition and Condition Number of a Hermitian Matrix

(1) **Function**

ASL_zbhfuc or ASL_cbhfuc uses the modified Cholesky method to perform an LDL* decomposition and obtain the condition number of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

ierr = ASL_zbhfuc (a, lna, n, ipvt, &cond, w1);

Single precision:

ierr = ASL_cbhfuc (a, lna, n, ipvt, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	cond	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
6	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array a. (See Fig. 2−11 in Section 2.11.1)

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

2.11.4 ASL_zbhfls, ASL_cbhfls Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix)

(1) **Function**

ASL_zbhfls or ASL_cbhfls solves the simultaneous linear equations $LDL^*\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbhfls (a, lna, n, b, ipvt);
Single precision:
ierr = ASL_cbhfls (a, lna, n, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: { int as for 32bit Integer
R:Single precision real	C:Single precision complex	long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z* C*}	lna×n	Input	Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	{Z* C*}	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.11.2 $\left\{\begin{array}{l}\text{ASL\_zbhfud}\\\text{ASL\_cbhfud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.11.3 $\left\{\begin{array}{l}\text{ASL\_zbhfuc}\\\text{ASL\_cbhfuc}\end{array}\right\}$. In addition, if you have already used 2.11.1 $\left\{\begin{array}{l}\text{ASL\_zbhfsl}\\\text{ASL\_cbhfsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.11.5 $\left\{\begin{array}{l}\text{ASL\_zbhfms}\\\text{ASL\_cbhfms}\end{array}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. (See Fig. 2−11 in Section 2.11.1)

(c) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions 2.11.2 $\left\{\begin{array}{l}\text{ASL\_zbhfud}\\\text{ASL\_cbhfud}\end{array}\right\}$, 2.11.3 $\left\{\begin{array}{l}\text{ASL\_zbhfuc}\\\text{ASL\_cbhfuc}\end{array}\right\}$ or 2.11.1 $\left\{\begin{array}{l}\text{ASL\_zbhfsl}\\\text{ASL\_cbhfsl}\end{array}\right\}$ which perform LDL* decomposition of matrix A.

2.11.5 ASL_zbhfms, ASL_cbhfms
Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Hermitian Matrix)

(1) **Function**

ASL_zbhfms or ASL_cbhfms solves the simultaneous linear equations $LDL^*\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
ierr = ASL_zbhfms (a, lna, n, b, lnb, m, ipvt);
Single precision:
ierr = ASL_cbhfms (a, lna, n, b, lnb, m, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer
R:Single precision real C:Single precision complex long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z* C*}	lna×n	Input	Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	{Z* C*}	lnb×m	Input	Constant vector $\boldsymbol{b_i}(i = 1, 2, \cdots, m)$
				Output	Solution $\boldsymbol{x_i}(i = 1, 2, \cdots, m)$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Number of right-hand side vectors, m
7	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[lnb ∗ (i − 1)] ← b[lnb ∗ (i − 1)])/a[0] (i= $1, 2, \cdots, m$) are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) Notes

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.11.2 $\left\{\begin{array}{c}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.11.3 $\left\{\begin{array}{c}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{array}\right\}$. In addition, if you have already used 2.11.1 $\left\{\begin{array}{c}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. (See Fig. 2−11 in Section 2.11.1)

(c) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions 2.11.2 $\left\{\begin{array}{c}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{array}\right\}$, 2.11.3 $\left\{\begin{array}{c}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{array}\right\}$ or 2.11.1 $\left\{\begin{array}{c}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{array}\right\}$ which perform LDL* decomposition of matrix A.

(7) Example

(a) Problem
Solve the following simultaneous linear equations.

$$\left[\begin{array}{rrrr} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{array}\right] \left[\begin{array}{cccc} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{array}\right] = \left[\begin{array}{rrrr} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{array}\right]$$

(b) Input data
Coefficient matrix A which has been LDL* decomposed by the modified Cholesky method, lna $= 11$, n $= 4$, constant vectors $\boldsymbol{b_i}(i = 1, 2, \cdots, m)$, lnb=11 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbhfms */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a, *b;
    double *wk;
    int *ipvt;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

    fp = fopen( "zbhfms.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbhfms ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &nb );
    fscanf( fp, "%d", &m );

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (nb*m) ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    ipvt = ( int * )malloc((size_t)( sizeof(int) * (n) ));
    if( ipvt == NULL )
    {
        printf( "no enough memory for array ipvt\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", n );

    printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        for( j=i ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf %lf", &tmp_re, &tmp_im );
            a[i+na*j] = tmp_re + tmp_im * _Complex_I;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                       " );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
```

```
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf %lf", &tmp_re, &tmp_im);
            b[i+nb*j] = tmp_re + tmp_im * _Complex_I;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }
    fclose( fp );

    ierr = ASL_zbhfud(a, na, n, ipvt,wk);
    ierr = ASL_zbhfms(a, na, n, b, nb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( wk );
    return 0;
}
```

(d) Output results

```
*** ASL_zbhfms ***

 ** Input **

n =      4
m =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-4.28e-16 ,         1) ( 5.35e-17 ,         1) (        1 ,         0)
(        1 ,  8.88e-17) (        1 , -1.78e-16) (-1.78e-16 ,         1) (        0 ,         1)
(-4.97e-17 ,         1) (        1 , -1.33e-16) (        1 ,         0) (        0 ,         1)
(-4.17e-17 ,         1) ( 8.34e-17 ,         1) (        1 , -8.34e-17) (        1 , -8.34e-17)
```

2.11.6 ASL_zbhfdi, ASL_cbhfdi Determinant and Inverse Matrix of a Hermitian Matrix

(1) **Function**

ASL_zbhfdi or ASL_cbhfdi obtains the determinant and inverse matrix of the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method.

(2) **Usage**

Double precision:
ierr = ASL_zbhfdi (a, lna, n, ipvt, det, isw, w1);
Single precision:
ierr = ASL_cbhfdi (a, lna, n, ipvt, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real	C:Single precision complex	

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (d))
5	det	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
6	isw	I	1	Input	Processing switch isw>0:Obtain determinant. isw=0:Obtain determinant and inverse matrix. isw<0:Obtain inverse matrix.
7	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \le \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/a[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Use any of the functions 2.11.2 $\begin{Bmatrix}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{Bmatrix}$, 2.11.3 $\begin{Bmatrix}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{Bmatrix}$, 2.11.1 $\begin{Bmatrix}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{Bmatrix}$ to perform the decomposition.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. Since the inverse matrix A^{-1} is a Hermitian matrix, only its upper triangular portion is stored in A. (See Fig. 2−11 in Section 2.11.1)

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) Information about partial pivoting performed during LDL* decomposition must be stored in ipvt. This information is given by the functions which perform LDL* decomposition of matrix A.

(e) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.11.7 ASL_zbhflx, ASL_cbhflx
Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

ASL_zbhflx or ASL_cbhflx uses an iterative method to improve the solution of the simultaneous linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_zbhflx (a, lna, n, al, b, x, &itol, nit, ipvt, w1);
Single precision:
ierr = ASL_cbhflx (a, lna, n, al, b, x, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

$$\text{I:} \begin{Bmatrix} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{Bmatrix}$$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\begin{Bmatrix} \text{Z}* \\ \text{C}* \end{Bmatrix}$	lna×n	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
2	lna	I	1	Input	Adjustable dimension of array a and al
3	n	I	1	Input	Order of matrix A
4	al	$\begin{Bmatrix} \text{Z}* \\ \text{C}* \end{Bmatrix}$	lna×n	Input	Coefficient matrix A after LDL* decomposition (See Note (a))
5	b	$\begin{Bmatrix} \text{Z}* \\ \text{C}* \end{Bmatrix}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\begin{Bmatrix} \text{Z}* \\ \text{C}* \end{Bmatrix}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	ipvt	I*	n	Output	Pivoting information. (See Note (a))
10	w1	$\begin{Bmatrix} \text{Z}* \\ \text{C}* \end{Bmatrix}$	n	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) Notes

(a) This function improves the solution obtained by the 2.11.1 $\left\{\begin{matrix}\text{ASL\_zbhfsl}\\\text{ASL\_cbhfsl}\end{matrix}\right\}$ or 2.11.4 $\left\{\begin{matrix}\text{ASL\_zbhfls}\\\text{ASL\_cbhfls}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.11.1 $\left\{\begin{matrix}\text{ASL\_zbhfsl}\\\text{ASL\_cbhfsl}\end{matrix}\right\}$, 2.11.2 $\left\{\begin{matrix}\text{ASL\_zbhfud}\\\text{ASL\_cbhfud}\end{matrix}\right\}$, or 2.11.3 $\left\{\begin{matrix}\text{ASL\_zbhfuc}\\\text{ASL\_cbhfuc}\end{matrix}\right\}$ functions and the pivoting information at that time must be given as input.

(b) Solution improvement is repreated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\mathrm{itol} \leq 0$$

or

$$\mathrm{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

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

2.12.1 ASL_zbhesl, ASL_cbhesl

Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_zbhesl (a, lna, n, b);
Single precision:
ierr = ASL_cbhesl (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{matrix}Z*\\C*\end{matrix}\right\}$	lna×n	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{matrix}Z*\\C*\end{matrix}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

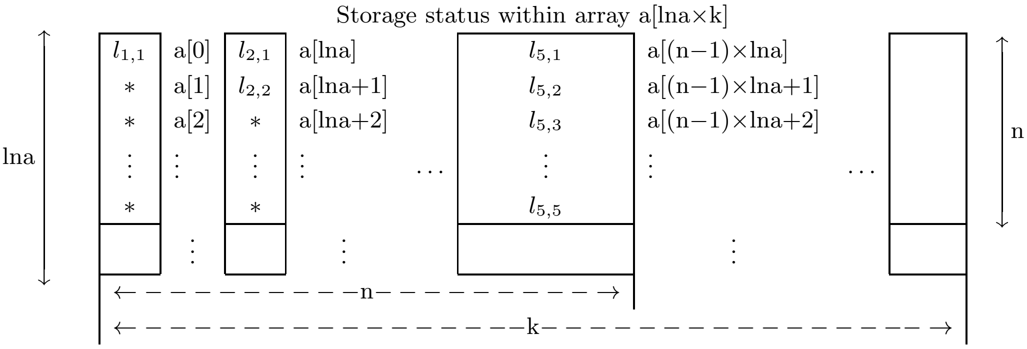
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function 2.12.4 $\left\{\begin{array}{l}\text{ASL\_zbhels}\\ \text{ASL\_cbhels}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the LDL* decomposition of matrix A only once.

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

$$
\begin{array}{cc}
\text{Matrix } L^* & \text{Matrix } D \\
\begin{bmatrix}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{bmatrix}
&
\begin{bmatrix}
1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5}
\end{bmatrix}
\end{array}
$$

$$\Downarrow$$



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−12 Storage Status of Matrix L^* and Contents of Matrix D

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_zbhesl */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a;
    int na;
    int n;
    double _Complex *b;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbhesl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbhesl ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    printf( "\t n = %6d\n", n );
    printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            double tmp_re;
            fscanf( fp, "%lf", &tmp_re );
            a[i+na*j] = tmp_re;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            double tmp_im;
            fscanf( fp, "%lf", &tmp_im );
            a[i+na*j] = a[i+na*j] + tmp_im * _Complex_I;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                      " );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        double tmp_re;
        fscanf( fp, "%lf", &tmp_re );
        b[i] = tmp_re;
    }
    for( i=0 ; i<n ; i++ )
    {
        double tmp_im;
        fscanf( fp, "%lf", &tmp_im );
        b[i] = b[i] + tmp_im * _Complex_I;
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
    }
    fclose( fp );

    ierr = ASL_zbhesl(a, na, n, b);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = (%8.3g , %8.3g)\n", i,creal(b[i]),cimag(b[i]));
    }

    free( a );
    free( b );

    return 0;
}
```

(d) Output results

```
*** ASL_zbhesl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

  x[     0] = (       1 , 9.87e-17)
  x[     1] = (       1 , 9.37e-17)
  x[     2] = (-1.02e-16 ,        1)
  x[     3] = (       0 ,        1)
```

2.12.2 ASL_zbheud, ASL_cbheud
LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_zbheud (a, lna, n);
Single precision:
  ierr = ASL_cbheud (a, lna, n);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{Z}*\\\text{C}*\end{array}\right\}$	lna×n	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array a. This function uses only the upper triangular portion of array a. (See Fig. 2−12 in Section 2.12.1)

2.12.3 ASL_zbheuc, ASL_cbheuc
LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting)

(1) **Function**

ASL_zbheuc or ASL_cbheuc uses the modified Cholesky method to perform an LDL* decomposition and obtain the condition number of the Hermitian matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
  ierr = ASL_zbheuc (a, lna, n, &cond, w1);
Single precision:
  ierr = ASL_cbheuc (a, lna, n, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	cond	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
5	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) The upper triangular matrix L^* is stored in the upper triangular portion of array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they are not stored in array a. (See Fig. 2−12 in Section 2.12.1)

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

2.12.4 ASL_zbhels, ASL_cbhels Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

(1) Function

ASL_zbhels or ASL_cbhels solves the simultaneous linear equations $LDL^*\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_zbhels (a, lna, n, b);
Single precision:
ierr = ASL_cbhels (a, lna, n, b);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$. In addition, if you have already used 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output. To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function 2.12.5 $\left\{\begin{matrix}\text{ASL\_zbhems}\\ \text{ASL\_cbhems}\end{matrix}\right\}$ to perform the calculations.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. (See Fig. 2−12 in Section 2.12.1)

2.12.5 ASL_zbhems, ASL_cbhems Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

(1) **Function**

ASL_zbhems or ASL_cbhems solves the simultaneous linear equations $LDL^*\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method as coefficient matrix. That is, when the $n \times m$ matrix B is defined by $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$, the function obtains $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$.

(2) **Usage**

Double precision:
ierr = ASL_zbhems (a, lna, n, b, lnb, m);
Single precision:
ierr = ASL_cbhems (a, lna, n, b, lnb, m);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer, long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z* C*}	lna×n	Input	Coefficient matrix A after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	{Z* C*}	lnb×m	Input	Constant vector $\boldsymbol{b_i}(i = 1, 2, \cdots, m)$
				Output	Solution $\boldsymbol{x_i}(i = 1, 2, \cdots, m)$
5	lnb	I	1	Input	Adjustable dimension of array b
6	m	I	1	Input	Number of right-hand side vectors, m
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

(b) $\mathrm{m} > 0$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[lnb ∗ (i − 1)] ← b[lnb ∗ (i − 1)])/a[0] (i= $1, 2, \cdots, m$) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Normally, you should decompose matrix A by calling the 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$. In addition, if you have already used 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LDL* decomposition obtained as part of its output.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. (See Fig. 2−12 in Section 2.12.1)

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

(b) Input data

Coefficient matrix A which has been LDL* decomposed by the modified Cholesky method, lna = 11, n = 4, constant vectors $\boldsymbol{b_i}(i = 1, 2, \cdots, m)$, lnb=11 and m=4.

(c) Main program

```
/*      C interface example for ASL_zbhems */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    double _Complex *a, *b;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

    fp = fopen( "zbhems.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbhems ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &nb );
    fscanf( fp, "%d", &m );

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (nb*m) ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
```

```
        return -1;
    }

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", n );

    printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        for( j=i ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf %lf", &tmp_re, &tmp_im );
            a[i+na*j] = tmp_re + tmp_im * _Complex_I;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                      " );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf %lf", &tmp_re, &tmp_im);
            b[i+nb*j] = tmp_re + tmp_im * _Complex_I;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }
    fclose( fp );

    ierr = ASL_zbheud(a, na, n);
    ierr = ASL_zbhems(a, na, n, b, nb, m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    return 0;
}
```

(d) Output results

```
*** ASL_zbhems ***

 ** Input **

n =      4
m =      4

Coefficient Matrix  (Real, Imaginary)

(       9 ,        0) (       7 ,        3) (       2 ,        5) (       1 ,        1)
                      (      10 ,        0) (       3 ,        2) (       2 ,        4)
                                            (       8 ,        0) (       5 ,        1)
                                                                  (       6 ,        0)

Constant Vector (Real, Imaginary)

(      10 ,        6) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,  2.96e-16) (        0 ,         1) ( -7.4e-17 ,         1) (        1 ,         0)
(        1 ,  6.25e-17) (        1 , -4.06e-16) (-6.25e-17 ,         1) (        0 ,         1)
(-1.02e-16 ,         1) (        1 , -2.04e-16) (        1 ,  1.02e-16) (        0 ,         1)
( 4.17e-17 ,         1) ( 6.67e-16 ,         1) (        1 , -4.17e-17) (        1 ,         0)
```

2.12.6 ASL_zbhedi, ASL_cbhedi Determinant and Inverse Matrix of a Hermitian Matrix (No Pivoting)

(1) **Function**

ASL_zbhedi or ASL_cbhedi obtains the determinant and inverse matrix of the Hermitian matrix A (two-dimensional array type) (upper triangular type) which has been LDL* decomposed by the modified Cholesky method.

(2) **Usage**

Double precision:
ierr = ASL_zbhedi (a, lna, n, det, isw, w1);
Single precision:
ierr = ASL_cbhedi (a, lna, n, det, isw, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b))
				Output	Inverse matrix of matrix A (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	det	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	2	Output	Determinant of matrix A (See Note (c))
5	isw	I	1	Input	Processing switch isw>0:Obtain determinant. isw=0:Obtain determinant and inverse matrix. isw<0:Obtain inverse matrix.
6	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/a[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LDL* decomposed before using this function. Use any of the functions 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$, 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ to perform the decomposition.

(b) The upper triangular matrix L^* must be stored in array a. Since the diagonal matrix D and the lower triangular matrix L are calculated from L^*, they need not be stored in array a. Since the inverse matrix A^{-1} is a Hermitian matrix, only its upper triangular portion is stored in A. (See Fig. 2−12 in Section 2.12.1)

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.12.7 ASL_zbhelx, ASL_cbhelx
Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) Function

ASL_zbhelx or ASL_cbhelx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the Hermitian matrix A (two-dimensional array type) (upper triangular type) as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_zbhelx (a, lna, n, al, b, x, &itol, nit, w1);
Single precision:
ierr = ASL_cbhelx (a, lna, n, al, b, x, &itol, nit, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{ \begin{array}{l} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
2	lna	I	1	Input	Adjustable dimension of array a and al
3	n	I	1	Input	Order of matrix A
4	al	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	lna×n	Input	Coefficient matrix A after LDL* decomposition (See Note (a))
5	b	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
6	x	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
7	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
8	nit	I	1	Input	Maximum number of iterations (See Note (d))
9	w1	$\left\{\begin{array}{c}Z*\\C*\end{array}\right\}$	n	Work	Work area
10	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a) was not satisfied.	Processing is aborted.
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) **Notes**

(a) This function improves the solution obtained by the 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ or 2.12.4 $\left\{\begin{matrix}\text{ASL\_zbhels}\\ \text{ASL\_cbhels}\end{matrix}\right\}$ function. Therefore, the coefficient matrix A after being decomposed by the 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ or 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$ function must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \le 0$$

or

$$\text{itol} \ge -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

2.13 REAL BAND MATRIX (BAND TYPE)

2.13.1 ASL_dbbdsl, ASL_rbbdsl
Simultaneous Linear Equations (Real Band Matrix)

(1) **Function**

ASL_dbbdsl or ASL_rbbdsl uses the Gauss method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a real band matrix (band type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbbdsl (a, lma, n, mu, ml, b, ipvt);
Single precision:
ierr = ASL_rbbdsl (a, lma, n, mu, ml, b, ipvt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

$$\text{I:}\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lma×n	Input	Coefficient matrix A (real band matrix, band type) (See Appendix B)
				Output	Upper triangular matrix U and unit lower triangular matrix L when A is decomposed into $A = LU$ (See Note (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	b	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
7	ipvt	I*	n	Output	Pivoting information ipvt[i-1]: Number of row exchanged with row i in the i-th processing step (See Note (b))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mu} \leq \mathrm{n} - 1$
$0 \leq \mathrm{ml} \leq \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \leq \mathrm{lma}$

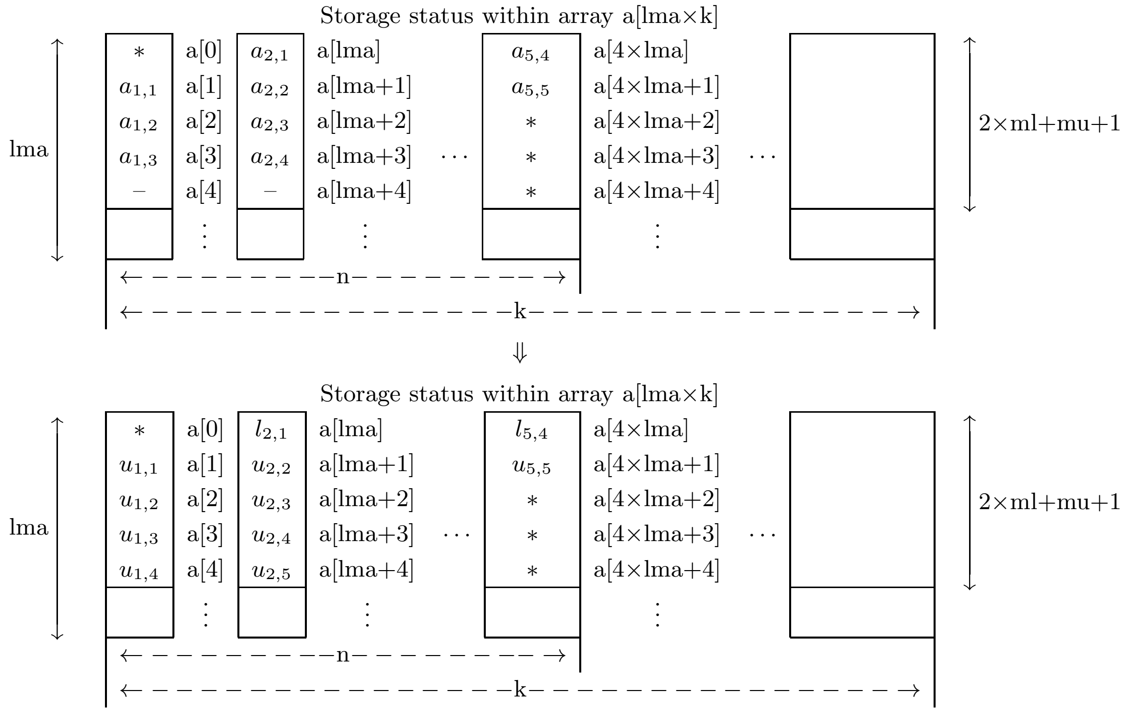
(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array a are not changed. b[0] ← b[0]/a[0] is performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this function only once and then call function 2.13.4 $\left\{\begin{array}{c}\text{ASL\_dbbdls}\\\text{ASL\_rbbdls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the LU decomposition of matrix A only once.

