# ADVANCED SCIENTIFIC LIBRARY 

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

## PROPRIETARY NOTICE

The information disclosed in this document is the property of NEC Corporation (NEC) and/or its licensors. NEC and/or its licensors, as appropriate, reserve all patent, copyright and other proprietary rights to this document, including all design, manufacturing, reproduction, use and sales rights thereto, except to extent said rights are expressly granted to others.

The information in this document is subject to change at any time, without 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 <br> item, and usage limitations. |
| 2 | Storage Mode <br> Conversion | Explanation of algorithms, method of using, and usage example <br> of function related to storage mode conversion of array data. |
| 3 | Basic Matrix Algebra | Explanation of algorithms, method of using, and usage example <br> of function related to basic calculations involving matrices. |
| 4 | Eigenvalues and <br> Eigenvectors | Explanation of algorithms, method of using, and usage example <br> of function related to <br> the standard eigenvalue problem for real matrices, complex <br> matrices, real symmetric matrices, Hermitian matrices, real sym- <br> metric band matrices, real symmetric tridiagonal matrices, real <br> symmetric random sparse matrices, Hermitian random sparse <br> matrices and <br> the generalized eigenvalue problem for real matrices, real <br> symmetric matrices, Hermitian matrices, real symmetric band <br> matrices. |

Basic Functions Volume 2

| Chapter | Title | Contents |
| :---: | :--- | :--- |
| 1 | Introduction | Explanation of the organization of this manual, how to view each <br> item, and usage limitations. |
| 2 | Simultaneous Linear <br> Equations <br> (Direct Method) | Explanation of algorithms, method of using, and usage example <br> of function related to simultaneous linear equations correspond- <br> ing to real matrices, complex matrices, positive symmetric ma- <br> trices, real symmetric matrices, Hermitian matrices, real band <br> matrices, positive symmetric band matrices, real tridiagonal ma- <br> trices, real upper triangular matrices, and real lower triangular <br> matrices. |

Basic Functions Volume 3

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

Basic Functions Volume 4

| Chapter | Title | Contents |
| :---: | :--- | :--- |
| 1 | Introduction | Differential Equations <br> and Their Applications <br> item, and usage limitations. |
| 2 | Explanation of algorithms, method of using, and usage example <br> of function related to <br> ordinary differential equations initial value problems for <br> high-order simultaneous ordinary differential equations, implicit <br> simultaneous ordinary differential equations, matrix type ordi- |  |
| nary differential equations, stiff problem high-order simultane- |  |  |
| ous ordinary differential equations, simultaneous ordinary dif- |  |  |
| ferential 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 or- |  |  |
| dinary differential equations, high-order linear ordinary differen- |  |  |
| tial equations, and second-order linear ordinary differential equa- |  |  |
| tions, 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. |  |  |$|$

Basic Functions Volume 5

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

Basic Functions Volume 6

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

Shared Memory Parallel Functions

| Chapter | Title | Contents |
| :---: | :--- | :--- |
| 1 | Introduction | Explanation of the organization of this manual, how to view each <br> item, and usage limitations. |
| 2 | Basic Matrix Algebra | Explanation of algorithms, method of using, and usage example <br> of function related to obtain the product of real matrices and <br> complex matrices. |
| 3 | Simultaneous Linear <br> Equations <br> (Direct Method) | Explanation of algorithms, method of using, and usage example <br> of function related to simultaneous linear equations correspond- <br> ing to real matrices, complex matrices, real symmetric matrices, <br> and Hermitian matrices. |
| 4 | Simultaneous Linear <br> Equations <br> (Iteration Method) | Explanation of algorithms, method of using, and usage example <br> of function related to simultaneous linear equations correspond- <br> ing to real positive definite symmetric sparse matrices, real sym- <br> metric sparse matrices and real asymmetric sparse matrices. |
| 5 | Eigenvalues and <br> Eigenvectors | Explanation of algorithms, method of using, and usage example <br> of function related to the eigenvalue problem for real symmetric <br> matrices and Hermitian matrices. |
| 6 | Fourier Transforms <br> and their applications | Explanation of algorithms, method of using, and usage exam- <br> ple of function related to one-, two- and three-dimensional com- <br> plex Fourier transforms and real Fourier transforms, two- and |
| 7 | Sorting | three-dimensional convolutions, correlations, and power spec- <br> trum analysis. |

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

1 INTRODUCTION ..... 1
1.1 OVERVIEW ..... 1
1.1.1 Introduction to The Advanced Scientific Library ASL C interface ..... 1
1.1.2 Distinctive Characteristics of ASL C interface ..... 1
1.2 KINDS OF LIBRARIES ..... 2
1.3 ORGANIZATION ..... 3
1.3.1 Introduction ..... 3
1.3.2 Organization of Function Description ..... 3
1.3.3 Contents of Each Item ..... 3
1.4 FUNCTION NAMES ..... 7
1.5 NOTES ..... 9
2 SIMULTANEOUS LINEAR EQUATIONS(DIRECT METHOD) ..... 11
2.1 INTRODUCTION ..... 11
2.1.1 Methods of using functions ..... 12
2.1.2 Notes ..... 14
2.1.3 Algorithms Used ..... 16
2.1.3.1 Crout Method ..... 16
2.1.3.2 Cholesky method ..... 17
2.1.3.3 Modified Cholesky method ..... 17
2.1.3.4 Gauss method ..... 18
2.1.3.5 Levinson method ..... 19
2.1.3.6 Vandermonde matrix ..... 20
2.1.3.7 Cyclic Reduction Method ..... 22
2.1.3.8 Calculating the inverse matrix ..... 28
2.1.3.9 Calculating the determinant ..... 28
2.1.3.10 Improving the solution ..... 28
2.1.3.11 Precise estimate of the approximate solution ..... 29
2.1.3.12 Condition Number ..... 29
2.1.4 Reference Bibliography ..... 31
2.2 REAL MATRIX (TWO-DIMENSIONAL ARRAY TYPE) ..... 32
2.2.1 ASL_dbgmsm, ASL_rbgmsm Simultaneous Linear Equations with Multiple Right-Hand Sides (Real Matrix) ..... 32
2.2.2 ASL_dbgmsl, ASL_rbgmsl Simultaneous Linear Equations (Real Matrix) ..... 37
2.2.3 ASL_dbgmlu, ASL_rbgmlu LU Decomposition of a Real Matrix ..... 42
2.2.4 ASL_dbgmlc, ASL_rbgmlc
LU Decomposition and Condition Number of a Real Matrix ..... 44
2.2.5 ASL_dbgmls, ASL_rbgmls
Simultaneous Linear Equations (LU-Decomposed Real Matrix) ..... 46
2.2.6 ASL_dbgmms, ASL_rbgmms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Real Matrix) ..... 48
2.2.7 ASL_dbgmdi, ASL_rbgmdi Determinant and Inverse Matrix of a Real Matrix ..... 52
2.2.8 ASL_dbgmlx, ASL_rbgmlx Improving the Solution of Simultaneous Linear Equations (Real Matrix) ..... 54
2.3 COMPLEX MATRIX (TWO DIMENSIONAL ARRAY TYPE) (REAL ARGUMENT TYPE) ..... 59
2.3.1 ASL_zbgmsm, ASL_cbgmsm
Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix) ..... 59
2.3.2 ASL_zbgmsl, ASL_cbgmsl Simultaneous Linear Equations (Complex Matrix) ..... 64
2.3.3 ASL_zbgmlu, ASL_cbgmlu LU Decomposition of a Complex Matrix ..... 70
2.3.4 ASL_zbgmlc, ASL_cbgmlc LU Decomposition and Condition Number of a Complex Matrix ..... 72
2.3.5 ASL_zbgmls, ASL_cbgmls Simultaneous Linear Equations (LU-Decomposed Complex Matrix) ..... 74
2.3.6 ASL_zbgmms, ASL_cbgmms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Complex Matrix) ..... 76
2.3.7 ASL_zbgmdi, ASL_cbgmdi Determinant and Inverse Matrix of a Complex Matrix ..... 80
2.3.8 ASL_zbgmlx, ASL_cbgmlx Improving the Solution of Simultaneous Linear Equations (Complex Matrix) ..... 82
2.4 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (COMPLEX ARGUMENT TYPE) ..... 84
2.4.1 ASL_zbgnsm, ASL_cbgnsm Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix) ..... 84
2.4.2 ASL_zbgnsl, ASL_cbgnsl Simultaneous Linear Equations (Complex Matrix) ..... 88
2.4.3 ASL_zbgnlu, ASL_cbgnlu LU Decomposition of a Complex Matrix ..... 92
2.4.4 ASL_zbgnlc, ASL_cbgnlc
LU Decomposition and Condition Number of a Complex Matrix ..... 94
2.4.5 ASL_zbgnls, ASL_cbgnls
Simultaneous Linear Equations (LU-Decomposed Complex Matrix) ..... 96
2.4.6 ASL_zbgnms, ASL_cbgnms Simultaneous Linear Equations with Multiple Right-Hand Sides (LU-Decomposed Complex Matrix) ..... 98
2.4.7 ASL_zbgndi, ASL_cbgndi Determinant and Inverse Matrix of a Complex Matrix ..... 102
2.4.8 ASL_zbgnlx, ASL_cbgnlx Improving the Solution of Simultaneous Linear Equations (Complex Matrix) ..... 104
2.5 POSITIVE SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIAN- GULAR TYPE) ..... 106
2.5.1 ASL_dbpdsl, ASL_rbpdsl Simultaneous Linear Equations (Positive Symmetric Matrix) ..... 106
2.5.2 ASL_dbpduu, ASL_rbpduu $L^{\mathrm{T}}$ Decomposition of a Positive Symmetric Matrix ..... 110
2.5.3 ASL_dbpduc, ASL_rbpduc
LL $^{\mathrm{T}}$ Decomposition and Condition Number of a Positive Symmetric Matrix ..... 112
2.5.4 ASL_dbpdls, ASL_rbpdls Simultaneous Linear Equations (LL${ }^{T}$-Decomposed Positive Symmetric Matrix) ..... 114
2.5.5 ASL_dbpddi, ASL_rbpddi Determinant and Inverse Matrix of a Positive Symmetric Matrix ..... 116
2.5.6 ASL_dbpdlx, ASL_rbpdlx Improving the Solution of Simultaneous Linear Equations (Positive Symmetric Matrix) ..... 118
2.6 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGU- LAR TYPE) ..... 120
2.6.1 ASL_dbspsl, ASL_rbspsl Simultaneous Linear Equations (Real Symmetric Matrix) ..... 120
2.6.2 ASL_dbspud, ASL_rbspud LDL ${ }^{\mathrm{T}}$ Decomposition of a Real Symmetric Matrix ..... 125
2.6.3 ASL_dbspuc, ASL_rbspuc LDL $^{\mathrm{T}}$ Decomposition and Condition Number of a Real Symmetric Matrix ..... 127
2.6.4 ASL_dbspls, ASL_rbspls
Simultaneous Linear Equations (LDL ${ }^{\mathrm{T}}$-Decomposed Real Symmetric Matrix) ..... 129
2.6.5 ASL_dbspms, ASL_rbspms Simultaneous Linear Equations with Multiple Right-Hand Sides ( LDL ${ }^{T}$ decomposed Real Matrix ) ..... 131
2.6.6 ASL_dbspdi, ASL_rbspdi Determinant and Inverse Matrix of a Real Symmetric Matrix ..... 135
2.6.7 ASL_dbsplx, ASL_rbsplx Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix) ..... 137
2.7 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGU- LAR TYPE)(NO PIVOTING) ..... 139
2.7.1 ASL_dbsmsl, ASL_rbsmsl Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting) ..... 139
2.7.2 ASL_dbsmud, ASL_rbsmud LDL $^{\mathrm{T}}$ Decomposition of a Real Symmetric Matrix (No Pivoting) ..... 144
2.7.3 ASL_dbsmuc, ASL_rbsmuc LDL $^{\mathrm{T}}$ Decomposition and Condition Number of a Real Symmetric Matrix (No Pivoting) . 146
2.7.4 ASL_dbsmls, ASL_rbsmls
Simultaneous Linear Equations (LDL ${ }^{T}$-Decomposed Real Symmetric Matrix) (No Pivoting) ..... 148
2.7.5 ASL_dbsmms, ASL_rbsmms Simultaneous Linear Equations with Multiple Right-Hand Sides ( LDL $^{T}$-Decomposed Real Matrix ) ( No Pivoting ) ..... 150
2.7.6 ASL_dbsmdi, ASL_rbsmdi Determinant and Inverse Matrix of a Real Symmetric Matrix (No Pivoting) ..... 154
2.7.7 ASL_dbsmlx, ASL_rbsmlx
Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting) ..... 156
2.8 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE, LOWER TRIANGU- LAR TYPE)(NO PIVOTING) ..... 158
2.8.1 ASL_dbsnsl, ASL_rbsnsl Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting) ..... 158
2.8.2 ASL_dbsnud, ASL_rbsnud $\mathrm{U}^{\mathrm{T}}$ DU Decomposition of a Real Symmetric Matrix (No Pivoting) ..... 163
2.8.3 ASL_dbsnls, ASL_rbsnls Simultaneous Linear Equations (U ${ }^{\mathrm{T}}$ DU-Decomposed Real Symmetric Matrix) (No Pivoting) 165
2.9 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) ..... 167
2.9.1 ASL_zbhpsl, ASL_cbhpsl Simultaneous Linear Equations (Hermitian Matrix) ..... 167
2.9.2 ASL_zbhpud, ASL_cbhpud
LDL* Decomposition of a Hermitian Matrix ..... 172
2.9.3 ASL_zbhpuc, ASL_cbhpuc LDL* Decomposition and Condition Number of a Hermitian Matrix ..... 174
2.9.4 ASL_zbhpls, ASL_cbhpls Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) ..... 176
2.9.5 ASL_zbhpms, ASL_cbhpms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Her- mitian Matrix) ..... 178
2.9.6 ASL_zbhpdi, ASL_cbhpdi Determinant and Inverse Matrix of a Hermitian Matrix ..... 182
2.9.7 ASL_zbhplx, ASL_cbhplx
Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) ..... 184
2.10 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) (NO PIVOTING) ..... 186
2.10.1 ASL_zbhrsl, ASL_cbhrsl Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) ..... 186
2.10.2 ASL_zbhrud, ASL_cbhrud LDL* Decomposition of a Hermitian Matrix (No Pivoting) ..... 191
2.10.3 ASL_zbhruc, ASL_cbhruc LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting) ..... 193
2.10.4 ASL_zbhrls, ASL_cbhrls Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting) ..... 195
2.10.5 ASL_zbhrms, ASL_cbhrms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Her- mitian Matrix) (No Pivoting) ..... 197
2.10.6 ASL_zbhrdi, ASL_cbhrdi Determinant and Inverse Matrix of a Hermitian Matrix (No Pivoting) ..... 201
2.10.7 ASL_zbhrlx, ASL_cbhrlx Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) 203
2.11 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) ..... 205
2.11.1 ASL_zbhfsl, ASL_cbhfsl Simultaneous Linear Equations (Hermitian Matrix) ..... 205
2.11.2 ASL_zbhfud, ASL_cbhfud LDL* Decomposition of a Hermitian Matrix ..... 210
2.11.3 ASL_zbhfuc, ASL_cbhfuc LDL* Decomposition and Condition Number of a Hermitian Matrix ..... 212
2.11.4 ASL_zbhfls, ASL_cbhfls Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) ..... 214
2.11.5 ASL_zbhfms, ASL_cbhfms Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Her- mitian Matrix) ..... 216
2.11.6 ASL_zbhfdi, ASL_cbhfdi Determinant and Inverse Matrix of a Hermitian Matrix ..... 220
2.11.7 ASL_zbhflx, ASL_cbhflx Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) ..... 222
2.12 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) (NO PIVOTING) ..... 224
2.12.1 ASL_zbhesl, ASL_cbhesl Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) ..... 224
2.12.2 ASL_zbheud, ASL_cbheud
LDL* Decomposition of a Hermitian Matrix (No Pivoting) ..... 229
2.12.3 ASL_zbheuc, ASL_cbheuc LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting) ..... 231
2.12.4 ASL_zbhels, ASL_cbhels Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting) ..... 233
2.12.5 ASL_zbhems, ASL_cbhems
Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*-Decomposed Her- mitian Matrix) (No Pivoting) ..... 235
2.12.6 ASL_zbhedi, ASL_cbhedi Determinant and Inverse Matrix of a Hermitian Matrix (No Pivoting) ..... 239
2.12.7 ASL_zbhelx, ASL_cbhelx Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) 241
2.13 REAL BAND MATRIX (BAND TYPE) ..... 243
2.13.1 ASL_dbbdsl, ASL_rbbdsl Simultaneous Linear Equations (Real Band Matrix) ..... 243
2.13.2 ASL_dbbdlu, ASL_rbbdlu LU Decomposition of a Real Band Matrix ..... 248
2.13.3 ASL_dbbdlc, ASL_rbbdlc
LU Decomposition and Condition Number of a Real Band Matrix ..... 250
2.13.4 ASL_dbbdls, ASL_rbbdls Simultaneous Linear Equations (LU-Decomposed Real Band Matrix) ..... 253
2.13.5 ASL_dbbddi, ASL_rbbddi Determinant of a Real Band Matrix ..... 255
2.13.6 ASL_dbbdlx, ASL_rbbdlx Improving the Solution of Simultaneous Linear Equations (Real Band Matrix) ..... 257
2.14 POSITIVE SYMMETRIC BAND MATRIX (SYMMETRIC BAND TYPE) ..... 262
2.14.1 ASL_dbbpsl, ASL_rbbpsl Simultaneous Linear Equations (Positive Symmetric Band Matrix) ..... 262
2.14.2 ASL_dbbpuu, ASL_rbbpuu $L^{T}$ Decomposition of a Positive Symmetric Band Matrix ..... 266
2.14.3 ASL_dbbpuc, ASL_rbbpuc
$L^{\mathrm{T}}$ Decomposition and Condition Number of a Positive Symmetric Band Matrix ..... 268
2.14.4 ASL_dbbpls, ASL_rbbpls Simultaneous Linear Equations (LL ${ }^{T}$-Decomposed Positive Symmetric Band Matrix) ..... 270
2.14.5 ASL_dbbpdi, ASL_rbbpdi Determinant of a Positive Symmetric Band Matrix ..... 272
2.14.6 ASL_dbbplx, ASL_rbbplx Improving the Solution of Simultaneous Linear Equations (Positive Symmetric Band Matrix)274
2.15 REAL TRIDIAGONAL MATRIX (VECTOR TYPE) ..... 276
2.15.1 ASL_dbtdsl, ASL_rbtdsl Simultaneous Linear Equations (Real Tridiagonal Matrix) ..... 276
2.15.2 ASL_dbtpsl, ASL_rbtpsl Simultaneous Linear Equations (Positive Symmetric Tridiagonal Matrix) ..... 280
2.16 REAL TRIDIAGONAL MATRIX (VECTOR TYPE) ..... 284
2.16.1 ASL_wbtdsl
Simultaneous Linear Equations (Real Tridiagonal Matrix) ..... 284
2.16.2 ASL_wbtdls
Simultaneous Linear Equations (Real Tridiagonal Matrix after Reduction Operations) ..... 288
2.17 FIXED COEFFICIENT REAL TRIDIAGONAL MATRIX (SCALAR TYPE) ..... 292
2.17.1 ASL_wbtcsl Simultaneous Linear Equations (Fixed Coefficient Real Tridiagonal Matrix) ..... 292
2.17.2 ASL_wbtcls Simultaneous Linear Equations (Fixed Coefficient Real Tridiagonal Matrix after Reduction Operations) ..... 297
2.18 VANDERMONDE MATRIX AND TOEPLITZ MATRIX ..... 302
2.18.1 ASL_dbtosl, ASL_rbtosl Simultaneous Linear Equations (Toeplitz Matrix) ..... 302
2.18.2 ASL_dbtssl, ASL_rbtssl Simultaneous Linear Equations (Symmetric Toeplitz Matrix) ..... 306
2.18.3 ASL_dbvmsl, ASL_rbvmsl Simultaneous Linear Equations (Vandermonde Matrix) ..... 310
2.19 REAL UPPER TRIANGULAR MATRIX (TWO-DIMENSIONAL ARRAY TYPE) ..... 314
2.19.1 ASL_dbtusl, ASL_rbtusl Simultaneous Linear Equations (Real Upper Triangular Matrix) ..... 314
2.19.2 ASL_dbtuco, ASL_rbtuco Condition Number of a Real Upper Triangular Matrix ..... 317
2.19.3 ASL_dbtudi, ASL_rbtudi Determinant and Inverse Matrix of a Real Upper Triangular Matrix ..... 319
2.20 REAL LOWER TRIANGULAR MATRIX
(TWO-DIMENSIONAL ARRAY TYPE) ..... 321
2.20.1 ASL_dbtlsl, ASL_rbtlsl Simultaneous Linear Equations (Real Lower Triangular Matrix) ..... 321
2.20.2 ASL_dbtlco, ASL_rbtlco Condition Number of a Real Lower Triangular Matrix ..... 324
2.20.3 ASL_dbtldi, ASL_rbtldi Determinant and Inverse Matrix of a Real Lower Triangular Matrix ..... 326
A GLOSSARY ..... 328
B METHODS OF HANDLING ARRAY DATA ..... 337
B. 1 Methods of handling array data corresponding to matrix ..... 337
B. 2 Data storage modes ..... 339
B.2.1 Real matrix (two-dimensional array type) ..... 339
B.2.2 Complex matrix ..... 340
B.2.3 Real symmetric matrix and positive symmetric matrix ..... 342
B.2.4 Hermitian matrix ..... 344
B.2.5 Real band matrix ..... 346
B.2.6 Real symmetric band matrix and positive symmetric matrix (symmetric band type) ..... 347
B.2.7 Real tridiagonal matrix (vector type) ..... 348
B.2.8 Real symmetric tridiagonal matrix and positive symmetric tridiagonal matrix (vector type) ..... 349
B.2.9 Fixed coefficient real tridiagonal matrix (scalar type) ..... 349
B.2.10 Triangular matrix ..... 350
B.2.11 Random sparse matrix (For symmetric matrix only) ..... 350
B.2.12 Random sparse matrix ..... 351
C MACHINE CONSTANTS USED IN ASL C INTERFACE ..... 352
C. 1 Units for Determining Error ..... 352
C. 2 Maximum and Minimum Values of Floating Point Data ..... 352

## 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 |  |
| 4 | 4 | int float | 32bit integer Single-precision function |  |
| 8 | 8 | long double | 64bit integer Double-precision function |  |
| 8 | 4 | long <br> 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.
$(* \mathbf{2})$ 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 $3 \times n$ or $n+m$.
(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:

$$
\text { ierr }=\text { ASL_dbgmlc }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \text { ipvt, \&cond, w1); }
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbgmlc }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{ipvt}, \& c o n d, \mathrm{w} 1) ;
$$

The explanation of the arguments and return value is as follows.
Table 1-3 Sample Arguments and Return Value

| D:Double precision real R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | a | Note | $\ln a \times n$ | Input | Real matrix $A$ (two-dimensional array) |
|  |  | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ |  | Output | The matrix $A$ decomposed into the matrix $L U$ where $U$ is a unit upper triangular matrix and $L$ is a lower triangular matrix. |
| 2 | $\operatorname{lna}$ | 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 | $\left\{\begin{array}{l}\text { D } \\ \mathrm{R}\end{array}\right\}$ | 1 | Output | Reciprocal of the condition number |
| 6 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 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 $\left\{\begin{array}{c}\text { double-precision } \\ \text { single-precision }\end{array}\right\}$ Note real array of size $[\ln a \times n]$, ipvt is an integer array of size $n$ and w1 is a $\left\{\begin{array}{c}\text { double-precision } \\ \text { single-precision }\end{array}\right\}$ real array of size $n$.
When the 64 -bit integer version is used, all integer-type arguments (lna, n, ipvt and ierr) must be declared by using 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 | $1000 \sim 2999$ | Processing is terminated under cer- <br> tain conditions. | Results are conditionally guaranteed. |
| Fatal | $3000 \sim 3499$ | Processing is aborted since an argu- <br> ment violated its restrictions. | Results are not guaranteed. |
|  | $3500 \sim 3999$ | Obtained results did not satisfy a cer- <br> tain condition. | Obtained results are returned (the <br> results are not guaranteed). |
|  | 4000 or more | A fatal error was detected during <br> processing. Usually, processing is <br> 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 $\rangle$.