(b) This function performs partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j ($i \leq j$), then j is stored in ipvt[i-1]. In addition, since columns i through n in rows i and j of matrix A actually are exchanged at this time, the storage area of array a increases only by size ml×n. Therefore, if $n < 2ml + mu + 1$, less memory is required to use the function for real matrices.



Remarks

a. Input time values of elements indicated by asterisks (∗) are guaranteed.

b. $u_{1,4}$, $u_{2,5}$ is set when corresponding rows are actually exchanged by partial pivoting.

c. mu is the upper band width and ml is the lower band width.

d. lma ≥ 2 × ml+mu+1 and k ≥ n must hold.

Figure 2−13 Storage Status of Array a before and after LU Decomposition

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 1 & -2 & 0 & 0 \\ -1 & 3 & 2 & 0 \\ 1 & -1 & 4 & -2 \\ 0 & 1 & -1 & 7 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 3 \\ -7 \\ 1 \\ 13 \end{bmatrix}$$

(b) Input data

Coefficient matrix $\mathrm{a}, \mathrm{lma} = 11, \mathrm{n} = 4, \mathrm{mu} = 1, \mathrm{ml} = 2$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbbdsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,wa;
    int ma;
    int n;
    int mu;
    int ml;
    double *b;
    int *kpvt;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

    fp = fopen( "dbbdsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbbdsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &mu );
    fscanf( fp, "%d", &ml );

    a = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    kpvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( kpvt == NULL )
    {
        printf( "no enough memory for array kpvt\n" );
        return -1;
    }

    printf( "\tn = %6d\n\n", n );
    printf( "\tUpper Band Width = %6d \n\n",mu);
    printf( "\tLower Band Width = %6d \n\n",ml);
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &wa );
            if(j-i<=mu && i-j<=ml){
            if(i-ml>=0){
            a[j-i+ml+ma*i]=wa;
            }
```

```
            else {
            a[j+ml-i+ma*i]=wa;
            }
            }
        }
    }
    printf( "\n\tCoefficient Matrix\n\n");
    printf( "\t%8c %8c %8.3g %8.3g\n",   SP,SP,  a[  2*ma],a[  3*ma] );
    printf( "\t%8c %8.3g %8.3g %8.3g\n", SP,a[1+ma],a[1+2*ma],a[1+3*ma] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n", a[2],a[2+ma],a[2+2*ma],a[2+3*ma] );
    printf( "\t%8.3g %8.3g %8.3g\n",       a[3],a[3+ma],a[3+2*ma]           );

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );

    ierr = ASL_dbbdsl(a, ma, n, mu, ml, b, kpvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
    }

    free( a );
    free( b );
    free( kpvt );

    return 0;
}
```

(d) Output results

```
*** ASL_dbbdsl ***

 ** Input **

n =      4

Upper Band Width =      1

Lower Band Width =      2


Coefficient Matrix

                         1        1
               -1       -1       -1
       1        3        4        7
      -2        2       -2

Constant Vector

       3
      -7
       1
      13

 ** Output **

ierr =      0

Solution

  x[     0] =      -29
  x[     1] =      -16
  x[     2] =        6
  x[     3] =        5
```

2.13.2 ASL_dbbdlu, ASL_rbbdlu LU Decomposition of a Real Band Matrix

(1) Function

ASL_dbbdlu or ASL_rbbdlu uses the Gauss method to perform an LU decomposition of the real band matrix A (band type).

(2) Usage

Double precision:
ierr = ASL_dbbdlu (a, lma, n, mu, ml, ipvt);
Single precision:
ierr = ASL_rbbdlu (a, lma, n, mu, ml, ipvt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\ R*\end{array}\right\}$	lma×n	Input	Real band matrix A (band type) (See Appendix B)
				Output	Upper triangular matrix U and unit lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	ipvt	I*	n	Output	Pivoting information ipvt[i-1]: Number of row exchanged with row i in the i-th processing step (See Note (b))
7	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $\mathrm{n} > 0$

(b) $0 \le \mathrm{mu} \le \mathrm{n} - 1$
$0 \le \mathrm{ml} \le \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \le \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array a are not changed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

(a) The unit lower triangular matrix L and the upper triangular matrix U are stored in band format in array a. However, since the diagonal elements of L always are 1.0, they are not stored in array a. (See Section 2.13.1 Figure 2−13.)

(b) This function performs partial pivoting. Pivoting information is stored in array ipvt for use by subsequent function. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in ipvt[i-1]. In addition, since columns i through n in rows i and j of matrix A actually are exchanged at this time, the storage area within array a increases only by size ml×n. Therefore, if n $<$ 2ml + mu + 1, less memory is required to use the function for real matrices. (See Section 2.13.1 Figure 2−13.)

2.13.3 ASL_dbbdlc, ASL_rbbdlc
LU Decomposition and Condition Number of a Real Band Matrix

(1) Function

ASL_dbbdlc or ASL_rbbdlc uses the Gauss method to perform an LU decomposition and obtain the condition number of the real band matrix A (band type).

(2) Usage

Double precision:
  ierr = ASL_dbbdlc (a, lma, n, mu, ml, ipvt, &cond, w1);
Single precision:
  ierr = ASL_rbbdlc (a, lma, n, mu, ml, ipvt, &cond, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\ R*\end{array}\right\}$	lma×n	Input	Real band matrix A (band type) (See Appendix B)
				Output	Upper triangular matrix U and unit lower triangular matrix L when A is decomposed into $A = LU$ (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	ipvt	I*	n	Output	Pivoting information ipvt[i-1]: Number of row exchanged with row i in the i-th processing step (See Note (b))
7	cond	$\left\{\begin{array}{c}D*\\ R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
8	w1	$\left\{\begin{array}{c}D*\\ R*\end{array}\right\}$	n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mu} \leq \mathrm{n} - 1$
$0 \leq \mathrm{ml} \leq \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \leq \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array a are not changed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the i-th processing step. A is nearly singular.	Processing is aborted. The condition number is not obtained.

(6) **Notes**

(a) The unit lower triangular matrix L and the upper triangular matrix U are stored in band format in array a. However, since the diagonal elements of L always are 1.0, they are not stored in array a. (See 2.13.1 Figure 2−13.)

(b) This function performs partial pivoting. Pivoting information is stored in array ipvt for use by subsequent function. If the pivot row in the i-th step is row j (i $\leq$ j), then j is stored in ipvt[i-1]. In addition, since columns i through n in rows i and j of matrix A actually are exchanged at this time, the storage area within array a increases only by size ml×n. Therefore, if n $<$ 2ml + mu + 1, less memory is required to use the function for real matrices. (See 2.13.1 Figure 2−13.)

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

2.13.4 ASL_dbbdls, ASL_rbbdls Simultaneous Linear Equations (LU-Decomposed Real Band Matrix)

(1) Function

ASL_dbbdls or ASL_rbbdls solves the simultaneous linear equations $LU\boldsymbol{x} = \boldsymbol{b}$ having the real band matrix A (band type) which has been LU decomposed by the Gauss method as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_dbbdls (a, lma, n, mu, ml, b, ipvt);
Single precision:
ierr = ASL_rbbdls (a, lma, n, mu, ml, b, ipvt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lma×n	Input	Coefficient matrix A after LU decomposition (real band matrix, band type) (See Appendix B) (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
7	ipvt	I*	n	Input	Pivoting information ipvt[i-1]: Number of row exchanged with row i in the i-th processing step (See Note (c))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $\text{n} > 0$

(b) $0 \leq \text{mu} \leq \text{n} - 1$
$0 \leq \text{ml} \leq \text{n} - 1$

(c) $\min(2 \times \text{ml} + \text{mu} + 1, \text{n} + \text{ml}) \leq \text{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	L has a 0.0 diagonal element. i is the number of the first 0.0 diagonal element.	

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using this function. Normally you should decompose matrix A by calling the 2.13.2 $\left\{\begin{array}{c}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.13.3 $\left\{\begin{array}{c}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{array}\right\}$. In addition, if you have already used 2.13.1 $\left\{\begin{array}{c}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LU decomposition obtained as part of its output.

(b) The unit lower triangular matrix L and the upper triangular matrix U must be stored in band format in array a. However, since the diagonal elements of L always are 1.0, they should not be stored in array a. (See 2.13.1 Figure 2−13.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the 2.13.2 $\left\{\begin{array}{c}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{array}\right\}$, 2.13.3 $\left\{\begin{array}{c}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{array}\right\}$, 2.13.1 $\left\{\begin{array}{c}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{array}\right\}$ functions which perform LU decomposition of matrix A.

2.13.5 ASL_dbbddi, ASL_rbbddi Determinant of a Real Band Matrix

(1) **Function**

ASL_dbbddi or ASL_rbbddi obtains the determinant of the real band matrix A (band type) which has been LU decomposed by the Gauss method.

(2) **Usage**

Double precision:
ierr = ASL_dbbddi (a, lma, n, mu, ml, ipvt, det);
Single precision:
ierr = ASL_rbbddi (a, lma, n, mu, ml, ipvt, det);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D∗ / R∗}	lma×n	Input	Real band matrix A (band type) (See Appendix B) after LU decomposition (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	ipvt	I*	n	Input	Pivoting information ipvt[i-1]: Number of row exchanged with row i in the i-th processing step (See Note (c))
7	det	{D∗ / R∗}	2	Output	Determinant of matrix A (See Note (d))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mu} \leq \mathrm{n} - 1$
$0 \leq \mathrm{ml} \leq \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \leq \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 (See Note (d))
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LU decomposed before using this function. Use any of the 2.13.2 $\left\{\begin{matrix}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{matrix}\right\}$, 2.13.3 $\left\{\begin{matrix}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{matrix}\right\}$, 2.13.1 $\left\{\begin{matrix}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{matrix}\right\}$ functions to perform the decomposition.

(b) The unit lower triangular matrix L and the upper triangular matrix U must be stored in band format in array a. However, since the diagonal elements of L always are 1.0, they need not be stored in array a. (See 2.13.1 Figure 2−13.)

(c) Information about partial pivoting performed during LU decomposition must be stored in ipvt. This information is given by the function that performs the LU decomposition of matrix A.

(d) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(e) Since the inverse matrix of a band matrix generally is a dense matrix, it is not obtained in this function.

2.13.6 ASL_dbbdlx, ASL_rbbdlx
Improving the Solution of Simultaneous Linear Equations (Real Band Matrix)

(1) **Function**

ASL_dbbdlx or ASL_rbbdlx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real band matrix A (band type) as coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_dbbdlx (a, lma, n, mu, ml, alu, b, x, &itol, nit, ipvt, w1);

Single precision:

ierr = ASL_rbbdlx (a, lma, n, mu, ml, alu, b, x, &itol, nit, ipvt, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: {int as for 32bit Integer, long as for 64bit Integer}

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lma×n	Input	Coefficient matrix A (real band matrix, band type) (See Appendix B)
2	lma	I	1	Input	Adjustable dimension of arrays a and alu
3	n	I	1	Input	Order of matrix A
4	mu	I	1	Input	Upper band width of matrix A
5	ml	I	1	Input	Lower band width of matrix A
6	alu	{D*, R*}	lma×n	Input	Coefficient matrix A after LU decomposition (See Note (a))
7	b	{D*, R*}	n	Input	Constant vector $\boldsymbol{b}$
8	x	{D*, R*}	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
9	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
10	nit	I	1	Input	Maximum number of iterations (See Note (d))
11	ipvt	I*	n	Input	Pivoting information (See Note (a))
12	w1	{D*, R*}	n	Work	Work area
13	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $n > 0$

(b) $0 \leq mu \leq n-1$
$0 \leq ml \leq n-1$

(c) $\min(2 \times ml + mu + 1, n + ml) \leq lma$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	The i-th diagonal element of alu was equal to 0.0.	
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) Notes

(a) This function improves the solution obtained by the 2.13.1 $\left\{\begin{array}{c}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{array}\right\}$ or 2.13.4 $\left\{\begin{array}{c}\text{ASL\_dbbdls}\\ \text{ASL\_rbbdls}\end{array}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.13.1 $\left\{\begin{array}{c}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{array}\right\}$, 2.13.2 $\left\{\begin{array}{c}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{array}\right\}$ or 2.13.3 $\left\{\begin{array}{c}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{array}\right\}$ function and the pivoting information at that time must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

(7) Example

(a) Problem

Solve the following simultaneous linear equations and improve the solution.

$$\begin{bmatrix} 10 & 9 & 8 & 7 & 6 & 0 & 0 & 0 & 0 & 0 \\ 9 & 9 & 8 & 7 & 6 & 5 & 0 & 0 & 0 & 0 \\ 8 & 8 & 8 & 7 & 6 & 5 & 4 & 0 & 0 & 0 \\ 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 0 & 0 \\ 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 0 \\ 0 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\ 0 & 0 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 0 & 0 & 0 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 2 \\ 2 \\ 4 \\ -2 \\ -2 \\ 2 \\ 2 \\ 0 \end{bmatrix}$$

(b) Input data

Coefficient matrix a, $\mathrm{lna} = 21, \mathrm{n} = 10, \mathrm{mu} = 4, \mathrm{ml} = 4$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbbdlx */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*sa,wa;
    int ma;
    int n;
    int mu;
    int ml;
    double *b,*sb;
    int itol=0;
    int nnit=0;
    int *kpvt;
    double *wk;


    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbbdlx.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbbdlx ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &mu );
    fscanf( fp, "%d", &ml );

    a = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    sa = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( sa == NULL )
    {
        printf( "no enough memory for array sa\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
```

```
        return -1;
    }

    sb = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( sb == NULL )
    {
        printf( "no enough memory for array sb\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    kpvt = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( kpvt == NULL )
    {
        printf( "no enough memory for array kpvt\n" );
        return -1;
    }

    printf( "\t n  = %6d \n\t mu = %6d \n\t ml = %6d\n", n,mu,ml );

    for( i=0 ; i<ma ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            a[i+ma*j] = 0.0;
            sa[i+ma*j] = 0.0;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &wa );
            if(j-i<=mu && i-j<=ml){
            if(i-ml>=0){
            a[j-i+ml+ma*i]=wa;
            sa[j-i+ml+ma*i]=wa;
            }
            else {
            a[j+ml-i+ma*i]=wa;
            sa[j+ml-i+ma*i]=wa;
            }
            }
        }
    }
    printf( "\n\tCoefficient Matrix a\n\n" );
    for( j=0 ; j<mu+ml+1 ; j++ ){
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        printf( "%8.3g", a[j+ma*i] );
        printf( "\n");
    }

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        sb[i] = b[i];
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );


    ierr = ASL_dbbdsl(a, ma, n, mu, ml, b, kpvt);


    printf( "\n\tOriginal Solution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n",i,b[i] );
    }


    ierr = ASL_dbbdlx(sa, ma, n, mu, ml, a, sb, b, &itol, nnit, kpvt, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tImproved Solution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
```

```
    }

    free( a );
    free( sa );
    free( b );
    free( sb );
    free( wk );
    free( kpvt );

    return 0;
}
```

(d) Output results

```
*** ASL_dbbdlx ***

 ** Input **

 n  =     10
 mu =      4
 ml =      4

Coefficient Matrix a

       0       0       0       0       6       5       4       3       2       1
       0       0       0       7       6       5       4       3       2       1
       0       0       8       7       6       5       4       3       2       1
       0       9       8       7       6       5       4       3       2       1
      10       9       8       7       6       5       4       3       2       1
       9       8       7       6       5       4       3       2       1       0
       8       7       6       5       4       3       2       1       0       0
       7       6       5       4       3       2       1       0       0       0
       6       5       4       3       2       1       0       0       0       0

Constant Vector

       8
       7
       2
       2
       4
      -2
      -2
       2
       2
       0

Original Solution

  x[     0] =         1
  x[     1] =         0
  x[     2] =        -1
  x[     3] =         0
  x[     4] =         1
  x[     5] = -3.78e-17
  x[     6] =        -1
  x[     7] = -6.29e-16
  x[     8] =         1
  x[     9] = 4.88e-16

 ** Output **

ierr =      0

Improved Solution

  x[     0] =         1
  x[     1] = 7.89e-32
  x[     2] =        -1
  x[     3] = -6.57e-32
  x[     4] =         1
  x[     5] = -4.93e-32
  x[     6] =        -1
  x[     7] = 9.86e-32
  x[     8] =         1
  x[     9] = 2.96e-31
```

2.14 POSITIVE SYMMETRIC BAND MATRIX (SYMMETRIC BAND TYPE)

2.14.1 ASL_dbbpsl, ASL_rbbpsl
Simultaneous Linear Equations (Positive Symmetric Band Matrix)

(1) **Function**

ASL_dbbpsl or ASL_rbbpsl uses the Cholesky method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric band matrix A (symmetric band type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbbpsl (a, lma, n, mb, b);
Single precision:
ierr = ASL_rbbpsl (a, lma, n, mb, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lma×n	Input	Positive symmetric band matrix A (symmetric band type) (See Appendix B)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	b	{D* R*}	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	$a[0] \leftarrow \sqrt{a[0]}$ and $b[0] \leftarrow b[0]/a[0]$ are performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step of the $\mathrm{LL^T}$ decomposition of coefficient matrix A. A is nearly singular.	

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector differs, call this function only once and then call function 2.14.4 $\left\{\begin{array}{l}\text{ASL\_dbbpls}\\ \text{ASL\_rbbpls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the $\mathrm{LL^T}$ decomposition of matrix A only once.

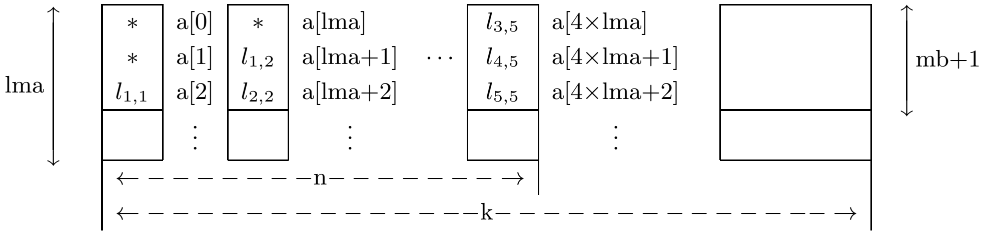
(b) Only the upper triangular matrix L^T is stored in array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a.