Figure 1-1 Function Name Components

ASL_

"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 | 4 |
|  | 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.hmust 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 " $\langle 6$ lowercase letters $\rangle$ 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 $\pi$ or function approximation constants, which are fixed in double-precision operations, for example, are also to be subject to calculations that depend on the length of the multiple length arithmetic operations, the calculation efficiency will be worse than for normal operations.
(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 $\left\{\begin{array}{c}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}$

$$
\text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmsl } \\
\text { ASL_rbgmsl }
\end{array}\right\}(A, \cdots, \boldsymbol{b}, \cdots)
$$

Performs a triangular decomposition of coefficient matrix $A$ and solves $A \boldsymbol{x}=\boldsymbol{b}$.
(2) Using $\left\{\begin{array}{c}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ and $\left\{\begin{array}{c}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$

$$
\begin{aligned}
& \text { ierr }=\left\{\begin{array}{l}
\text { ASL_dbgmlu } \\
\text { ASL_rbgmlu }
\end{array}\right\}(A, \cdots) ; \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}(A, \cdots, \boldsymbol{b}, \cdots) ;
\end{aligned}
$$

$\left\{\begin{array}{c}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ performs a triangular decomposition of coefficient matrix $A$, and $\left\{\begin{array}{c}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$ solves $A \boldsymbol{x}=\boldsymbol{b}$.
(3) Obtaining the condition number in addition to solving simultaneous linear equations

$$
\begin{aligned}
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmlc } \\
\text { ASL_rbgmlc }
\end{array}\right\}(A, \cdots, \& \text { cond }, \cdots) \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}(A, \cdots, \boldsymbol{b}, \cdots)
\end{aligned}
$$

$\left\{\begin{array}{l}\text { ASL_dbgmlc } \\ \text { ASL_rbgmlc }\end{array}\right\}$ calculates the condition number and performs a triangular decomposition of coefficient matrix $A$, and $\left\{\begin{array}{c}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$ solves $A \boldsymbol{x}=\boldsymbol{b}$.
(2) Determinant and inverse matrix

$$
\begin{aligned}
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmlu } \\
\text { ASL_rbgmlu }
\end{array}\right\}(A, \cdots) ; \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmdi } \\
\text { ASL_rbgmdi }
\end{array}\right\}(A, \cdots, \text { det }, \cdots)
\end{aligned}
$$

$\left\{\begin{array}{l}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ performs a triangular decomposition of matrix $A$, and $\left\{\begin{array}{l}\text { ASL_dbgmdi } \\ \text { ASL_rbgmdi }\end{array}\right\}$ obtains the determinant and inverse matrix.
(3) Improving the solution

$$
\begin{aligned}
& \text { (1) Using }\left\{\begin{array}{c}
\text { ASL_dbgmsl } \\
\text { ASL_rbgmsl }
\end{array}\right\} \\
& \begin{array}{l}
A_{2} \leftarrow A
\end{array} \\
& \boldsymbol{b}_{\mathbf{2}} \leftarrow \boldsymbol{b} \\
& \qquad \operatorname{ierr}=\left\{\begin{array}{c}
\text { ASL_dbgmsl } \\
\text { ASL_rbgmsl }
\end{array}\right\}\left(A_{2}, \cdots, \boldsymbol{b}_{2}, \cdots\right)
\end{aligned}
$$

$$
\text { ierr }=\left\{\begin{array}{l}
\text { ASL_dbgmlx } \\
\text { ASL_rbgmlx }
\end{array}\right\}\left(A, \cdots, A_{2}, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_{2}, \cdots\right) ;
$$

The function shown above improves the solution obtained by using $\left\{\begin{array}{l}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}$.
(2) Using $\left\{\begin{array}{l}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ and $\left\{\begin{array}{l}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$
$A_{2} \leftarrow A$
$b_{2} \leftarrow b$

$$
\begin{aligned}
& \text { ierr }=\left\{\begin{array}{l}
\text { ASL_dbgmlu } \\
\text { ASL_rbgmlu }
\end{array}\right\}\left(A_{2}, \cdots\right) ; \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}\left(A_{2}, \cdots, \boldsymbol{b}_{2}, \cdots\right) ; \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmlx } \\
\text { ASL_rbgmlx }
\end{array}\right\}\left(A, \cdots, A_{2}, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_{2}, \cdots\right) ;
\end{aligned}
$$

$\left\{\begin{array}{c}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ performs a triangular decomposition of matrix $A,\left\{\begin{array}{l}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$ solves $A \boldsymbol{x}=\boldsymbol{b}$, and $\left\{\begin{array}{l}\text { ASL_dbgmlx } \\ \text { ASL_rbgmlx }\end{array}\right\}$ 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:
or

$$
\begin{aligned}
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmsl } \\
\text { ASL_rbgmsl }
\end{array}\right\}\left(A, \cdots, \boldsymbol{b}_{1}, \cdots\right) ; \\
& \text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}\left(A, \cdots, \boldsymbol{b}_{2}, \cdots\right) ;
\end{aligned}
$$

$$
\text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmlu } \\
\text { ASL_rbgmlu }
\end{array}\right\}(A, \cdots) ;
$$

$$
\text { ierr }=\left\{\begin{array}{l}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}\left(A, \cdots, \boldsymbol{b}_{1}, \cdots\right) ;
$$

$$
\text { ierr }=\left\{\begin{array}{c}
\text { ASL_dbgmls } \\
\text { ASL_rbgmls }
\end{array}\right\}\left(A, \cdots, \boldsymbol{b}_{2}, \cdots\right) ;
$$

$\left\{\begin{array}{c}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}$ or $\left\{\begin{array}{c}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ 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 $U$.

$$
A=L U
$$



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


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 $L^{T}$.

$$
A=L L^{T}
$$



Assume $A=\left(a_{i j}\right), L=\left(l_{i j}\right)$ and $L^{T}=\left(l_{i j}^{\prime}\right)(i, j=1,2, \cdots, N)$. 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.


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 $L^{T}$.

$$
A=L D L^{T}
$$

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


Assume $A=\left(a_{i j}\right), L=\left(l_{i j}\right), D=\left(d_{i j}\right)$ and $L^{T}=\left(l_{i j}^{\prime}\right)(i, j=1,2, \cdots, N)$. Then, the algorithm is as follows.

$$
l_{1 j}^{\prime} \leftarrow a_{1 j}(j=1,2,3, \cdots, N)
$$


$w$ 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 $U$.

$$
A=L U
$$



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

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:

$$
R=\left[\begin{array}{cccccc}
r_{0} & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\
r_{1} & r_{0} & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\
\vdots & \vdots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \vdots & \vdots \\
r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_{0} & r_{-1} \\
r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_{1} & r_{0}
\end{array}\right]
$$

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 $(m=1)$

$$
\begin{aligned}
& x_{1}^{(1)}=\frac{b_{1}}{r_{0}} \\
& g_{1}^{(1)}=\frac{r_{-1}}{r_{0}} \\
& h_{1}^{(1)}=\frac{r_{1}}{r_{0}}
\end{aligned}
$$

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

$$
\begin{aligned}
& x^{(n u)}=\sum_{j=1}^{m-1} r_{m-j} x_{j}-b_{m} \\
& x^{(d e)}=\sum_{j=1}^{m-1} r_{m-j} g_{m-j}^{(m-1)}-r_{0} \\
& x_{m}^{(m)}=\frac{x^{(n u)}}{x^{(d e)}} \\
& x_{j}^{(m)}=x_{j}^{(m-1)}-x_{m}^{(m)} g_{m-j}^{(m-1)} \quad(j=1,2, \cdots, m-1) \\
& g^{(n u)}=\sum_{j=1}^{m-1} r_{j-m} g_{j}^{(m-1)}-r_{-m} \\
& g^{(d e)}=\sum_{j=1}^{m-1} r_{j-m} h_{m-j}^{(m-1)}-r_{0}
\end{aligned}
$$

$$
\begin{aligned}
& h^{(n u)}=\sum_{j=1}^{m-1} r_{m-j} h_{j}^{(m-1)}-r_{m} \\
& g_{m}^{(m)}=\frac{g^{(n u)}}{g^{(d e)}} \\
& h_{m}^{(m)}=\frac{h^{(n u)}}{x^{(d e)}} \\
& g_{j}^{(m)}=g_{j}^{(m-1)}-g_{m}^{(m)} h_{m-j}^{(m-1)} \quad(j=1,2, \cdots, m-1) \\
& h_{j}^{(m)}=h_{j}^{(m-1)}-h_{m}^{(m)} g_{m-j}^{(m-1)} \quad(j=1,2, \cdots, m-1)
\end{aligned}
$$

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.

$$
V=\left[\begin{array}{cccccc}
1 & v_{1} & v_{1}^{2} & \cdots & v_{1}^{n-2} & v_{1}^{n-1} \\
1 & v_{2} & v_{2}^{2} & \cdots & v_{2}^{n-2} & v_{2}^{n-1} \\
\vdots & \vdots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \vdots & \vdots \\
1 & v_{n-1} & v_{n-1}^{2} & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\
1 & v_{n} & v_{n}^{2} & \cdots & v_{n}^{n-2} & v_{n}^{n-1}
\end{array}\right]
$$

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:

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

the relationship $P_{i}^{(n)}\left(v_{k}\right)=\delta_{i k}$ (where $\delta_{i k}$ 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 $U=\left\{u_{i, j}\right\}$, the relationship $U V^{T}=E$ (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.

$$
P^{(n)}(x)=\prod_{k=1}^{n}\left(x-v_{k}\right)
$$

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

$$
\begin{aligned}
& w_{1}^{(i)}=w_{1}^{(i-1)}-v_{i} \quad(i=2, \cdots, n) \\
& w_{j}^{(i)}=w_{j}^{(i-1)}-v_{i} w_{j-1}^{(i-1)} \quad(j=i, i-1, \cdots, 2 ; i=2,3, \cdots, n)
\end{aligned}
$$

where, the following equations hold.

$$
\begin{aligned}
& w_{1}^{(1)}=-v_{1} \\
& w_{j}^{(j-1)}=0 \quad(j=2,3, \cdots, n)
\end{aligned}
$$

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

$$
\left.\frac{d P^{(n)}(x)}{d x}\right|_{x=v_{i}}=\prod_{\substack{k=1 \\(k \neq i)}}^{n}\left(v_{i}-v_{k}\right)
$$

and this value can be calculated from the following:

$$
\left.\frac{d P^{(n)}(x)}{d x}\right|_{x=v_{i}}=n v_{i}^{n-1}+(n-1) w_{1}^{(n)} v_{i}^{n-2}+\cdots+w_{n-1}^{(n)}
$$

Also, since:

$$
P_{i}^{(n)}(x)=\frac{P^{(n)}(x)}{\left.\left(x-v_{i}\right) \frac{d P^{(n)}(x)}{d x}\right|_{x=v_{i}}}
$$

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 $\frac{P^{(n)}(x)}{\left(x-v_{i}\right)}$. 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:

$$
\begin{equation*}
A \boldsymbol{x}=\boldsymbol{b} \tag{2.1}
\end{equation*}
$$

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

$$
A=\left[\begin{array}{ccccc}
d_{1} & u_{1} & & & 0 \\
\ell_{2} & d_{2} & u_{2} & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & \cdot & u_{n-1} \\
& 0 & & \ell_{n} & d_{n}
\end{array}\right], \boldsymbol{x}=\left[\begin{array}{c}
x_{1} \\
x_{2} \\
\cdot \\
\cdot \\
x_{n}
\end{array}\right], \boldsymbol{b}=\left[\begin{array}{c}
b_{1} \\
b_{2} \\
\cdot \\
\cdot \\
b_{n}
\end{array}\right]
$$

then:

$$
\begin{equation*}
\ell_{i} x_{i-1}+d_{i} x_{i}+u_{i} x_{i+1}=b_{i} \tag{2.2}
\end{equation*}
$$

This algorithm repeatedly performs a reduction operation $\left\lfloor\log _{2}(n)\right\rfloor$ 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.

$$
\begin{align*}
d x & =b \\
x & =b / d \tag{2.3}
\end{align*}
$$

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:

$$
\begin{align*}
& \ell_{i}^{\prime} x_{i-2} \tag{2.4}
\end{align*}+d_{i}^{\prime} x_{i}+u_{i}^{\prime} x_{i+2}=b_{i}^{\prime} .
$$

from the three rows:

$$
\left\{\begin{array}{rlrll}
\ell_{i-1} x_{i-2}+d_{i-1} x_{i-1}+u_{i-1} x_{i} & & & =b_{i-1} \\
& \ell_{i} x_{i-1} & +d_{i} x_{i}+u_{i} x_{i+1} & & =b_{i} \\
& & \ell_{i+1} x_{i}+d_{i+1} x_{i+1}+u_{i+1} x_{i+2} & =b_{i+1}
\end{array}\right.
$$

where, $i$ is an even number.
By applying (2.4) to all even numbered rows contained in (2.1) $\left(x_{0}=x_{n+1}=0\right)$, 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:

$$
\begin{align*}
& \ell_{n}^{\prime} x_{n-2}+d_{n}^{\prime} x_{n}=b_{n}^{\prime}  \tag{2.5}\\
& \left\{\begin{aligned}
\ell_{n}^{\prime} & =\ell_{n-1} \ell_{n} \\
d_{n}^{\prime} & =\ell_{n} u_{n-1}-d_{n} d_{n-1} \\
b_{n}^{\prime} & =\ell_{n} b_{n-1}-d_{n-1} d_{n}
\end{aligned}\right.
\end{align*}
$$

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

$$
\left\{\begin{array}{rlll}
\ell_{n-1} x_{n-2}+d_{n-1} x_{n-1} & +u_{n-1} x_{n} & =b_{n-1} \\
\ell_{n} x_{n-1} & +d_{n} x_{n} & =b_{n}
\end{array}\right.
$$

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:

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

(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:

$$
\left[\begin{array}{ccccc}
d & s & & & 0  \tag{2.6}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & \cdot & s \\
0 & & & s & d
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0
$$

$$
\left[\begin{array}{ccccc}
d & s & & & 0  \tag{2.7}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & \cdot & s \\
0 & & & 2 \cdot s & d
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0
$$

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).

$$
\left[\begin{array}{ccccc}
d & s & & & 0  \tag{2.8}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & d & s \\
0 & & & s & e
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0, \quad e \neq 0
$$

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:

$$
\begin{align*}
& s^{\prime} x_{i-2}+d^{\prime} x_{i}+s^{\prime} x_{i+2}=b_{i}^{\prime}  \tag{2.9}\\
& \left\{\begin{aligned}
s^{\prime} & =s^{2} \\
d^{\prime} & =2 \cdot s^{2}-d^{2} \\
b^{\prime} & =s\left(b_{i-1}+b_{i+1}\right)-d b_{i}
\end{aligned}\right.
\end{align*}
$$

from the three rows:

$$
\left\{\begin{array}{rlrl}
s x_{i-2}+d x_{i-1} & +s x_{i} & & \\
& =b_{i-1} \\
s x_{i-1} & +d x_{i}+s x_{i+1} & & =b_{i} \\
& & s x_{i}+d x_{i+1}+s x_{i+2} & =b_{i+1}
\end{array}\right.
$$

where, i is an even number.
By applying (2.9) to all even numbered rows contained in (2.8) $\left(x_{0}=x_{n+1}=0\right)$, 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:

$$
\begin{align*}
& s^{\prime} x_{n-3}+e^{\prime} x_{n-1}=b_{n-1}^{\prime}  \tag{2.10}\\
& \begin{cases}s^{\prime} & =e \cdot s^{2} \\
e^{\prime} & =e \cdot s^{2}-e \cdot d^{2}+d \cdot s^{2} \\
b_{n-1}^{\prime} & =e \cdot s \cdot b_{n-2}-e \cdot d \cdot b_{n-1}+d \cdot s \cdot b_{n}\end{cases}
\end{align*}
$$

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:

$$
\begin{align*}
& s^{\prime} x_{n-2}+e^{\prime} x_{n}=b_{n-1}  \tag{2.11}\\
& \begin{cases}s^{\prime} & =s^{2} \\
d^{\prime} & =s^{2}-d \cdot e \\
b_{n-1}^{\prime} & =s \cdot b_{n-1}-d \cdot b_{n}\end{cases}
\end{align*}
$$

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

$$
\left\{\begin{aligned}
s x_{n-2}+d x_{n-1} & +s x_{n}=b_{n-1} \\
s x_{n-1} & +e x_{n}=b_{n}
\end{aligned}\right.
$$

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 $\left\lfloor\log _{2}(n)\right\rfloor$ times until, ultimately, a single linear equation is created from which a single solution is obtained.

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

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:

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

Next, consider the following coefficient matrices:

$$
\left[\begin{array}{ccccc}
d & 2 \cdot s & & & 0  \tag{2.12}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & \cdot & s \\
0 & & & s & d
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0
$$

$$
\left[\begin{array}{ccccc}
d & 2 \cdot s & & & 0  \tag{2.13}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & \cdot & s \\
0 & & & 2 \cdot s & d
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0
$$

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).

$$
\left[\begin{array}{ccccc}
d & 2 \cdot s & & & 0  \tag{2.14}\\
s & d & s & & \\
& \cdot & \cdot & \cdot & \\
& & \cdot & d & s \\
0 & & & s & e
\end{array}\right] \quad, \quad d \neq 0, \quad s \neq 0, \quad e \neq 0
$$

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:

$$
\begin{equation*}
\left(d^{2}-2 \cdot s^{2}\right) x_{1}-2 \cdot s^{2} x_{3}=d b_{1}-2 s b_{2} \tag{2.15}
\end{equation*}
$$

from the two rows:

$$
\left\{\begin{array}{lll}
d x_{1}+2 \cdot s x_{2} & =b_{1} \\
s x_{1}+d x_{2}+s x_{3} & =b_{2}
\end{array}\right.
$$

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

$$
\begin{align*}
& s^{\prime} x_{2 i-1}+d^{\prime} x_{2 i+1}+s^{\prime} x_{2 i+3}=b_{2 i+1}^{\prime}  \tag{2.16}\\
& \begin{cases}s^{\prime} & =s^{2} \\
d^{\prime} & =2 \cdot s^{2}-d^{2} \\
b_{2 i+1}^{\prime} & =s \cdot b_{2 i}-d \cdot b_{2 i+1}+s \cdot b_{2 i+2}\end{cases}
\end{align*}
$$

from the three rows:

$$
\left\{\begin{aligned}
s x_{2 i-1}+d x_{2 i}+s x_{2 i+1} & & =b_{2 i} \\
s x_{2 i}+d x_{2 i+1}+s x_{2 i+2} & & =b_{2 i+1} \\
& s d_{2 i+1}+s x_{2 i+2}+s x_{2 i+3} & =b_{2 i+2}
\end{aligned}\right.
$$

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 $2 i+1 \leq n-2$.

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:

$$
\begin{align*}
& s^{\prime} x_{n-3}+e^{\prime} x_{n-1}=b_{n-1}^{\prime}  \tag{2.17}\\
& \begin{cases}s^{\prime} & =e \cdot s^{2} \\
e^{\prime} & =e \cdot s^{2}-e \cdot d^{2}+d \cdot s^{2} \\
b_{n-1}^{\prime} & =s \cdot e \cdot b_{n-2}-d \cdot e \cdot b_{n-1}+d \cdot s \cdot b_{n}\end{cases}
\end{align*}
$$

from the three rows:

$$
\left\{\begin{array}{rlrl}
s x_{n-3}+d x_{n-2} & +s x_{n-1} & & =b_{n-2} \\
s x_{n-2} & +d x_{n-1}+s x_{n} & =b_{n-1} \\
& s x_{n-1}+e x_{n} & =b_{n}
\end{array}\right.
$$

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

$$
\begin{align*}
& s^{\prime} x_{n-2}+e^{\prime} x_{n}=b_{n}^{\prime}  \tag{2.18}\\
& \left\{\begin{array}{l}
s^{\prime}=s^{2} \\
e^{\prime}=s^{2}-e \cdot d \\
b_{n}^{\prime}=s \cdot b_{n-1}-d \cdot b_{n}
\end{array}\right.
\end{align*}
$$

from the two rows:

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 $\left\lfloor\log _{2}(n-1)\right\rfloor$ times until, ultimately, a set of equations having the following coefficient matrix is obtained:

$$
\left[\begin{array}{ll}
d^{(\mathrm{m})} & 2 \\
1 & e^{(\mathrm{m})}
\end{array}\right], \quad m=\lfloor(n-1) / 2\rfloor+1
$$

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 $1 / \mathrm{ep}$ (ep: Units for determining error) at an intermediate stage of the reduction operation.

Consider the following as one such assumption:

$$
\begin{equation*}
\left|l_{i}^{(k)}\right|,\left|u_{i}^{(k)}\right|<\left|d_{i}^{(k)} / 2\right|, \quad 1 \leq i \leq n \tag{2.19}
\end{equation*}
$$

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:

$$
\begin{equation*}
\left(\cdots, l_{i}^{(k)} / d_{i}{ }^{(k)}, 1, u_{i}{ }^{(k)} / d_{i}{ }^{(k)}, \cdots\right) \tag{2.20}
\end{equation*}
$$

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 $\boldsymbol{e}=\max _{i}\left(l_{i}{ }^{(k)} / d_{i}{ }^{(k)}, u_{i}{ }^{(k)} / d_{i}{ }^{(k)}\right)$ and consider the matrix $(\cdots, \boldsymbol{e}, 1, \boldsymbol{e}, \cdots)$ obtained by replacing all $l_{i}(k)$ and $u_{i}^{i}(k)$ 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:

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

where, $\boldsymbol{d}^{(k)}$ is the diagonal element computed during the $k$-th iteration. Let's try to measure whether $\left|\boldsymbol{d}^{(k)}\right|$ increases towards $1 / \mathrm{ep}$ 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:

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

and it follows that:

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

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):

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

is assumed to be the reduction frequency up to truncation. Moreover, truncation will not occur if $k \geq\left\lfloor\log _{2}(n)\right\rfloor$.

## (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 $A=L U$, 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 $\left(L^{T}\right)^{-1}$ 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 $A=L U$, then

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

where, $L=\left(l_{i j}\right)$ and $U=\left(u_{i j}\right)$.