Matrix L^T

$$\begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & 0 & 0 \\ 0 & l_{2,2} & l_{2,3} & l_{2,4} & 0 \\ 0 & 0 & l_{3,3} & l_{3,4} & l_{3,5} \\ 0 & 0 & 0 & l_{4,4} & l_{4,5} \\ 0 & 0 & 0 & 0 & l_{5,5} \end{bmatrix}$$

$$\Downarrow$$

Storage status within array a[lma×k]



Remarks

a. Input time values of elements indicated by asterisks (∗) are guaranteed.

b. mb is the band width.

c. lma ≥ mb+1 and k ≥ n must hold.

Figure 2−14 Storage Status of Matrix L^T

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 10 & -2 & 1 & 0 \\ -2 & 9 & -1 & 2 \\ 1 & -1 & 8 & -3 \\ 0 & 2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 72 \\ 9 \\ 62 \\ -4 \end{bmatrix}$$

(b) Input data

Coefficient matrix a, $\mathrm{lma} = 11, \mathrm{n} = 4, \mathrm{mb} = 2$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbbpsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>


int main()
{
    double *a,*wa;
    int ma;
    int n;
    int m;
    double *b;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

    fp = fopen( "dbbpsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbbpsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );

    a = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    wa = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    printf( "\t n = %6d \n\t Band Width = %6d\n", n,m );

    for( i=0 ; i<n ; i++ )
        for( j=0 ; j<n ; j++ )
            fscanf( fp, "%lf", &wa[i+ma*j]);

    for( j=0 ; j<n; j++ ){
        if(i-m<=0){
        for( i=0 ; i<=j ; i++ )
            a[i+(m-j)+ma*j] = wa[i+ma*j];
        } else {
        for( i=j-m ; i<=j ; i++ ){
        if(i>=0)a[i+(m-j)+ma*j] = wa[i+ma*j];
        }
        }
    }
```

```
    printf( "\n\tCoefficient Matrix\n\n" );
    printf( "\t%8c %8c %8.3g %8.3g\n",  SP,SP, a[  ma*2],a[  ma*3] );
    printf( "\t%8c %8.3g %8.3g %8.3g\n",SP,a[1+ma],a[1+ma*2],a[1+ma*3] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n",a[2],a[2+ma],a[2+ma*2],a[2+ma*3] );

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );

    ierr = ASL_dbbpsl(a, ma, n, m, b);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
    }

    free( a );
    free( wa );
    free( b );

    return 0;
}
```

(d) Output results

```
    *** ASL_dbbpsl ***

     ** Input **

     n =      4
     Band Width =      2

    Coefficient Matrix

                              1        2
                  -2        -1        -3
          10        9         8         7

    Constant Vector

          72
           9
          62
          -4

     ** Output **

    ierr =      0

    Solution

      x[     0] =        7
      x[     1] =        3
      x[     2] =        8
      x[     3] =        2
```

2.14.2 ASL_dbbpuu, ASL_rbbpuu

$\mathrm{LL^T}$ Decomposition of a Positive Symmetric Band Matrix

(1) Function

ASL_dbbpuu or ASL_rbbpuu uses the Cholesky method to perform an $\mathrm{LL^T}$ decomposition of the positive symmetric band matrix A (symmetric band type).

(2) Usage

Double precision:
ierr = ASL_dbbpuu (a, lma, n, mb);
Single precision:
ierr = ASL_rbbpuu (a, lma, n, mb);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lma×n	Input	Positive symmetric band matrix A (symmetric band type) (See Appendix B)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (a))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	$\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}$ is performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step.	

(6) **Notes**

(a) The upper triangular matrix L^T is stored in array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a. (See 2.14.1 Figure 2−14.)

2.14.3 ASL_dbbpuc, ASL_rbbpuc

LL^T Decomposition and Condition Number of a Positive Symmetric Band Matrix

(1) **Function**

ASL_dbbpuc or ASL_rbbpuc uses the Cholesky method to perform an LL^T decomposition and obtain the condition number of the positive symmetric band matrix A (symmetric band type).

(2) **Usage**

Double precision:
ierr = ASL_dbbpuc (a, lma, n, mb, &cond, w1);
Single precision:
ierr = ASL_rbbpuc (a, lma, n, mb, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	lma×n	Input	Positive symmetric band matrix A (symmetric band type) (See Appendix B)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LL^T$ (See Note (a))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	cond	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
6	w1	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Work	Work area
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	a[0] ← $\sqrt{\text{a[0]}}$ and cond ← 1.0 are performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	A diagonal element became less than or equal to 0.0 in the i-th processing step.	Processing is aborted. The condition number is not obtained.

(6) Notes

(a) The upper triangular matrix L^T is stored in array a. Since the lower triangular matrix L is calculated from L^T, it is not stored in array a. (See 2.14.1 Figure 2−14.)

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

2.14.4 ASL_dbbpls, ASL_rbbpls Simultaneous Linear Equations (LL^T-Decomposed Positive Symmetric Band Matrix)

(1) **Function**

ASL_dbbpls or ASL_rbbpls solves the simultaneous linear equations $LL^T\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric band matrix A (symmetric band type) which has been LL^T decomposed by the Cholesky method as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_dbbpls (a, lma, n, mb, b);
Single precision:
  ierr = ASL_rbbpls (a, lma, n, mb, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lma×n	Input	Coefficient matrix A after LL^T decomposition (positive symmetric band matrix, symmetric band type) (See Appendix B) (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0]^2 is performed.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	L^T has a diagonal element that is less than or equal to 0.0. i is the number of the first diagonal element that is less than or equal to 0.0.	

(6) **Notes**

(a) The coefficient matrix A must be LL$^{\mathrm{T}}$ decomposed before using this function. Normally, you should decompose matrix A by calling the 2.14.2 $\left\{\begin{matrix}\text{ASL\_dbbpuu}\\ \text{ASL\_rbbpuu}\end{matrix}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.14.3 $\left\{\begin{matrix}\text{ASL\_dbbpuc}\\ \text{ASL\_rbbpuc}\end{matrix}\right\}$. In addition, if you have already used 2.14.1 $\left\{\begin{matrix}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{matrix}\right\}$ to solve simultaneous linear equations having the same coefficient matrix A, you can use the LL$^{\mathrm{T}}$ decomposition obtained as part of its output.

(b) The upper triangular matrix L^T must be stored in array a. Since the lower triangular matrix L is calculated from L^T, it should not be stored in array a. (See 2.14.1 Figure 2−14.)

2.14.5 ASL_dbbpdi, ASL_rbbpdi
Determinant of a Positive Symmetric Band Matrix

(1) Function

ASL_dbbpdi or ASL_rbbpdi obtains the determinant of the positive symmetric band matrix A (symmetric band type) which has been LL^{T} decomposed by the Cholesky method.

(2) Usage

Double precision:
ierr = ASL_dbbpdi (a, lma, n, mb, det);
Single precision:
ierr = ASL_rbbpdi (a, lma, n, mb, det);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lma×n	Input	Upper triangular matrix L^T after LL^{T} decomposition (See Notes (a) and (b))
2	lma	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	det	{D* R*}	2	Output	Determinant of matrix A (See Note (c))
6	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 (See Note (c))
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.

(6) **Notes**

(a) The coefficient matrix A must be LL^{T} decomposed before using this function. Use any of the 2.14.1 $\left\{\begin{array}{l}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{array}\right\}$, 2.14.2 $\left\{\begin{array}{l}\text{ASL\_dbbpuu}\\ \text{ASL\_rbbpuu}\end{array}\right\}$, 2.14.3 $\left\{\begin{array}{l}\text{ASL\_dbbpuc}\\ \text{ASL\_rbbpuc}\end{array}\right\}$ functions to perform the decomposition.

(b) The upper triangular matrix L^T must be stored in array a. Since the lower triangular matrix L is calculated from L^T, it should not be stored in array a. (See 2.14.1 Figure 2−14.)

(c) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(d) Since the inverse matrix of a positive symmetric band matrix generally is a dense matrix, it is not obtained in this function.

2.14.6 ASL_dbbplx, ASL_rbbplx
Improving the Solution of Simultaneous Linear Equations (Positive Symmetric Band Matrix)

(1) Function

ASL_dbbplx or ASL_rbbplx uses an iterative method to improve the solution of the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the positive symmetric band matrix A (symmetric band type) as coefficient matrix.

(2) Usage

Double precision:
ierr = ASL_dbbplx (a, lma, n, mb, all, b, x, &itol, nit, w1);
Single precision:
ierr = ASL_rbbplx (a, lma, n, mb, all, b, x, &itol, nit, w1);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{matrix}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{matrix}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	lma×n	Input	Coefficient matrix A (positive symmetric band matrix, symmetric band type) (See Appendix B)
2	lma	I	1	Input	Adjustable dimension of arrays a and all
3	n	I	1	Input	Order of matrix A
4	mb	I	1	Input	Band width of matrix A
5	all	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	lma×n	Input	Coefficient matrix A after LL^T decomposition (See Note (a))
6	b	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
7	x	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Input	Approximate solution $\boldsymbol{x}$
				Output	Iteratively improved solution $\boldsymbol{x}$
8	itol	I*	1	Input	Number of digits to which solution is to be improved (See Note (b))
				Output	Approximate number of digits to which solution was improved (See Note (c))
9	nit	I	1	Input	Maximum number of iterations (See Note (d))
10	w1	$\left\{\begin{matrix}D*\\R*\end{matrix}\right\}$	n	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $n > 0$

(b) $0 \leq mb \leq n-1$

(c) $mb + 1 \leq lma$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	The solution is not improved.
3000	Restriction (a), (b) or (c) was not satisfied.	Processing is aborted.
$4000 + i$	The i-th diagonal element of array all was less than or equal to 0.0.	
5000	The solution did not converge within the maximum number of iterations.	Processing is aborted after calculating the itol output value.
6000	The solution could not be improved.	

(6) Notes

(a) This function improves the solution obtained by the 2.14.1 $\left\{\begin{array}{c}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{array}\right\}$ or 2.14.4 $\left\{\begin{array}{c}\text{ASL\_dbbpls}\\ \text{ASL\_rbbpls}\end{array}\right\}$ function. Therefore, the coefficient matrix A after it has been decomposed by the 2.14.1 $\left\{\begin{array}{c}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{array}\right\}$, 2.14.2 $\left\{\begin{array}{c}\text{ASL\_dbbpuu}\\ \text{ASL\_rbbpuu}\end{array}\right\}$ or 2.14.3 $\left\{\begin{array}{c}\text{ASL\_dbbpuc}\\ \text{ASL\_rbbpuc}\end{array}\right\}$ function must be given as input.

(b) Solution improvement is repeated until the high-order itol digits of the solution do not change. However, if the following condition is satisfied, solution improvement is repeated until the solution changes in at most the low order 1 bit.

$$\text{itol} \leq 0$$

or

$$\text{itol} \geq -\log_{10}(2 \times \varepsilon) \quad (\varepsilon : \text{Unit for determining error})$$

(c) If the required number of digits have not converged within the iteration count, the approximate number of digits in the improved solution that were unchanged is returned to itol.

(d) If the nit input value is less than or equal to zero, 40 is assumed as the default value.

2.15 REAL TRIDIAGONAL MATRIX (VECTOR TYPE)

2.15.1 ASL_dbtdsl, ASL_rbtdsl

Simultaneous Linear Equations (Real Tridiagonal Matrix)

(1) **Function**

ASL_dbtdsl or ASL_rbtdsl uses the Gauss method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a real tridiagonal matrix A (vector type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbtdsl (sdl, d, sdu, n, b);
Single precision:
ierr = ASL_rbtdsl (sdl, d, sdu, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{matrix}\text{int as for 32bit Integer}\\\text{long as for 64bit Integer}\end{matrix}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	sdl	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Lower subdiagonal component of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B)
				Output	Input-time contents are not saved.
2	d	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Diagonal component of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B)
				Output	Input-time contents are not saved.
3	sdu	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Upper subdiagonal component of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B)
				Output	Input-time contents are not saved.
4	n	I	1	Input	Order of matrix A
5	b	$\begin{Bmatrix}\text{D}*\\\text{R}*\end{Bmatrix}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/d[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	The pivot became 0.0 in the i-th processing step. A is nearly singular.	

(6) **Notes**

(a) This function performs partial pivoting.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 2 & 3 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 14 \\ 20 \\ 11 \end{bmatrix}$$

(b) Input data

Lower subdiagonal component sdl, diagonal component d, upper subdiagonal component sdu, n = 4 and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbtdsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a1,*wa;
    double *a2;
    double *a3;
    int nn;
    double *b;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

    fp = fopen( "dbtdsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbtdsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nn );

    a1 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    wa = ( double * )malloc((size_t)( sizeof(double) * nn * nn ));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\n" );
        return -1;
    }
```

```
        a2 = ( double * )malloc((size_t)( sizeof(double) * nn ));
        if( a2 == NULL )
        {
            printf( "no enough memory for array a2\n" );
            return -1;
        }
        a3 = ( double * )malloc((size_t)( sizeof(double) * nn ));
        if( a3 == NULL )
        {
            printf( "no enough memory for array a3\n" );
            return -1;
        }

        b = ( double * )malloc((size_t)( sizeof(double) * nn ));
        if( b == NULL )
        {
            printf( "no enough memory for array b\n" );
            return -1;
        }

        printf( "\tn = %6d ", nn );
        for( i=0 ; i<nn ; i++ )
        {
            for( j=0 ; j<nn ; j++ )
            {
                fscanf( fp, "%lf", &wa[i+nn*j] );
            }
        }

        for( i=0 ; i<nn ; i++ )
        {
            a2[i]=wa[i+(nn)*i];
        }
        for( i=0 ; i<nn-1 ; i++ )
        {
            a1[i+1]=wa[i+1+(nn)*i];
            a3[i]=wa[i+(nn)*(i+1)];
        }
        printf( "\n\n\tCoefficient Matrix\n\n" );
        printf( "\t%8c %8.3g %8.3g %8.3g\n",  SP,a1[1],a1[2],a1[3] );
        printf( "\t%8.3g %8.3g %8.3g %8.3g\n",a2[0],a2[1],a2[2],a2[3] );
        printf( "\t%8.3g %8.3g %8.3g\n",      a3[0],a3[1],a3[2]       );

        printf( "\n\n\tConstant Vector\n\n" );
        for( i=0 ; i<nn ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "\t%8.3g\n", b[i] );
        }

        fclose( fp );

        ierr = ASL_dbtdsl(a1, a2, a3, nn, b);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution \n\n" );
        for( i=0 ; i<nn ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
        }

        free( a1 );
        free( wa );
        free( a2 );
        free( a3 );
        free( b );

        return 0;
    }
```

(d) Output results

```
        *** ASL_dbtdsl ***

         ** Input **

        n =      4

        Coefficient Matrix

                         1        1        1
               2         2        2        2
               3         3        3

        Constant Vector

               8
              14
              20
              11
```

```
 ** Output **

ierr =      0

Solution

  x[     0] =        1
  x[     1] =        2
  x[     2] =        3
  x[     3] =        4
```

2.15.2 ASL_dbtpsl, ASL_rbtpsl
Simultaneous Linear Equations (Positive Symmetric Tridiagonal Matrix)

(1) **Function**

ASL_dbtpsl or ASL_rbtpsl uses the Gauss method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a positive symmetric tridiagonal matrix A (vector type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbtpsl (d, sd, n, b);
Single precision:
ierr = ASL_rbtpsl (d, sd, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	d	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Input	Diagonal component of coefficient matrix A (positive symmetric tridiagonal matrix, vector type) (See Appendix B)
				Output	Input-time contents are not saved.
2	sd	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Input	Subdiagonal component of coefficient matrix A (positive symmetric tridiagonal matrix, vector type) (See Appendix B)
				Output	Input-time contents are not saved.
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{l}D*\\R*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

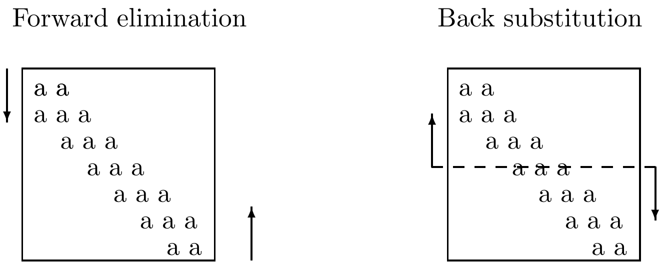
(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/d[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	The diagonal component became 0.0 during processing. *A* is nearly singular.	

(6) Notes

(a) This function performs Gaussian elimination concurrently from both ends of the diagonal of matrix *A*. Therefore, both forward elimination and back substitution are performed repeatedly along the diagonal.

Figure 2−15 Operations for a Positive Symmetric Tridiagonal Matrix


Forward elimination
Back substitution

(7) Example

(a) Problem
Solve the following simultaneous linear equations.

$$\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

(b) Input data
Diagonal component d, subdiagonal component sd, n = 4 and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbtpsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *wa;
    double *a2;
    double *a3;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbtpsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
```

```
        return -1;
    }

    printf( "    *** ASL_dbtpsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nn );

    wa = ( double * )malloc((size_t)( sizeof(double) * nn * nn));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    printf( "\tn = %6d\n", nn );
    for( i=0 ; i<nn ; i++ )
    {
        for( j=0 ; j<nn ; j++ )
        {
            fscanf( fp, "%lf", &wa[i+nn*j] );
        }
    }

    for( i=0 ; i<nn ; i++ )
    {
            a2[i]=wa[i+(nn)*i];
    }
    for( i=0 ; i<nn-1 ; i++ )
    {
            a3[i]=wa[i+(nn)*(i+1)];
    }
    printf( "\n\tCoefficient Matrix\n" );
    for( i=0 ; i<nn ; i++ )
            printf( "\t%8.3g",a2[i]);
            printf( "\n");
    for( i=0 ; i<(nn-1) ; i++ )
            printf( "\t%8.3g",a3[i]);
            printf( "\n");

    printf( "\n\tConstant Vector\n" );
    for( i=0 ; i<nn ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );

    ierr = ASL_dbtpsl(a2, a3, nn, b);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
    }

    free( wa );
    free( a2 );
    free( a3 );
    free( b );

    return 0;
}
```

(d) Output results

```
*** ASL_dbtpsl ***

 ** Input **

n =      4

Coefficient Matrix
      -2          -2          -2          -2
       1           1           1

Constant Vector
      -1
       0
       0
       0

 ** Output **

ierr =      0

Solution

  x[     0] =      0.8
  x[     1] =      0.6
  x[     2] =      0.4
  x[     3] =      0.2
```

2.16 REAL TRIDIAGONAL MATRIX (VECTOR TYPE)

2.16.1 ASL_wbtdsl
Simultaneous Linear Equations (Real Tridiagonal Matrix)

(1) **Function**

ASL_wbtdsl uses the cyclic reduction method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real tridiagonal matrix A (vector type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_wbtdsl (sdl, d, sdu, n, b, iw, w1);
Single precision:
Nothing

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: int as for 32bit Integer
R:Single precision real	C:Single precision complex	I: long as for 64bit Integer

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	sdl	D*	n	Input	Lower subdiagonal components of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B.)
				Output	Input-time contents are not retained.
2	d	D*	n	Input	Diagonal components of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B.)
				Output	Input-time contents are not retained.
3	sdu	D*	n	Input	Upper subdiagonal components of coefficient matrix A (real tridiagonal matrix, vector type) (See Appendix B.)
				Output	Input-time contents are not retained.
4	n	I	1	Input	Order of matrix A
5	b	D*	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
6	iw	I*	See Contents	Work	Work area (See Note (a)) **Size**: $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	$4 \times \mathrm{n}$	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n = 1	b[0] ← b[0]/d[0]
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	a is nearly singular.	

(6) **Notes**

(a) $\lfloor \log_2(\mathrm{n}) \rfloor$ is the value obtained by truncating the fractional part of $\log_2(\mathrm{n})$.

(b) The single-precision version of the function is not supported.

(7) **Example**

(a) Problem
Solve

$$\begin{bmatrix} 6 & 2 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 6 & 2 \\ 0 & 0 & 1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 19 \\ 28 \\ 27 \end{bmatrix}$$

(b) Input data
Lower subdiagonal components sdl, diagonal components d, upper subdiagonal components sdu, n = 4 and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_wbtdsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a1;
    double *a2;
    double *a3;
    int n;
    double *b;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,niwk;
    char SP=' ';
    FILE *fp;

    fp = fopen( "wbtdsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_wbtdsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
```

```
        log2n++;
    }

    a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * (4*n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

    printf( "\tn=%6d\n", n );

    printf( "\n\tCoefficient Matrix\n\n" );
    for( i=1 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a1[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a2[i] );
    }
    for( i=0 ; i<n-1 ; i++ )
    {
        fscanf( fp, "%lf", &a3[i] );
    }
    printf( "\t%8c %8.3g %8.3g %8.3g\n",   SP,a1[1],a1[2],a1[3] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n", a2[0],a2[1],a2[2],a2[3] );
    printf( "\t%8.3g %8.3g %8.3g\n",       a3[0],a3[1],a3[2]       );

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "    %8.3g\n", b[i] );
    }

    fclose( fp );

    ierr = ASL_wbtdsl(a1, a2, a3, n, b, iw, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\tx[%6d  ] = %8.3g\n", i,b[i] );
    }

    free( a1 );
    free( a2 );
    free( a3 );
    free( b );
    free( iw );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_wbtdsl ***

 ** Input **

n=     4

Coefficient Matrix

              1        1        1
       6      6        6        6
       2      2        2

Constant Vector

      10
      19
      28
      27

 ** Output **

ierr =      0

Solution

x[     0  ] =        1
x[     1  ] =        2
x[     2  ] =        3
x[     3  ] =        4
```