### 2.1.3.10 Improving the solution

Consider improving the solution of the simultaneous linear equations $\boldsymbol{A} \boldsymbol{x}=\boldsymbol{b}$. 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:

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

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

$$
\begin{aligned}
& \boldsymbol{x}^{(k+1)} \rightarrow \boldsymbol{x} \\
& \boldsymbol{r}^{(k+1)} \rightarrow 0
\end{aligned} \quad(k \rightarrow \infty)
$$

Moreover, if

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

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

### 2.1.3.11 Precise estimate of the approximate solution

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

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

The relative error of the solution $\frac{\left\|\boldsymbol{x}-\boldsymbol{x}^{(k)}\right\|_{\infty}}{\left\|\boldsymbol{x}^{(k)}\right\|_{\infty}}$ can be replaced by $\frac{\left\|\boldsymbol{y}^{(k)}\right\|_{\infty}}{\left\|\boldsymbol{x}^{(k)}\right\|_{\infty}}$ 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:

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

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:

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

where:

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

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

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

where:

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

Therefore, if the condition number is on the order of $10^{\alpha}$, the precision of the derived solution may be approximately $\alpha$ 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):

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

(2) Calculating the condition number Although the condition number

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

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

$$
\Sigma=\left[\begin{array}{ccccc}
\sigma_{1} & & & & 0 \\
& \ddots & & & \\
& & \ddots & & \\
& & & \ddots & \\
0 & & & & \sigma_{n}
\end{array}\right]
$$

$\sigma_{i}$ : Singular value
$\sigma_{1} \geq \sigma_{2} \geq \cdots \geq \sigma_{n}$
Consider the system of equations $A \boldsymbol{x}=\boldsymbol{y}$. If $\boldsymbol{y}$ is represented as:

$$
\boldsymbol{y}=\|\boldsymbol{y}\| \sum_{i=1}^{n} \alpha_{i} u_{i} \quad\left(\sum_{i} \alpha_{i}^{2}=1\right)
$$

where $u_{i}$ (a column vector of $U$ ) is a basis, then, the following relationship holds:

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

As long as $\alpha_{n}$ is not particularly small, the size of the right side is on the order of $\sigma_{n}{ }^{-1}\left(=\left\|A^{-1}\right\|\right)$ 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 $\boldsymbol{y}=\boldsymbol{u}_{n}\left(\alpha_{n}=1, \alpha_{i}=0 ; i=1,2, \cdots, n-1\right)$. Therefore, $\boldsymbol{y}$ should be determined so that it has $\boldsymbol{u}_{n}$ as its principle elements. Actually, for:

$$
\boldsymbol{z}=\left(\begin{array}{c} 
\pm 1 \\
\pm 1 \\
\vdots \\
\pm 1
\end{array}\right)
$$

$\boldsymbol{y}$ should be obtained in $A^{T} \boldsymbol{y}=\boldsymbol{z}$ by determining the sign of each element of $\boldsymbol{z}$ so that $\frac{\|\boldsymbol{y}\|}{\|\boldsymbol{z}\|}$ is maximized.
Using this $\boldsymbol{y}$ to solve $A \boldsymbol{x}=\boldsymbol{y}, \frac{\|\boldsymbol{x}\|}{\|\boldsymbol{y}\|}$ is the approximate value of $\left\|A^{-1}\right\|$.
The actual procedure for obtaining the condition number is as follows.
(a) Obtain $\|A\|$.
(b) Perform a triangular decomposition of $A$ into $A=L U$.
(c) Obtain $\boldsymbol{w}$ by determining $\boldsymbol{z}$ in $U^{T} \boldsymbol{w}=\boldsymbol{z}$ so that $\frac{\|\boldsymbol{w}\|}{\|\boldsymbol{z}\|}$ is maximized.
(d) Obtain $\boldsymbol{y}$ by solving $L^{T} \boldsymbol{y}=\boldsymbol{w}$.
(e) Obtain $\boldsymbol{x}$ by solving $L U \boldsymbol{x}=\boldsymbol{y}$.
(f) Obtain $\frac{\|\boldsymbol{y}\|}{\|\boldsymbol{x}\|\|A\|}$ (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", PrenticeHall, 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 <br> 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 $A \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=$ $1,2, \cdots, m$ ) having real matrix $A$ (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix $B$ is defined by $B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$, the function obtains $\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\boldsymbol{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B$.
(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 R :Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | ab | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | See <br> Contents | Input | Matrix (real matrix, two-dimensional array type) consisting of coefficient matrix $A$ and right-hand side vectors $\boldsymbol{b}_{\boldsymbol{i}}\left[A, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$ Size: $(\operatorname{lna} \times(\mathrm{n}+\mathrm{m}))$ |
|  |  |  |  | Output | Matrix (real matrix, two-dimensional array type) consisting of the factored matrix $A^{\prime}$ of coefficient matrix $A$ and solution vectors $\boldsymbol{x}_{\boldsymbol{i}}$ $\left[A^{\prime}, \boldsymbol{x}_{\boldsymbol{1}}, \boldsymbol{x}_{\boldsymbol{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]$ (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) $0<\mathrm{n} \leq \ln \mathrm{a}$
(b) $0<\mathrm{m}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :---: | :---: |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 . | $\begin{aligned} & \mathrm{ab}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \\ & \leftarrow \mathrm{ab}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] / \mathrm{ab}[0] \\ & (i=1,2, \cdots, \mathrm{~m}) \text { is performed. } \end{aligned}$ |
| 2100 | There existed the diagonal element which was close to zero in the $L U$ 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 $A$. <br> $A$ 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, 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 $U$

Matrix $L$

$$
\left.\left[\begin{array}{ccccc}
1.0 & 0.0 & 0.0 & \cdots & 0.0 \\
l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\
l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0
\end{array}\right] \underset{\Downarrow}{\mid} \begin{array}{cccccc}
u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\
0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\
0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & u_{5,5}
\end{array}\right]
$$

Storage status of array $\mathrm{ab}[\operatorname{lna} \times \mathrm{k}]$

## Remarks

a. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{n}+\mathrm{m} \leq \mathrm{k}$ must be hold.

## (7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
2 & 4 & -1 & 6 \\
-1 & -5 & 4 & 2 \\
1 & 2 & 3 & 1 \\
3 & 5 & -1 & -3
\end{array}\right]\left[\begin{array}{ll}
x_{1,1} & x_{1,2} \\
x_{2,1} & x_{2,2} \\
x_{3,1} & x_{3,2} \\
x_{4,1} & x_{4,2}
\end{array}\right]=\left[\begin{array}{rr}
36 & 11 \\
15 & 0 \\
22 & 7 \\
-6 & 4
\end{array}\right]
$$

(b) Input data

Array ab in which coefficient matrix $A$ and constant vectors $\boldsymbol{b}_{\mathbf{1}}$ and $\boldsymbol{b}_{\mathbf{2}}$ are stored, lna=11, $\mathrm{n}=4$ and $\mathrm{m}=2$.
(c) Main program
/* C interface example for ASL_dbgmsm */
\#include <stdio.h>
\#include <stdlib.h>
\#include <asl.h>
int main()

$\mathrm{fp}=$ fopen( "dbgmsm.dat", "r" );
if ( $\mathrm{fp}==$ NULL $)$
\{
printf( "file open error\n" );
\}
printf( " $\quad * * *$ ASL_dbgmsm $* * * \backslash \mathrm{n} ")$;
printf( "\n $\quad * *$ Input $* * \backslash \mathrm{n} \backslash \mathrm{n} ")$;

```
    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" );
        }
        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 ***
```

*** ASL_dbgmsm ***
** Input **
** Input **
n = 4 m = 2
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
\begin{tabular}{ll}
1 & 1 \\
2 & 1 \\
4 & 1 \\
5 & 1
\end{tabular}

\subsection*{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 ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Coefficient matrix \(A\) (real matrix, twodimensional array type) \\
\hline & & & & Output & Upper triangular matrix \(U\) and lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\). (See Notes (b) and (c)) \\
\hline 2 & lna & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{n} & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Solution vector \(\boldsymbol{x}\) \\
\hline 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)) \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step of the LU decomposition of coef- \\
ficient matrix \(A\). \\
\(A\) is nearly singular.
\end{tabular} \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{c}\text { ASL_dbgmsm } \\ \text { ASL_rbgmsm }\end{array}\right\}\) to perform the calculations. However, when 2.2.1 \(\left\{\begin{array}{c}\text { ASL_dbgmsm } \\ \text { ASL_rbgmsm }\end{array}\right\}\) cannot be used such as when all of the righthand 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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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.
(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.

Matrix \(L\)
Matrix \(U\)
\[
\left[\begin{array}{ccccc}
1.0 & 0.0 & 0.0 & \cdots & 0.0 \\
l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\
l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0
\end{array}\right] \underset{\Downarrow}{\left[\begin{array}{ccccc}
u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\
0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\
0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & u_{5,5}
\end{array}\right]}
\]

\section*{Remarks}
a. \(\quad \ln \mathrm{a} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) must hold.

Figure 2-2 Storage Status of Matrices \(L\) and \(U\)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations and obtain the condition number.
\[
\left[\begin{array}{rrrr}
2 & 4 & -1 & 6 \\
-1 & -5 & 4 & 2 \\
1 & 2 & 3 & 1 \\
3 & 5 & -1 & -3
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
36 \\
15 \\
22 \\
-6
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(\mathrm{a}, \ln \mathrm{a}=11, \mathrm{n}=4\), 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 *ipvt
int i,j;
int i,j;;
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%%.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
Constant Vector
36
** Output **
ierr = 0
Solution

```
\(\begin{array}{lll}\mathrm{x}[ & 0]= & 1 \\ \mathrm{x}[ & 1]= & 2 \\ \mathrm{x}[ & 2]= & 4 \\ \mathrm{x}[ & 3]= & 5\end{array}\)

\subsection*{2.2.3 ASL_dbgmlu, ASL_rbgmlu}

\section*{LU Decomposition of a Real Matrix}

\section*{(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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real Matrix \(A\) (two-dimensional array type) \\
\hline & & & & Output & Unit upper triangular matrix \(U\) and lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step. \\
\(A\) is nearly singular.
\end{tabular} \\
\hline
\end{tabular}
(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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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.

\subsection*{2.2.4 ASL_dbgmlc, ASL_rbgmlc}

\section*{LU Decomposition and Condition Number of a Real Matrix}

\section*{(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
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real Matrix \(A\) (two-dimensional array type) \\
\hline & & & & Output & Unit upper triangular matrix \(U\) and lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline 2 & lna & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & \begin{tabular}{l} 
Contents of array a are not changed and \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step. \\
\(A\) is nearly singular.
\end{tabular} \\
\hline
\end{tabular}
(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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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) Although the condition number is defined by \(\|A\| \cdots\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.2.5 ASL_dbgmls, ASL_rbgmls}

\section*{Simultaneous Linear Equations (LU-Decomposed Real Matrix)}

\section*{(1) Function}

ASL_dbgmls or ASL_rbgmls solves the simultaneous linear equations \(L U \boldsymbol{x}=\boldsymbol{b}\) 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 ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt}) ;\)
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after LU decomposition (real matrix, two-dimensional array type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & \{ \(\mathrm{D} *\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & R* \(\}\) & & Output & Solution vector \(\boldsymbol{x}\) \\
\hline 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)) \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & \(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_dbgmms } \\ \text { ASL_rbgmms }\end{array}\right\}\) 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 \(A\).

\subsection*{2.2.6 ASL_dbgmms, ASL_rbgmms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides (LUDecomposed Real Matrix)}
(1) Function

ASL_dbgmms or ASL_rbgmms solves the simultaneous linear equations \(L U \boldsymbol{x}=\boldsymbol{b}\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{1}, \boldsymbol{x}_{2}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
ierr \(=\) ASL_dbgmms (a, lna, n, b, lnb, m, ipvt);
Single precision:
ierr \(=\) ASL_rbgmms ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for } 64 \text { bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after LU decomposition (real matrix, two-dimensional array type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Matrix consisting of constant vector \(\boldsymbol{b}_{\boldsymbol{i}}\) \(\left[A^{\prime}, \boldsymbol{b}_{1}, \boldsymbol{b}_{2}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\) \\
\hline & & & & Output & Matrix consisting of Solution vector \(\boldsymbol{x}_{\boldsymbol{i}}\) \(\left[A^{\prime}, x_{1}, x_{2}, \cdots, x_{m}\right]\) \\
\hline 5 & \(\operatorname{lnb}\) & I & 1 & Input & Adjustable dimension of array b \\
\hline 6 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 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)). \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a
(b) \(0<\mathrm{m}\)
(c) \(0<\operatorname{ipvt}[\mathrm{i}-1] \leq \mathrm{n}(\mathrm{i}=1, \ldots, \mathrm{n})\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|c|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & \begin{tabular}{c}
\(\mathrm{b}[\ln \mathrm{a} *(\mathrm{i}-1)] \leftarrow \mathrm{b}[\ln \mathrm{n} *(\mathrm{i}-1)] / \mathrm{a}[0]\) \\
\((i=1,2, \cdots, \mathrm{~m})\) is performed.
\end{tabular} \\
\hline 3000 & Restriction (a) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 3010 & Restriction (b) was not satisfied. & \multirow{3}{*}{} \\
\hline 3020 & Restriction (c) was not satisfied. & \\
\hline
\end{tabular}

\section*{(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 \(A\).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\(\left[\begin{array}{rrrr}2 & 4 & -1 & 6 \\ -1 & -5 & 4 & 2 \\ 1 & 2 & 3 & 1 \\ 3 & 5 & -1 & -3\end{array}\right]\left[\begin{array}{ll}x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2}\end{array}\right]=\left[\begin{array}{rr}36 & 11 \\ 15 & 0 \\ 22 & 7 \\ -6 & 4\end{array}\right]\)
(b) Input data

Coefficient matrix a , \(\ln \mathrm{a}=10, \mathrm{n}=4\), matrix consisting of constant vector \(B, \operatorname{lnb}=\mathrm{B}\) and \(\mathrm{m}=2\).
(c) Main program
```

/* C interface example for ASL_dbgmms */
\#include <stdio.h>
\#include <stdlib.h>
\#include <asl.h>

```
```

int main()

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

    int n=4;
    ```
```

double *b;
int lnb=11
int m=2;
int *ipvt;
int ierr_lu,ierr_ms;
int i,j;;
fp = fopen( "dbgmms.dat", "r" );
if( fp == NULL )
printf( "file open error\n" );
}
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" );
}
b = ( double * )malloc((size_t)( sizeof(double) * (lnb*m) ));
if( b == NULL )
printf( "no enough memory for array b\n" );
}
ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
if( ipvt == NULL )
{
printf( "no enough memory for array ipvt\n" );
}
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 )
ree( ipvt )
return 0;
}
(d) Output results

```
```

*** ASL_dbgmms ***

```
*** ASL_dbgmms ***
** Input **
** Input **
n= 4
n= 4
Coefficient Matrix a
Coefficient Matrix a
    2 
    2 
Constant Vectors b
Constant Vectors b
    36
    36
** Output **
** Output **
ierr ( ASL_dbgmlu ) = 0
ierr ( ASL_dbgmlu ) = 0
ierr ( ASL_dbgmms ) = 0
ierr ( ASL_dbgmms ) = 0
Solution
Solution
\begin{tabular}{ll}
1 & 1 \\
2 & 1 \\
4 & 1 \\
5 & 1
\end{tabular}
```


### 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 <br> R:Single precision real | Z:Double precision complex <br> C:Single precision complex |
| :--- | :--- |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\ln a \times n$ | Input | Real matrix $A$ (two-dimensional array type) after LU decomposition (See Notes (a) and (b)) |
|  |  |  |  | Output | Inverse matrix of matrix $A$ |
| 2 | $\operatorname{lna}$ | 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 $[\mathrm{i}-1]$ : Number of row exchanged with row i in the i-th processing step. (See Note (c)) |
| 5 | det | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 2 | Output | Determinant of matrix $A$ (See Note (d)) |
| 6 | isw | I | 1 | Input | Processing switch <br> isw $>0$ : Obtain determinant. <br> isw $=0:$ Obtain determinant and inverse matrix. <br> isw $<0$ : Obtain inverse matrix. |
| 7 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 8 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$

## (5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 | $\operatorname{det}[0] \leftarrow \mathrm{a}[0]$ (See Note (d)) |
|  |  | $\operatorname{det}[1] \leftarrow 0.0$ |
|  |  | $\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]$. |
| 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 $\left\{\begin{array}{c}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}, 2.2 .4\left\{\begin{array}{c}\text { ASL_dbgmlc } \\ \text { ASL_rbgmlc }\end{array}\right\}, 2.2 .2\left\{\begin{array}{c}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}$ 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:

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

Scaling is performed at this time so that:

$$
1.0 \leq|\operatorname{det}[0]|<10.0
$$

(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 $A X=B$ 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 <br> 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 <br> R:Single precision real | Z:Double precision complex <br> C:Single precision complex |
| :--- | :--- |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lna\times n}$ | Input | Coefficient matrix $A$ (real matrix, twodimensional array type) |
| 2 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of arrays a and alu |
| 3 | n | I | 1 | Input | Order of matrix $A$ |
| 4 | alu | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lna} \times \mathrm{n}$ | Input | Coefficient matrix $A$ after LU decomposition (See Note (a)) |
| 5 | b | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
| 6 | x | $\{\mathrm{D} *\}$ | n | Input | Approximate solution $\boldsymbol{x}$ |
|  |  | R* $\}$ |  | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 11 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(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 <br> maximum number of iterations. | Processing is aborted after calculating the <br> itol output value. |
| 6000 | The solution could not be improved. |  |

## (6) Notes

(a) This function improves the solution obtained by the 2.2.2 $\left\{\begin{array}{l}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}$ or 2.2.5 $\left\{\begin{array}{l}\text { ASL_dbgmls } \\ \text { ASL_rbgmls }\end{array}\right\}$ function. Therefore, the coefficient matrix $A$ after it has been decomposed 2.2.2 $\left\{\begin{array}{c}\text { ASL_dbgmsl } \\ \text { ASL_rbgmsl }\end{array}\right\}, 2.2 .3$ $\left\{\begin{array}{l}\text { ASL_dbgmlu } \\ \text { ASL_rbgmlu }\end{array}\right\}$ or 2.2.4 $\left\{\begin{array}{c}\text { ASL_dbgmlc } \\ \text { ASL_rbgmlc }\end{array}\right\}$ 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.
$\left[\begin{array}{rrrrrrrrrr}10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 8 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{array}\right]\left[\begin{array}{l}x_{1} \\ x_{2} \\ x_{3} \\ x_{4} \\ x_{5} \\ x_{6} \\ x_{7} \\ x_{8} \\ x_{9} \\ x_{10}\end{array}\right]=\left[\begin{array}{l}6 \\ 5 \\ 4 \\ 4 \\ 4 \\ 3 \\ 2 \\ 2 \\ 2 \\ 1\end{array}\right]$
(b) Input data

Coefficient matrix $A$, 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" );
    }
    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
        printf( "no enough memory for array b\n" );
        return -1;
    }
    sb}=(\mathrm{ double * )malloc((size_t)( sizeof(double) * n ));
    if( sb == NULL )
        printf( "no enough memory for array sb\n" );
```

\}
return -1;
wk $=\left(\right.$ double $\left.{ }^{*}\right)$ 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
if
printf( "no enough memory for array ipvt $\backslash n$ " );
return -1;
\}
printf( "\t $n=\% 6 d \backslash n ", n)$;
printf( "\n\tCoefficient Matrix\n\n");
for ( i=0 ; i<n ; i++ )
printf( "\t" );
for ( $\mathrm{j}=0$; $\mathrm{j}<\mathrm{n}$; $\mathrm{j}++$ )
\{
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] );
$\mathrm{sb}[\mathrm{i}]=\mathrm{b}[\mathrm{i}] ;$
printf $($ " $\backslash \mathrm{t} \% \mathrm{~B} .3 \mathrm{~g} \backslash \mathrm{n"}, \mathrm{~b}[\mathrm{i}])$;
\}
fclose( fp );
ierr = ASL_dbgmsl(a, lna, $n$, b, ipvt);
printf( "\n\tOriginal Solution\n\n" );
for ( i=0 ; i<n ; i++ )
printf( "\t $x[\% 6 d]=\% 8.3 g \backslash n ", i, b[i])$;
\}
ierr $=$ ASL_dbgmlx (sa, lna, $n, ~ a, ~ s b, ~ b, ~ \& i t o l, ~ n n i t, ~ i p v t, ~ w k) ; ~$
printf( "\n ** Output $* * \backslash \mathrm{n} \backslash \mathrm{n} ")$;
printf( "\tierr = \%6d\n\n", ierr );
printf( "\tImproved Solution\n\n");
for ( $i=0$; $i<n$; i++ )
printf( "\t $x[\% 6 d]=\% 8.3 g \backslash n ", i, b[i])$;
\}
free ( a );
free ( b );
free ( ba);
free (sa);
free ( sb $) ;$
free ( sb );
free( wk );
free( ipvt )
return 0 ;
\}
(d) Output results

```
*** ASL_dbgmlx ***
** Input **
n = 10
```

Coefficient Matrix

| 10 | 9 | 8 | 7 | 6 | 5 |
| ---: | ---: | ---: | ---: | :--- | :--- |
| 9 | 9 | 8 | 7 | 6 | 5 |
| 8 | 8 | 8 | 7 | 6 | 5 |
| 7 | 7 | 7 | 7 | 6 | 5 |
| 6 | 6 | 6 | 6 | 6 | 5 |
| 5 | 5 | 5 | 5 | 5 | 5 |


| 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 Vec
6
5
4
4
4
4
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 |  | $7.99 \mathrm{e}-16$ |
| x [ | 6 | = | -1 |
| x [ | 73 |  | -7.4e-17 |
| x [ | 8 | = | 1 |
| x [ | 9] | = | 0 |

** Output **
ierr $=0$
Improved Solution

### 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 $A \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=$ $1,2, \cdots, m$ ) having complex matrix $A$ (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix $B$ is defined by $B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$, the function obtains $\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B$.
(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 R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ Output | Contents |
| 1 | abr | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | See Contents | Input | Real part of matrix (complex matrix, twodimensional array type) consisting of coefficient matrix $A$ and right-hand side vectors $\boldsymbol{b}_{\boldsymbol{i}}$ $\left[A, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$ <br> Size: $(\operatorname{lna} \times(\mathrm{n}+\mathrm{m}))$ |
|  |  |  |  | Output | Real part of matrix (complex matrix, twodimensional array type) consisting of the factored matrix $A^{\prime}$ of coefficient matrix $A$ and solution vectors $\boldsymbol{x}_{\boldsymbol{i}}\left[A^{\prime}, \boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]$ (See Notes (a) and (b)). |
| 2 | abi | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | See <br> Contents | Input | Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix $A$ and right-hand side vectors $\boldsymbol{b}_{\boldsymbol{i}}\left[A, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$ <br> Size: $(\operatorname{lna} \times(\mathrm{n}+\mathrm{m}))$ |
|  |  |  |  | Output | Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix $A^{\prime}$ of coefficient matrix $A$ and solution vectors $\boldsymbol{x}_{\boldsymbol{i}}\left[A^{\prime}, \boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]$ (See Notes (a) and (b)) |
| 3 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of arrays abr and abi |


| No. | Argument and <br> Return Value | Type | Size | Input/ <br> 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 | $\mathrm{I}^{*}$ | n | Output | Pivoting information <br> ipvt[i -1$]:$ Number of row exchanged with <br> row i in the i-th processing step (See Note <br> (a)). |
| 7 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 8 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(b) $0<\mathrm{m}$

## (5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :---: | :---: |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 | $\begin{aligned} & \text { abr }[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \\ & \leftarrow(\operatorname{abr}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \times \operatorname{abr}[0] \\ & \quad+\operatorname{abi}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \times \operatorname{abi}[0]) \\ & /\left(\operatorname{abr}[0]^{2}+\operatorname{abi}[0]^{2}\right) \\ & \\ & \mathrm{abi}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \\ & \leftarrow(\operatorname{abi}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \times \operatorname{abr}[0] \\ & \quad-\operatorname{abr}[\ln a *(\mathrm{n}+\mathrm{i}-1)] \times \operatorname{abi}[0]) \\ & /\left(\operatorname{abr}[0]^{2}+\operatorname{abi}[0]^{2}\right) \\ & (\mathrm{i}=1,2, \cdots, \mathrm{~m}) \end{aligned}$ |
| 2100 | There existed the diagonal element which was close to zero in the $L U$ 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 $A$. <br> $A$ 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, 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.
(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 $U$

$$
\left[\begin{array}{ccccc}
1.0 & 0.0 & 0.0 & \cdots & 0.0 \\
l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\
l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0
\end{array}\right] \quad\left[\begin{array}{ccccc}
u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\
0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\
0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & u_{5,5}
\end{array}\right]
$$

$\Downarrow$
Storage status within array abr $[\ln a \times k]$



## Remarks

a. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{n}+\mathrm{m} \leq \mathrm{k}$ must be hold.
(7) Example
(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
4+2 i & 3+9 i & 4+i & 7+9 i \\
6+7 i & 4 i & 4+7 i & 2+5 i \\
9+3 i & 6+2 i & 9+5 i & 8+5 i \\
1+5 i & 7+9 i & 3+5 i & 2+4 i
\end{array}\right]\left[\begin{array}{llll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$

(b) Input data

Array abr and abi in which coefficient matrix $A$, constant vectors $\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \boldsymbol{b}_{\mathbf{3}}$ and $\boldsymbol{b}_{\mathbf{4}}$ are stored, lna=11,

$$
\mathrm{n}=4 \text { and } \mathrm{m}=4 .
$$

(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;
    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" );
    }
    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


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

| ,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 , | $0.0686)$ | . 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 <br> R:Single precision real | Z:Double precision complex <br> C:Single precision complex |
| :--- | :--- |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | ar | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\ln a \times n$ | 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 $A=L U$ (See Notes (b) and (c)) |
| 2 | ai | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\ln a \times n$ | 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 $A=L U$ (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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Input | Real part of constant vector $\boldsymbol{b}$ |
|  |  |  |  | Output | Real part of solution $\boldsymbol{x}$ |
| 6 | bi | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 9 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :---: | :---: |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 | $\begin{aligned} & \text { br }[0] \\ & \leftarrow\{\operatorname{br}[0] \times \operatorname{ar}[0]+\operatorname{bi}[0] \times \operatorname{ai}[0]\} \\ & \quad /\left\{\operatorname{ar}[0]^{2}+\operatorname{ai}[0]^{2}\right\} \\ & \operatorname{bi}[0] \\ & \leftarrow\{\operatorname{bi}[0] \times \operatorname{ar}[0]-\operatorname{br}[0] \times \operatorname{ai}[0]\} \\ & \quad /\left\{\operatorname{ar}[0]^{2}+\operatorname{ai}[0]^{2}\right\} \\ & \hline \end{aligned}$ |
| 2100 | There existed the diagonal element which was close to zero in the $L U$ 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 $A$. <br> $A$ 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 righthand 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 $A=(\mathrm{ar}$, ai). If the pivot row in the $\mathrm{i}-\mathrm{th}$ step is row $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, 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 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.

Matrix $L$
$\left[\begin{array}{ccccc}1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0\end{array}\right]\left[\begin{array}{ccccc}u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\ 0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\ 0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & u_{5,5}\end{array}\right]$

Storage status within array $\operatorname{ar}[\ln \mathrm{a} \times \mathrm{k}]$



## Remarks

a. $\quad \operatorname{lna} \geq \mathrm{n}, \mathrm{n} \leq \mathrm{k}$ must hold.

Figure 2-4 Storage Status of Matrices $L$ and $U$

## (7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{cccc}
5+8 i & 7+i & 6+3 i & 1+2 i \\
1+i & 9+5 i & 4+i & 5 \\
4 i & 3+3 i & 4+2 i & 6+9 i \\
7+8 i & 6 & 7+6 i & 10+4 i
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
3+20 i \\
-6+7 i \\
-6 i \\
13 i
\end{array}\right]
$$

(b) Input data

Coefficient matrix real part ar and imaginary part ai, $\ln \mathrm{a}=11, \mathrm{n}=4$ 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 ierr;
    int i,j;;
    fp = fopen( "zbgmsl.dat", "r" );
    if( fp == NULL )
    {f
        printf( "file open error\n" );
    }
    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" );
    }
    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" );
    }
    w = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w == NULL )
        printf( "no enough memory for array w\n" );
    }
    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( W );
    return 0;
}
(d) Output results
```

```
*** ASL_zbgmsl ***
```

*** ASL_zbgmsl ***
** Input **
** Input **
n = 4
n = 4
Coefficient Matrix (Real, Imaginary)

```

```

Constant Vector (Real, Imaginary)
$\begin{array}{lrr}( & 3, & 20) \\ ( & -6, & 7 \\ ( & 0, & -6) \\ ( & 0, & 13)\end{array}$

```
** Output **
ierr \(=0\)
Solution (Real, Imaginary)


\subsection*{2.3.3 ASL_zbgmlu, ASL_cbgmlu}

\section*{LU Decomposition of a Complex Matrix}

\section*{(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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of coefficient matrix \(A\) (two-dimensional array type) \\
\hline & & & & Output & Real parts of unit upper triangular matrix \(U\) and low triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of coefficient matrix (two-dimensional array type) \\
\hline & & & & Output & Imaginary parts of unit upper triangular matrix \(U\) and lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 6 & w1 & \(\underline{\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array ar and ai are not \\
changed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}

\section*{(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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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.

\subsection*{2.3.4 ASL_zbgmlc, ASL_cbgmlc}

\section*{LU Decomposition and Condition Number of a Complex Matrix}

\section*{(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
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Real part of coefficient matrix \(A\) (two-dimensional array type) \\
\hline & & & & Output & Real parts of unit upper triangular matrix \(U\) and low triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Imaginary part of coefficient matrix (two-dimensional array type) \\
\hline & & & & Output & Imaginary parts of unit upper triangular matrix \(U\) and lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Notes (a) and (b)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 6 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & Contents \\
\hline 7 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(2 \times \mathrm{n}\) & Work & Work area \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array ar and ai are not \\
changed and cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} \\
\hline 2100 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step. \\
\(A\) is nearly singular.
\end{tabular} & \multicolumn{2}{|c|}{\begin{tabular}{l} 
Processing is aborted.
\end{tabular}} \\
\hline \(4000+i\) &
\end{tabular}
(6) Notes
(a) The unit lower triangular matrix \(L\) is stored in the lower triangular portion of array ar and aith 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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), then j is stored in \(\operatorname{ipvt}[\mathrm{i}-1]\). 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) Although the condition number is defined by \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function

\subsection*{2.3.5 ASL_zbgmls, ASL_cbgmls}

\section*{Simultaneous Linear Equations (LU-Decomposed Complex Matrix)}

\section*{(1) Function}

ASL_zbgmls or ASL_cbgmls solves the simultaneous linear equations \(L U \boldsymbol{x}=\boldsymbol{b}\) 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
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Imaginary parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & br & \{ \(\mathrm{D} *\}\) & n & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline & & R \(*\) \} & & Output & Real part of solution \(\boldsymbol{x}\) \\
\hline 6 & bi & \{ \(\mathrm{D} *\}\) & n & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline & & R * \(\}\) & & Output & Imaginary part of solution \(\boldsymbol{x}\) \\
\hline 7 & ipvt & I* & n & Input & Pivoting information \(\operatorname{ipvt}[i-1]\) : Number of row exchanged with row i in the \(i\)-th processing step. (See Note (c)) \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \(\mathrm{br}[0] \leftarrow\) \\
& & \(\{\mathrm{br}[0] \times \operatorname{ar}[0]+\mathrm{bi}[0] \times \mathrm{ai}[0]\}\) \\
& & \(/\left\{\operatorname{ar}[0]^{2}+\mathrm{ai}[0]^{2}\right\}\) \\
& & \(\mathrm{bi}[1] \leftarrow\) \\
& & \(\{\mathrm{bi}[0] \times \operatorname{ar}[0]-\mathrm{br}[0] \times \mathrm{ai}[0]\}\) \\
& & \(/\left\{\operatorname{ar}[0]^{2}+\mathrm{ai}[0]^{2}\right\}\) \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.3.4 \(\left\{\begin{array}{l}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}\). In addition, if you have already used 2.3.2 \(\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) 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 2.3.6 \(\left\{\begin{array}{l}\text { ASL_zbgmms } \\ \text { ASL_cbgmms }\end{array}\right\}\) 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 \(\left\{\begin{array}{l}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}, 2.3 .4\left\{\begin{array}{l}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}, 2.3 .2\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) functions which perform LU decomposition of matrix \(A\).

\subsection*{2.3.6 ASL_zbgmms, ASL_cbgmms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides (LUDecomposed Complex Matrix)}
(1) Function

ASL_zbgmms or ASL_cbgmms uses Gauss' method to solve the simultaneous linear equations \(A \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=\) \(1,2, \cdots, m\) ) having complex matrix \(A\) (two-dimensional array type) as coefficient matrix. That is, when the \(n \times m\) matrix \(B\) is defined by \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt); }
\]

Single precision:
ierr \(=\) ASL_cbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Real parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Imaginary parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline 3 & lna & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & br & \{ \(\mathrm{D} *\}\) & \(\operatorname{lnb} \times \mathrm{m}\) & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline & & R** & & Output & Real part of solution \(\boldsymbol{x}\) \\
\hline 6 & bi & \{ \(\mathrm{D} *\}\) & \(\operatorname{lnb} \times \mathrm{m}\) & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline & & R** & & Output & Imaginary part of solution \(\boldsymbol{x}\) \\
\hline 7 & \(\operatorname{lnb}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 8 & m & I & 1 & Input & Order of matrix \(B\) \\
\hline 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)) \\
\hline 10 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \operatorname{lna}, \operatorname{lnb}\)
(b) \(\mathrm{m}>0\)
(c) \(0<\operatorname{ipvt}[\mathrm{i}-1] \leq \mathrm{n}(\mathrm{i}=1, \ldots, \mathrm{n})\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \[
\begin{aligned}
& \mathrm{br}[\operatorname{lnb} \times i] \leftarrow \\
& \{\mathrm{br}[\operatorname{lnb} \times i] \times \operatorname{ar}[0]+\mathrm{bi}[\operatorname{lnb} \times i] \times \mathrm{ai}[0]\} \\
& \quad /\left\{\operatorname{ar}[0]^{2}+\mathrm{ai}[0]^{2}\right\} \\
& \mathrm{bi}[\operatorname{lnb} \times i] \leftarrow \\
& \{\mathrm{bi}[\operatorname{lnb} \times i] \times \operatorname{ar}[0]-\mathrm{br}[\ln \mathrm{~b} \times i] \times \mathrm{ai}[0]\} \\
& \quad /\left\{\operatorname{ar}[0]^{2}+\mathrm{ai}[0]^{2}\right\} \\
& (i=0,1, \cdots, \mathrm{~m}-1) \text { is performed } .
\end{aligned}
\] \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline 3020 & Restriction (c) was not satisfied. & \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.3.4 \(\left\{\begin{array}{l}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}\). In addition, if you have already used 2.3.2 \(\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) 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 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 \(\left\{\begin{array}{c}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}\), 2.3.4 \(\left\{\begin{array}{c}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}\), 2.3.2 \(\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) functions which perform LU decomposition of matrix \(A\).

\section*{(7) Example}
(a) ProblemSolve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
4+2 i & 3+9 i & 4+i & 7+9 i \\
6+7 i & 4 i & 4+7 i & 2+5 i \\
9+3 i & 6+2 i & 9+5 i & 8+5 i \\
1+5 i & 7+9 i & 3+5 i & 2+4 i
\end{array}\right]\left[\begin{array}{llll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
\]
(b) Input data

Array abr and abi in which coefficient matrix \(A\), constant vectors \(\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \boldsymbol{b}_{\mathbf{3}}\) and \(\boldsymbol{b}_{\mathbf{4}}\) are stored, lna \(=11\), \(\ln \mathrm{b}=11, \mathrm{n}=4\) and \(\mathrm{m}=4\).
(c) Main program
```

/* C interface example for ASL_zbgmms */
\#include <stdio.h>
\#include <stdlib.h>
\#include <asl.h>
int main()
double *ar
double *ai
nt lna=11
int lnb=11;
int n;
nt m;
double *br;
double *bi;
int *ipvt;
double *W
int ierr,kerr
int i,j;
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) \((\operatorname{sizeof}(\) double) \(*(\operatorname{lna} * \mathrm{n})))\);
    if ( ar == NULL \()\)
    printf( "no enough memory for array ar\n" );
        return -1;
    \}
    ai \(=(\) double \(*)\) malloc ( (size_t) ( sizeof (double) * (lna*n) ));
    if \((a i==\) NULL \()\)
        printf( "no enough memory for array ai\n" );
    \}
    br = ( double * ) malloc ((size_t) ( sizeof (double) * (lnb*m) ));
    if ( br == NULL \({ }^{*}\) )
        printf( "no enough memory for array br \(\backslash \mathrm{n}\) " );
    \}
    bi \(=(\) double \(*)\) malloc \(((\) size_t) \((\operatorname{sizeof}(\) double) \(*(\operatorname{lnb} * m))\);
    if ( bi == NULL )
        printf( "no enough memory for array bi\n" );
        return -1 ;
    \}
    \(\mathrm{W}=(\) double \(*)\) malloc \(((\) size_t) ( sizeof(double) * (2*n) ));
    \(\operatorname{if}_{\{ }(\mathrm{w}==\mathrm{NULL})\)
        printf( "no enough memory for array w w " ) ;
        return -1;
    \}
    ipvt \(=(\) int \(*\) )malloc ((size_t) ( sizeof(int) \(* \mathrm{n})\) );
    if (ipvt == NULL )
    \{
        printf( "no enough memory for array ipvt\n" );
        return -1 ;
    \(\}\)
    printf( "\tn \(=\% 6 d \backslash t m=\% 6 d \backslash n ", n, m\) );
    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( ar );
free(br );
free( bi );
free( w );
free( ipvt );
return 0;
}

```
(d) Output results
```

*** ASL_zbgmms ***
** Input **

```
\(\mathrm{n}=\quad 4 \mathrm{~m}=\quad 4\)
Coefficient Matrix (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline 4, & 2) & 3 , & 9) & 4, & 1) & 7, & 9) \\
\hline 6, & 7) & 0, & 4) & 4, & 7) & 2, & 5) \\
\hline 9, & 3) & 6 , & 2) & 9, & 5) & 8, & 5) \\
\hline 1, & 5) & 7, & 9) & 3, & 5) & 2 , & 4) \\
\hline
\end{tabular}

Constant Vectors (Real,Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline 1 , & 0) & 0 , & 0) & 0 , & 0) & 0 & 0) \\
\hline 0 , & 0) & 1, & 0) & 0 , & 0) & 0 & 0) \\
\hline 0 , & 0) & 0 , & 0) & 1, & 0) & 0 & 0) \\
\hline 0 , & 0) & 0 , & \(0)\) & 0 , & 0) & 1, & 0) \\
\hline
\end{tabular}
** Output **
zbgmlu ierr \(=\quad 0\)
zbgmms ierr =
0
0
Solution (Real,Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline 0.0133 , & -0.073) & 0.181 , & -0.247) & -0.184, & \(0.178)\) & -0.104, & -0.056) \\
\hline -0.0178, & -0.0189) & -0.068, & -0.0696) & -0.0128, & \(0.1)\) & 0.0415 , & -0.0657) \\
\hline -0.0353, & \(0.138)\) & ( -0.0585, & \(0.17)\) & ( 0.133, & -0.241) & 0.131 , & \(0.0191)\) \\
\hline 0.0494 , & -0.0686) & (-0.00961, & \(0.13)\) & 0.0885 , & -0.0709) & ( -0.0462, & \(0.0662)\) \\
\hline
\end{tabular}

\subsection*{2.3.7 ASL_zbgmdi, ASL_cbgmdi}

Determinant and Inverse Matrix of a Complex Matrix

\section*{(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
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array}\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Real parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Real parts of inverse matrix of matrix \(A\) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary parts of coefficient matrix \(A\) after LU decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Imaginary parts inverse matrix of matrix \(A\) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 6 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 3 & Output & Determinant of matrix \(A\) (See Note (d)) \\
\hline 7 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
isw \(>0\) : Obtain determinant. \\
isw \(=0\) : Obtain determinant and inverse matrix. \\
isw \(<0\) : Obtain inverse matrix.
\end{tabular} \\
\hline 8 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times n\) & Work & Work area \\
\hline 9 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{det}[0] \leftarrow \operatorname{ar}[0]\) \\
& & \(\operatorname{det}[1] \leftarrow \operatorname{ai}[0]\) \\
& & \(\operatorname{det}[2] \leftarrow 0.0\) \\
& & \(\operatorname{ar}[0] \leftarrow \operatorname{ar}[0] /\left\{\operatorname{ar}[0]^{2}+\operatorname{ai}[0]^{2}\right\}\) \\
& & \(\operatorname{ai}[0] \leftarrow-\operatorname{ai}[0] /\left\{\operatorname{ar}[0]^{2}+\operatorname{ai}[0]^{2}\right\}\) \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be LU decomposed before using this function. Use any of the 2.3.3 \(\left\{\begin{array}{c}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}, 2.3 .4\left\{\begin{array}{c}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}, 2.3 .2\left\{\begin{array}{c}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) 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: \(\Re\{\operatorname{det}(A)\}=\operatorname{det}[0] \times 10^{\operatorname{det}[2]}\)
\(\Im\{\operatorname{det}(A)\}=\operatorname{det}[1] \times 10^{\operatorname{det}[2]}\)
Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|+|\operatorname{det}[1]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.3.8 ASL_zbgmlx, ASL_cbgmlx \\ Improving the Solution of Simultaneous Linear Equations (Complex Matrix)}

\section*{(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
\(\begin{array}{ll}\begin{array}{l}\text { D:Double precision real } \\ \text { R:Single precision real }\end{array} & \begin{array}{l}\text { Z:Double precision complex } \\ \text { C:Single precision complex }\end{array}\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Real parts of coefficient matrix \(A\) (two-dimensional array type) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Imaginary parts of coefficient matrix \(A\) (two-dimensional array type) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array ar, ai, alr, and ali \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & alr & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Real parts of coefficient matrix \(A\) after LU decomposition (See Note (a)) \\
\hline 6 & ali & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Imaginary parts of coefficient matrix \(A\) after LU decomposition (See Note (a)) \\
\hline 7 & br & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline 8 & bi & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline \multirow[t]{2}{*}{9} & \multirow[t]{2}{*}{xr} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Real part of approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Real part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline \multirow[t]{2}{*}{10} & \multirow[t]{2}{*}{xi} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Imaginary part of approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Imaginary part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline
\end{tabular}
\begin{tabular}{|l|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 11 & itol & \(\mathrm{I}^{*}\) & 1 & Input & \begin{tabular}{l} 
Number of digits to which solution is to be \\
improved (See Note (b))
\end{tabular} \\
\cline { 3 - 6 } & & Output & \begin{tabular}{l} 
Approximate number of digits to which solu- \\
tion was improved (See Note (c))
\end{tabular} \\
\hline 12 & nit & I & 1 & Input & \begin{tabular}{l} 
Maximum number of iterations (See Note \\
(d))
\end{tabular} \\
\hline 13 & ipvt & \(\mathrm{I}^{*}\) & n & Input & Pivoting information (See Note (a)) \\
\hline 14 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(3 \times \mathrm{n}\) & Work & Work area \\
\hline 15 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculation the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.3.2 \(\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\) or 2.3.5 \(\left\{\begin{array}{l}\text { ASL_zbgmls } \\ \text { ASL_cbgmls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after being decomposed by 2.3.2 \(\left\{\begin{array}{l}\text { ASL_zbgmsl } \\ \text { ASL_cbgmsl }\end{array}\right\}\), 2.3.3 \(\left\{\begin{array}{l}\text { ASL_zbgmlu } \\ \text { ASL_cbgmlu }\end{array}\right\}, 2.3 .4\left\{\begin{array}{l}\text { ASL_zbgmlc } \\ \text { ASL_cbgmlc }\end{array}\right\}\) 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.

\subsection*{2.4 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (COMPLEX ARGUMENT TYPE)}

\subsection*{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 \(A \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=\) \(1,2, \cdots, m\) ) having complex matrix \(A\) (two-dimensional array type) as coefficient matrix. That is, when the \(n \times m\) matrix \(B\) is defined by \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(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
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 b i t \text { Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ab & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \begin{tabular}{l}
See \\
Contents
\end{tabular} & Input & \begin{tabular}{l}
Matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix \(A\) and right-hand side vectors \(\boldsymbol{b}_{\boldsymbol{i}}\) \(\left[A, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\) \\
Size: \((\operatorname{lna} \times(\mathrm{n}+\mathrm{m}))\)
\end{tabular} \\
\hline & & & & Output & Matrix (complex matrix, two-dimensional array type) consisting of the factored matrix \(A^{\prime}\) of coefficient matrix \(A\) and solution vectors \(\boldsymbol{x}_{\boldsymbol{i}}\left[A^{\prime}, \boldsymbol{x}_{\boldsymbol{1}}, \boldsymbol{x}_{\boldsymbol{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]\) (See Notes (a) and (b)) \\
\hline 2 & lna & I & 1 & Input & Adjustable dimension of array ab \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 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)). \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(b) \(0<\mathrm{m}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \[
\begin{aligned}
& \hline \mathrm{ab}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] \\
& \leftarrow \mathrm{ab}[\operatorname{lna} *(\mathrm{n}+\mathrm{i}-1)] / \mathrm{ab}[0] \\
& (i=1,2, \cdots, \mathrm{~m}) \text { is performed. }
\end{aligned}
\] \\
\hline 2100 & There existed the diagonal element which was close to zero in the \(L U\) decomposition of the coefficient matrix \(A\). The result may not be obtained with a good accuracy. & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline \(4000+i\) & The pivot became 0.0 in the \(i\)-th processing step of the LU decomposition of coefficient matrix \(A\). \(A\) is nearly singular. & \\
\hline
\end{tabular}
(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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
4+2 i & 3+9 i & 4+i & 7+9 i \\
6+7 i & 4 i & 4+7 i & 2+5 i \\
9+3 i & 6+2 i & 9+5 i & 8+5 i \\
1+5 i & 7+9 i & 3+5 i & 2+4 i
\end{array}\right]\left[\begin{array}{llll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
\]
(b) Input data

Array \(a b\) in which coefficient matrix \(A\), constant vectors \(\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \boldsymbol{b}_{\mathbf{3}}\) and \(\boldsymbol{b}_{\mathbf{4}}\) are stored, lna=11, \(\mathrm{n}=4\) 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

```
double _Complex *ab
int lna=11, lma=5;
int lna=11, lma=5;
int n;
int n;
int m;
int m;
int *ipvt;
int *ipvt;
int ierr;
int ierr;
int i,j;
int i,j;
fp = fopen( "zbgnsm.dat", "r" );
fp = fopen( "zbgnsm.dat", "r" );
if( fp == NULL )
if( fp == NULL )
    printf( "file open error\n" );
    printf( "file open error\n" );
        return -1;
        return -1;
}
printf( " *** ASL_zbgnsm ***\n" );
printf( " *** ASL_zbgnsm ***\n" );
printf( "\n ** Input **\n\n" );
printf( "\n ** Input **\n\n" );
fscanf( fp, "%d", &n );
fscanf( fp, "%d", &n );
fscanf( fp, "%d", &m );
fscanf( fp, "%d", &m );
printf( "\t n = %6d m = %6d\n", n, m );
printf( "\t n = %6d m = %6d\n", n, m );
ab = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*(lna+lma)) ));
ab = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*(lna+lma)) ));
if( ab == NULL )
if( ab == NULL )
    printf( "no enough memory for array ab\n" );
    printf( "no enough memory for array ab\n" );
}
ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
ipvt = ( int * )malloc((size_t)( sizeof(int) * n ));
if( ipvt == NULL )
if( ipvt == NULL )
{
{
    printf( "no enough memory for array ipvt\n" );
    printf( "no enough memory for array ipvt\n" );
    return -1;
    return -1;
}
printf( "\n\tCoefficient Matrix\n\n");
printf( "\n\tCoefficient Matrix\n\n");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    printf( "\t" );
    printf( "\t" );
    for( j=0 ; j<n ; j++ )
    for( j=0 ; j<n ; j++ )
    {
    {
            double tmp_re, tmp_im;
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            fscanf( fp, "%lf", &tmp_im );
            ab[i+lna*j] = tmp_re + tmp_im * _Complex_I;
            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( "(%8.3g,%8.3g)", creal(ab[i+lna*j]), cimag(ab[i+lna*j]) );
        }
        }
    printf( "\n" );
    printf( "\n" );
}
}
printf( "\n\tConstant Vectors\n\n");
printf( "\n\tConstant Vectors\n\n");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    printf( "\t" );
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    for( j=0 ; j<m ; j++ )
    {
    {
            double tmp_re, tmp_im;
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            fscanf( fp, "%lf", &tmp_im );
            ab[i+lna*(n+j)] = tmp_re + tmp_im * _Complex_I;
            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( "(%8.3g,%8.3g)",creal(ab[i+lna*(n+j)]),cimag(ab[i+lna*(n+j)]) );
    }
    }
    printf( "\n" );
    printf( "\n" );
}
}
fclose( fp );
fclose( fp );
ierr = ASL_zbgnsm(ab, lna, n, m, ipvt);
ierr = ASL_zbgnsm(ab, lna, n, m, ipvt);
printf( "\n ** Output **\n\n" );
printf( "\n ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tSolution\n\n" );
printf( "\n\tSolution\n\n" );
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
{ printf( "\t" );
{ printf( "\t" );
    for( j=0 ; j<m ; j++ )
    for( j=0 ; j<m ; j++ )
    {
    {
        printf( "(%8.3g,%8.3g)", creal(ab[i+lna*(n+j)]),cimag(ab[i+lna*(n+j)]) );
        printf( "(%8.3g,%8.3g)", creal(ab[i+lna*(n+j)]),cimag(ab[i+lna*(n+j)]) );
    }
    }
    printf( "\n" );
    printf( "\n" );
}
free( ab );
free( ab );
free( ipvt');
```

free( ipvt');

```
\}
(d) Output results
```

*** ASL_zbgnsm ***
** Input **
n = 4m= 4

```

Coefficient Matrix


Constant Vectors
\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline 1, & 0)( & 0 , & 0) & 0, & 0) & 0, & 0) \\
\hline 0, & 0) \((\) & 1 , & 0) & 0, & 0) & 0 , & 0) \\
\hline 0 , & 0) & 0, & 0) & 1 , & 0) & 0 , & 0) \\
\hline 0 , & \(0)(\) & 0 , & 0) & 0 , & \(0)\) & 1 , & \(0)\) \\
\hline
\end{tabular}
** Output **
ierr \(=0\)
Solution
\((0.0133,-0.073)(0.181,-0.247)(-0.184,0.178)(-0.104,-0.056)\)
\((-0.0178,-0.0189)(-0.068,-0.0696)(-0.0128,10.1)(0.0415,-0.0657)\)
\(\begin{array}{rrrr}(-0.0353, & 0.138)(-0.0585, & 0.17)(0.133, & -0.241)(0.131, \\ 0.0494, & -0.0686)(-0.00961, & 0.13)(0.0191) \\ 0.0885, & -0.0709)(-0.0462, & 0.0662)\end{array}\)

\subsection*{2.4.2 ASL_zbgnsl, ASL_cbgnsl}

Simultaneous Linear Equations (Complex Matrix)

\section*{(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 ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\
\mathrm{C} *\end{array}\right\}\) & & lna \(\times \mathrm{n}\) & Input \\
\cline { 4 - 6 } & & & \begin{tabular}{l} 
Output \\
(complex matrix, two-dimensional array \\
type)
\end{tabular} \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step of the LU decomposition of coef- \\
ficient matrix \(A\). \\
\(A\) is nearly singular.
\end{tabular} \\
\hline
\end{tabular}

\section*{(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 righthand 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 \(\mathrm{i}-\mathrm{th}\) step is row \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), 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 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).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
5+8 i & 7+i & 6+3 i & 1+2 i \\
1+i & 9+5 i & 4+i & 5 \\
4 i & 3+3 i & 4+2 i & 6+9 i \\
7+8 i & 6 & 7+6 i & 10+4 i
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
3+20 i \\
-6+7 i \\
-6 i \\
13 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\), lna \(=11, \mathrm{n}=4\) 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;
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" );
}
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;
}
f for
or( 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(
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]) );
}
}
for( i=0 ; i<n ; i++ )
double tmp_re;
fscanf( fp, "%lf", \&tmp_re );
b[i] = tmp_re;