2.16.2 ASL_wbtdls Simultaneous Linear Equations (Real Tridiagonal Matrix after Reduction Operations)

(1) **Function**

ASL_wbtdls uses the cyclic reduction method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the real tridiagonal matrix A (vector type) after reduction operations have been performed as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_wbtdls (sdl, d, sdu, n, b, iw, w1);
Single precision:
Nothing

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

I: $\left\{ \begin{array}{ll} \text{int} & \text{as for 32bit Integer} \\ \text{long} & \text{as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	sdl	D*	n	Input	Lower subdiagonal components of coefficient matrix A after reduction operations (real tridiagonal matrix, vector type) (See Appendix B.) (See Note (a))
2	d	D*	n	Input	Diagonal components of coefficient matrix A after reduction operations (real tridiagonal matrix, vector type) (See Appendix B.) (See Note (a))
3	sdu	D*	n	Input	Upper subdiagonal components of coefficient matrix A after reduction operations (real tridiagonal matrix, vector type) (See Appendix B.) (See Note (a))
4	n	I	1	Input	Order of matrix A
5	b	D*	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
6	iw	I*	See Contents	Input	Reduction operation information (See Notes (a) and (b)) **Size**: $3 \times \lfloor \log_2(\text{n}) \rfloor + 1$
7	w1	D*	$4 \times$ n	Input	Reduction operation information (See Note (a))
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n = 1	b[0] ← b[0]/d[0]
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000	a is nearly singular (Only when n = 1)	

(6) **Notes**

(a) This function can be used to solve multiple sets of simultaneous linear equations having the same coefficient matrix but different constant vectors. First, use 2.16.1 ASL_wbtdsl to perform reduction operations for the coefficient matrix and obtain solutions.
Then, repeatedly use this function to only obtain solutions for the different constant vectors. The contents of arguments sdl, d, sdu, iw, and w1 from this function must be retained since they become input values for this function 2.16.1 ASL_wbtdsl.

(b) $\lfloor \log_2(n) \rfloor$ is the value obtained by truncating the fractional part of $\log_2(n)$.

(c) The single-precision version of the function is not supported.

(7) **Example**

(a) Problem
Solve simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}_1$ and $A\boldsymbol{y} = \boldsymbol{b}_2$ with unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$ where,

$$A = \begin{bmatrix} 6 & 2 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 6 & 2 \\ 0 & 0 & 1 & 6 \end{bmatrix}, \ \boldsymbol{b}_1 = \begin{bmatrix} 10 \\ 19 \\ 28 \\ 27 \end{bmatrix}, \ \boldsymbol{b}_2 = \begin{bmatrix} 30 \\ 26 \\ 17 \\ 8 \end{bmatrix}$$

(b) Input data
Lower subdiagonal components sdl, diagonal components d, upper subdiagonal components sdu, n = 4 and constant vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$.

(c) Main program

```
/*      C interface example for ASL_wbtdls */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a1;
    double *a2;
    double *a3;
    int n;
    double *b1;
    double *b2;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,niwk;
    char SP=' ';
    FILE *fp;

    fp = fopen( "wbtdls.dat", "r" );
    if( fp == NULL )
    {
```

```
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_wbtdls ***\n" );
printf( "\n     ** Input **\n" );

fscanf( fp, "%d", &n );
/* get floor(log2(n)) */
nn=n;
log2n=0;
while (nn>1)
{
    nn=(nn>>1);
    log2n++;
}

a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a1 == NULL )
{
    printf( "no enough memory for array a1\n" );
    return -1;
}
a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a2 == NULL )
{
    printf( "no enough memory for array a2\n" );
    return -1;
}
a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a3 == NULL )
{
    printf( "no enough memory for array a3\n" );
    return -1;
}
b1 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( b1 == NULL )
{
    printf( "no enough memory for array b1\n" );
    return -1;
}
b2 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( b2 == NULL )
{
    printf( "no enough memory for array b2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * (4*n) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
niwk=3*log2n+1;
iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iw == NULL )
{
    printf( "no enough memory for array iw\n" );
    return -1;
}

printf( "\n\tn=%6d\n", n );

printf( "\n\tCoefficient Matrix\n\n" );
for( i=1 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &a1[i] );
}
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &a2[i] );
}
for( i=0 ; i<n-1 ; i++ )
{
    fscanf( fp, "%lf", &a3[i] );
}
printf( "\t%8c %8.3g %8.3g %8.3g\n",   SP,a1[1],a1[2],a1[3] );
printf( "\t%8.3g %8.3g %8.3g %8.3g\n", a2[0],a2[1],a2[2],a2[3] );
printf( "\t%8.3g %8.3g %8.3g\n",       a3[0],a3[1],a3[2]       );

printf( "\n\tConstant Vector\n\n" );
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &b1[i] );
}
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &b2[i] );
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t%8.3g %8.3g\n", b1[i],b2[i] );
```

```
    }

    fclose( fp );

    ierr = ASL_wbtdsl(a1, a2, a3, n, b1, iw, wk);

    ierr = ASL_wbtdls(a1, a2, a3, n, b2, iw, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution x\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d  ] = %8.3g\n", i,b1[i] );
    }

    printf( "\n\tSolution y\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  y[%6d  ] = %8.3g\n", i,b2[i] );
    }

    free( a1 );
    free( a2 );
    free( a3 );
    free( b1 );
    free( b2 );
    free( iw );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_wbtdls ***

 ** Input **

n=     4

Coefficient Matrix

                1        1        1
       6        6        6        6
       2        2        2

Constant Vector

      10       30
      19       26
      28       17
      27        8

 ** Output **

ierr =      0

Solution x

  x[     0  ] =        1
  x[     1  ] =        2
  x[     2  ] =        3
  x[     3  ] =        4

Solution y

  y[     0  ] =        4
  y[     1  ] =        3
  y[     2  ] =        2
  y[     3  ] =        1
```

2.17 FIXED COEFFICIENT REAL TRIDIAGONAL MATRIX (SCALAR TYPE)

2.17.1 ASL_wbtcsl

Simultaneous Linear Equations (Fixed Coefficient Real Tridiagonal Matrix)

(1) **Function**

ASL_wbtcsl uses the cyclic reduction method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the fixed coefficient real tridiagonal matrix a (scalar type) as coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_wbtcsl (&d, &sd, n, b, isw, iw, w1);

Single precision:

Nothing

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer, long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	d	D*	1	Input	Diagonal components of coefficient matrix A (fixed coefficient real tridiagonal matrix, scalar type) (See Note (a))
				Output	Input-time contents are not retained.
2	sd	D*	1	Input	Subdiagonal components of coefficient matrix A (fixed coefficient real tridiagonal matrix, scalar type) (See Note (a))
				Output	Input-time contents are not retained.
3	n	I	1	Input	Order of matrix A
4	b	D*	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
5	isw	I	1	Input	Specifies the type of coefficient matrix A. (See Note (a)) isw=1, 2, 3 or 4
6	iw	I*	See Contents	Work	Work area (See Note (b)) **Size**: $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	See Contents	Work	Work area **Size**: $\mathrm{n} + 3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 2$
8	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $n > 0$

(b) $\mathrm{isw} \in \{1, 2, 3, 4\}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	$n = 1$	$b[0] \leftarrow b[0]/d[0]$
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000	A is nearly singular.	

(6) Notes

(a) Coefficient matrix A is a fixed coefficient real tridiagonal matrix of the types shown below corresponding to isw = 1, 2, 3, and 4.

For isw = 1

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ 0 & & & \cdot & \mathrm{d} & \mathrm{sd} \\ & & & & \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 2

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ 0 & & & \cdot & \mathrm{d} & \mathrm{sd} \\ & & & & 2 \times \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 3

$$\begin{bmatrix} \mathrm{d} & 2 \times \mathrm{sd} & & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ 0 & & & \cdot & \mathrm{d} & \mathrm{sd} \\ & & & & \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 4

$$\left[\begin{array}{ccccccc}
\mathrm{d} & 2\times\mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & & \\
 & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\
 & & \cdot & \cdot & \cdot & & \\
 & & & \cdot & \cdot & & \cdot \\
 & 0 & & \cdot & \mathrm{d} & & \mathrm{sd} \\
 & & & & 2\times\mathrm{sd} & & \mathrm{d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

Coefficient matrices of the types shown above appear when discretizing the Dirichlet or Neumann boundary value problem.

(b) $\lfloor \log_2(\mathrm{n}) \rfloor$ is the value obtained by truncating the fractional part of $\log_2(\mathrm{n})$.

(c) The single-precision version of the function is not supported.

(7) **Example**

(a) Problem

Solve

$$\begin{bmatrix} 6 & 2 & 0 & 0 \\ 2 & 6 & 2 & 0 \\ 0 & 2 & 6 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 10 \\ 10 \\ 8 \end{bmatrix}$$

(b) Input data

Diagonal components d, subdiagonal components sd, n = 4, isw = 1, and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_wbtcsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a1;
    double *a2;
    double *a3;
    double d;
    double sd;
    int n;
    double *b;
    int isw;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,nwk,niwk;
    char SP=' ';
    FILE *fp;

    fp = fopen( "wbtcsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_wbtcsl ***\n" );
    printf( "\n     ** Input **\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &isw );
    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
        log2n++;
```

```
    }

    a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    nwk=n+3*log2n+2;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

    printf( "\n\tn  =%6d\n", n );
    printf( "\tisw=%6d\n", isw );

    printf( "\n\tCoefficient Matrix\n\n" );
    for( i=1 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a1[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a2[i] );
    }
    for( i=0 ; i<n-1 ; i++ )
    {
        fscanf( fp, "%lf", &a3[i] );
    }
    printf( "\t%8c %8.3g %8.3g %8.3g\n",   SP, a1[1],a1[2],a1[3] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n",a2[0],a2[1],a2[2],a2[3] );
    printf( "\t%8.3g %8.3g %8.3g\n",      a3[0],a3[1],a3[2]       );

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    d=a2[0];
    sd=a1[1];

    fclose( fp );

    ierr = ASL_wbtcsl(&d, &sd, n, b, isw, iw, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d  ] = %8.3g\n", i,b[i] );
    }

    free( a1 );
    free( a2 );
    free( a3 );
    free( b );
    free( iw );
    free( wk );

    return 0;
```

```
}
```

(d) Output results

```
*** ASL_wbtcsl ***

 ** Input **

n  =     4
isw=     1

Coefficient Matrix

                2        2        2
       6        6        6        6
       2        2        2

Constant Vector

       8
      10
      10
       8

 ** Output **

ierr =      0

Solution

  x[     0  ] =        1
  x[     1  ] =        1
  x[     2  ] =        1
  x[     3  ] =        1
```

2.17.2 ASL_wbtcls
Simultaneous Linear Equations (Fixed Coefficient Real Tridiagonal Matrix after Reduction Operations)

(1) **Function**

ASL_wbtcls uses the cyclic reduction method to solve the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having the fixed coefficient real tridiagonal matrix A (scalar type) after reduction operations have been performed as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_wbtcls (d, sd, n, b, isw, iw, w1);
Single precision:
Nothing

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	d	D	1	Input	Diagonal components of coefficient matrix A after reduction operations (fixed coefficient real tridiagonal matrix, scalar type)(See Notes (a) and (b))
2	sd	D	1	Input	Subdiagonal components of coefficient matrix A after reduction operations (fixed coefficient real tridiagonal matrix, scalar type)(See Notes (a) and (b))
3	n	I	1	Input	Order of matrix A
4	b	D*	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution vector $\boldsymbol{x}$
5	isw	I	1	Input	Specifies the type of coefficient matrix A (See Note (b)) isw=1, 2, 3 or 4
6	iw	I*	See Contents	Input	Reduction operation information (See Notes (a) and (c)) **Size**: $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	See Contents	Input	Reduction operation information (See Notes (a) and (c)) **Size**: $\mathrm{n} + 3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 2$
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{n} > 0$

(b) $\mathrm{isw} \in \{1, 2, 3, 4\}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	$\mathrm{n} = 1$	$\mathrm{b}[0] \leftarrow \mathrm{b}[0]/\mathrm{d}[0]$
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
4000	A is nearly singular. (Only when $\mathrm{n} = 1$)	

(6) **Notes**

(a) This function can be used to solve multiple sets of simultaneous linear equations having the same coefficient matrix but different constant vectors. First, use 2.17.1 ASL_wbtcsl to perform reduction operations for the coefficient matrix and obtain solutions. Then, repeatedly use this function to only obtain solutions for the different constant vectors. The contents of arguments d, sd, iw, and w1 from 2.17.1 ASL_wbtcsl must be retained since they become input values for this function.

(b) Coefficient matrix A is a fixed coefficient real tridiagonal matrix of the types shown below corresponding to isw = 1, 2, 3, and 4.
For isw = 1

$$\begin{bmatrix} \mathrm{d} & & \mathrm{sd} & & & & 0 & \\ \mathrm{sd} & & \mathrm{d} & \mathrm{sd} & & & & \\ & & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & & \cdot \\ & 0 & & & \cdot & \mathrm{d} & & \mathrm{sd} \\ & & & & & \mathrm{sd} & & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 2

$$\begin{bmatrix} \mathrm{d} & & \mathrm{sd} & & & & 0 & \\ \mathrm{sd} & & \mathrm{d} & \mathrm{sd} & & & & \\ & & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & & \cdot \\ & 0 & & & \cdot & \mathrm{d} & & \mathrm{sd} \\ & & & & & 2 \times \mathrm{sd} & & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 3

$$\begin{bmatrix} \mathrm{d} & 2 \times \mathrm{sd} & & & & 0 & \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & & \cdot \\ & 0 & & \cdot & \mathrm{d} & & \mathrm{sd} \\ & & & & \mathrm{sd} & & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

For isw = 4

$$\left[\begin{array}{ccccccc} \mathrm{d} & 2\times\mathrm{sd} & & & & 0 & \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & & \cdot \\ & 0 & & \cdot & \mathrm{d} & & \mathrm{sd} \\ & & & & 2\times\mathrm{sd} & & \mathrm{d} \end{array}\right], \mathrm{d}\neq 0, \mathrm{sd}\neq 0$$

Coefficient matrices of the types shown above appear when discretizing the Dirichlet or Neumann boundary value problem.

(c) $\lfloor \log_2(\mathrm{n}) \rfloor$ is the value obtained by truncating the fractional part of $\log_2(\mathrm{n})$.

(d) The single-precision version of the function is not supported.

(7) **Example**

(a) Problem

Solve simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}_1$ and $A\boldsymbol{y} = \boldsymbol{b}_2$ with unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$. Where,

$$A = \begin{bmatrix} 6 & 2 & 0 & 0 \\ 2 & 6 & 2 & 0 \\ 0 & 2 & 6 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix}, \ \boldsymbol{b}_1 = \begin{bmatrix} 8 \\ 10 \\ 10 \\ 8 \end{bmatrix}, \ \boldsymbol{b}_2 = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 30 \end{bmatrix}$$

(b) Input data

Diagonal components d, subdiagonal components sd, n = 4, isw = 1 and constant vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$.

(c) Main program

```
/*      C interface example for ASL_wbtcls */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    double d;
    double sd;
    int n;
    double *b1;
    double *b2;
    int isw;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,nwk,niwk;
    FILE *fp;

    fp = fopen( "wbtcls.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_wbtcls ***\n" );
    printf( "\n     ** Input **\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &isw );

    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
        log2n++;
    }
```

```
    b1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b1 == NULL )
    {
        printf( "no enough memory for array b1\n" );
        return -1;
    }
    b2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b2 == NULL )
    {
        printf( "no enough memory for array b2\n" );
        return -1;
    }
    nwk=n+3*log2n+2;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

    printf( "\n\tn  =%6d\n", n );
    printf( "\tisw=%6d\n", isw );
    fscanf( fp, "%lf", &d );
    fscanf( fp, "%lf", &sd );
    printf( "\n\tCoefficient Matrix\n\n" );
    printf( "\t%8.3g\n", d );
    printf( "\t%8.3g\n", sd );
    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b1[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b2[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g %8.3g\n", b1[i],b2[i] );
    }

    fclose( fp );


    ierr = ASL_wbtcsl(&d, &sd, n, b1, isw, iw, wk);
    ierr = ASL_wbtcls(d, sd, n, b2, isw, iw, wk);


    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution x\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d  ] = %8.3g\n", i,b1[i] );
    }

    printf( "\n\tSolution y\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  y[%6d  ] = %8.3g\n", i,b2[i] );
    }

    free( b1 );
    free( b2 );
    free( iw );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_wbtcls ***

 ** Input **

n  =     4
isw=     1

Coefficient Matrix

       6
```

```
       2

Constant Vector

       8       10
      10       20
      10       30
       8       30

 ** Output **

ierr =      0

Solution x

  x[     0  ] =         1
  x[     1  ] =         1
  x[     2  ] =         1
  x[     3  ] =         1

Solution y

  y[     0  ] =         1
  y[     1  ] =         2
  y[     2  ] =         3
  y[     3  ] =         4
```

2.18 VANDERMONDE MATRIX AND TOEPLITZ MATRIX

2.18.1 ASL_dbtosl, ASL_rbtosl
Simultaneous Linear Equations (Toeplitz Matrix)

(1) **Function**

The Toeplitz matrix R of order n consisting of $2 \times n - 1$ elements r_k $(k = -n+1, -n+2, \cdots, n-1)$ is represented as follows.

$$R = \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\ r_1 & r_0 & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_{-1} \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

The ASL_dbtosl or ASL_rbtosl solves the following simultaneous linear equations $R\boldsymbol{x} = \boldsymbol{b}$ having this Toeplitz matrix R as coefficient matrix:

$$\sum_{j=1}^{n} r_{i-j} x_j = b_i \quad (i = 1, \cdots, n)$$

or the following simultaneous linear equations $R^T\boldsymbol{x} = \boldsymbol{b}$ having the matrix R^T as coefficient matrix:

$$\sum_{j=1}^{n} r_{j-i} x_j = b_i \quad (i = 1, \cdots, n)$$

(2) **Usage**

Double precision:

ierr = ASL_dbtosl (r, n, b, x, w, isw);

Single precision:

ierr = ASL_rbtosl (r, n, b, x, w, isw);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	r	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	2 × n − 1	Input	Components r_k $(k = -n+1, -n+2, \cdots, n-1)$ of Toeplitz matrix R
2	n	I	1	Input	Order of matrix R
3	b	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
4	x	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Output	Solution vector $\boldsymbol{x}$
5	w	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	2 × n	Work	Work area
6	isw	I	1	Input	Processing switch 1: Solve $R\boldsymbol{x} = \boldsymbol{b}$ 2: Solve $R^T\boldsymbol{x} = \boldsymbol{b}$
7	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) isw $\in \{1, 2\}$

(b) n > 0

(c) r[n − 1] $\neq 0$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	x[0] ← b[0]/r[n − 1] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	The divisor $x^{(de)}$ was zero.	
4010	The divisor $g^{(de)}$ was zero.	

(6) Notes

(a) Since this function makes practical use of the properties of the matrix, it is superior to 2.2.2 $\left\{\begin{array}{l}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ in terms of memory usage and calculation efficiency. However, the solution may not be obtained theoretically even if the matrix is regular. In particular, if $x^{(de)}$ or $g^{(de)}$, which are divisors, are close to zero during the calculation process, the reliability of the solution obtained will not be guaranteed. (See Section 2.1.3 “Algorithms Used”.)

(7) **Example**

(a) ProblemSolve the following simultaneous linear equations.

$$\begin{bmatrix} r_0 & r_{-1} & r_{-2} & r_{-3} \\ r_1 & r_0 & r_{-1} & r_{-2} \\ r_2 & r_1 & r_0 & r_{-1} \\ r_3 & r_2 & r_1 & r_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

(b) Input data

Array $\mathrm{r} = \{\mathrm{r}_{-3}, \mathrm{r}_{-2}, \mathrm{r}_{-1}, \mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3\}$ in which matrix R components are stored, n=4, isw=1 and constant vector $\boldsymbol{b}$.

Note The same problem can be solved by storing matrix R components as $\mathrm{r} = \{\mathrm{r}_3, \mathrm{r}_2, \mathrm{r}_1, \mathrm{r}_0, \mathrm{r}_{-1}, \mathrm{r}_{-2}, \mathrm{r}_{-3}\}$ and setting isw=2.

(c) Main program