```
```

    {
    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 );
ree(
free( ipvt );
return 0;
}
(d) Output results

```
```

*** ASL_zbgnsl ***

```
*** ASL_zbgnsl ***
** Input **
** Input **
\(\mathrm{n}=4\)
\(\mathrm{n}=4\)
Coefficient Matrix (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline 5 & 8) & 7 & 1) & 6 & 3) & 1 & & 2) \\
\hline 1 & 1) & 9 & 5) & 4 & 1) & 5 & & 0) \\
\hline 0 & 4) & 3 & 3) & 4 & 2) & 6 & & 9) \\
\hline 7 & 8) & 6 & 0) & 7 & 6) & 10 & & 4) \\
\hline
\end{tabular}
Constant Vector (Real, Imaginary)
\begin{tabular}{|c|c|c|c|}
\hline ( & 3 & , & 20) \\
\hline ( & -6 & , & 7) \\
\hline ( & 0 & , & -6) \\
\hline ( & 0 & & 13) \\
\hline
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)
\begin{tabular}{|c|c|c|c|}
\hline x [ & 0 & \(=\left(\begin{array}{ll}1\end{array}\right.\) & 1) \\
\hline X & 1 & \(=(-1.67 \mathrm{e}-16\) & 1) \\
\hline x & 2 & \(=(1\) & , \(-2.78 \mathrm{e}-16\) ) \\
\hline x [ & 3 & \(=\left(\begin{array}{ll}1\end{array}\right.\) & -1) \\
\hline
\end{tabular}
```


### 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 |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{Z} * \\ \mathrm{C} * \end{array}\right\}$ | $\operatorname{lna\times n}$ | Input | Complex matrix $A$ (two-dimensional array type) |
|  |  |  |  | Output | Upper triangular matrix $U$ and lower triangular matrix $L$ when $A$ is decomposed into $A=L U$. (See Notes (a) and (b)) |
| 2 | $\operatorname{lna}$ | 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) $0<\mathrm{n} \leq \ln \mathrm{a}$
(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 <br> was close to zero in the $L U$ decompo- <br> sition of the coefficient matrix $A$. The <br> result may not be obtained with a good <br> accuracy. | Processing continues. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| $4000+i$ | The pivot became 0.0 in the $i$-th process- <br> ing step. <br> $A$ 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, 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 ( $\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{ipvt}, \& c o n d, \mathrm{w} 1)$;
(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real | Z:Double precision complex <br> C:Single precision complex |
| :--- | :--- | I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{Z} * \\ \mathrm{C} * \end{array}\right\}$ | $\ln a \times n$ | Input | Complex matrix (two-dimensional array type) |
|  |  |  |  | Output | Upper triangular matrix $U$ and lower triangular matrix $L$ when $A$ is decomposed into $A=L U$ (See Notes (a) and (b)) |
| 2 | $\operatorname{lna}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 1 | Output | Reciprocal of the condition number |
| 6 | w1 | $\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}$ | n | Work | Work area |
| 7 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. | The contents of array a are unchanged. <br> cond $\leftarrow 1.0$ is performed. |
| 1000 | n was equal to 1. | Processing continues. <br> 2100 <br> There existed the diagonal element which <br> sas close to zero in the $L U$ decompo- <br> sition of the coefficient matrix $A . \quad$ The <br> result may not be obtained with a good <br> accuracy. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. <br> $4000+i$ <br> The pivot became 0.0 in the $i$-th process- <br> ing step. <br> $A$ is nearly singular. <br> Processing is aborted. The condition <br> 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, 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) Although the condition number is defined by $\|A\| \cdot\left\|A^{-1}\right\|$, 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 $L U \boldsymbol{x}=\boldsymbol{b}$ 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:

$$
\text { ierr }=\text { ASL_zbgnls }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \text { ipvt }) ;
$$

Single precision:
ierr $=$ ASL_cbgnls ( $\mathrm{a}, \ln \mathrm{l}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt}) ;$
(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |
| :--- |
| Z:Double precision complex <br> C:Single precision complex$\quad$ I: $\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}$ |
| No.Argument and <br> Return Value |
| 1 |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]$ 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 $\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.4.4 $\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}$. In addition, if you have already used 2.4.2 $\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}$ 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 2.4.6 $\left\{\begin{array}{l}\text { ASL_zbgnms } \\ \text { ASL_cbgnms }\end{array}\right\}$ 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 $\left\{\begin{array}{c}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}, 2.4 .4\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}, 2.4 .2\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}$ 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) Decomposed Complex Matrix)
(1) Function

ASL_zbgmsm or ASL_cbgnms uses Gauss' method to solve the simultaneous linear equations $A \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=$ $1,2, \cdots, m$ ) having complex matrix $A$ (two-dimensional array type) as coefficient matrix. That is, when the $n \times m$ matrix $B$ is defined by $B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]$, the function obtains $\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B$.
(2) Usage

Double precision:

$$
\text { ierr }=\text { ASL_zbgnms }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \ln b, \mathrm{~m}, \text { ipvt }) ;
$$

Single precision:
ierr $=$ ASL_cbgnms ( $\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m}, \mathrm{ipvt}) ;$
(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |
| :--- |
| Z:Double precision complex <br> C:Single precision complex |
| No. | | Argument and |
| :--- |
| Return Value |$\quad$ Type | Size |
| :--- |
| 1 |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}, \ln \mathrm{b}$
(b) $\mathrm{m}>0$
(c) $0<\operatorname{ipvt}[\mathrm{i}-1] \leq \mathrm{n}(\mathrm{i}=1, \ldots, \mathrm{n})$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 | $\mathrm{b}[\ln \mathrm{b} \times i] \leftarrow \mathrm{b}[\ln \mathrm{b} \times i] / \mathrm{a}[0](i=0,1, \cdots, \mathrm{~m}-1)$ <br> 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 $\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.4.4 $\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}$. In addition, if you have already used 2.4.2 $\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}$ 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 $L$ 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 $\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}$, 2.4.4 $\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}, 2.4 .2\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}$ functions which perform LU decomposition of matrix $A$.

## (7) Example

(a) ProblemSolve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
4+2 i & 3+9 i & 4+i & 7+9 i \\
6+7 i & 4 i & 4+7 i & 2+5 i \\
9+3 i & 6+2 i & 9+5 i & 8+5 i \\
1+5 i & 7+9 i & 3+5 i & 2+4 i
\end{array}\right]\left[\begin{array}{llll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$

(b) Input data

Coefficient matrix $A$, constant vectors $\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \boldsymbol{b}_{\mathbf{3}}$ and $\boldsymbol{b}_{\mathbf{4}}$ are stored, $\ln \mathrm{a}=11, \ln \mathrm{~b}=11, \mathrm{n}=4$ and $\mathrm{m}=4$.
(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;
    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" );
    }
    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 ***
```

*** ASL_zbgnms ***
** Input **
** Input **
n = 4 m = 4
n = 4 m = 4
Coefficient Matrix (Real,Imaginary)

| 4, | 2) | 3 , | 9) | 4 , | 1) | 7, | 9) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 6, | 7) | 0, | 4) | 4 , | 7) | 2, | ) |
| 9, | 3) | 6, | 2) | 9, | 5) | 8, | $)$ |
| 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 **
$\begin{array}{ll}\text { zbgnlu ierr }= & 0 \\ \text { zbgnms ierr } & =\end{array}$
Solution (Real,Imaginary)

```


\subsection*{2.4.7 ASL_zbgndi, ASL_cbgndi}

Determinant and Inverse Matrix of a Complex Matrix

\section*{(1) Function}

ASL_zbgndi or ASL_cbgndi obtains the determinant and inverse matrix of the complex matrix \(A\) (twodimensional 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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|l|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{\(\left.\begin{array}{l}\text { Zontents } \\
\mathrm{C} *\end{array}\right\}\)}
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{cdet} \leftarrow \mathrm{a}[0]\) \\
& & \(\operatorname{det} \leftarrow 0.0\) (See Note (d)) and \\
& & \(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]\) are performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}\), 2.4.4 \(\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}, 2.4 .2\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}\) 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 \(\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}\), 2.4.4 \(\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}, 2.4 .2\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}\) functions which perform LU decomposition of matrix \(A\).
(d) The determinant is given by the following expression:
\[
\operatorname{det}(A)=\operatorname{cdet} \times 10^{\operatorname{det}}
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\Re\{\operatorname{cdet}\}|+|\Im\{\operatorname{cdet}\}|<10.0
\]
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 \(A X=B\) 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.

\subsection*{2.4.8 ASL_zbgnlx, ASL_cbgnlx}

Improving the Solution of Simultaneous Linear Equations (Complex Matrix)

\section*{(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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{\(\left.\begin{array}{l}\text { Contents } \\
\mathrm{C}_{*}\end{array}\right\}\)}
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) This function improves the solution obtained by the 2.4.2 \(\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}\) or 2.4.5 \(\left\{\begin{array}{l}\text { ASL_zbgnls } \\ \text { ASL_cbgnls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after being decomposed by 2.4.2 \(\left\{\begin{array}{l}\text { ASL_zbgnsl } \\ \text { ASL_cbgnsl }\end{array}\right\}\), 2.4.3 \(\left\{\begin{array}{l}\text { ASL_zbgnlu } \\ \text { ASL_cbgnlu }\end{array}\right\}\), or 2.4.4 \(\left\{\begin{array}{l}\text { ASL_zbgnlc } \\ \text { ASL_cbgnlc }\end{array}\right\}\) 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.

\subsection*{2.5 POSITIVE SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)}

\subsection*{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
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Coefficient matrix \(A\) (positive symmetric matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{T}\) when \(A\) is decomposed into \(A=L L^{T}\) (See Note (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \begin{tabular}{l}
\(\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}\) and \\
\(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\)
\end{tabular} \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L L^{T}\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Processing continues. \\
\hline 3000
\end{tabular} \\
\hline \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} & \multirow{2}{*}{ Processing is aborted. } \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became less than or \\
equal to 0.0 in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_dbpdls } \\ \text { ASL_rbpdls }\end{array}\right\}\) required number of times varying only the contents of \(b\). This enables you to eliminate unnecessary calculations by performing the \(L L^{T}\) 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 \(L^{T}\)
\[
\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \\
& & \Downarrow & &
\end{array}\right]
\]

Storage status within array a[lna \(\times \mathrm{k}\) ]


\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) must hold.
b. Input time values of elements indicated by asterisks \((*)\) are not guaranteed.

Figure 2-5 Storage status of Matrix \(L^{T}\)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
5 & 7 & 6 & 5 \\
7 & 10 & 8 & 7 \\
6 & 8 & 10 & 9 \\
5 & 7 & 9 & 10
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{l}
23 \\
32 \\
33 \\
31
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\), 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;
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;
}}=(\mathrm{ double *
if( b == NULL )
printf( "no enough memory for array b\n" )
}
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
** Output **
ierr = 0
Solution

| $\mathrm{x}[$ | $0]=$ | 1 |
| :--- | :--- | :--- |
| $\mathrm{x}[$ | $1]=$ | 1 |
| $\times[$ | $2]=$ | 1 |
| $\times[$ | $3]=$ | 1 |

```

\subsection*{2.5.2 ASL_dbpduu, ASL_rbpduu}

\section*{\(L^{T}\) Decomposition of a Positive Symmetric Matrix}
(1) Function

ASL_dbpduu or ASL_rbpduu uses the Cholesky method to perform an \(L L^{T}\) 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 ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}\) );
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & & \(\ln \times \mathrm{n}\) & Input \\
\cline { 3 - 6 } & & & \begin{tabular}{l} 
Positive symmetric matrix \(A\) \\
(two-dimensional array type) (upper triangu- \\
lar type)
\end{tabular} \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline \begin{tabular}{l} 
Upper triangular matrix \(L^{T}\) when \(A\) is de- \\
composed into \(A=L L^{T}\) (See Note (a))
\end{tabular} \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L L^{T}\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became less than or \\
equal to 0.0 in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \multicolumn{2}{|c|}{} \\
\hline
\end{tabular}
(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)

\subsection*{2.5.3 ASL_dbpduc, ASL_rbpduc}

\section*{\(L^{T}\) Decomposition and Condition Number of a Positive Symmetric Matrix}
(1) Function

ASL_dbpduc or ASL_rbpduc uses the Cholesky method to perform an \(L L^{T}\) 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
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|l|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & & \(\ln \times \mathrm{n}\) & Input \\
\cline { 4 - 6 } & & \begin{tabular}{l} 
Output \\
(two-dimensional array type) (upper triangu- \\
lar type)
\end{tabular} \\
\hline \begin{tabular}{l} 
Upper triangular matrix \(L^{T}\) when \(A\) is de- \\
composed into \(A=L L^{T}\) (See Note (a))
\end{tabular} \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 5 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l}
\(\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}\) and \\
cond \(\leftarrow 1.0\) are performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L L^{T}\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became less than or \\
equal to 0.0 in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline \(4000+i\) & \\
\hline
\end{tabular}
(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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.5.4 ASL_dbpdls, ASL_rbpdls}

\section*{Simultaneous Linear Equations (LL \({ }^{\mathrm{T}}\)-Decomposed Positive Symmetric Matrix)}
(1) Function

ASL_dbpdls or ASL_rbpdls solves the simultaneous linear equations \(L L^{T} \boldsymbol{x}=\boldsymbol{b}\) having the positive symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) which has been \(L L^{T}\) decomposed by the Cholesky method as coefficient matrix.
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_dbpdls } \quad(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
\]

Single precision:
ierr \(=\) ASL_rbpdls (a, lna, n, b);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after \(L L^{T}\) decomposition (positive symmetric matrix, twodimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & \{ \(\mathrm{D} *\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & R* \(\}\) & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{|c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]^{2}\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be \(L L^{T}\) 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 \(L L^{T}\) 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).

\subsection*{2.5.5 ASL_dbpddi, ASL_rbpddi}

Determinant and Inverse Matrix of a Positive Symmetric Matrix

\section*{(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 \(L L^{T}\) decomposed by the Cholesky method.
(2) Usage

Double precision:
ierr = ASL_dbpddi (a, lna, n, det, isw);

Single precision: ierr \(=\) ASL_rbpddi ( \(\mathrm{a}, \ln \mathrm{n}, \mathrm{n}\), det, isw);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Positive symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) after \(L L^{T}\) decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 2 & Output & Determinant of matrix \(A\) (See Note (c)) \\
\hline 5 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
isw \(>0\) : Obtain determinant. \\
isw \(=0:\) Obtain determinant and inverse matrix. \\
isw \(<0\) : Obtain inverse matrix.
\end{tabular} \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \(\operatorname{det}[0] \leftarrow \mathrm{a}[0]^{2}\) \\
& & \(\operatorname{det}[1] \leftarrow 1.0\) \\
& & \(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]^{2}\) \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be \(L L^{T}\) decomposed before using this function. Use any of the 2.5.2 \(\left\{\begin{array}{c}\text { ASL_dbpduu } \\ \text { ASL_rbpduu }\end{array}\right\}, 2.5 .3\left\{\begin{array}{c}\text { ASL_dbpduc } \\ \text { ASL_rbpduc }\end{array}\right\}, 2.5 .1\left\{\begin{array}{c}\text { ASL_dbpdsl } \\ \text { ASL_rbpdsl }\end{array}\right\}\) 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:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times 10^{\operatorname{det}[1]}
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.5.6 ASL_dbpdlx, ASL_rbpdlx Improving the Solution of Simultaneous Linear Equations (Positive Symmetric Matrix)}

\section*{(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:
\[
\text { ierr }=\text { ASL_dbpdlx (a, lna, n, all, b, x, \&itol, nit, w1); }
\]

Single precision:
ierr \(=\) ASL_rbpdlx (a, lna, n, all, b, x, \&itol, nit, w1);
(3) Arguments and Return Value
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} Z:Double precision complex
C:Single precision complex \(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating itol \\
output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) This function improves the solution obtained by the 2.5.1 \(\left\{\begin{array}{c}\text { ASL_dbpdsl } \\ \text { ASL_rbpdsl }\end{array}\right\}\) or 2.5.4 \(\left\{\begin{array}{c}\text { ASL_dbpdls } \\ \text { ASL_rbpdls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after it has been decomposed 2.5.1 \(\left\{\begin{array}{l}\text { ASL_dbpdsl } \\ \text { ASL_rbpdsl }\end{array}\right\}, 2.5 .2\) \(\left\{\begin{array}{c}\text { ASL_dbpduu } \\ \text { ASL_rbpduu }\end{array}\right\}\), or 2.5.3 \(\left\{\begin{array}{l}\text { ASL_dbpduc } \\ \text { ASL_rbpduc }\end{array}\right\}\) 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.
\[
\mathrm{itol} \leq 0
\]
or
\[
\text { itol } \geq-\log _{10}(2 \times \varepsilon)(\varepsilon: \text { Unit for determining error })
\]
(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.

\subsection*{2.6 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)}

\subsection*{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:
\[
\text { ierr }=\text { ASL_dbspsl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \text { ipvt, wk }) ;
\]

Single precision:
ierr \(=\) ASL_rbspsl ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}\), ipvt, wk);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & & \(\ln \times \mathrm{n}\) & Input \\
\cline { 4 - 6 } & & & \begin{tabular}{l} 
Output \\
Coefficient matrix \(A\) (real symmetric matrix, \\
two-dimensional array type, upper triangular \\
type)
\end{tabular} \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step of the \(L D L^{T}\) \\
decomposition of coefficient matrix \(A\). \\
\(A\) is nearly singular.
\end{tabular} & \multicolumn{2}{|c|}{} \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_dbspls } \\ \text { ASL_rbspls }\end{array}\right\}\) 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.

Matrix \(L^{T}\)
Matrix \(D\)
\[
\begin{aligned}
& {\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right]\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]} \\
& \Downarrow
\end{aligned}
\]


\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(\operatorname{LDL}^{\mathrm{T}}\) 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 \() \mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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.

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
1 \\
-1 \\
4 \\
-4
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A, \operatorname{lna}=11, \mathrm{n}=4\) 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()
\[
\begin{aligned}
& \text { double *a; } \\
& \text { int na; } \\
& \text { int n; } \\
& \text { double *b; } \\
& \text { int *ipvt; } \\
& \text { double *wk; } \\
& \text { int ierr; }
\end{aligned}
\]
```

int i,j;
FILE *fp;
fp = fopen( "dbspsl.dat", "r" );
if( fp == NULL )
printf( "file open error\n" );
prinur-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" );
}
b}=(\mathrm{ double *)malloc((size_t)( sizeof(double) * n ));
if(b 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( "<br>\tConstant Vector\n\n" );
for( i=0 ; i<n ; i++ )
fscanf( fp, "%lf", \&b[i] );
printf( "\t%%.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( ipvt );
free( wk );
return 0;

```
(d) Output results
*** ASL_dbspsl ***
** Input **
\(\mathrm{n}=4\)
Coefficient Matrix
\begin{tabular}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{tabular}

Constant Vector
\[
\begin{array}{r}
1 \\
-1 \\
4 \\
-4
\end{array}
\]
** Output **
ierr \(=0\)
Solution
\(\begin{array}{llr}\mathrm{x}[ & 0]= & 1 \\ \times[ & 1]= & -1 \\ \times[ & 2]= & 2 \\ \mathrm{x}[ & 3]= & -2\end{array}\)

\subsection*{2.6.2 ASL_dbspud, ASL_rbspud}

\section*{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:
\[
\text { ierr }=\text { ASL_rbspud (a, lna, n, ipvt, wk); }
\]
(3) Arguments and Return Value
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Real symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{T}\) when \(A\) is decomposed into \(A=L D L^{T}\) (See Note (a)) \\
\hline 2 & \(\operatorname{lna}\) & 1 & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & 1 & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & wk & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LU decomposi- \\
tion of the coefficient matrix \(A\). The re- \\
sult may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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 \(\mathrm{LDL}^{\mathrm{T}}\) 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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in \(\mathrm{ipvt}[\mathrm{i}-1]\). 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.

\subsection*{2.6.3 ASL_dbspuc, ASL_rbspuc}

\section*{LDL \(^{\mathrm{T}}\) 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 \(^{\mathrm{T}}\) 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_dbspuc (a, lna, n, ipvt, \&cond, wk);
Single precision:
ierr \(=\) ASL_rbspuc ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{ipvt}, \& c o n d, \mathrm{wk})\);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{T}\) when \(A\) is decomposed into \(A=L D L^{T}\) (See Note (a)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 6 & wk & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array a are not changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LU decomposi- \\
tion of the coefficient matrix \(A\). The re- \\
sult may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline \(4000+i\) & & \\
\hline
\end{tabular}
(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 \(\mathrm{LDL}^{\mathrm{T}}\) 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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.6.4 ASL_dbspls, ASL_rbspls}

Simultaneous Linear Equations (LDL \({ }^{\mathrm{T}}\)-Decomposed Real Symmetric Matrix)
(1) Function

ASL_dbspls or ASL_rbspls solves the simultaneous linear equations having the real symmetric matrix \(A\) (twodimensional array type) which has been LDL \(^{\mathrm{T}}\) decomposed by the modified Cholesky method as coefficient matrix.
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_dbspls }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \text { ipvt }) ;
\]

Single precision: ierr \(=\) ASL_rbspls ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b}\), ipvt);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Coefficient matrix \(A\) after \(\mathrm{LDL}^{\mathrm{T}}\) decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension af array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & \{ \(\mathrm{D} *\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & R** & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ipvt & I* & n & Output & \begin{tabular}{l}
Pivoting information \\
ipvt \([i-1]\) : Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
\end{tabular} \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be \(\mathrm{LDL}^{\mathrm{T}}\) 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 \(\left\{\begin{array}{l}\text { ASL_dbspuc } \\ \text { ASL_rbspuc }\end{array}\right\}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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 2.6.5 \(\left\{\begin{array}{c}\text { ASL_dbspms } \\ \text { ASL_rbspms }\end{array}\right\}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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.

\subsection*{2.6.5 ASL_dbspms, ASL_rbspms}

Simultaneous Linear Equations with Multiple Right-Hand Sides ( LDL \({ }^{\mathrm{T}}\) decomposed Real Matrix )

\section*{(1) Function}

ASL_dbspms or ASL_rbspms solves the simultaneous linear equations \(L D L^{T} \boldsymbol{x}=\boldsymbol{b}\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\boldsymbol{1}}, \boldsymbol{x}_{2}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
ierr \(=\) ASL_dbspms (a, lna, n, b, lnb, m, ipvt);
Single precision:
ierr \(=\) ASL_rbspms ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\text { D* } \\ R *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Coefficient matrix \(A\) after LDL \(^{\mathrm{T}}\) decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & lna & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Matrix consisting of constant vector \(\boldsymbol{b}_{\boldsymbol{i}}\) \(\left[A^{\prime}, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\) \\
\hline & & & & Output & Matrix consisting of Solution vector \(\boldsymbol{x}_{\boldsymbol{i}}\) \(\left[A^{\prime}, \boldsymbol{x}_{\boldsymbol{1}}, \boldsymbol{x}_{2}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]\) \\
\hline 5 & lnb & I & 1 & Input & Adjustable dimension of array b \\
\hline 6 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 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)) \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(b) \(0<\mathrm{m}\)
(c) \(0<\operatorname{ipvt}[\mathrm{i}-1] \leq \mathrm{n}(\mathrm{i}=1, \ldots, \mathrm{n})\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & \begin{tabular}{l}
\(\mathrm{b}[\ln \mathrm{a} *(\mathrm{i}-1)] \leftarrow \mathrm{b}[\ln a *(\mathrm{i}-1)] / \mathrm{a}[0]\) \\
\((i=1,2, \cdots, \mathrm{~m})\) is performed.
\end{tabular} \\
\hline 3000 & & Restriction (a) was not satisfied. \\
\hline \multirow{2}{*}{ Processing is aborted. } \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be \(\operatorname{LDL}^{\mathrm{T}}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) decomposition of matrix \(A\).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\(\left[\begin{array}{llll}5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4\end{array}\right]\left[\begin{array}{ll}x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2}\end{array}\right]=\left[\begin{array}{rr}1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13\end{array}\right]\)
(b) Input data

Coefficient matrix \(\mathrm{a}, \operatorname{lna}=10, \mathrm{n}=4\), matrix consisting of constant vector \(B, \ln \mathrm{~b}=\mathrm{B}\) and \(\mathrm{m}=2\).
(c) Main program
/* C interface example for ASL_dbspms */
\#include <stdio.h>
\#include <stdlib.h>
\#include <asl.h>

\section*{int main()}
double *a;
int lna=11;
int \(\ln \mathrm{n}=4\);
int \(n=4 ;\)
double *b;
int \(\operatorname{lnb}=11 ;\)
int \(m=2\);
double *wk
int *ipvt;
int ierr_ud,ierr_ms;
```

intili,j;
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}=(\mathrm{ 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(
free( wk );
free( ipvt );
return 0;

```
(d) Output results
```

*** ASL_dbspms ***
** Input **
n= 4
Coefficient Matrix a

| 5 | 4 | 1 | 1 |
| :--- | :--- | :--- | :--- |
| 4 | 5 | 1 | 1 |
| 1 | 1 | 4 | 2 |
| 1 | 1 | 2 | 4 |

```
Constant Vectors b
    \(\begin{array}{rr}1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13\end{array}\)
** Output **
ierr ( ASL_dbspud ) = 0
ierr ( ASL_dbspms ) = 0
Solution
\(\begin{array}{rr}1 & -2 \\ -1 & 1 \\ 2 & 1 \\ -2 & 3\end{array}\)

\subsection*{2.6.6 ASL_dbspdi, ASL_rbspdi}

\section*{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 \(\mathrm{LDL}^{\mathrm{T}}\) 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
\(\left.\begin{array}{l}\begin{array}{l}\text { D:Double precision real } \\ \text { R:Single precision real }\end{array} \\ \begin{array}{|c|c|c|c|c|l|}\text { Z:Double precision complex } \\ \text { C:Single precision complex }\end{array} \quad \mathrm{I}:\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right.\end{array}\right\}\)

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{det}[0] \leftarrow \mathrm{a}[0], \operatorname{det}[1] \leftarrow 0.0\) \\
& & \begin{tabular}{l}
\(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]\) \\
\\
\\
\end{tabular} \\
\hline 3000 & Restriction performed. (See Note \((\mathrm{c})\) ) a\()\) \\
\hline
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be LDL \({ }^{\mathrm{T}}\) decomposed before using this function. Use any of the 2.6.2 \(\left\{\begin{array}{c}\text { ASL_dbspud } \\ \text { ASL_rbspud }\end{array}\right\}, 2.6 .3\left\{\begin{array}{c}\text { ASL_dbspuc } \\ \text { ASL_rbspuc }\end{array}\right\}, 2.6 .1\left\{\begin{array}{c}\text { ASL_dbspsl } \\ \text { ASL_rbspsl }\end{array}\right\}\) 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times 10^{\operatorname{det}[1]}
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.6.7 ASL_dbsplx, ASL_rbsplx}

\section*{Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix)}

\section*{(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.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_dbsplx (a, lna, n, ald, b, x, \&itol, nit, ipvt, wk);
Single precision:
ierr \(=\) ASL_rbsplx ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}\), ald, \(\mathrm{b}, \mathrm{x}\), \&itol, nit, ipvt, wk);
(3) Arguments and Return Value
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}\(\quad \mathrm{I}:\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculation the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.6.1 \(\left\{\begin{array}{l}\text { ASL_dbspsl } \\ \text { ASL_rbspsl }\end{array}\right\}\) or 2.6.4 \(\left\{\begin{array}{l}\text { ASL_dbspls } \\ \text { ASL_rbspls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after it has been decomposed by the 2.6.1 \(\left\{\begin{array}{l}\text { ASL_dbspsl } \\ \text { ASL_rbspsl }\end{array}\right\}\), 2.6.2 \(\left\{\begin{array}{c}\text { ASL_dbspud } \\ \text { ASL_rbspud }\end{array}\right\}\), or \(2.6 .3\left\{\begin{array}{c}\text { ASL_dbspuc } \\ \text { ASL_rbspuc }\end{array}\right\}\) 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.
\[
\mathrm{itol} \leq 0
\]
or
\[
\text { itol } \geq-\log _{10}(2 \times \varepsilon)(\varepsilon: \text { Unit for determining error })
\]
(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.

\subsection*{2.7 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE)(NO PIVOTING)}

\subsection*{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:
\[
\text { ierr }=\text { ASL_dbsmsl }(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{~b}, \mathrm{w} 1)
\]

Single precision:
\[
\text { ierr }=\text { ASL_rbsmsl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \mathrm{w} 1) ;
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Coefficient matrix \(A\) (real symmetric matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{T}\) when \(A\) is decomposed into \(A=L D L^{T}\) (See Note (b)) \\
\hline 2 & \(\ln a\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\(\frac{\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}}{\text { ( }}\)} & \multirow[t]{2}{*}{n} & Input & Constant vector b \\
\hline & & & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work Area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(\mathrm{LDL}^{\mathrm{T}}\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
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.
\end{tabular} & \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{c}\text { ASL_dbsmls } \\ \text { ASL_rbsmls }\end{array}\right\}\) 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.
\[
\begin{aligned}
& \begin{array}{c}
\text { Matrix } L^{T} \\
{\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right]\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]}
\end{array} \\
& \Downarrow
\end{aligned}
\]

Storage status within array a[lna \(\times \mathrm{k}\) ]


\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(D\)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
1 \\
-1 \\
4 \\
-4
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\), 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" );
}
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 ));
b=( double * )
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 );
return 0;

```
\}
(d) Output results
*** ASL_dbsmsl ***
** Input **
\(\mathrm{n}=4\)
Coefficient Matrix
\begin{tabular}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{tabular}

Constant Vector
1
-1
4
-4
** Output **
ierr \(=0\)
Solution
\(\begin{array}{llr}\mathrm{x}[ & 0] & = \\ \mathrm{x}[ & 1]= & 1 \\ \mathrm{x}[ & 2] & = \\ \mathrm{x}[ & 3] & = \\ & & -2\end{array}\)

\subsection*{2.7.2 ASL_dbsmud, ASL_rbsmud}

\section*{LDL \({ }^{\mathrm{T}}\) Decomposition of a Real Symmetric Matrix (No Pivoting)}
(1) Function

ASL_dbsmud or ASL_rbsmud 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:
\[
\text { ierr = ASL_dbsmud }(\mathrm{a}, \operatorname{lna} \mathrm{n}, \mathrm{w} 1) ;
\]

Single precision:
ierr \(=\) ASL_rbsmud (a, lna, n, w1);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|l|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & \begin{tabular}{l} 
Real symmetric matrix \(A\) (two-dimensional \\
array type) (upper triangular type)
\end{tabular} \\
\cline { 4 - 6 } & & Output & \begin{tabular}{l} 
Upper triangular matrix \(L^{T}\) when \(A\) is de- \\
\\
lomposed into \(A=L D L^{T}\) (See Note (a))
\end{tabular} \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln a\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL \\
T decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}

\section*{(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 \(2-7\).)

\subsection*{2.7.3 ASL_dbsmuc, ASL_rbsmuc}

\section*{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:
\[
\text { ierr }=\text { ASL_dbsmuc (a, lna, n, \&cond, w1); }
\]

Single precision:
\[
\text { ierr }=\text { ASL_rbsmuc } \quad(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \& \text { cond, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R :Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{T}\) when \(A\) is decomposed into \(A=L D L^{T}\) (See Note (a)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 5 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array a are not changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL \\
T \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
\hline 3000 \\
\hline \(4000+i\) \\
\end{tabular} \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} \\
\begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline
\end{tabular}
(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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.7.4 ASL_dbsmls, ASL_rbsmls}

\section*{Simultaneous Linear Equations (LDL \({ }^{\mathrm{T}}\)-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 \(^{\mathrm{T}}\) decomposed by the modified Cholesky method as coefficient matrix.
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_dbsmls }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
\]

Single precision:
ierr \(=\) ASL_rbsmls ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}\) );
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Coefficient matrix \(A\) after LDL \({ }^{\mathrm{T}}\) decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\ln a\) & I & 1 & Input & Adjustable dimension af array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & \{ \(\mathrm{D} *\}\) & n & Input & Constant vector b \\
\hline & & R* \(\}\) & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{|c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be \(\operatorname{LDL}^{\mathrm{T}}\) 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 \(\left\{\begin{array}{l}\text { ASL_dbsmuc } \\ \text { ASL_rbsmuc }\end{array}\right\}\) function. 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 \(\mathrm{LDL}^{\mathrm{T}}\) 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.)

\subsection*{2.7.5 ASL_dbsmms, ASL_rbsmms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides ( \(\mathrm{LDL}^{T}\) Decomposed Real Matrix ) ( No Pivoting )}
(1) Function

ASL_dbsmms or ASL_rbsmms solves the simultaneous linear equations \(L D L^{T} \boldsymbol{x}=\boldsymbol{b}\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
ierr \(=\) ASL_dbsmms ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b}, \ln \mathrm{b}, \mathrm{m})\);
Single precision:
ierr \(=\) ASL_rbsmms \((\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m})\);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Coefficient matrix \(A\) after LDL \(^{\mathrm{T}}\) decomposition (real symmetric matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Matrix consisting of constant vector \(\boldsymbol{b}_{\boldsymbol{i}}\) \(\left[A^{\prime}, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\) \\
\hline & & & & Output & Matrix consisting of Solution vector \(\boldsymbol{x}_{\boldsymbol{i}}\) \(\left[A^{\prime}, \boldsymbol{x}_{1}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]\) \\
\hline 5 & \(\operatorname{lnb}\) & I & 1 & Input & Adjustable dimension of array b \\
\hline 6 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(b) \(0<\mathrm{m}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n is equal to 1 & \begin{tabular}{l}
\(\mathrm{b}[\ln \mathrm{na} *(\mathrm{i}-1)] \leftarrow \mathrm{b}[\operatorname{lna} *(\mathrm{i}-1)] / \mathrm{a}[0]\) \\
\((i=1,2, \cdots, \mathrm{~m})\) is performed.
\end{tabular} \\
\hline 3000 & & Restriction (a) was not satisfied.
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be \(\mathrm{LDL}^{\mathrm{T}}\) 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 \(\left\{\begin{array}{l}\text { ASL_dbsmsl } \\ \text { ASL_rbsmsl }\end{array}\right\}\) to solve simultaneous linear equations having the same coefficient matrix \(A\), you can use the \(\mathrm{LDL}^{\mathrm{T}}\) 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.
\(\left[\begin{array}{llll}5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4\end{array}\right]\left[\begin{array}{ll}x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2}\end{array}\right]=\left[\begin{array}{rr}1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13\end{array}\right]\)
(b) Input data

Coefficient matrix a , \(\ln \mathrm{a}=10, \mathrm{n}=4\), matrix consisting of constant vector \(B, \operatorname{lnb}=\mathrm{B}\) and \(\mathrm{m}=2\).
(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;
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" );
}
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) ));
iff 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 );
}
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
Coefficient Matrix a

```
\begin{tabular}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{tabular} Constant Vectors b
\begin{tabular}{rr}
1 & -2 \\
-1 & 1 \\
4 & 9 \\
-4 & 13
\end{tabular}
** Output **
ierr ( ASL_dbsmud )= 0
ierr ( ASL_dbsmms )= 0
Solution
\begin{tabular}{rr}
1 & -2 \\
-1 & 1 \\
2 & 1 \\
-2 & 3
\end{tabular}

\subsection*{2.7.6 ASL_dbsmdi, ASL_rbsmdi}

Determinant and Inverse Matrix of a Real Symmetric Matrix (No Pivoting)

\section*{(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 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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & Z:Double precision complex \\
C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna} \times \mathrm{n}\)} & Input & Real symmetric matrix \(A\) (two-dimensional array type) (upper triangular type) after \(\mathrm{LDL}^{\mathrm{T}}\) decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimensional pf array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 2 & Output & Determinant of matrix \(A\) (See Note (c) ) \\
\hline 5 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
isw \(>0\) : Obtain determinant. \\
isw \(=0:\) Obtain determinant and inverse matrix. \\
isw \(<0\) : Obtain inverse matrix.
\end{tabular} \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{det}[0] \leftarrow \mathrm{a}[0], \operatorname{det}[1] \leftarrow 0.0\) \\
& & \(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]\) \\
& & are performed. (See Note (c)) \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times 10^{\operatorname{det}[1]}
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.7.7 ASL_dbsmlx, ASL_rbsmlx}

Improving the Solution of Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

\section*{(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.

\section*{(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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) (real symmetric matrix, two-dimensional array type, upper triangular type) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a and ald \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & ald & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after \(\mathrm{LDL}^{\mathrm{T}}\) decomposition (See Note (a)) \\
\hline 5 & b & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline 6 & x & \{ \(\mathrm{D} *\}\) & n & Input & Approximate solution \(\boldsymbol{x}\) \\
\hline & & (R* \(\}\) & & Output & Iteratively improved solution \(\boldsymbol{x}\) \\
\hline 7 & itol & I* & 1 & Input & Number of digits to which solution is to be improved (See Note (b)) \\
\hline & & & & Output & Approximate number of digits to which solution was improved (See Note (c)) \\
\hline 8 & nit & I & 1 & Input & Maximum number of iterations (See Note (d)) \\
\hline 9 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 10 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculation the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) This function improves the solution obtained by the 2.7.1 \(\left\{\begin{array}{c}\text { ASL_dbsmsl } \\ \text { ASL_rbsmsl }\end{array}\right\}\) 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 \(\left\{\begin{array}{c}\text { ASL_dbsmsl } \\ \text { ASL_rbsmsl }\end{array}\right\}\), 2.7.2 \(\left\{\begin{array}{c}\text { ASL_dbsmud } \\ \text { ASL_rbsmud }\end{array}\right\}\), or 2.7.3 \(\left\{\begin{array}{c}\text { ASL_dbsmuc } \\ \text { ASL_rbsmuc }\end{array}\right\}\) 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.

\title{
2.8 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE, LOWER TRIANGULAR TYPE)(NO PIVOTING)
}

\subsection*{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 ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b})\);
Single precision:
ierr \(=\) ASL_rbsnsl ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b})\);
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \\
\begin{tabular}{|c|c|c|c|c|l|}
\multicolumn{1}{l}{} \\
\begin{tabular}{|c|c|c|c|} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} & I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\end{tabular} \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the U \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) decom- \\
position of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step of the UT DU \\
decomposition of coefficient matrix \(A\). \\
\(A\) is nearly singular.
\end{tabular} & \multicolumn{2}{|c|}{} \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{c}\text { ASL_dbsnls } \\ \text { ASL_rbsnls }\end{array}\right\}\) you to eliminate unnecessary calculations by performing the \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) 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.
\[

\]
\(\Downarrow\)
Storage status within array a[lna \(\times \mathrm{k}\) ]


\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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.
\[
\left[\begin{array}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
1 \\
-1 \\
4 \\
-4
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\), 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" );
}
b = ( double * )malloc((size_t)( sizeof(double) * n ));
if( b == NULL )
printf( "no enough memory for array b\n" );
}
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 **
$\mathrm{n}=\quad 4$

```
Coefficient Matrix
\begin{tabular}{llll}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4
\end{tabular}

Constant Vector
1
-1
4
-4
** Output **
ierr \(=0\)
Solution
\begin{tabular}{lll}
\(\mathrm{x}[\) & \(0]\) & \(=\) \\
\(\mathrm{x}\left[\begin{array}{ll}1\end{array}\right]\) & \(=\) & 1 \\
\(\mathrm{x}[\) & \(2]\) & \(=\) \\
\(\mathrm{x}[\) & \(3]\) & \(=\) \\
&
\end{tabular}

\subsection*{2.8.2 ASL_dbsnud, ASL_rbsnud}
\(\mathrm{U}^{\mathrm{T}}\) DU Decomposition of a Real Symmetric Matrix (No Pivoting)
(1) Function

ASL_dbsnud or ASL_rbsnud uses the modified Cholesky method to perform an \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) decomposition of the real symmetric matrix \(A\) (two-dimensional array type) (lower triangular type).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_dbsnud }(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}) ;
\]

Single precision:
ierr \(=\) ASL_rbsnud ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n})\);
(3) Arguments and Return Value
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{ Contents } \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(\ln \times \mathrm{n}\) & Input & \begin{tabular}{l} 
Real symmetric matrix \(A\) (two-dimensional \\
array type) (lower triangular type)
\end{tabular} \\
\cline { 4 - 6 } & & Output & \begin{tabular}{l} 
Lower triangular matrix \(U^{T}\) when \(A\) is de- \\
composed into \(A=U^{T} D U\) (See Note (a))
\end{tabular} \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the UT DU decom- \\
position of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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.)

\subsection*{2.8.3 ASL_dbsnls, ASL_rbsnls}

Simultaneous Linear Equations ( \(\mathrm{U}^{\mathrm{T}}\) DU-Decomposed Real Symmetric Matrix) (No Pivoting)

\section*{(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 \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) decomposed by the modified Cholesky method as coefficient matrix.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_dbsnls ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b})\);
Single precision:
\[
\text { ierr }=\text { ASL_rbsnls }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R :Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Coefficient matrix \(A\) after \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) decomposition (real symmetric matrix, two-dimensional array type, lower triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\ln a\) & I & 1 & Input & Adjustable dimension af array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & ( \(\mathrm{D} *\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & (R* \(\}\) & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) 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 \(\mathrm{U}^{\mathrm{T}} \mathrm{DU}\) 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.)

\subsection*{2.9 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)}

\subsection*{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 \(\boldsymbol{A} \boldsymbol{x}=\boldsymbol{b}\) having a Hermitian matrix (two-dimensional array type) (upper triangular type) as coefficient matrix.
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbhpsl (ar, ai, lna, n, br, bi, ipvt, w1); }
\]

Single precision:
ierr \(=\) ASL_cbhpsl (ar, ai, lna, n, br, bi, ipvt, w1);
(3) Arguments and Return Value
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Real part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (b)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Imaginary part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (b)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{5} & \multirow[t]{2}{*}{br} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Real part of solution \(\boldsymbol{x}\) \\
\hline \multirow[t]{2}{*}{6} & \multirow[t]{2}{*}{bi} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Imaginary part of solution \(\boldsymbol{x}\) \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 7 & ipvt & \(\mathrm{I}^{*}\) & n & Output & \begin{tabular}{l} 
Pivoting information \\
ipvt[i - 1]: Number of the row(column) ex- \\
changed with row(column) i in the i-th pro- \\
cessing step. (See Note (c))
\end{tabular} \\
\hline 8 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 9 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 . & Contents of arrays ar and ai are not changed. \(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{ar}[0]\) is performed. \\
\hline 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. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & 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. & \\
\hline
\end{tabular}
(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 2.9.4 \(\left\{\begin{array}{l}\text { ASL_zbhpls } \\ \text { ASL_cbhpls }\end{array}\right\}\) 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.

Matrix \(L^{*}\)
\[
\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right]\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]
\]


Storage status within array ai[lna \(\times \mathrm{k}\) ]


Remarks
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(\mathrm{i}-\mathrm{th}\) step is row (column) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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.

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
9 & 7+3 i & 2+5 i & 1+i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+i \\
1-i & 2-4 i & 5-i & 6
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
10+6 i \\
11+2 i \\
4+6 i \\
4+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix real part ar and Imaginary part ai, \(\ln \mathrm{a}=11, \mathrm{n}=4\) 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;
fp = fopen( "zbhpsl.dat", "r" );
if( fp == NULL )
{
printf( "file open error\n" );
}
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" );
}
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] );
}
f
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 **
$\mathrm{n}=4$

```

Coefficient Matrix (Real, Imaginary)


Constant Vector (Real, Imaginary)
\begin{tabular}{lrl}
\((\) & 10, & \(6)\) \\
\((\) & 11, & \(2)\) \\
\((\) & 4 \\
\((\) & \(6)\) & \(6)\)
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|}
\hline x [ & 0 & = & 1 & & ) \\
\hline x & 1 & = & - & & \\
\hline x [ & 2 & = & (-4.97e-17 & & 1) \\
\hline x[ & 3. & = & (-4.17e-17 & & \\
\hline
\end{tabular}

\subsection*{2.9.2 ASL_zbhpud, ASL_cbhpud}

\section*{LDL* Decomposition of a Hermitian Matrix}

\section*{(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
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for } 64 \text { bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Real part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Imaginary part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 3 & lna & I & 1 & Input & Adjustable dimension of array ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times n\) & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 . & Contents of arrays ar and ai are not changed. \\
\hline 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. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & A diagonal element became equal to 0.0 in the \(i\)-th processing step. \(A\) is nearly singular. & \\
\hline
\end{tabular}
(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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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.

\subsection*{2.9.3 ASL_zbhpuc, ASL_cbhpuc}

\section*{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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{\(\left.\begin{array}{l}\text { ar Contents } \\
\mathrm{R} *\end{array}\right\}\)}
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of arrays ar and ai are not \\
changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL* decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline \(4000+i\) &
\end{tabular}

\section*{(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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.9.4 ASL_zbhpls, ASL_cbhpls}

Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix)

\section*{(1) Function}

ASL_zbhpls or ASL_cbhpls solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}=\boldsymbol{b}\) having the Hermitian matrix \(A\) (two-dimensional array type) (upper triangular type) which has been \(\mathrm{LDL}^{*}\) 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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \(\begin{array}{l}\text { Argument and } \\
\text { Return Value }\end{array}\) & Type & Size & \(\begin{array}{l}\text { Input/ } \\
\text { Output }\end{array}\) & \multicolumn{1}{|c|}{\(\begin{array}{l}\text { Contents } \\
\hline 1\end{array}\)} \\
\hline ar \(*\)
\end{tabular}\(\}\)
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] /\) ar \([0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbhpud } \\ \text { ASL_cbhpud }\end{array}\right\}\). However, if you also want to obtain the condition number, you should use 2.9.3 \(\left\{\begin{array}{l}\text { ASL_zbhpuc } \\ \text { ASL_cbhpuc }\end{array}\right\}\). In addition, if you have already used 2.9.1 \(\left\{\begin{array}{l}\text { ASL_zbhpsl } \\ \text { ASL_cbhpsl }\end{array}\right\}\) 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 2.9.5 \(\left\{\begin{array}{l}\text { ASL_zbhpms } \\ \text { ASL_cbhpms }\end{array}\right\}\) 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 \(A\).