```
/*      C interface example for ASL_dbtosl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *r;
    int n;
    double *b;
    double *x;
    double *w;
    int isw;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

    fp = fopen( "dbtosl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbtosl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &isw );
    fscanf( fp, "%d", &n );
    printf( "\t isw = %6d n = %6d\n", isw, n );

    r = ( double * )malloc((size_t)( sizeof(double) * (2*lna-1) ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    w = ( double * )malloc((size_t)( sizeof(double) * (2*lna) ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    for( i=0 ; i<2*n-1 ; i++ )
    {
        fscanf( fp, "%lf", &r[i] );
    }

    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
```

```
            printf( "\t" );
            for( j=0 ; j<n ; j++ )
            {
                printf( "%8.3g  ", r[n+i-j-1] );
            }
            printf( "\n" );
        }
        printf( "\n\tConstant Vector\n\n");
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "%8.3g  ", b[i] );
        }
        fclose( fp );

        ierr = ASL_dbtosl(r, n, b, x, w, isw);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tSolution\n\n" );
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "%8.3g  ", x[i] );
        }
        printf( "\n" );

        free( r );
        free( b );
        free( x );
        free( w );

        return 0;
    }
```

(d) Output results

```
        *** ASL_dbtosl ***

         ** Input **

         isw =      1 n =      4

        Coefficient Matrix

               1        -2        -3        -4
               2         1        -2        -3
               3         2         1        -2
               4         3         2         1

        Constant Vector

              -8        -2         4        10
         ** Output **

        ierr =      0

        Solution

               1         1         1         1
```

2.18.2 ASL_dbtssl, ASL_rbtssl Simultaneous Linear Equations (Symmetric Toeplitz Matrix)

(1) Function

The symmetric Toeplitz matrix R of order n consisting of n elements r_k $(k = 0, 1, \cdots, n-1)$ is represented as follows.

$$R = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-2} & r_{n-1} \\ r_1 & r_0 & r_1 & \cdots & r_{n-3} & r_{n-2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_1 \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

ASL_dbtssl or ASL_rbtssl solves the following simultaneous linear equations $R\boldsymbol{x} = \boldsymbol{b}$ having this symmetric Toeplitz matrix R as coefficient matrix:

$$\sum_{j=1}^{n} r_{|i-j|} x_j = b_i \quad (i = 1, \cdots, n)$$

(2) Usage

Double precision:
ierr = ASL_dbtssl (r, n, b, x, w);
Single precision:
ierr = ASL_rbtssl (r, n, b, x, w);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{ \begin{array}{ll} \text{int} & \text{as for 32bit Integer} \\ \text{long} & \text{as for 64bit Integer} \end{array} \right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	r	$\left\{ \begin{array}{c} \text{D}* \\ \text{R}* \end{array} \right\}$	n	Input	Components r_k $(k = 0, 1, \cdots, n-1)$ of symmetric Toeplitz matrix R
2	n	I	1	Input	Order of matrix R
3	b	$\left\{ \begin{array}{c} \text{D}* \\ \text{R}* \end{array} \right\}$	n	Input	Constant vector $\boldsymbol{b}$
4	x	$\left\{ \begin{array}{c} \text{D}* \\ \text{R}* \end{array} \right\}$	n	Output	Solution vector $\boldsymbol{x}$
5	w	$\left\{ \begin{array}{c} \text{D}* \\ \text{R}* \end{array} \right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) n > 0

(b) r[0] ≠ 0

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	x[0] ← b[0]/r[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	The divisor $x^{(de)}$ was zero.	

(6) Notes

(a) Since this function makes practical use of the properties of the matrix, it is superior to 2.2.2 $\left\{\begin{array}{l}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ in terms of memory usage and calculation efficiency. However, the solution may not be obtained theoretically even if the matrix is regular. In particular, if $x^{(de)}$, which is divisor, is close to zero during the calculation process, the reliability of the solution obtained will not be guaranteed. (See Section 2.1.3 “Algorithms Used”).

(7) Example

(a) ProblemSolve the following simultaneous linear equations.

$$\begin{bmatrix} r_0 & r_1 & r_2 & r_3 \\ r_1 & r_0 & r_1 & r_2 \\ r_2 & r_1 & r_0 & r_1 \\ r_3 & r_2 & r_1 & r_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

(b) Input data
Array r = $\{\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3\}$ in which matrix R components are stored, n=4 and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbtssl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *r;
    int n;
    double *b;
    double *x;
    double *w;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

    fp = fopen( "dbtssl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbtssl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &n );
```

```
    printf( "\t n = %6d\n", n );

    r = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    w = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &r[i] );
    }
    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            printf( "%8.3g  ", r[abs(i-j)] );
        }
        printf( "\n" );
    }
    printf( "\n\tConstant Vector\n\n");
    printf( "\t" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "%8.3g  ", b[i] );
    }
    fclose( fp );

    ierr = ASL_dbtssl(r, n, b, x, w);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    printf( "\t" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "%8.3g  ", x[i] );
    }
    printf( "\n" );

    free( r );
    free( b );
    free( x );
    free( w );

    return 0;
}
```

(d) Output results

```
*** ASL_dbtssl ***

 ** Input **

 n =      4

Coefficient Matrix

       1         2         3         4
       2         1         2         3
       3         2         1         2
       4         3         2         1

Constant Vector

      10         8         8        10
 ** Output **

ierr =      0

Solution

       1         1         1         1
```

2.18.3 ASL_dbvmsl, ASL_rbvmsl Simultaneous Linear Equations (Vandermonde Matrix)

(1) **Function**

The Vandermonde matrix V of order n consisting of n different elements v_k $(k = 1, 2, \cdots, n)$ is represented as follows.

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & \cdots & v_1^{n-2} & v_1^{n-1} \\ 1 & v_2 & v_2^2 & \cdots & v_2^{n-2} & v_2^{n-1} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 1 & v_{n-1} & v_{n-1}^2 & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\ 1 & v_n & v_n^2 & \cdots & v_n^{n-2} & v_n^{n-1} \end{bmatrix}$$

ASL_dbvmsl or ASL_rbvmsl solves the following simultaneous linear equations $V\boldsymbol{x} = \boldsymbol{b}$ having this Vandermonde matrix V as coefficient matrix:

$$\sum_{j=1}^{n} v_i^{j-1} x_j = b_i \quad (i = 1, \cdots, n)$$

or the following simultaneous linear equations $V^T\boldsymbol{x} = \boldsymbol{b}$ having the matrix V^T as coefficient matrix:

$$\sum_{j=1}^{n} v_j^{i-1} x_j = b_i \quad (i = 1, \cdots, n)$$

The simultaneous linear equations having the Vandermonde matrix as the coefficient matrix essentially are ill-conditioned, and it is difficult to obtain a solution with good precision except when n is extremely small (See Note (a)).

(2) **Usage**

Double precision:
ierr = ASL_dbvmsl (v, n, b, x, w, isw);
Single precision:
ierr = ASL_rbvmsl (v, n, b, x, w, isw);

(3) Arguments and Return Value

D:Double precision real	Z:Double precision complex	I: $\left\{\begin{array}{ll}\text{int} & \text{as for 32bit Integer}\\ \text{long} & \text{as for 64bit Integer}\end{array}\right\}$
R:Single precision real	C:Single precision complex	

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	v	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Components v_k $(k = 1, 2, \cdots, n)$ of Vandermonde matrix V
2	n	I	1	Input	Order n of matrix V
3	b	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
4	x	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Output	Solution vector $\boldsymbol{x}$
5	w	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Work	Work area (See Note (b))
6	isw	I	1	Input	Processing switch 1: Solve $V\boldsymbol{x} = \boldsymbol{b}$ 2: Solve $V^T\boldsymbol{x} = \boldsymbol{b}$
7	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) isw $\in \{1, 2\}$

(b) n > 0

(c) v[i $-$ 1] $\neq 0$ (i $= 1, \ldots,$ n)

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n = 1 is specified.	x[0] ← b[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000	A division by zero occurred during an operation.	

(6) Notes

(a) Since this function makes practical use of the properties of the matrix, it is superior to 2.2.2 $\left\{\begin{array}{c}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{array}\right\}$ in terms of memory usage. However, the part that obtains the solution via the inverse matrix without performing pivoting may be inferior in terms of calculation precision. In any event, the simultaneous linear equations having the Vandermonde matrix as the coefficient matrix essentially are ill-conditioned, and it is difficult to obtain a solution with good precision except when n is extremely small. When double precision function is used, the maximum value of n, which is the size of the problem for which solutions can be obtained, is about 15. Also, the simultaneous linear equations having V^T as coefficient matrix usually has better properties than the simultaneous linear equations having V as coefficient matrix.

(b) The coefficients w_j of the terms of the master polynomial $P(x)$ defined by the following equation are stored in Work area w.

$$P(x) = \prod_{k=1}^{n} (x - v_k) = x^n + w_1 x^{n-1} + \cdots + w_{n-1} x + w_n$$

(7) **Example**

(a) ProblemSolve the following simultaneous linear equations.

$$\begin{bmatrix} 1 & v_1 & v_1^2 & v_1^3 \\ 1 & v_2 & v_2^2 & v_2^3 \\ 1 & v_3 & v_3^2 & v_3^3 \\ 1 & v_4 & v_4^2 & v_4^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

(b) Input data

Array v = $\{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \mathrm{v}_4\}$ in which matrix V components are stored, n=4, isw=1 and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbvmsl */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    double *v;
    int n;
    double *b;
    double *x;
    double *w;
    double t;
    int isw;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

    fp = fopen( "dbvmsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbvmsl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &isw );
    fscanf( fp, "%d", &n );
    printf( "\t isw = %6d n = %6d\n", isw, n );

    v = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( v == NULL )
    {
        printf( "no enough memory for array v\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( x == NULL )
```

```
        {
            printf( "no enough memory for array x\n" );
            return -1;
        }
        w = ( double * )malloc((size_t)( sizeof(double) * lna ));
        if( w == NULL )
        {
            printf( "no enough memory for array w\n" );
            return -1;
        }
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &v[i] );
        }

        printf( "\n\tCoefficient Matrix\n\n");
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<n ; j++ )
            {
                t=j;
                printf( "%8.3g  ", pow(v[i], t) );
            }
            printf( "\n" );
        }
        printf( "\n\tConstant Vector\n\n");
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "%8.3g  ", b[i] );
        }
        fclose( fp );

        ierr = ASL_dbvmsl(v, n, b, x, w, isw);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tSolution\n\n" );
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "%8.3g  ", x[i] );
        }
        printf( "\n" );

        free( v );
        free( b );
        free( x );
        free( w );

        return 0;
    }
```

(d) Output results

```
    *** ASL_dbvmsl ***

     ** Input **

     isw =      1 n =      4

    Coefficient Matrix

           1         2         4         8
           1         3         9        27
           1         4        16        64
           1         5        25       125

    Constant Vector

          15        40        85       156
     ** Output **

    ierr =      0

    Solution

           1         1         1         1
```

2.19 REAL UPPER TRIANGULAR MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

2.19.1 ASL_dbtusl, ASL_rbtusl

Simultaneous Linear Equations (Real Upper Triangular Matrix)

(1) **Function**

ASL_dbtusl or ASL_rbtusl solves the simultaneous linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ having a real upper triangular matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
ierr = ASL_dbtusl (a, lna, n, b);
Single precision:
ierr = ASL_rbtusl (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex

$$\text{I: } \left\{ \begin{array}{l} \text{int as for 32bit Integer} \\ \text{long as for 64bit Integer} \end{array} \right\}$$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	lna×n	Input	Coefficient matrix A (real upper triangular matrix, two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\left\{\begin{array}{l}\text{D}*\\\text{R}*\end{array}\right\}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
2100	There existed the diagonal element which was close to zero in the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	The coefficient matrix A has a 0.0 diagonal element. A is singular.	

(6) Notes

None

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 1 & 2 & -3 & 4 \\ 0 & 4 & -1 & 1 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -10 \\ -9 \\ -3 \\ -16 \end{bmatrix}$$

(b) Input data

Coefficient matrix A, $\mathrm{lna} = 11, \mathrm{n} = 4$ and constant vector $\boldsymbol{b}$.

(c) Main program

```
/*      C interface example for ASL_dbtusl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbtusl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbtusl ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &nn );

    a = ( double * )malloc((size_t)( sizeof(double) * (na*nn) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
```

```
        b = ( double * )malloc((size_t)( sizeof(double) * nn ));
        if( b == NULL )
        {
            printf( "no enough memory for array b\n" );
            return -1;
        }

        printf( "\t n = %6d\n", nn );
        printf( "\n\tCoefficient Matrix\n" );
        for( i=0 ; i<nn ; i++ )
        {
            for( j=0 ; j<nn ; j++ )
            {
                fscanf( fp, "%lf", &a[i+na*j] );
                printf( "\t%8.3g", a[i+na*j] );
            }
            printf( "\n" );
        }

        printf( "\n\tConstant Vector\n" );
        for( i=0 ; i<nn ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "\t%8.3g\n", b[i] );
        }

        fclose( fp );

        ierr = ASL_dbtusl(a, na, nn, b);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution\n" );
        for( i=0 ; i<nn ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
        }

        free( a );
        free( b );

        return 0;
    }
```

(d) Output results

```
        *** ASL_dbtusl ***

         ** Input **

         n =      4

        Coefficient Matrix
               1           2          -3           4
               0           4          -1           1
               0           0           5          -1
               0           0           0           8

        Constant Vector
             -10
              -9
              -3
             -16

         ** Output **

        ierr =      0

        Solution
          x[     0] =       -1
          x[     1] =       -2
          x[     2] =       -1
          x[     3] =       -2
```

2.19.2 ASL_dbtuco, ASL_rbtuco
Condition Number of a Real Upper Triangular Matrix

(1) **Function**

ASL_dbtuco or ASL_rbtuco obtains the condition number of the real upper triangular matrix A (two-dimensional array type).

(2) **Usage**

Double precision:
ierr = ASL_dbtuco (a, lna, n, &cond, w1);
Single precision:
ierr = ASL_rbtuco (a, lna, n, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	lna×n	Input	Real upper triangular matrix A (two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	cond	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	1	Output	Reciprocal of the condition number
5	w1	$\left\{\begin{array}{c}D*\\R*\end{array}\right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < n \leq lna$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	cond ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	Matrix A has a 0.0 diagonal element. i is the number of the first 0.0 diagonal element.	

(6) Notes

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

2.19.3 ASL_dbtudi, ASL_rbtudi Determinant and Inverse Matrix of a Real Upper Triangular Matrix

(1) **Function**

ASL_dbtudi or ASL_rbtudi obtains the determinant and inverse matrix of the real upper triangular matrix A (two-dimensional array type).

(2) **Usage**

Double precision:
ierr = ASL_dbtudi (a, lna, n, det, isw);
Single precision:
ierr = ASL_rbtudi (a, lna, n, det, isw);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer, long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* R*}	lna×n	Input	Real upper triangular matrix A (two-dimensional array type)
				Output	inverse matrix of matrix A (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	det	{D* R*}	2	Output	Determinant of matrix A((See Note (b))
5	isw	I	1	Input	Processing switch isw>0:Obtain determinant. isw=0:Obtain determinant and inverse matrix. isw<0:Obtain inverse matrix.
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/a[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) Notes

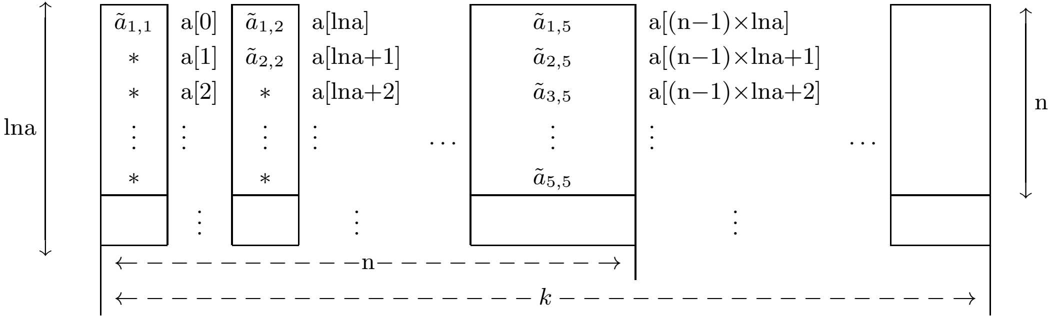
(a) Since the inverse matrix of an upper triangular matrix is an upper triangular matrix, the inverse matrix A^{-1} is stored only in the upper triangular portion of array a.

Inverse matrix A^{-1}

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

$$\Downarrow$$

Storage status within array a[lna×k]



Remarks

a. lna ≥ n and n ≤ k must hold.

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

Figure 2−16 Storage Status of the Inverse Matrix (Upper Triangular Matrix)

(b) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(c) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

2.20 REAL LOWER TRIANGULAR MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

2.20.1 ASL_dbtlsl, ASL_rbtlsl

Simultaneous Linear Equations (Real Lower Triangular Matrix)

(1) **Function**

ASL_dbtlsl or ASL_rbtlsl solves the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ having a real lower triangular matrix A (two-dimensional array type) as coefficient matrix.

(2) **Usage**

Double precision:
  ierr = ASL_dbtlsl (a, lna, n, b);
Single precision:
  ierr = ASL_rbtlsl (a, lna, n, b);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\begin{Bmatrix}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{Bmatrix}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\begin{Bmatrix}D*\\R*\end{Bmatrix}$	lna×n	Input	Coefficient matrix A (real lower triangular matrix, two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	$\begin{Bmatrix}D*\\R*\end{Bmatrix}$	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
2100	There existed the diagonal element which was close to zero in the coefficient matrix *A*. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	The coefficient matrix *A* has a 0.0 diagonal element. *i* is the number of the first 0.0 diagonal element. The matrix *A* is singular.	

(6) Notes

None

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$\begin{bmatrix} 5 & 0 & 0 & 0 \\ -1 & 4 & 0 & 0 \\ 2 & 1 & 2 & 0 \\ 3 & 2 & 7 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 5 \\ 22 \end{bmatrix}$$

(b) Input data

Coefficient matrix a, $\text{lna} = 11, \text{n} = 4$ and constant vector b.

(c) Main program

```
/*      C interface example for ASL_dbtlsl */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "dbtlsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dbtlsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &nn );

    a = ( double * )malloc((size_t)( sizeof(double) * (na*nn) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
```

```
            return -1;
        }

        b = ( double * )malloc((size_t)( sizeof(double) * nn ));
        if( b == NULL )
        {
            printf( "no enough memory for array b\n" );
            return -1;
        }

        printf( "\t n = %6d\n", nn );
        printf( "\n\tCoefficient Matrix\n\n" );
        for( i=0 ; i<nn ; i++ )
        {
            for( j=0 ; j<nn ; j++ )
            {
                fscanf( fp, "%lf", &a[i+na*j] );
                printf( "\t%8.3g", a[i+na*j] );
            }
            printf( "\n" );
        }

        printf( "\n\tConstant Vector\n\n" );
        for( i=0 ; i<nn ; i++ )
        {
            fscanf( fp, "%lf", &b[i] );
            printf( "\t%8.3g\n", b[i] );
        }

        fclose( fp );

        ierr = ASL_dbtlsl(a, na, nn, b);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution\n\n" );
        for( i=0 ; i<nn ; i++ )
        {
            printf( "\t  x[%6d] = %8.3g\n", i,b[i] );
        }

        free( a );
        free( b );

        return 0;
    }
```

(d) Output results

```
    *** ASL_dbtlsl ***

     ** Input **

     n =      4

    Coefficient Matrix

           5           0           0           0
          -1           4           0           0
           2           1           2           0
           3           2           7          10

    Constant Vector

           5
           3
           5
          22

     ** Output **

    ierr =      0

    Solution

      x[     0] =        1
      x[     1] =        1
      x[     2] =        1
      x[     3] =        1
```

2.20.2 ASL_dbtlco, ASL_rbtlco Condition Number of a Real Lower Triangular Matrix

(1) **Function**

ASL_dbtlco or ASL_rbtlco obtains the condition number of the real lower triangular matrix A (two-dimensional array type).

(2) **Usage**

Double precision:
ierr = ASL_dbtlco (a, lna, n, &cond, w1);
Single precision:
ierr = ASL_rbtlco (a, lna, n, &cond, w1);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real lower triangular matrix A (two-dimensional array type)
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	cond	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	1	Output	Reciprocal of the condition number
5	w1	$\left\{\begin{array}{l}\text{D}*\\ \text{R}*\end{array}\right\}$	n	Work	Work area
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{n} \leq \text{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	cond ← 1.0 is performed.
2100	There existed the diagonal element which was close to zero in the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
$4000 + i$	Matrix A has a 0.0 diagonal element. i is the number of the first 0.0 diagonal element.	

(6) **Notes**

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

2.20.3 ASL_dbtldi, ASL_rbtldi
Determinant and Inverse Matrix of a Real Lower Triangular Matrix

(1) **Function**

ASL_dbtldi or ASL_rbtldi obtains the determinant and inverse matrix of the real lower triangular matrix A (two-dimensional array type).

(2) **Usage**

Double precision:
  ierr = ASL_dbtldi (a, lna, n, det, isw);
Single precision:
  ierr = ASL_rbtldi (a, lna, n, det, isw);

(3) **Arguments and Return Value**

D:Double precision real	Z:Double precision complex	I: $\left\{\begin{array}{l}\text{int as for 32bit Integer}\\ \text{long as for 64bit Integer}\end{array}\right\}$
R:Single precision real	C:Single precision complex	

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	lna×n	Input	Real lower triangular matrix A (two-dimensional array type)
				Output	Inverse matrix of matrix A (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	det	$\left\{\begin{array}{c}\text{D}*\\ \text{R}*\end{array}\right\}$	2	Output	Determinant of matrix A((See Note (b))
5	isw	I	1	Input	Processing switch isw>0:Obtain determinant isw=0:Obtain determinant and inverse matrix isw<0:Obtain inverse matrix
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/a[0] are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.

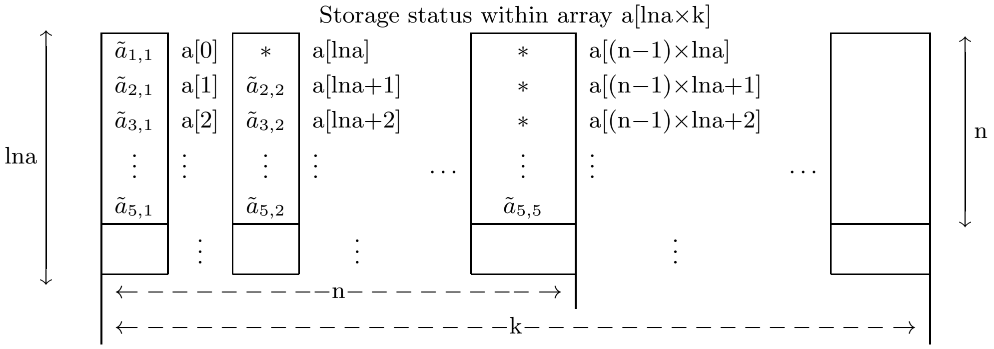
(6) Notes

(a) Since the inverse matrix of an lower triangular matrix is an lower triangular matrix, the inverse matrix A^{-1} is stored only in the lower triangular portion of array a.

Inverse matrix A^{-1}

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

$$\Downarrow$$



Remarks

a. lna $\geq$ n and n $\leq$ k must hold.

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

Figure 2−17 Storage Status of the Inverse Matrix (Lower Triangular Matrix)

(b) The determinant is given by the following expression:

$$\det(A) = \det[0] \times (10.0^{\det[1]})$$

Scaling is performed at this time so that:

$$1.0 \leq |\det[0]| < 10.0$$

(c) **The inverse matrix should not be calculated, except the inverse matrix itself is required, or the order of the matrix is sufficiently small (less than 100).** In many cases, inverse matrix appears in the form $A^{-1}\boldsymbol{b}$ or $A^{-1}B$ in the numerical calculations, it must be calculated by solving the simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$ for the vector $\boldsymbol{x}$ or by solving the simultaneous linear equations with multiple right-hand sides $AX = B$ for the matrix X, respectively. Mathematically, solving these kinds of simultaneous linear equations is the same as obtaining inverse matrix, and multiplying the inverse matrix and a vector or multiplying the inverse matrix and a matrix. However, in numerical calculations, these are usually extremely different. The calculation efficiency for obtaining inverse matrix, and multiplying the inverse matrix and vector or multiplying the inverse matrix and matrix is worse than for solving the simultaneous linear equations, and the calculation precision also declines.

Appendix A

GLOSSARY

(1) **Matrix**

An $m \times n$ matrix A is rectangular array of $m \times n$ elements $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ as shown below.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

The element $a_{i,j}$ is called the (i, j)-th element of matrix A. The elements of a matrix are considered to be complex or real numbers. In particular, a matrix having complex numbers as its elements is called a complex matrix, and a matrix having real numbers as its elements is called a real matrix. Also, if $m = n$, the matrix A is called square matrix.

The matrix A is sometimes denotes as (a_{ij}). In this manual, $(a_{i,j})$ is used for distinguishing between the row subscript i and column subscript j as necessary.

(2) **(Number) vector**

1 x n matrix is called a row vector of size n, and an m x 1 matrix is called a column vector of size m. Unless it is specifically necessary to distinguish between them, both of these are simply called vectors. Mathematically, a vector is defined as a more abstract concept. The "vector" described here is called a number vector. For the definition of an abstract vector, see the explanation of "vector space."

(3) **Matrix product**

The matrix product $AB = (c_{i,l})$ of the two matrices $A = (a_{i,j})$ and $B = (b_{k,l})$ is defined as follows

$$c_{i,l} = \sum_j a_{i,j} \cdot b_{j,l}$$

only when the number of columns in matrix A is equal to the number of rows in matrix B.

(4) **Matrix-vector product**

If the matrix B in the matrix product AB is a column vector $\boldsymbol{x}$, then the product $A\boldsymbol{x}$ is called the matrix-vector product.

(5) **Transpose of matrix**

The matrix $A' = (a_{j,i})$ formed by interchanging the rows and columns in $m \times n$ matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ is called the transpose of matrix A and is represented by A^T. The transpose may be also represented as ${}^tA$.

(6) **(Main) diagonal of a matrix**

The list of elements $a_{i,i}$ $(i = 1, 2, \cdots, n)$ in an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ is called the (main) diagonal, and the elements are called the (main) diagonal elements. Also, a matrix having nonzero elements only on the diagonal is called a diagonal matrix.

(7) **Unit matrix**

An $n \times n$ matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ in which all the diagonal elements $a_{i,i}$ $(i = 1, 2, \cdots, n)$ are 1 and all the non-diagonal elements are 0 is called a unit matrix and is represented using the symbol E or I. This satisfies $AE = EA = A$ for any matrix A.

(8) **Inverse matrix**

For a square matrix A, if a square matrix B exist that satisfies $AB = BA = E$ (where E is the unit matrix), then the matrix B is called the inverse matrix of matrix A and is represented by the symbol A^{-1}.

(9) **General inverse matrix**

For an $m \times n$ matrix A, an $n \times m$ matrix X that satisfies the following relationships exists uniquely. This matrix X, which is called the (Moore-Penrose) general inverse matrix of matrix A, is represented by the symbol $A^{\dagger}$.

- $AXA \ = \ A$
- $XAX \ = \ X$
- $(AX)^T \ = \ AX$
- $(XA)^T \ = \ XA$

(10) **Lower triangle and upper triangle of a matrix**

The collection of elements $a_{i,j}$ $(i > j)$ in an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ is called the lower triangle and the collection of elements $a_{i,j}$ $(i < j)$ is called the upper triangle. The diagonal may also be included in the definition of the upper and lower triangles. A matrix having nonzero elements only in the lower triangle that includes the diagonal is called a lower triangular matrix, and a matrix having nonzero elements only in the upper triangle that includes the diagonal is called an upper triangular matrix.

(11) **Conjugate transpose matrix**

The transpose of a matrix having the complex conjugates of the elements of a complex matrix A as elements is called conjugate transpose matrix and is represented by the symbol A^*. If the elements of a matrix are real numbers, then $A^* = A^T$.

(12) **Symmetric matrix**

A square matrix for which $A = A^T$ holds is called a symmetric matrix. In a symmetric matrix, $a_{i,j} = a_{j,i}$.

(13) **Hermitian matrix**

A square matrix for which $A = A^*$ holds is called a Hermitian matrix. In a Hermitian matrix, $a_{i,j}$ and $a_{j,i}$ are complex conjugates.

(14) **Unitary matrix**

The square matrix U for which $UU^* = I$ (I is the unit matrix) holds is called the unitary matrix.

(15) **Orthogonal matrix**

The real square matrix A for which $AA^T = I$ (I is the unit matrix) holds is called the orthogonal matrix.

(16) **Subdiagonal of a matrix**

The list of elements $a_{i,i+p}$ $(i = 1, 2, \cdots, n - p)$ in an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ is called the p-th upper subdiagonal, and the list of elements $a_{i+q,i}$ $(i = 1, 2, \cdots, n - q)$ is called the q-th lower subdiagonal. The elements are called the p-th upper subdiagonal elements and q-th lower subdiagonal elements, respectively. Also, both of these collectively may be referred to simply as subdiagonal elements.

(17) **Band matrix**

A matrix having nonzero elements only on the main diagonal and in several upper and lower subdiagonals near the main diagonal in an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ is called a band matrix. If the subdiagonals containing nonzero elements that are furthest from the diagonal are the u-th upper subdiagonal and l-th lower subdiagonal, the values u and l are called the upper bandwidth and lower bandwidth, respectively. if $u = l$, this is simply called the bandwidth.

(18) **Tridiagonal matrix**

A matrix in which the upper and lower bandwidths are both 1 is called a tridiagonal matrix.

(19) **Hessenberg matrix**

A matrix in which all lower triangle elements except the first lower subdiagonal are zero in an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ is called a Hessenberg matrix. To obtain the eigenvalues of a matrix, a general matrix is converted to this kind of matrix.

(20) **Quasi-upper triangular matrix**

An $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ for which at least one of every two consecutive subdiagonal elements of the first lower subdiagonal is 0 and all lower triangular elements excluding the first lower subdiagonal are 0 is called a quasi-upper triangular matrix. This is a special case of a Hessenberg matrix.

(21) **Sparse matrix**

In general, a matrix in which the number of nonzero elements is relatively small compared to the total number of elements is called a sparse matrix. If the arrangement of the elements within a sparse matrix has some kind of regularity and an effective method of solving a problem is created by making practical use of this regularity, this matrix is called a regular sparse matrix. A sparse matrix that is not a regular sparse matrix is called an irregular sparse matrix. For example, a band matrix having a small bandwidth is a type of regular sparse matrix.

(22) **Regular and singular matrices**

If a square matrix A has an inverse matrix, the matrix A is said to be regular. A matrix that is not regular is said to be singular. The solutions of system of simultaneous linear equations having a regular matrix as coefficients are uniquely determined. However, since calculations are actually performed using a finite number of digits, the effects of rounding errors cannot be avoided, and the distinction between a regular and singular matrix becomes ambiguous. For example, solutions may apparently be obtained even when a system of simultaneous linear equations is solved numerically using a mathematically singular matrix. Therefore, when solving a system of simultaneous linear equations having a nearly singular matrix as coefficients, sufficient testing is required concerning the appropriateness of solutions that are apparently obtained.

(23) **LU decomposition**

To use a direct method to solve the system of simultaneous linear equations $A\boldsymbol{x} = \boldsymbol{b}$, first decompose the coefficient matrix A into the product $A = LU$ of the lower triangular matrix L and upper triangular matrix U. This decomposition is called an LU decomposition, If this kind of decomposition is performed, the solution $\boldsymbol{x}$ of the system of simultaneous linear equations is obtained by sequentially solving the following equations:

$$\begin{aligned} L\boldsymbol{y} &= \boldsymbol{b} \\ U\boldsymbol{x} &= \boldsymbol{y} \end{aligned}$$

Since the coefficient matrix of these two simultaneous linear equations is a triangular matrix, they can be easily solved by using forward-substitution and backward-substitution. If the matrix A is regular, for example, if the diagonal elements of matrix L are fixed at 1, the LU decomposition of the matrix A is uniquely determined. Also, when solving a system of simultaneous linear equations, since LU decomposition generally is performed while performing partial pivoting, if P is a row exchange matrix due to pivoting, triangular matrices L and U for which $PA = LU$ is satisfied are obtained, respectively.

(24) U$^{\mathrm{T}}$DU **decomposition**

If the coefficient matrix of a system of simultaneous linear equations is a symmetric matrix, the relationship $L = U^T D$ holds between the lower triangular matrix L and upper triangular matrix U obtained by performing an LU decomposition without performing pivoting. Here, D is a diagonal matrix. Therefore, the system of simultaneous linear equations can be solved by explicitly obtaining only D and one of L and U. The decomposition that explicitly obtains U and D from coefficient matrix is called the U$^{\mathrm{T}}$DU decomposition.

(25) U*DU **decomposition**

If the coefficient matrix of a system of simultaneous linear equations is a Hermitian matrix, the relationship $L = U^* D$ holds between the lower triangular matrix L and upper triangular matrix U obtained by performing an LU decomposition without performing pivoting. Here, D is a diagonal matrix. Therefore, the system of simultaneous linear equations can be solved by explicitly obtaining only D and one of L and U. The decomposition that explicitly obtains U and D from coefficient matrix is called the U*DU decomposition.

(26) **Positive definite**

If a real symmetric matrix or Hermitian matrix A satisfies $\boldsymbol{x}^* A \boldsymbol{x} > 0$ for an arbitrary vector $\boldsymbol{x}$ ($\boldsymbol{x} \neq \boldsymbol{0}$), it is said to be positive (definite). If it satisfies $\boldsymbol{x}^* A \boldsymbol{x} < 0$, it is said to be negative. The fact that the matrix A is a positive definite matrix is equivalent to the following two condition.

(a) All of the eigenvalues of matrix A are positive.

(b) All principal minors of matrix A are positive.

Although, mathematically, an LU decomposition can be performed for a positive definite matrix without performing pivoting, if pivoting is not actually performed, an LU decomposition may not be able to be performed numerically with stability.