\subsection*{2.9.5 ASL_zbhpms, ASL_cbhpms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*Decomposed Hermitian Matrix)}

\section*{(1) Function}

ASL_zbhpms or ASL_cbhpms solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m)\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbhpms (ar, ai, lna, n, br, bi, lnb, m, ipvt); }
\]

Single precision:
ierr \(=\) ASL_cbhpms (ar, ai, lna, n, br, bi, lnb, m, ipvt);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R :Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Real part of Coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Imaginary part of Coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & br & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lnb} \times \mathrm{m}\) & Input & Real part of Constant vector \(\boldsymbol{b}_{\boldsymbol{i}}\) \((i=1,2, \cdots, m)\) \\
\hline & & & & Output & Real part of Solution \(\boldsymbol{x}_{\boldsymbol{i}}(i=1,2, \cdots, m)\) \\
\hline 6 & bi & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lnb} \times \mathrm{m}\) & Input & Imaginary part of Constant vector \(\boldsymbol{b}_{\boldsymbol{i}}\) \((i=1,2, \cdots, m)\) \\
\hline & & & & Output & Imaginary part of Solution \(\boldsymbol{x}_{\boldsymbol{i}}\) \((i=1,2, \cdots, m)\) \\
\hline 7 & \(\operatorname{lnb}\) & I & 1 & Input & Adjustable dimension of array br and bi \\
\hline 8 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 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)) \\
\hline 10 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\) n \(\leq \ln a, \ln b\)
(b) \(\mathrm{m}>0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 . & \[
\begin{aligned}
& \mathrm{br}[\operatorname{lnb} *(\mathrm{i}-1)] \leftarrow \mathrm{br}[\operatorname{lnb} *(\mathrm{i}-1)] / \operatorname{ar}[0], \\
& \mathrm{bi}[\operatorname{lnb} *(\mathrm{i}-1)] \leftarrow \mathrm{bi}[\operatorname{lnb} *(\mathrm{i}-1)] / \operatorname{ar}[0] \\
& (\mathrm{i}=1,2, \cdots, m) \text { are performed } .
\end{aligned}
\] \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_zbhpud } \\ \text { ASL_cbhpud }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.9.3 \(\left\{\begin{array}{l}\text { ASL_zbhpuc } \\ \text { ASL_cbhpuc }\end{array}\right\}\). In addition, if you have already used 2.9.1 \(\left\{\begin{array}{l}\text { ASL_zbhpsl } \\ \text { ASL_cbhpsl }\end{array}\right\}\) 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. 2-9 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 \(A\).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
9 & 7+3 i & 2+5 i & 1+1 i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+1 i \\
1-1 i & 2-4 i & 5-1 i & 6
\end{array}\right]\left[\begin{array}{rlll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{rrr}
10+6 i & 8+18 i & 22 i \\
11+2 i & 12+11 i & 8+23 i \\
7+14 i \\
4+6 i & 15+5 i & 20+6 i \\
4+6 i & 8+2 i & 16+2 i \\
42+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\) which has been LDL* decomposed by the modified Cholesky method, lna \(=\) \(11, \mathrm{n}=4\), constant vectors \(\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m), \operatorname{lnb}=11\) and \(\mathrm{m}=4\).
(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" );
}
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" );
}
br = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
iff( br == NULL )
printf( "no enough memory for array br\n" );
}
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) ));
{k( 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
    {for( i=0 ; i<n ; i++ )
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
            {or( 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=
Coefficient Matrix (Real, Imaginary)

```

Constant Vector (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline ( & 10 & & 6) & 8 & & 18) & 0 & & 22) & 2 & & 10) \\
\hline ( & 11 & , & 2) & 12 & , & 11) & 8 & , & 23) & 7 & & 14) \\
\hline ( & 4 & & 6) & 15 & & 5) & 20 & & 6) & 9 & & 7) \\
\hline ( & 4 & & 6) & 8 & , & 2) & 16 & & 2) & 12 & & 6) \\
\hline
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)


\subsection*{2.9.6 ASL_zbhpdi, ASL_cbhpdi}

Determinant and Inverse Matrix of a Hermitian Matrix

\section*{(1) Function}

ASL_zbhpdi or ASL_cbhpdi obtains the determinant and inverse matrix of the Hermitian matrix \(A\) (twodimensional 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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{\(\left.\begin{array}{l}\text { ar Contents } \\
\mathrm{R} *\end{array}\right\}\)}
\end{tabular}
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & Contents \\
\hline 8 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 9 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l}
\(\operatorname{det}[0] \leftarrow \mathrm{a}[0]\) \\
\(\operatorname{det}[1] \leftarrow 0.0\) \\
\(\operatorname{ar}[0] \leftarrow 1.0 /\) ar \([0]\) are performed.
\end{tabular} \\
\hline & & Processing is aborted. \\
\hline 3000 & Restriction (a) was not satisfied. & \\
\hline
\end{tabular}

\section*{(6) Notes}
(a) The coefficient matrix \(A\) must be LDL* decomposed before using this function. Use any of the 2.9.2 \(\left\{\begin{array}{c}\text { ASL_zbhpud } \\ \text { ASL_cbhpud }\end{array}\right\}, 2.9 .3\left\{\begin{array}{l}\text { ASL_zbhpuc } \\ \text { ASL_cbhpuc }\end{array}\right\}, 2.9 .1\left\{\begin{array}{l}\text { ASL_zbhpsl } \\ \text { ASL_cbhpsl }\end{array}\right\}\) 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 2.9.1 Figure 2-9.)
(c) The determinant is given by the following expression:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times\left(10.0^{\operatorname{det}[1]}\right)
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.9.7 ASL_zbhplx, ASL_cbhplx Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix)}

\section*{(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
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Imaginary part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar, ai, alr and ali \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & alr & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna} \times n\) & Input & Real part of coefficient matrix \(A\) after LDL* decomposition (See Note (a)) \\
\hline 6 & ali & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Imaginary part of coefficient matrix \(A\) after LDL* decomposition (See Note (a)) \\
\hline 7 & br & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Real part of constant vector b \\
\hline 8 & bi & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline 9 & xr & \{ \(\mathrm{D} *\}\) & n & Input & Real part of approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Real part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline 10 & xi & & n & Input & Imaginary part of approximate solution \(\boldsymbol{x}\) \\
\hline & & \(\{\mathrm{R} *\) \} & & Output & Imaginary part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 11 & itol & \(\mathrm{I}^{*}\) & 1 & \multicolumn{1}{|c|}{ Input } & \begin{tabular}{l} 
Number of digits to which solution is to be \\
improved (See Note (b))
\end{tabular} \\
\cline { 4 - 6 } & & & Output & \begin{tabular}{l} 
Approximate number of digits to which solu- \\
tion was improved (See Note (c))
\end{tabular} \\
\hline 12 & nit & I & 1 & Input & \begin{tabular}{l} 
Maximum number of iterations (See Note \\
(d))
\end{tabular} \\
\hline 13 & ipvt & \(\mathrm{I}^{*}\) & n & Output & Pivoting information. (See Note (a)) \\
\hline 14 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(3 \times \mathrm{n}\) & Work & Work area \\
\hline 15 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.9.1 \(\left\{\begin{array}{l}\text { ASL_zbhpsl } \\ \text { ASL_cbhpsl }\end{array}\right\}\) or 2.9.4 \(\left\{\begin{array}{l}\text { ASL_zbhpls } \\ \text { ASL_cbhpls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after it has been decomposed by the 2.9.1 \(\left\{\begin{array}{l}\text { ASL_zbhpsl } \\ \text { ASL_cbhpsl }\end{array}\right\}\), 2.9.2 \(\left\{\begin{array}{l}\text { ASL_zbhpud } \\ \text { ASL_cbhpud }\end{array}\right\}\), or 2.9.3 \(\left\{\begin{array}{l}\text { ASL_zbhpuc } \\ \text { ASL_cbhpuc }\end{array}\right\}\) 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.
itol \(\leq 0\) or itol \(\geq-\log _{10}(2 \times \varepsilon) \quad(\varepsilon:\) 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.

\subsection*{2.10 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) (NO PIVOTING)}

\subsection*{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:
\[
\text { ierr }=\text { ASL_zbhrsl }(\text { ar, ai, lna, n, br, bi, w1 })
\]

Single precision:
ierr \(=\) ASL_cbhrsl (ar, ai, lna, n, br, bi, w1);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Real part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (b)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Imaginary part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (b)) \\
\hline 3 & lna & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{5} & \multirow[t]{2}{*}{br} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Real part of solution \(\boldsymbol{x}\) \\
\hline \multirow[t]{2}{*}{6} & \multirow[t]{2}{*}{bi} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Imaginary part of solution \(\boldsymbol{x}\) \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|c|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & Contents \\
\hline 7 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & \(2 \times \mathrm{n}\) & Work & Work area \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln\) a
(5) Error indicator (Return Value)
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 . & \begin{tabular}{l}
Contents of arrays ar and ai are not changed. \\
\(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{ar}[0]\) is performed.
\end{tabular} \\
\hline 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. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & 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. & \\
\hline
\end{tabular}
(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).

Matrix \(L^{*}\)
Matrix \(D\)
\[
\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right]\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]
\]
\(\Downarrow\)


Storage status within array ai[lna \(\times \mathrm{k}\) ]


\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(D\)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
9 & 7+3 i & 2+5 i & 1+i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+i \\
1-i & 2-4 i & 5-i & 6
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
10+6 i \\
11+2 i \\
4+6 i \\
4+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix real part ar and Imaginary part ai, \(\operatorname{lna}=11, \mathrm{n}=4\) 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;;
fp = fopen( "zbhrsl.dat", "r" );
if( fp == NULL )
{
printf( "file open error\n" );
}
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" );
}
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\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 )

```
\}
(d) Output results
```

*** ASL_zbhrsl ***
** Input **
n = 4

```
Coefficient Matrix (Real, Imaginary)


Constant Vector (Real, Imaginary)
\begin{tabular}{lrl}
\((\) & 10, & \(6)\) \\
\((\) & 11, & \(2)\) \\
\((\) & 4, & \(6)\) \\
\((\) & 4 & \(6)\)
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)
\(\left.\begin{array}{lll}\mathrm{x}[ & 0] & =( \\ \mathrm{x}[ & 1] & =(1\end{array}\right)\)

\subsection*{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:
\[
\text { ierr }=\text { ASL_zbhrud (ar, ai, lna, n, w1) }
\]

Single precision:
\[
\text { ierr }=\text { ASL_cbhrud (ar, ai, lna, n, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Real part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Imaginary part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times n\) & Work & Work area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|c|c|}
\hline ierr value & Meaning & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 . & Contents of arrays ar and ai are not changed. \\
\hline 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. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l}
A diagonal element became equal to 0.0 in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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).

\subsection*{2.10.3 ASL_zbhruc, ASL_cbhruc}

\section*{LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting)}

\section*{(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).

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_zbhruc (ar, ai, lna, n, \&cond, w1);
Single precision:
\[
\text { ierr }=\text { ASL_cbhruc (ar, ai, lna, n, \&cond, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R :Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna} \times \mathrm{n}\)} & Input & Real part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Real part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Imaginary part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) \\
\hline & & & & Output & Imaginary part of upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 3 & \(\ln a\) & I & 1 & Input & Adjustable dimension of arrays ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times \mathrm{n}\) & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of arrays ar and ai are not \\
changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL* decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline \(4000+i\) & \\
\hline
\end{tabular}
(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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.10.4 ASL_zbhrls, ASL_cbhrls}

Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

\section*{(1) Function}

ASL_zbhrls or ASL_cbhrls solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}=\boldsymbol{b}\) 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.

\section*{(2) Usage}

Double precision:
\[
\text { ierr }=\text { ASL_zbhrls (ar, ai, lna, n, br, bi); }
\]

Single precision:
\[
\text { ierr }=\text { ASL_cbhrls (ar, ai, lna, n, br, bi); }
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Real part of coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Imaginary part of coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 3 & lna & I & 1 & Input & Adjustable dimension of array ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & br & (D* & n & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline & & R* \(\}\) & & Output & Real part of solution \(\boldsymbol{x}\) \\
\hline 6 & bi & \{ \(\mathrm{D} *\}\) & n & Input & Imaginary part of constant vector \(\boldsymbol{b}\) \\
\hline & & R* \(\}\) & & Output & Imaginary part of solution \(\boldsymbol{x}\) \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \operatorname{ar}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_zbhrud } \\ \text { ASL_cbhrud }\end{array}\right\}\). However, if you also want to obtain the condition number, you should use 2.10.3 \(\left\{\begin{array}{l}\text { ASL_zbhruc } \\ \text { ASL_cbhruc }\end{array}\right\}\). In addition, if you have already used 2.10.1 \(\left\{\begin{array}{l}\text { ASL_zbhrsl } \\ \text { ASL_cbhrsl }\end{array}\right\}\) 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 2.10.5 \(\left\{\begin{array}{l}\text { ASL_zbhrms } \\ \text { ASL_cbhrms }\end{array}\right\}\) 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).

\subsection*{2.10.5 ASL_zbhrms, ASL_cbhrms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*Decomposed Hermitian Matrix) (No Pivoting)}

\section*{(1) Function}

ASL_zbhrms or ASL_cbhrms solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbhrms (ar, ai, lna, n, br, bi, lnb, m); }
\]

Single precision:
ierr \(=\) ASL_cbhrms (ar, ai, lna, n, br, bi, lnb, m);
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} \\
\(\left.\begin{array}{|c|c|c|c|l|l|}\hline \text { I: }\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right.\end{array}\right\}\) \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\) n \(\leq \ln \mathrm{a}, \operatorname{lnb}\)
(b) \(\mathrm{m}>0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l}
\(\mathrm{br}[\operatorname{lnb} *(\mathrm{i}-1)] \leftarrow \mathrm{br}[\operatorname{lnb} *(\mathrm{i}-1)] / \operatorname{ar}[0]\), \\
\(\mathrm{bi}[\operatorname{lnb} *(\mathrm{i}-1)] \leftarrow \mathrm{bi}[\operatorname{lnb} *(\mathrm{i}-1)] / \mathrm{ar}[0]\) \\
\((\mathrm{i}=1,2, \cdots, m)\) are performed.
\end{tabular} \\
\hline 3000 & & Restriction (a) was not satisfied. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_zbhrud } \\ \text { ASL_cbhrud }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.10.3 \(\left\{\begin{array}{l}\text { ASL_zbhruc } \\ \text { ASL_cbhruc }\end{array}\right\}\). In addition, if you have already used 2.10.1 \(\left\{\begin{array}{l}\text { ASL_zbhrsl } \\ \text { ASL_cbhrsl }\end{array}\right\}\) 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. 2-10 in Section 2.10.1).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrrr}
9 & 7+3 i & 2+5 i & 1+1 i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+1 i \\
1-1 i & 2-4 i & 5-1 i & 6
\end{array}\right]\left[\begin{array}{rrrr}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{rrrr}
10+6 i & 8+18 i & 22 i & 2+10 i \\
11+2 i & 12+11 i & 8+23 i & 7+14 i \\
4+6 i & 15+5 i & 20+6 i & 9+7 i \\
4+6 i & 8+2 i & 16+2 i & 12+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\) which has been LDL* decomposed by the modified Cholesky method, lna \(=\) \(11, \mathrm{n}=4\), constant vectors \(\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m), \operatorname{lnb}=11\) and \(\mathrm{m}=4\).
(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" );
}
bi = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
if( bi == NULL )
printf( "no enough memory for array bi\n" );
}
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( ai )
        free( br )
        free( bi )
        free( wk )
    ```
        \}
(d) Output results


\subsection*{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 \(A\) (twodimensional 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:
\[
\text { ierr }=\text { ASL_cbhrdi (ar, ai, lna, n, det, isw, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{ar} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Real part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Real part of the Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline \multirow[t]{2}{*}{2} & \multirow[t]{2}{*}{ai} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Imaginary part of Hermitian matrix \(A\) (twodimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Imaginary part of the Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline 3 & \(\ln a\) & I & 1 & Input & Adjustable dimension of array ar and ai \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 2 & Output & Determinant of matrix \(A\) (See Note (c) ) \\
\hline 6 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch isw \(>0\) :Obtain determinant. \\
isw \(=0:\) Obtain determinant and inverse matrix. \\
isw \(<0\) :Obtain inverse matrix.
\end{tabular} \\
\hline 7 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l}
\(\operatorname{det}[0] \leftarrow \mathrm{a}[0]\) \\
\(\operatorname{det}[1] \leftarrow 0.0\) \\
\(\operatorname{ar}[0] \leftarrow 1.0 /\) ar \([0]\) are performed.
\end{tabular} \\
\hline 3000 & & Restriction (a) was not satisfied.
\end{tabular} Processing is aborted. \begin{tabular}{c} 
\\
\hline
\end{tabular}
(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:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times\left(10.0^{\operatorname{det}[1]}\right)
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.10.7 ASL_zbhrlx, ASL_cbhrlx}

\section*{Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)}

\section*{(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.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_zbhrlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, \&itol, nit, w1);
Single precision:
\[
\text { ierr }=\text { ASL_cbhrlx (ar, ai, lna, n, alr, ali, br, bi, xr, xi, \&itol, nit, w1); }
\]

\section*{(3) Arguments and Return Value}
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R :Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & ar & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline 2 & ai & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Imaginary part of coefficient matrix \(A\) (Hermitian matrix, two-dimensional array type, upper triangular type) \\
\hline 3 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of arrays ar, ai, alr and ali \\
\hline 4 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 5 & alr & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Real part of coefficient matrix \(A\) after LDL* decomposition (See Note (a)) \\
\hline 6 & ali & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Imaginary part of coefficient matrix \(A\) after LDL* decomposition (See Note (a)) \\
\hline 7 & br & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Real part of constant vector \(\boldsymbol{b}\) \\
\hline 8 & bi & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Imaginary part of constant vector b \\
\hline
\end{tabular}
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{9} & \multirow[t]{2}{*}{xr} & \multirow[t]{2}{*}{\(\underline{\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}}\)} & \multirow[t]{2}{*}{n} & Input & Real part of approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Real part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline \multirow[t]{2}{*}{10} & \multirow[t]{2}{*}{xi} & \multirow[t]{2}{*}{\(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\)} & \multirow[t]{2}{*}{n} & Input & Imaginary part of approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Imaginary part of iteratively improved solution \(\boldsymbol{x}\) \\
\hline \multirow[t]{2}{*}{11} & \multirow[t]{2}{*}{itol} & \multirow[t]{2}{*}{I*} & \multirow[t]{2}{*}{1} & Input & Number of digits to which solution is to be improved (See Note (b)) \\
\hline & & & & Output & Approximate number of digits to which solution was improved (See Note (c)) \\
\hline 12 & nit & I & 1 & Input & Maximum number of iterations (See Note (d)) \\
\hline 13 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(3 \times \mathrm{n}\) & Work & Work area \\
\hline 14 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.10.1 \(\left\{\begin{array}{l}\text { ASL_zbhrsl } \\ \text { ASL_cbhrsl }\end{array}\right\}\) or 2.10.4 \(\left\{\begin{array}{l}\text { ASL_zbhrls } \\ \text { ASL_cbhrls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after it has been decomposed by the 2.10.1 \(\left\{\begin{array}{l}\text { ASL_zbhrsl } \\ \text { ASL_cbhrsl }\end{array}\right\}\), 2.10.2 \(\left\{\begin{array}{l}\text { ASL_zbhrud } \\ \text { ASL_cbhrud }\end{array}\right\}\), or 2.10.3 \(\left\{\begin{array}{l}\text { ASL_zbhruc } \\ \text { ASL_cbhruc }\end{array}\right\}\) 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.

\subsection*{2.11 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE)}

\subsection*{2.11.1 ASL_zbhfsl, ASL_cbhfsl}

\section*{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:
\[
\text { ierr }=\text { ASL_zbhfsl (a, lna, n, b, ipvt, w1); }
\]

Single precision:
\[
\text { ierr }=\text { ASL_cbhfsl (a, lna, n, b, ipvt, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} \\
\begin{tabular}{|c|c|c|c|c|l|}
\hline I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\end{tabular} \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l} 
Contents of array a are not changed. \\
\(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed.
\end{tabular} \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL* decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline \(4000+i\) & \begin{tabular}{l} 
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.
\end{tabular} & \multicolumn{2}{|c|}{\begin{tabular}{|c|}
\hline
\end{tabular}} \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_zbhfls } \\ \text { ASL_cbhfls }\end{array}\right\}\) 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.

Matrix \(L^{*}\)
\[
\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right] \quad\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]
\]
\(\Downarrow\)

\section*{Remarks}
a. \(\quad \operatorname{lna} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(\mathrm{i}-\mathrm{th}\) step is row (column) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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.

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
9 & 7+3 i & 2+5 i & 1+i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+i \\
1-i & 2-4 i & 5-i & 6
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
10+6 i \\
11+2 i \\
4+6 i \\
4+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\), 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;
nt n;
double _Complex *b;
int *ipvt;
double *W1;
int ierr;

```
```

int i,j;;
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" );
}
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 );

```
\}
(d) Output results
```

*** ASL_zbhfsl ***
** Input **
n = 4

```
Coefficient Matrix (Real, Imaginary)


Constant Vector (Real, Imaginary)

** Output **
ierr \(=0\)
Solution (Real, Imaginary)
\begin{tabular}{|c|c|c|c|}
\hline x [ & -1 & = 1 , & 0) \\
\hline x [ & \(1]\) & = 1 , & 8.88e-17) \\
\hline x [ & \(2]\) & \(=(-4.97 \mathrm{e}-17\) & 1) \\
\hline x [ & 3] & \(=(-4.17 \mathrm{e}-17\) & ) \\
\hline
\end{tabular}

\subsection*{2.11.2 ASL_zbhfud, ASL_cbhfud}

\section*{LDL* Decomposition of a Hermitian Matrix}

\section*{(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
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & Input/ Output & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Hermitian matrix \(A\) (two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & w1 & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Work & Work area \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL* decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in \(\mathrm{ipvt}[\mathrm{i}-1]\). 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.

\subsection*{2.11.3 ASL_zbhfuc, ASL_cbhfuc}

\section*{LDL* Decomposition and Condition Number of a Hermitian Matrix}

\section*{(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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & Z:Double precision complex \\
C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{Z} * \\
\mathrm{C} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\ln a \times n\)} & Input & Hermitian matrix \(A\) (two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 2 & lna & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & ipvt & I* & n & Output & Pivoting information \(\operatorname{ipvt}[\mathrm{i}-1]\) : Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b)) \\
\hline 5 & cond & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 1 & Output & Reciprocal of the condition number \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array a are not changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the LDL* decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline \(4000+i\) & \\
\hline
\end{tabular}

\section*{(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) \(\mathrm{j}(\mathrm{i}<\mathrm{j})\), then j is stored in ipvt[ \(\mathrm{i}-1]\). 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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.11.4 ASL_zbhfls, ASL_cbhfls \\ Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix)}

\section*{(1) Function}

ASL_zbhfls or ASL_cbhfls solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}=\boldsymbol{b}\) 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 ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt})\);
Single precision:
ierr \(=\) ASL_cbhfls ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular} I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\operatorname{lna} \times \mathrm{n}\) & Input & Coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline \multirow[t]{2}{*}{4} & \multirow[t]{2}{*}{b} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{Z} * \\
\mathrm{C} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{n} & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & & & Output & Solution \(\boldsymbol{x}\) \\
\hline 5 & ipvt & I* & n & Output & Pivoting information ipvt \([\mathrm{i}-1]\) : Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c)) \\
\hline 6 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}
(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 \(\left\{\begin{array}{l}\text { ASL_zbhfuc } \\ \text { ASL_cbhfuc }\end{array}\right\}\). 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 \(A\).

\subsection*{2.11.5 ASL_zbhfms, ASL_cbhfms}

\section*{Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*Decomposed Hermitian Matrix)}

\section*{(1) Function}

ASL_zbhfms or ASL_cbhfms solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbhfms } \quad(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{~b}, \ln \mathrm{~b}, \mathrm{~m}, \text { ipvt }) ;
\]

Single precision:
ierr \(=\) ASL_cbhfms ( \(\mathrm{a}, \operatorname{lna} \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m}, \mathrm{ipvt})\);
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular}\(\quad\)\begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} \\
\begin{tabular}{|c|c|c|c|c|l|}
\hline I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\) \\
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{c|}{ Contents }
\end{tabular} \\
\hline 1 \\
a \\
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}, \ln \mathrm{b}\)
(b) \(\mathrm{m}>0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l}
\(\mathrm{b}[\operatorname{lnb} *(\mathrm{i}-1)] \leftarrow \mathrm{b}[\operatorname{lnb} *(\mathrm{i}-1)]) / \mathrm{a}[0]\) \\
\((\mathrm{i}=1,2, \cdots, m)\) are performed.
\end{tabular} \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline
\end{tabular}
(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.
(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 \(A\).

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
9 & 7+3 i & 2+5 i & 1+1 i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+1 i \\
1-1 i & 2-4 i & 5-1 i & 6
\end{array}\right]\left[\begin{array}{rlll}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{rrrr}
10+6 i & 8+18 i & 22 i & 2+10 i \\
11+2 i & 12+11 i & 8+23 i & 7+14 i \\
4+6 i & 15+5 i & 20+6 i & 9+7 i \\
4+6 i & 8+2 i & 16+2 i & 12+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\) which has been LDL* decomposed by the modified Cholesky method, lna \(=\) \(11, \mathrm{n}=4\), constant vectors \(\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m), \operatorname{lnb}=11\) and \(\mathrm{m}=4\).
(c) Main program
```