(27) **Real eigenvalue**

The eigenvalue of a real square matrix are all real if and only if the matrix is a product of two real symmetric matrices. Also, the eigenvalue of a complex square matrix are all real if and only if the matrix is a product of two Hermitian matrices.

(28) **Diagonally dominant**

If the following holds for an $n \times n$ square matrix $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$

$$|a_{i,i}| \ > \ \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| \ \ (i = 1, 2, \cdots, n)$$

matrix A is called a diagonally dominant matrix. Although, mathematically, an LU decomposition can be performed for a diagonally dominant matrix without performing pivoting, if pivoting is not actually performed, an LU decomposition may not be able to be performed numerically with stability.

(29) **Fill-in**

When an LU decomposition of a sparse matrix is performed, changing elements that had originally been zero to nonzero values due to the calculation is called fill-in.

(30) **Envelope method**

When performing a $\mathrm{U^TDU}$ decomposition of an $n \times n$ symmetric sparse matrix A, the envelope method executes the decomposition by selecting the first nonzero element of each row of matrix A and the diagonal elements as an envelope and considering only the elements within the envelope. This technique uses the fact that fill-in occurs only within the envelope when $\mathrm{U^TDU}$ decomposition of the matrix is performed.

The envelope method performs the decomposition by considering the lower triangular portion of the symmetric matrix. A technique that performs a similar decomposition by considering the upper triangular portion is known as the skyline method.

(31) **Vector space**

If the set V satisfies conditions (a) and (b) V is called a vector space and its elements are called vectors.

(a) The sum $\boldsymbol{a}+\boldsymbol{b}$ of two elements $\boldsymbol{a}$ and $\boldsymbol{b}$ of V is uniquely determined as an element of V and satisfies the following properties.

  i. $(\boldsymbol{a}+\boldsymbol{b})+\boldsymbol{c}=\boldsymbol{a}+(\boldsymbol{b}+\boldsymbol{c})$ (associative law)
  Where, $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are arbitrary elements of V.

  ii. $\boldsymbol{a}+\boldsymbol{b}=\boldsymbol{b}+\boldsymbol{a}$ (commutative law)
  Where, $\boldsymbol{a}$ and $\boldsymbol{b}$ are arbitrary elements of V.

  iii. An element $\mathbf{0}$ of V, which is called the zero vector, exists and satisfies $\boldsymbol{a}+\mathbf{0}=\boldsymbol{a}$ for an arbitrary element $\boldsymbol{a}$ of V.

  iv. For an arbitrary element $\boldsymbol{a}$ of V, exactly one element $\boldsymbol{b}$ of V exists for which $\boldsymbol{a}+\boldsymbol{b}=\mathbf{0}$. This element $\boldsymbol{b}$ is represented as $-\boldsymbol{a}$.

(b) For an arbitrary element $\boldsymbol{a}$ of V and complex number c, $c\boldsymbol{a}$ (the c multiple of $\boldsymbol{a}$) is uniquely determined as an element of V and satisfies the following properties (scalar multiple).

  i. $c(\boldsymbol{a}+\boldsymbol{b})=c\boldsymbol{a}+c\boldsymbol{b}$ (vector distributive law)

  ii. $(c+d)\boldsymbol{a}=c\boldsymbol{a}+d\boldsymbol{a}$ (scalar distributive law)

  iii. $(cd)\boldsymbol{a}=c(d\boldsymbol{a})$

  iv. $1\boldsymbol{a}=\boldsymbol{a}$

(32) **Linear combination, linearly independent and linearly dependent**

The vector

$$c_1\boldsymbol{a}_1+\cdots+c_k\boldsymbol{a}_k$$

created from the k vectors $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k$ of vector space V and complex numbers $c_1, \cdots, c_k$ is called the linear combination of $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k$, and $c_1, \cdots, c_k$ are called its coefficients. For certain coefficients $c_1, \cdots, c_k$ that are not all zero, the set of vectors $\{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k\}$ is said to be linearly dependent if

$$c_1\boldsymbol{a}_1+\cdots+c_k\boldsymbol{a}_k=\mathbf{0}$$

and is said to be linearly independent otherwise.

(33) **Basis**

Let S be an arbitrary subset of vector space V, and let a collection of linearly independent vectors contained in S be $\{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k\}$. For an arbitrary vector $\boldsymbol{b}$ of S, if $\{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k, \boldsymbol{b}\}$ is linearly dependent, $\{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k\}$ is said to be the maximum set in S. When the vector space V itself is taken as S, this collection of linearly independent vectors is called the basis of vector space V. The number of vectors constituting the basis of V is called the dimension of V. Also, if we let an arbitrary basis of an n-dimensional vector space V_n be $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$, then an arbitrary vector $\boldsymbol{a}$ of V_n is represented uniquely as a linear combination of $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$.

(34) **(Vector) subspace**

A subset L of vector space V is called a (vector) subspace of V if the following conditions (a) and (b) are satisfied.

(a) If $\boldsymbol{a}, \boldsymbol{b} \in L$, then $\boldsymbol{a} + \boldsymbol{b} \in L$

(b) If $\boldsymbol{a} \in L$ and c is a complex number, $c\boldsymbol{a} \in L$

(35) **Linear transformation**

Let V_n and V_m be n-dimensional and m-dimensional vector spaces, respectively. If the mapping $\boldsymbol{A} : V_n \to V_m$ that associates each element $\boldsymbol{x}$ of V_n with an element $\boldsymbol{A}(\boldsymbol{x})$ of V_m satisfies the following two conditions, $\boldsymbol{A}$ is said to be a linear transformation from V_n to V_m.

(a) $\boldsymbol{A}(\boldsymbol{x}_1 + \boldsymbol{x}_2) = \boldsymbol{A}(\boldsymbol{x}_1) + \boldsymbol{A}(\boldsymbol{x}_1) \quad \boldsymbol{x}_1, \boldsymbol{x}_2 \in V_n$

(b) $\boldsymbol{A}(c\boldsymbol{x}) = c\boldsymbol{A}(\boldsymbol{x}) \quad \boldsymbol{x} \in V_n$ and $c :$ a complex number

If we let a single basis of V_n and V_m, respectively, be $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ and $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m\}$, then $\boldsymbol{A}(\boldsymbol{x})$ is determined for an arbitrary $\boldsymbol{x} \in V_n$ according to the coefficient matrix $A = (a_{i,j})$ of

$$\boldsymbol{A}(\boldsymbol{u}_j) = \sum_{i=1}^{m} a_{i,j}\boldsymbol{v}_i \quad (j = 1, \cdots, n)$$

The matrix A is called the representation matrix of the linear transformation $\boldsymbol{A}$ related to this basis. Also, if $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{x}$ for $\boldsymbol{x} \in V_n$, it defines the linear transformation $\boldsymbol{E} : V_n \to V_n$, which is called the identity transformation. The representation matrix of the identity transformation always is the unit matrix E regardless of how the basis is taken.

(36) **Eigenvalue and eigenvector**

For a linear transformation $\boldsymbol{A}$ within an n-dimensional vector space V_n, if there exists a number λ and a vector $\boldsymbol{x}$ ($\boldsymbol{x} \neq \boldsymbol{0}$) such that

$$\boldsymbol{A}(\boldsymbol{x}) = \lambda\boldsymbol{x}, \text{ that is, } (\boldsymbol{A} - \lambda\boldsymbol{E})(\boldsymbol{x}) = \boldsymbol{0}$$

is satisfied, then λ is called an eigenvalue of $\boldsymbol{A}$ and $\boldsymbol{x}$ is called the eigenvector belonging to the eigenvalue λ. Here, $\boldsymbol{E}$ is the identity transformation. If we fix a single basis within V_n, let the representation matrix of the linear transformation $\boldsymbol{A}$ be A, and let the number vector corresponding to the eigenvector $\boldsymbol{x}$ be $\hat{\boldsymbol{x}}$, then the eigenvalue λ and $\hat{\boldsymbol{x}}$ satisfy the following equation.

$$A\hat{\boldsymbol{x}} = \lambda\hat{\boldsymbol{x}}$$

Here, $\hat{\boldsymbol{x}}$ is represented using the components $x_1,\ \cdots,\ x_n$ of $\boldsymbol{x}$ as

$$\hat{\boldsymbol{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Normally, λ and $\hat{\boldsymbol{x}}$ are called the eigenvalue and eigenvector of matrix A, respectively. These terms are also used in this manual. Also, no distinction is made between the number vector and vector, which are represented as $\boldsymbol{x}$. Since the collection of all the vectors belonging to eigenvalue λ of the linear transformation $\boldsymbol{A} : V_n \to V_n$ together with the zero vector $\mathbf{0}$ form a single vector space, this is called the eigenvector space belonging to the eigenvalue λ of $\boldsymbol{A}$.

(37) **Invariant subspace**

For the linear transformation $\boldsymbol{A}$ within the vector space V_n, if the subspace U of V_n has the property

$$\boldsymbol{A}(U) \subseteq U$$

that is, if $\boldsymbol{Ax} \in U$ for an arbitrary vector $\boldsymbol{x}$, then U is said to be invariant relative to $\boldsymbol{A}$. In particular, the eigenvector space of $\boldsymbol{A}$ is invariant relative to $\boldsymbol{A}$. An invariant subvector space is called an invariant subspace.

(38) **Plane rotation**

The orthogonal transformation specified by the following kind of matrix $S_{k:l}(\theta)$ is called a plane rotation.

$$S_{kl}(\theta) = \begin{bmatrix} E_{1:k-1} & O_{1:k-1,k:l} & O_{1:k-1,l:n} \\ O_{k:l,1:k-1} & T_{k:l}(\theta) & O_{k:l,l:n} \\ O_{l:n,1:k-1} & O_{l:n,k:l} & E_{l:n} \end{bmatrix}$$

Here, $T_{k:l}(\theta)$ is defined as follows:

$$T_{k:l}(\theta) = \begin{bmatrix} \cos\theta & O_{k:k,k+1:l-1} & -\sin\theta \\ O_{k+1:l-1,k:k} & E_{k+1:l-1} & O_{k+1:l-1,l:l} \\ \sin\theta & O_{l:l,k+1:l-1} & \cos\theta \end{bmatrix}$$

$E_{p:q}$ is the $q-p+1$-dimensional unit matrix shown below:

$$E_{p:q} = \begin{array}{l} \begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \end{bmatrix} \end{array} \begin{array}{l} (p \\ (p+1 \\ \vdots \\ (q \end{array}$$

and $O_{p:r,q:s}$ is the $r-p+1 \times s-q+1$-dimensional zero matrix shown below:

$$O_{p:r,q:s} = \begin{array}{cccc} \overset{q}{\smile} & \overset{q+1}{\smile} & \cdots & \overset{s}{\smile} \\ \end{array}$$
$$\qquad\qquad \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{array}{l} (p \\ (p+1 \\ \vdots \\ (r \end{array}$$

Now, if the submatrix $A_{p:r,q:s}$ of $A = (a_{i,j})$ $(i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$ is defined as follows:

$$A_{p:r,q:s} = \begin{bmatrix} a_{p,q} & a_{p,q+1} & \cdots & a_{p,s} \\ a_{p+1,q} & a_{p+1,q+1} & \cdots & a_{p+1,s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r,q} & a_{r,q+1} & \cdots & a_{r,s} \end{bmatrix}$$

the matrix A is represented as follows:

$$A = \begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l} & A_{1:k-1,l+1:n} \\ A_{k:l,1:k-1} & A_{k:l,k:l} & A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l} & A_{l+1:n,l+1:n} \end{bmatrix}$$

At this time, since $S_{k:l}(\theta)A$ and $T_{k:l}(\theta)A_{k:l,q:s}$ are as follows:

$$S_{k:l}(\theta)A = \begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l} & A_{1:k-1,l+1:n} \\ T_{k:l}(\theta)A_{k:l,1:k-1} & T_{k:l}(\theta)A_{k:l,k:l} & T_{k:l}(\theta)A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l} & A_{l+1:n,l+1:n} \end{bmatrix}$$

$$T_{k:l}(\theta)A_{k:l,q:s} = \begin{bmatrix} \cos\theta a_{k,q} - \sin\theta a_{l,q} & \cdots & \cos\theta a_{k,r} - \sin\theta a_{l,s} \\ a_{k+1,q} & \cdots & a_{k+1,r} \\ \vdots & \ldots & \vdots \\ a_{l-1,q} & \cdots & a_{l-1,r} \\ \sin\theta a_{k,q} + \cos\theta a_{l,q} & \cdots & \sin\theta a_{k,r} + \cos\theta a_{l,s} \end{bmatrix}$$

if θ is determined so that $\tan\theta = \frac{a_{l,i}}{a_{k,i}}$ or $\tan\theta = -\frac{a_{l,i}}{a_{k,i}}(i = q, \cdots, s)$ is satisfied, then an arbitrary element among the elements of column k and column l of $S_{k:l}(\theta)A$ can be set to zero. Now, since the following relationship holds:

$$AS_{k:l}(-\theta) = \begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l}T_{k:l}(-\theta) & A_{1:k-1,l+1:n} \\ A_{k:l,1:k-1} & A_{k:l,k:l}T_{k:l}(-\theta) & A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l}T_{k:l}(-\theta) & A_{l+1:n,l+1:n} \end{bmatrix}$$

$$A_{p:r,k:l}T_{k:l}(-\theta) = \begin{bmatrix} \cos\theta a_{p,k} - \sin\theta a_{p,l} & a_{p,k+1} & \cdots & a_{p,l-1} & \sin\theta a_{p,k} + \cos\theta a_{p,l} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cos\theta a_{r,k} - \sin\theta a_{r,l} & a_{r,k+1} & \cdots & a_{r,l-1} & \sin\theta a_{r,k} + \cos\theta a_{r,l} \end{bmatrix}$$

if θ is determined so that $\tan\theta = \frac{a_{i,l}}{a_{i,k}}$ or $\tan\theta = -\frac{a_{i,l}}{a_{i,k}}(i = p, \cdots, r)$ is satisfied, then an arbitrary element among the elements of column k and column l of $AS_{k:l}(-\theta)$ can be set to zero. Now, since the following relationship holds:

$$S_{k:l}(-\theta) = S_{k:l}(\theta)^T$$

and since $\tilde{A} = S_{k:l}(\theta)AS_{k:l}(-\theta)$ is as follows:

$$\tilde{A} = S_{k:l}(\theta)AS_{k:l}(-\theta) = \begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l}T_{k:l}(-\theta) & A_{1:k-1,l+1:n} \\ T_{k:l}(\theta)A_{k:l,1:k-1} & T_{k:l}(\theta)A_{k:l,k:l}T_{k:l}(-\theta) & T_{k:l}(\theta)A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l}T_{k:l}(-\theta) & A_{l+1:n,l+1:n} \end{bmatrix}$$

if matrix A is a symmetric matrix, then by adjusting θ, either:

$$\tilde{a}_{k,j} = \tilde{a}_{j,k} = 0$$

or

$$\tilde{a}_{l,j} = \tilde{a}_{j,l} = 0$$

can be set for some $j(j \neq k, j \neq l)$, where the elements of $\tilde{A} = S_{k:l}(\theta)AS_{k:l}(-\theta)$ are represented by $(\tilde{a}_{i,j})$.

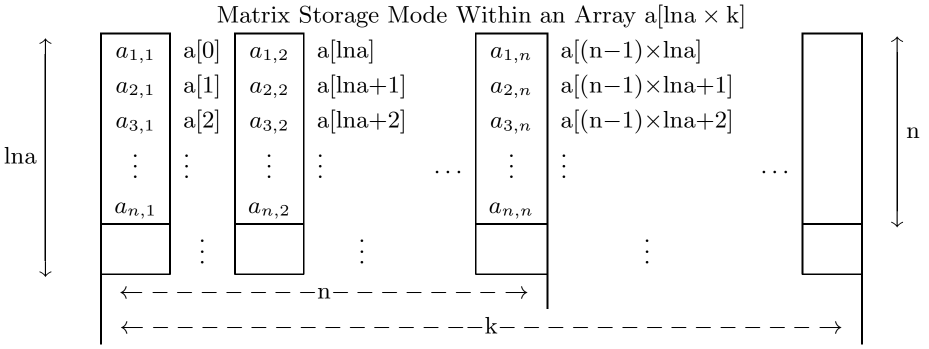
Appendix B

METHODS OF HANDLING ARRAY DATA

B.1 Methods of handling array data corresponding to matrix

Since the ASL C interface function library uses array data corresponding to matrix, this section describes various methods of handling arrays.

To call a function that uses array data, you must declare that array in advance in the calling program. If the declared array is a[lna × k], then $n \times n$ matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$ is stored in array a as shown in the figure below.



Remarks

a. lna $\geq n$ and k $\geq n$ must hold.

b. Matrix element $a_{i,j}$ corresponds to the array element a[$(i-1) + \text{lna} \times (j-1)$].

Figure B−1 Matrix Storage Mode Within an Array a[lna × k]

lna is called an adjustable dimension. If a two-dimensional array is used as an argument, the adjustable must be passed to the function as an argument in addition to the array name and order of the array. Because the matrix storage mode in ASL C interface corresponds to that of FORTRAN and the matrix elements $a_{i,j}$ $(i = 1, 2, \cdots, \text{lna}; j = 1, 2, \cdots, \text{k})$ must correspond to the array element a[ℓ] $(\ell = 0, 1, 2, \cdots, \text{lna} \times \text{k} - 1)$, as follows on the main memory.

$$\begin{array}{cccccccc} a_{1,1} & a_{2,1} & \cdots & a_{\text{lna},1} & a_{1,2} & a_{2,2} & \cdots \\ \updownarrow & \updownarrow & \cdots & \updownarrow & \updownarrow & \updownarrow & \cdots \\ \text{a}[0] & \text{a}[1] & \cdots & \text{a}[\text{lna}-1] & \text{a}[\text{lna}] & \text{a}[\text{lna}+1] & \cdots \end{array}$$

Example ASL_dam1ad (Real matrix addition)

Add 3×2 matrices A and B placing the sum in matrix C. If you declare arrays of size $[5 \times 4]$, the declaration is as follows.

```
/*      C interface example for ASL_dam1ad */
#include <stdio.h>
#include <stdlib.h>

#include <asl.h>

int main()
{
    double *a, *b, *c;
    int lma, lmb, lmc;
    int m, n, ierr;
```

```
    int k;
    lma = lmb = lmc = 5;
    k = 4;
    m = 3;
    n = 2;
    a = (double *)malloc((size_t) sizeof(double) * lma*k);
    if(a == NULL)
    {
        printf("no enough memory for array a\n");
        return -1;
    }
    b = (double *)malloc((size_t) sizeof(double) * lmb*k);
    if(b == NULL)
    {
        printf("no enough memory for array b\n");
        return -1;
    }
    c = (double *)malloc((size_t) sizeof(double) * lmc*k);
    if(c == NULL)
    {
        printf("no enough memory for array c\n");
        return -1;
    }

           ⋮
    ierr = ASL _dam1ad(a, lma , m, n, b, lmb, c, lmc);
           ⋮

    free(a);
    free(b);
    free(c);
    return 0;
}
```

Data is stored in a as follows. Data are stored in b and c in the same way.

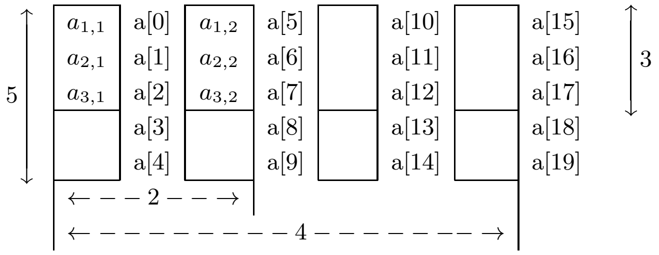


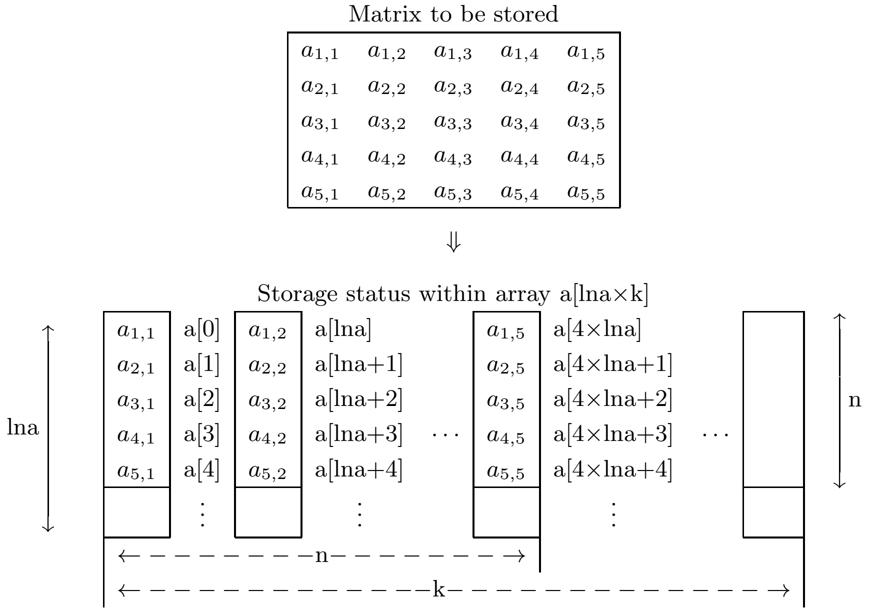
Figure B−2 Matrix Storage Mode Within an Array a[lna × k]

If you will be manipulating several arrays having different orders as data, you can prepare one array having lna equal to the largest order and use that array successively for each array. However, you must always assign the lna value as an adjustable dimension.

B.2 Data storage modes

Matrix data storage modes differ according to the matrix type. Storage modes for each type of matrix are shown below.

B.2.1 Real matrix (two-dimensional array type)



Remarks

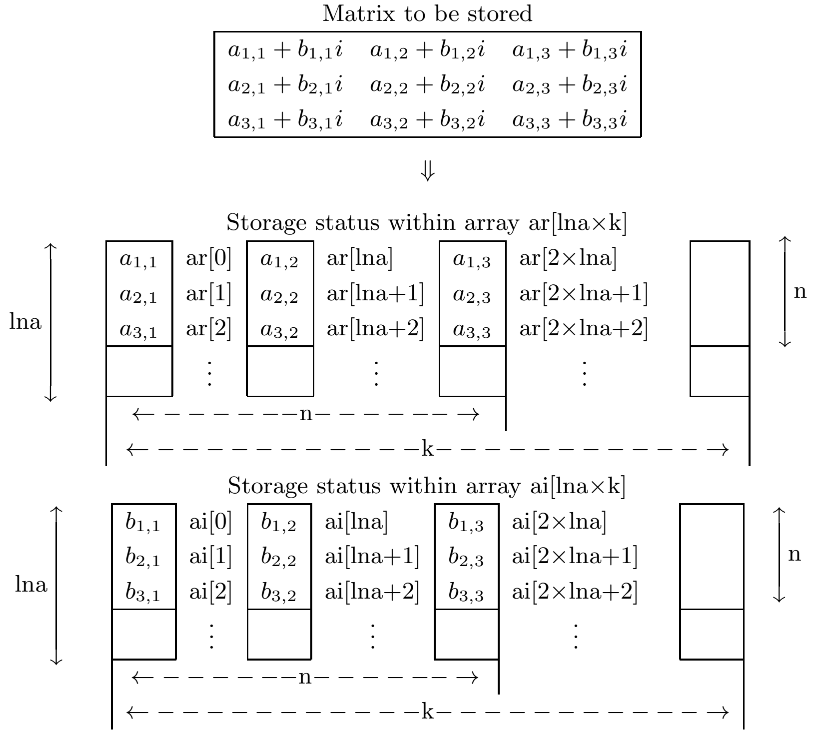
a. lna ≥ n and k ≥ n must hold.

Figure B−3 Real Matrix (Two-Dimensional Array Type) Storage Mode

B.2.2 Complex matrix

(1) Two-dimensional array type, real argument type

Real and imaginary parts are stored in separate arrays.

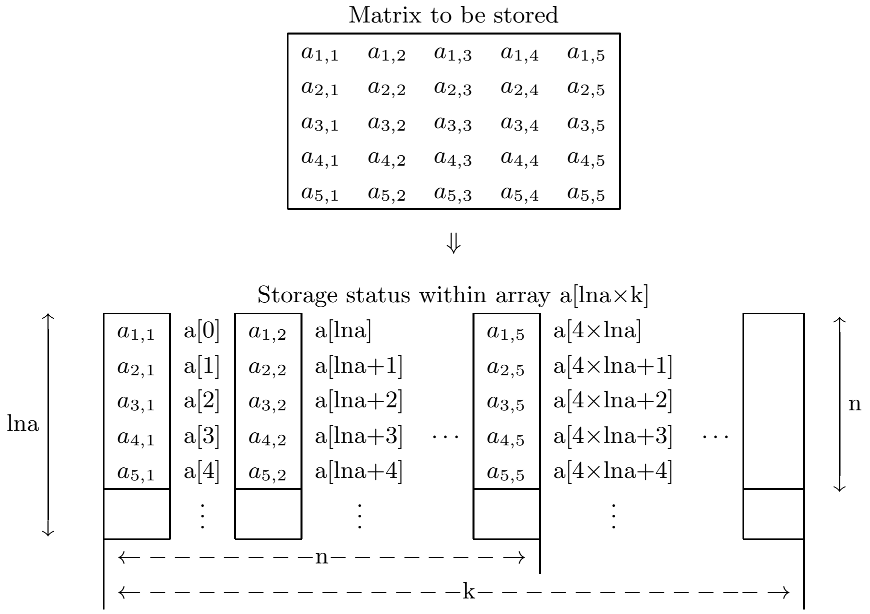


Remarks

a. lna ≥ n and k ≥ n must hold.

Figure B−4 Complex Matrix (Two-dimensional Array Type) (Real Argument Type) Storage Mode

(2) **Two-dimensional array type, complex argument type**



Remarks

a. lna ≥ n and k ≥ n must hold.

Figure B−5 Complex Matrix (Two-dimensional Array Type)(Complex Argument Type) Storage Mode

B.2.3 Real symmetric matrix and positive symmetric matrix

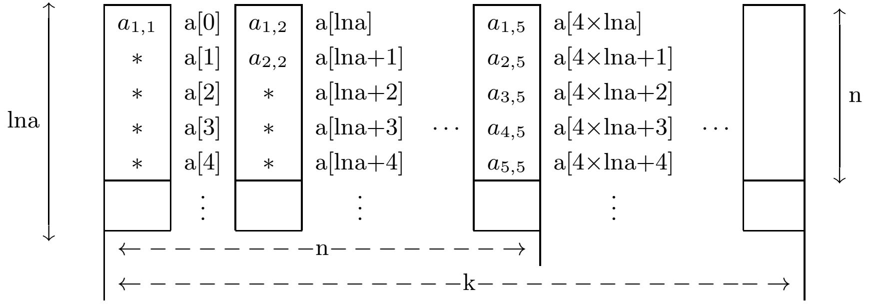
(1) Two-dimensional array type, upper triangular type

Matrix to be stored

$$\begin{array}{|ccccc|}\hline a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{1,3} & a_{2,3} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} & a_{4,5} \\ a_{1,5} & a_{2,5} & a_{3,5} & a_{4,5} & a_{5,5} \\ \hline \end{array}$$

$$\Downarrow$$

Storage status within array a[lna×k]



Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. lna ≥ n and k ≥ n must hold.

Figure B−6 Real Symmetric Matrix (Two-dimensional Array Type) (Upper Triangular Type) Storage mode

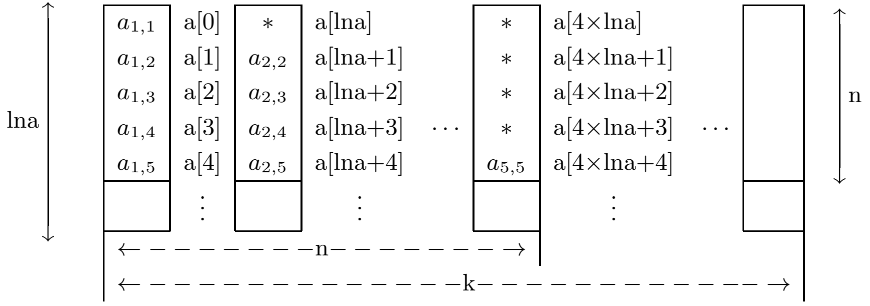
(2) **Two-dimensional array type, lower triangular type**

Matrix to be stored

$$\begin{array}{|ccccc|}\hline a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{1,3} & a_{2,3} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} & a_{4,5} \\ a_{1,5} & a_{2,5} & a_{3,5} & a_{4,5} & a_{5,5} \\ \hline \end{array}$$

$$\Downarrow$$

Storage status within array a[lna×k]



Remarks

a. The asterisk ($*$) indicates an arbitrary value.

b. lna $\geq$ n and k $\geq$ n must hold.

Figure B−7 Real Symmetric Matrix (Two-dimensional Array Type, Lower Triangular Type) Storage mode

B.2.4 Hermitian matrix

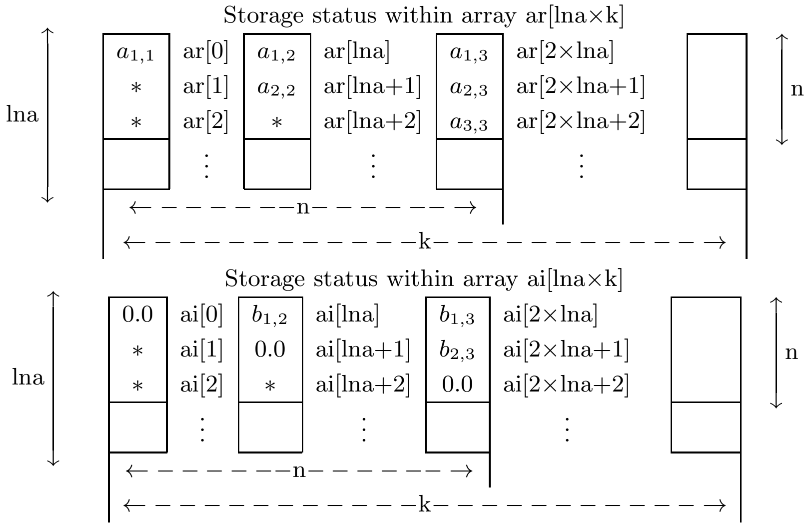
(1) **Two-dimensional array type, real argument type, upper triangular type**

Upper triangular portions of the real and imaginary parts are stored in separate arrays.

Matrix to be stored

$$\boxed{\begin{array}{ccc} a_{1,1} & a_{1,2}+b_{1,2}i & a_{1,3}+b_{1,3}i \\ a_{2,1}-b_{2,1}i & a_{2,2} & a_{2,3}+b_{2,3}i \\ a_{3,1}-b_{3,1}i & a_{3,2}-b_{3,2}i & a_{3,3} \end{array}}$$

$$\Downarrow$$



Storage status within array ar[lna×k]

Storage status within array ai[lna×k]

Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. lna $\geq$ n and k $\geq$ n must hold.

Figure B−8 Hermitian Matrix (Two-dimensional Array Type) (Real Argument Type) (Upper Triangular Type) Storage Mode

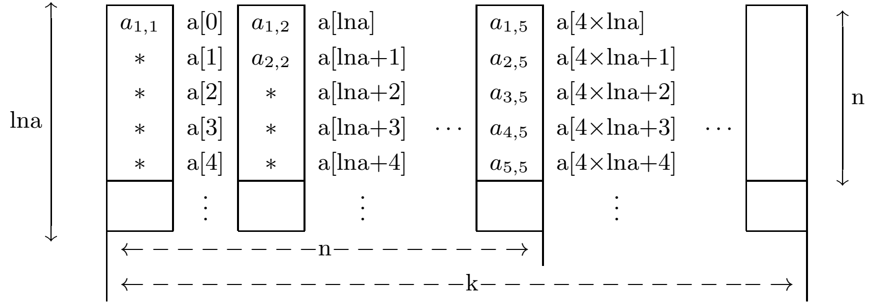
(2) **Two-dimensional array type, complex argument type, upper triangular type**

Matrix to be stored

$$\begin{array}{|ccccc|}\hline a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ \overline{a_{1,2}} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \overline{a_{1,3}} & \overline{a_{2,3}} & a_{3,3} & a_{3,4} & a_{3,5} \\ \overline{a_{1,4}} & \overline{a_{2,4}} & \overline{a_{3,4}} & a_{4,4} & a_{4,5} \\ \overline{a_{1,5}} & \overline{a_{2,5}} & \overline{a_{3,5}} & \overline{a_{4,5}} & a_{5,5} \\ \hline \end{array}$$

$$\Downarrow$$

Storage status within array a[lna×k]

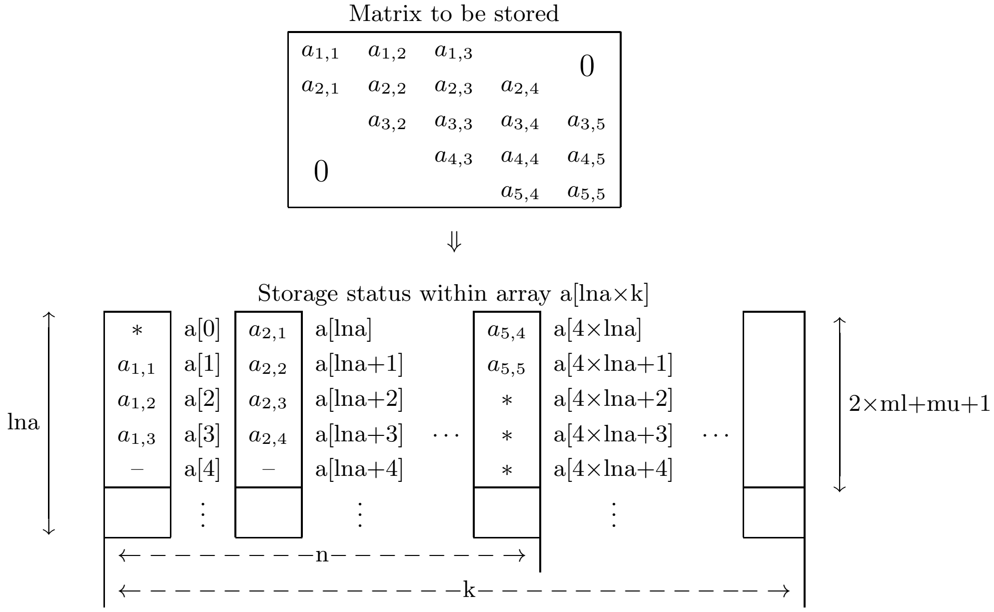


Remarks

- a. The $\overline{x}$ indicates the complex conjugate of x.
- b. The asterisk $*$ indicates an arbitrary value.
- c. lna $\geq$ n and k $\geq$ n must hold.

Figure B−9 Hermitian Matrix (Two-dimensional Array Type) (Complex Argument Type) (Upper Triangular Type) Storage Mode

B.2.5 Real band matrix



Remarks

a. The asterisk ∗ indicates an arbitrary value.

b. The area indicated by dashes (–) is required for an LU decomposition of the matrix.

c. mu is the upper band width and ml is the lower band width.

d. lna $\geq$ 2 $\times$ ml + mu + 1 and k $\geq$ n must hold. (However, if no LU decomposition is to be performed, lna $\geq$ ml + mu + 1 and k $\geq$ n is sufficient.)

Figure B−10 Real Band Matrix (Band Type) Storage Mode

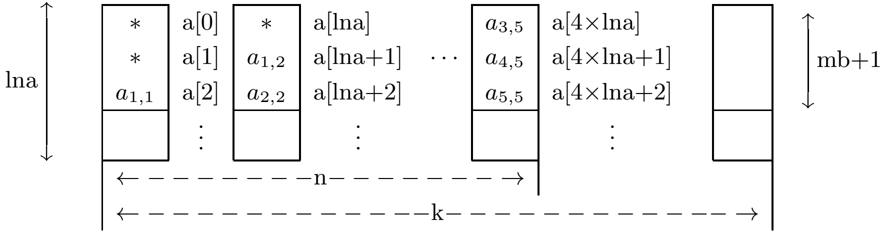
B.2.6 Real symmetric band matrix and positive symmetric matrix (symmetric band type)

Matrix to be stored

$$\begin{array}{|ccccc|}
\hline
a_{1,1} & a_{1,2} & a_{1,3} & & \\
a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} & 0 \\
a_{1,3} & a_{2,3} & a_{3,3} & a_{3,4} & a_{3,5} \\
0 & a_{2,4} & a_{3,4} & a_{4,4} & a_{4,5} \\
 & & a_{3,5} & a_{4,5} & a_{5,5} \\
\hline
\end{array}$$

⇓

Storage status within array a[lna×k]



Remarks

a. The asterisk ∗ indicates an arbitrary value.

b. mb is the band width.

c. lna $\geq$ mb + 1 and k $\geq$ n must hold.

Figure B−11 Real Symmetric Band Matrix (Symmetric Band Type) Storage Mode

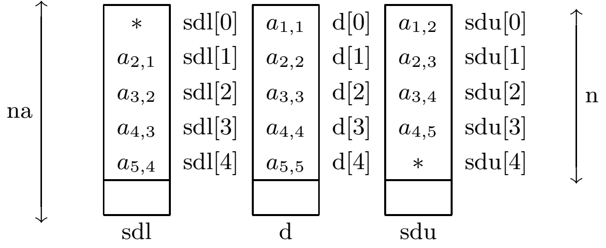
B.2.7 Real tridiagonal matrix (vector type)

Matrix to be stored

$$\begin{array}{|ccccc|} a_{1,1} & a_{1,2} & & & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & & \\ & a_{3,2} & a_{3,3} & a_{3,4} & \\ & & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & & & a_{5,4} & a_{5,5} \end{array}$$

$$\Downarrow$$

Storage status within arrays sdl[na]
(lower subdiagonal component),
d[na](diagonal component) and
sdu[na](upper subdiagonal component)



Remarks

a. The asterisk $*$ indicates an arbitrary value.

b. na $\geq$ n must hold.

Figure B−12 Real Tridiagonal Matrix (Vector Type) Storage Mode

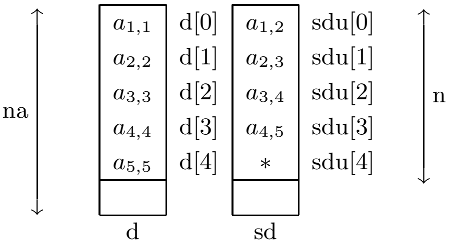
B.2.8 Real symmetric tridiagonal matrix and positive symmetric tridiagonal matrix (vector type)

Matrix to be stored

$$\begin{array}{|ccccc|}\hline a_{1,1} & a_{1,2} & & & 0 \\ a_{1,2} & a_{2,2} & a_{2,3} & & \\ & a_{2,3} & a_{3,3} & a_{3,4} & \\ 0 & & a_{3,4} & a_{4,4} & a_{4,5} \\ & & & a_{4,5} & a_{5,5} \\ \hline \end{array}$$

$$\Downarrow$$

Storage status within arrays d[na]
(diagonal component) and sd[na]
(subdiagonal component)



Remarks

- a. The asterisk $*$ indicates an arbitrary value.
- b. na $\geq$ n must hold.

Figure B−13 Real Symmetric Tridiagonal Matrix (Vector Type) Storage Mode

B.2.9 Fixed coefficient real tridiagonal matrix (scalar type)

Matrix to be stored (a) Matrix to be stored (b) Matrix to be stored (c) Matrix to be stored (d)

$$\begin{array}{|ccccc|ccccc|ccccc|ccccc|}\hline d & s & & & 0 & d & s & & & 0 & d & 2\times s & & & 0 & d & 2\times s & & & 0 \\ s & d & s & & & s & d & s & & & s & d & s & & & s & d & s & & \\ & s & d & s & & & s & d & s & & & s & d & s & & & s & d & s & \\ 0 & & s & d & s & 0 & & s & d & s & 0 & & s & d & s & 0 & & s & d & s \\ & & & s & d & & & & 2\times s & d & & & & s & d & & & & 2\times s & d \\ \hline \end{array}$$

$$\Downarrow$$

Storage status within variables d
(diagonal component) and sd
(subdiagonal component)

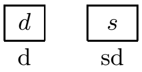


Figure B−14 Fixed Coefficient Real Tridiagonal Matrix (Scalar Type)

B.2.10 Triangular matrix

(1) **Two-dimensional array type**

The storage mode is the same as for a real symmetric matrix (two-dimensional array type) (upper triangular type) or a real symmetric matrix (two-dimensional array type) (lower triangular type).

B.2.11 Random sparse matrix (For symmetric matrix only)

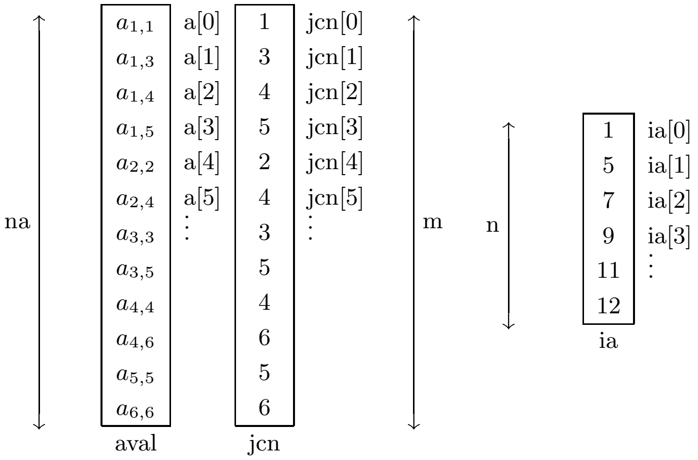
(1) **Sparse format (Symmetric case)**

Matrix A to be stored

$$\begin{array}{|cccccc|}\hline a_{1,1} & 0.0 & a_{1,3} & a_{1,4} & a_{1,5} & 0.0 \\ 0.0 & a_{2,2} & 0.0 & a_{2,4} & 0.0 & 0.0 \\ a_{1,3} & 0.0 & a_{3,3} & 0.0 & a_{3,5} & 0.0 \\ a_{1,4} & a_{2,4} & 0.0 & a_{4,4} & 0.0 & a_{4,6} \\ a_{1,5} & 0.0 & a_{3,5} & 0.0 & a_{5,5} & 0.0 \\ 0.0 & 0.0 & 0.0 & a_{4,6} & 0.0 & a_{6,6} \\ \hline \end{array}$$

$$\Downarrow$$

Storage status of arrays aval(na),jcn(na) and ia(n)



Remarks

a. m is the number of nonzero elements in the upper triangular part of the original matrix A including the diagonal.

b. Array aval contains the nonzero upper triangular elements of the original matrix A, stored sequentially beginning with the first row.

c. Array jcn contains the column numbers in the original matrix A of the elements stored in array aval.

d. Array ia contains values equal to one added to the positions in array aval of the diagonal elements.

e. $n \leq m < na$ must hold.

Figure B−15 Storage of Random Symmetric Sparse Matrix (Sparse format)

B.2.12 Random sparse matrix

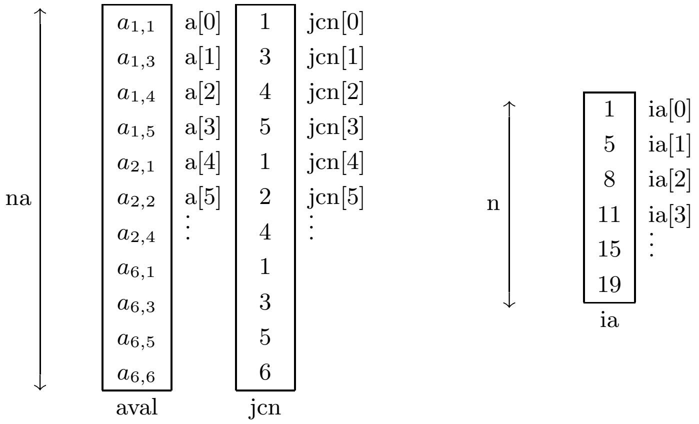
(1) Sparse format

Matrix A to be stored

$$\begin{array}{|cccccc|}
\hline
a_{1,1} & 0.0 & a_{1,3} & a_{1,4} & a_{1,5} & 0.0 \\
a_{2,1} & a_{2,2} & 0.0 & a_{2,4} & 0.0 & 0.0 \\
0.0 & a_{3,2} & a_{3,3} & 0.0 & a_{3,5} & 0.0 \\
a_{4,1} & a_{4,2} & 0.0 & a_{4,4} & 0.0 & a_{4,6} \\
a_{5,1} & 0.0 & a_{5,3} & a_{5,4} & a_{5,5} & 0.0 \\
a_{6,1} & 0.0 & a_{6,3} & 0.0 & a_{6,5} & a_{6,6} \\
\hline
\end{array}$$

$$\Downarrow$$

Storage status of arrays aval[na], jcn[na] and ia[n]



Remarks

a. na is the number of nonzero elements of the original matrix A.

b. Array aval contains the nonzero elements of the original matrix A, stored sequentially beginning with the first row.

c. Array jcn contains the column indices in the original matrix A of the elements stored in array aval.

d. Array ia contains values equal to one added to the positions in array aval of the first nonzero element in each row.

e. n < na must hold.

Figure B−16 Storage of Real Asymmetric Random Sparse Matrix (Sparse Format)

Appendix C

MACHINE CONSTANTS USED IN ASL C INTERFACE

C.1 Units for Determining Error

The table below shows values in ASL C interface 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 C interface uses these units for determining convergence and zeros.

Table C−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.

C.2 Maximum and Minimum Values of Floating Point Data

The table below shows maximum and minimum values of floating point data defined within ASL C interface. 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 C−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

(*) SMP Functions = Shared Memory Parallel Processing Functions