/* C interface example for ASL_zbhfms */
\#include <stdio.h>
\#include <stdlib.h>
\#include <stdlib.h>
\#include <complex
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" );
}
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" );
}
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" );
}
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++ )
{ , j=1 , j<n ; j++ )
printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
}
}
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 );
ree( b );
free( wk);
}
(d) Output results

```
```

*** ASL_zbhfms ***
** Input **
n=
Coefficient Matrix (Real, Imaginary)

```
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline \multirow[t]{4}{*}{(} & \multirow[t]{4}{*}{9 ,} & \multirow[t]{4}{*}{0) \((\)} & \multirow[t]{4}{*}{7
10} & \multirow[t]{4}{*}{3)} & \multirow[t]{4}{*}{2
3
8} & \multicolumn{2}{|l|}{\multirow[t]{4}{*}{\begin{tabular}{l}
5) \\
2) \\
0)
\end{tabular}}} & \multicolumn{3}{|l|}{\multirow[t]{4}{*}{\begin{tabular}{ll}
1, & \(1)\) \\
2, & \(4)\) \\
5 &, \\
6 & \(1)\)
\end{tabular}}} \\
\hline & & & & & & & & & & \\
\hline & & & & & & & & & & \\
\hline & & & & & & & & & & \\
\hline
\end{tabular}
Constant Vector (Real, Imaginary)

* Output **
ierr = 0
Solution (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline ( 1 & 0) & (-4.28e-16 & 1) & ( \(5.35 \mathrm{e}-17\) & & 1) & ( & 1 & & 0) \\
\hline ( 1 & 8.88e-17) & ( 1 & , \(-1.78 \mathrm{e}-16)\) & (-1.78e-16 & & 1) & ( & 0 & & 1) \\
\hline (-4.97e-17 & 1) & ( 1 & , \(-1.33 \mathrm{e}-16\) ) & ( 1 & & 0) & ( & 0 & & 1) \\
\hline (-4.17e-17 & 1) & ( 8.34e-17 & 1) & ( 1 & & -8.34e-17) & & 1 & & \(-8.34 e-17)\) \\
\hline
\end{tabular}

\subsection*{2.11.6 ASL_zbhfdi, ASL_cbhfdi}

\section*{Determinant and Inverse Matrix of a Hermitian Matrix}

\section*{(1) Function}

ASL_zbhfdi or ASL_cbhfdi obtains the determinant and inverse matrix of the Hermitian matrix \(A\) (twodimensional 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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{Z} * \\
\mathrm{C} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Hermitian matrix \(A\) (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 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)) \\
\hline 5 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 2 & Output & Determinant of matrix \(A\) (See Note (c)) \\
\hline 6 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
isw \(>0\) :Obtain determinant. \\
isw \(=0\) :Obtain determinant and inverse matrix. \\
isw \(<0\) :Obtain inverse matrix.
\end{tabular} \\
\hline 7 & w1 & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & n & Work & Work area \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{det}[0] \leftarrow \mathrm{a}[0]\) \\
& & \(\operatorname{det}[1] \leftarrow 0.0\) \\
& & \(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]\) are performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be LDL* decomposed before using this function. Use any of the functions 2.11.2 \(\left\{\begin{array}{c}\text { ASL_zbhfud } \\ \text { ASL_cbhfud }\end{array}\right\}, 2.11 .3\left\{\begin{array}{c}\text { ASL_zbhfuc } \\ \text { ASL_cbhfuc }\end{array}\right\}, 2.11 .1\left\{\begin{array}{l}\text { ASL_zbhfsl } \\ \text { ASL_cbhfsl }\end{array}\right\}\) 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. 2-11 in Section 2.11.1)
(c) The determinant is given by the following expression:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times\left(10.0^{\operatorname{det}[1]}\right)
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.11.7 ASL_zbhflx, ASL_cbhflx}

\section*{Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix)}

\section*{(1) Function}

ASL_zbhflx or ASL_cbhflx 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_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
\begin{tabular}{ll}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} & \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & \multicolumn{1}{|c|}{ Size } & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{\(\left.\begin{array}{l}\text { Contents } \\
\mathrm{C} *\end{array}\right\}\)}
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln\) a
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.11.1 \(\left\{\begin{array}{l}\text { ASL_zbhfsl } \\ \text { ASL_cbhfsl }\end{array}\right\}\) or 2.11.4 \(\left\{\begin{array}{l}\text { ASL_zbhfls } \\ \text { ASL_cbhfls }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after it has been decomposed by the 2.11.1 \(\left\{\begin{array}{l}\text { ASL_zbhfsl } \\ \text { ASL_cbhfsl }\end{array}\right\}\), 2.11.2 \(\left\{\begin{array}{l}\text { ASL_zbhfud } \\ \text { ASL_cbhfud }\end{array}\right\}\), or 2.11.3 \(\left\{\begin{array}{l}\text { ASL_zbhfuc } \\ \text { ASL_cbhfuc }\end{array}\right\}\) 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.
\[
\text { itol } \leq 0
\]
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.

\subsection*{2.12 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) (NO PIVOTING)}

\subsection*{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:
\[
\text { ierr }=\text { ASL_zbhesl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
\]

Single precision:
ierr \(=\) ASL_cbhesl ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b})\);
(3) Arguments and Return Value
D:Double precision real
R:Single precision real \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l} 
Contents of array a are not changed. \\
\(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed.
\end{tabular} \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline \(4000+i\) & \begin{tabular}{l} 
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.
\end{tabular} & \multicolumn{2}{|c|}{\begin{tabular}{|c} 
\\
\hline
\end{tabular}} \\
\hline
\end{tabular}
(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.

Matrix \(L^{*}\)
\[
\left[\begin{array}{ccccc}
l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\
0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\
0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & l_{5,5}
\end{array}\right]
\]
\[
\begin{gathered}
\text { Matrix } D \\
{\left[\begin{array}{ccccc}
1 / l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\
0.0 & 1 / l_{2,2} & 0.0 & \cdots & 0.0 \\
0.0 & 0.0 & 1 / l_{3,3} & \cdots & 0.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0.0 & 0.0 & 0.0 & \cdots & 1 / l_{5,5}
\end{array}\right]}
\end{gathered}
\]
\(\Downarrow\)

\section*{Remarks}
a. \(\quad \ln \mathrm{a} \geq \mathrm{n}\) and \(\mathrm{n} \leq \mathrm{k}\) 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 \(D\)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
9 & 7+3 i & 2+5 i & 1+i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+i \\
1-i & 2-4 i & 5-i & 6
\end{array}\right]\left[\begin{array}{c}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
10+6 i \\
11+2 i \\
4+6 i \\
4+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A, \operatorname{lna}=11, \mathrm{n}=4\) and constant vector \(\boldsymbol{b}\).
(c) Main program
```

/* C interface example for ASL_zbhesl */
\#include <stdio.h>
\#include <stdio.h>
\#include <complex.h>
\#include <asl.h>
int main()
double _Complex *a;
int na;
double _Complex *b;
int ierr;
int ierr;
int i,j;;
fp = fopen( "zbhesl.dat", "r" );
if( fp == NULL )
{
printf( "file open error\n" );
}
printf( " *** ASL_zbhesl ***\n" );
printf( "\n ** Input **\n\n" );

```
```

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

```
fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );
fscanf( fp, "%d", &n );
a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
if( a == NULL )
if( a == NULL )
    printf( "no enough memory for array a\n" );
    printf( "no enough memory for array a\n" );
    return -1;
    return -1;
}
}
b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
b = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
if( b == NULL )
if( b == NULL )
    printf( "no enough memory for array b\n" );
    printf( "no enough memory for array b\n" );
        return -1;
        return -1;
}
}
printf( "\t n = %6d\n", n );
printf( "\t n = %6d\n", n );
printf( "\n\tCoefficient Matrix (Real, Imaginary)\n\n");
printf( "\n\tCoefficient Matrix (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    for( j=0 ; j<n ; j++ )
    for( j=0 ; j<n ; j++ )
            double tmp_re;
            double tmp_re;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_re );
            a[i+na*j] = tmp_re;
            a[i+na*j] = tmp_re;
        }
        }
}
}
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
        for( j=0 ; j<n ; j++ )
        for( j=0 ; j<n ; j++ )
        { double tmp_im;
        { double tmp_im;
            fscanf( fp, "%lf", &tmp_im );
            fscanf( fp, "%lf", &tmp_im );
            a[i+na*j] = a[i+na*j] + tmp_im * _Complex_I;
            a[i+na*j] = a[i+na*j] + tmp_im * _Complex_I;
        }
        }
}
}
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
        printf( "\t" );
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        for( j=0 ; j<i ; j++ )
        { printf( " " );
        { printf( " " );
        }
        }
        for( j=i ; j<n ; j++ )
        for( j=i ; j<n ; j++ )
        printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
        printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
    f printf( "\n" );
    f printf( "\n" );
}
}
printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
        double tmp_re;
        double tmp_re;
        fscanf( fp, "%lf", &tmp_re );
        fscanf( fp, "%lf", &tmp_re );
        b[i] = tmp_re;
        b[i] = tmp_re;
}
}
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    double tmp_im;
    double tmp_im;
    fscanf( fp, "%lf", &tmp_im );
    fscanf( fp, "%lf", &tmp_im );
        b[i] = b[i] + tmp_im * _Complex_I;
        b[i] = b[i] + tmp_im * _Complex_I;
}
}
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
    printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
f flose( fp );
f flose( fp );
ierr=ASL_zbhesl(a, na, n, b);
ierr=ASL_zbhesl(a, na, n, b);
printf( "\tierr = %6d\n", ierr );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tSolution (Real, Imaginary)\n\n" );
printf( "\n\tSolution (Real, Imaginary)\n\n" );
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    printf( "\t x[%6d] = (%8.3g , %8.3g)\n", i,creal(b[i]),cimag(b[i]));
    printf( "\t x[%6d] = (%8.3g , %8.3g)\n", i,creal(b[i]),cimag(b[i]));
}
}
free( a );
free( a );
return 0;
```

return 0;

```
```

            double tmp_im;
    ```
            double tmp_im;
        printf( "\n" );
        printf( "\n" );
printf( "\n ** Output **\n\n" );
```

printf( "\n ** Output **\n\n" );

```
\}
(d) Output results
```

*** ASL_zbhesl ***
** Input **
n = 4

```
Coefficient Matrix (Real, Imaginary)


Constant Vector (Real, Imaginary)
\begin{tabular}{lrl}
\((\) & 10, & \(6)\) \\
\((\) & 11, & \(2)\) \\
\((\) & 4, & \(6)\)
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)

\subsection*{2.12.2 ASL_zbheud, ASL_cbheud}

\section*{LDL* Decomposition of a Hermitian Matrix (No Pivoting)}

\section*{(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:
\[
\text { ierr }=\text { ASL_zbheud }(a, \ln a, n)
\]

Single precision:
ierr \(=\) ASL_cbheud ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n})\);
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Hermitian matrix \(A\) (two-dimensional array type, upper triangular type) \\
\hline & & & & Output & Upper triangular matrix \(L^{*}\) when \(A\) is decomposed into \(A=L D L^{*}\) (See Note (a)) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & Contents of array a are not changed. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A\). The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & Processing continues. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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)

\subsection*{2.12.3 ASL_zbheuc, ASL_cbheuc}

\section*{LDL* Decomposition and Condition Number of a Hermitian Matrix (No Pivoting)}

\section*{(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:
\[
\text { ierr }=\text { ASL_cbheuc }(a, \operatorname{lna}, \mathrm{n}, \& \text { cond, w1 })
\]
(3) Arguments and Return Value
\begin{tabular}{l} 
D:Double precision real \\
\begin{tabular}{c} 
R:Single precision real
\end{tabular} \\
\begin{tabular}{|c|c|c|c|c|l|}
\hline Z:Double precision complex \\
No.Single precision complex
\end{tabular} \\
\begin{tabular}{c} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \begin{tabular}{l} 
Contents of array a are not changed. \\
cond \(\leftarrow 1.0\) is performed.
\end{tabular} \\
\hline 1000 & n was equal to 1. & Processing continues. \\
\hline 2100 & \begin{tabular}{l} 
There existed the diagonal element which \\
was close to zero in the \(L U\) decompo- \\
sition of the coefficient matrix \(A . \quad\) The \\
result may not be obtained with a good \\
accuracy.
\end{tabular} & \begin{tabular}{l} 
Restriction (a) was not satisfied.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
A diagonal element became equal to 0.0 \\
in the \(i\)-th processing step. \\
\(A\) is nearly singular.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted. \\
The condition number is not obtained.
\end{tabular} \\
\hline 4000
\end{tabular}
(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 \(\|A\| \cdot\left\|A^{-1}\right\|\), an approximate value is obtained by this function.

\subsection*{2.12.4 ASL_zbhels, ASL_cbhels}

Simultaneous Linear Equations (LDL*-Decomposed Hermitian Matrix) (No Pivoting)

\section*{(1) Function}

ASL_zbhels or ASL_cbhels solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}=\boldsymbol{b}\) 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.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_zbhels ( \(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{b})\);
Single precision:
\[
\text { ierr }=\text { ASL_cbhels }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
\]
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \\
\begin{tabular}{|c|c|c|c|c|l|}
\hline \multicolumn{1}{l}{\begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}} & I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\end{tabular} \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbheud } \\ \text { ASL_cbheud }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.12.3 \(\left\{\begin{array}{l}\text { ASL_zbheuc } \\ \text { ASL_cbheuc }\end{array}\right\}\). In addition, if you have already used 2.12.1 \(\left\{\begin{array}{l}\text { ASL_zbhesl } \\ \text { ASL_cbhesl }\end{array}\right\}\) 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 2.12.5 \(\left\{\begin{array}{l}\text { ASL_zbhems } \\ \text { ASL_cbhems }\end{array}\right\}\) 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. 2-12 in Section 2.12.1)

\subsection*{2.12.5 ASL_zbhems, ASL_cbhems \\ Simultaneous Linear Equations with Multiple Right-Hand Sides (LDL*Decomposed Hermitian Matrix) (No Pivoting)}

\section*{(1) Function}

ASL_zbhems or ASL_cbhems solves the simultaneous linear equations \(L D L^{*} \boldsymbol{x}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m\) 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 \(B=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \cdots, \boldsymbol{b}_{\boldsymbol{m}}\right]\), the function obtains \(\left[\boldsymbol{x}_{\mathbf{1}}, \boldsymbol{x}_{\mathbf{2}}, \cdots, \boldsymbol{x}_{\boldsymbol{m}}\right]=A^{-1} B\).
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_zbhems }(a, \operatorname{lna}, \mathrm{n}, \mathrm{~b}, \ln b, \mathrm{~m}) ;
\]

Single precision:
ierr \(=\) ASL_cbhems ( \(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \mathrm{b}, \operatorname{lnb}, \mathrm{m}\) );
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline & \begin{tabular}{l}
ouble precision rea \\
ngle precision real
\end{tabular} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after LDL* decomposition (Hermitian matrix, two-dimensional array type, upper triangular type) (See Notes (a) and (b)) \\
\hline 2 & \(\ln a\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & 1 & 1 & Input & Order of matrix \(A\) \\
\hline 4 & b & \{ \(\mathrm{Z} *\) & \(\operatorname{lnb} \times \mathrm{m}\) & Input & Constant vector \(\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m)\) \\
\hline & & C* \(\}\) & & Output & Solution \(\boldsymbol{x}_{\boldsymbol{i}}(i=1,2, \cdots, m)\) \\
\hline 5 & \(\operatorname{lnb}\) & I & 1 & Input & Adjustable dimension of array b \\
\hline 6 & m & I & 1 & Input & Number of right-hand side vectors, \(m\) \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \operatorname{lna}, \ln \mathrm{b}\)
(b) \(\mathrm{m}>0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\mathrm{~b}[\ln \mathrm{~b} *(\mathrm{i}-1)] \leftarrow \mathrm{b}[\operatorname{lnb} *(\mathrm{i}-1)]) / \mathrm{a}[0](\mathrm{i}=\) \\
& & \(1,2, \cdots, m)\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline
\end{tabular}

\section*{(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 \(\left\{\begin{array}{l}\text { ASL_zbheud } \\ \text { ASL_cbheud }\end{array}\right\}\) function. However, if you also want to obtain the condition number, you should use 2.12.3 \(\left\{\begin{array}{l}\text { ASL_zbheuc } \\ \text { ASL_cbheuc }\end{array}\right\}\). In addition, if you have already used 2.12.1 \(\left\{\begin{array}{l}\text { ASL_zbhesl } \\ \text { ASL_cbhesl }\end{array}\right\}\) 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. \(2-12\) in Section 2.12.1)

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrrr}
9 & 7+3 i & 2+5 i & 1+1 i \\
7-3 i & 10 & 3+2 i & 2+4 i \\
2-5 i & 3-2 i & 8 & 5+1 i \\
1-1 i & 2-4 i & 5-1 i & 6
\end{array}\right]\left[\begin{array}{rrrrrr}
x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\
x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\
x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\
x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4}
\end{array}\right]=\left[\begin{array}{rrrr}
10+6 i & 8+18 i & 22 i & 2+10 i \\
11+2 i & 12+11 i & 8+23 i & 7+14 i \\
4+6 i & 15+5 i & 20+6 i & 9+7 i \\
4+6 i & 8+2 i & 16+2 i & 12+6 i
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(A\) which has been LDL* decomposed by the modified Cholesky method, \(\ln \mathrm{a}=11, \mathrm{n}=4\), constant vectors \(\boldsymbol{b}_{\boldsymbol{i}}(i=1,2, \cdots, m), \operatorname{lnb}=11\) and \(\mathrm{m}=4\).
(c) Main program
```

/* C interface example for ASL_zbhems */
\#include <stdio.h>
\#include <stdio.h>
\#include <stdlib.h>
\#include <complex
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" );
}
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" );
}
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\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;
}
}
{
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);
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=
Coefficient Matrix (Real, Imaginary)

```
( 9
0) (

7
10,
3) ( \(\begin{aligned} & 2 \\ & 0) \\ & \text { 0) } \\ & \\ & 8\end{aligned} \quad\)
5)
2)
0)

1
2
5
6,
1)
4)
1)
0)

Constant Vector (Real, Imaginary)
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline ( & 10 & & 6) & 8 & 18) & 0 & 22) & 2 & & 10) \\
\hline ( & 11 & , & 2) & 12 & 11) & 8 & 23) & 7 & & 14) \\
\hline ( & 4 & & 6) & 15 & 5) & 20 & 6) & 9 & & 7) \\
\hline ( & 4 & & 6) & 8 & 2) & 16 & 2) & 12 & & 6) \\
\hline
\end{tabular}
** Output **
ierr \(=0\)
Solution (Real, Imaginary)


\subsection*{2.12.6 ASL_zbhedi, ASL_cbhedi \\ Determinant and Inverse Matrix of a Hermitian Matrix (No Pivoting)}

\section*{(1) Function}

ASL_zbhedi or ASL_cbhedi obtains the determinant and inverse matrix of the Hermitian matrix \(A\) (twodimensional 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
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{Z} * \\
\mathrm{C} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lna\times n}\)} & Input & Hermitian matrix \(A\) (two-dimensional array type) (upper triangular type) after LDL* decomposition (See Notes (a) and (b)) \\
\hline & & & & Output & Inverse matrix of matrix \(A\) (See Note (b)) \\
\hline 2 & \(\ln a\) & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & det & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & 2 & Output & Determinant of matrix \(A\) (See Note (c)) \\
\hline 5 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
isw \(>0\) :Obtain determinant. \\
isw \(=0\) :Obtain determinant and inverse matrix. \\
isw \(<0\) :Obtain inverse matrix.
\end{tabular} \\
\hline 6 & w1 & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & n & Work & Work area \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \(\operatorname{det}[0] \leftarrow \mathrm{a}[0]\) \\
& & \begin{tabular}{l}
\(\operatorname{det}[1] \leftarrow 0.0\) \\
\(\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]\) are performed.
\end{tabular} \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline
\end{tabular}
(6) Notes
(a) The coefficient matrix \(A\) must be LDL* decomposed before using this function. Use any of the functions 2.12.2 \(\left\{\begin{array}{l}\text { ASL_zbheud } \\ \text { ASL_cbheud }\end{array}\right\}, 2.12 .3\left\{\begin{array}{l}\text { ASL_zbheuc } \\ \text { ASL_cbheuc }\end{array}\right\}, 2.12 .1\left\{\begin{array}{l}\text { ASL_zbhesl } \\ \text { ASL_cbhesl }\end{array}\right\}\) 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:
\[
\operatorname{det}(A)=\operatorname{det}[0] \times\left(10.0^{\operatorname{det}[1]}\right)
\]

Scaling is performed at this time so that:
\[
1.0 \leq|\operatorname{det}[0]|<10.0
\]
(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 \(A X=B\) 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.

\subsection*{2.12.7 ASL_zbhelx, ASL_cbhelx \\ Improving the Solution of Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)}

\section*{(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.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_zbhelx (a, lna, n, al, b, x, \&itol, nit, w1);
Single precision:
\[
\text { ierr }=\text { ASL_cbhelx (a, lna, n, al, b, x, \&itol, nit, w1); }
\]
(3) Arguments and Return Value
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{2}{|l|}{D:Double precision real R:Single precision real} & \multicolumn{3}{|l|}{Z:Double precision complex C:Single precision complex} & \[
\text { I: }\left\{\begin{array}{l}
\text { int as for } 32 \text { bit Integer } \\
\text { long as for } 64 \text { bit Integer }
\end{array}\right\}
\] \\
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & a & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\ln a \times n\) & Input & Coefficient matrix \(A\) (Hermitian matrix, twodimensional array type, upper triangular type) \\
\hline 2 & \(\operatorname{lna}\) & I & 1 & Input & Adjustable dimension of array a and al \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & al & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & \(\operatorname{lna\times n}\) & Input & Coefficient matrix \(A\) after LDL* decomposition (See Note (a)) \\
\hline 5 & b & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline 6 & x & (Z* & n & Input & Approximate solution \(\boldsymbol{x}\) \\
\hline & & & & Output & Iteratively improved solution \(\boldsymbol{x}\) \\
\hline 7 & itol & I* & 1 & Input & Number of digits to which solution is to be improved (See Note (b)) \\
\hline & & & & Output & Approximate number of digits to which solution was improved (See Note (c)) \\
\hline 8 & nit & I & 1 & Input & Maximum number of iterations (See Note (d)) \\
\hline 9 & w1 & \(\left\{\begin{array}{l}\mathrm{Z} * \\ \mathrm{C} *\end{array}\right\}\) & n & Work & Work area \\
\hline 10 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}

\section*{(4) Restrictions}
(a) \(0<\mathrm{n} \leq \ln \mathrm{a}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & The solution is not improved. \\
\hline 3000 & Restriction (a) was not satisfied. & Processing is aborted. \\
\hline 5000 & \begin{tabular}{l} 
The solution did not converge within the \\
maximum number of iterations.
\end{tabular} & \begin{tabular}{l} 
Processing is aborted after calculating the \\
itol output value.
\end{tabular} \\
\hline 6000 & The solution could not be improved. & \\
\hline
\end{tabular}
(6) Notes
(a) This function improves the solution obtained by the 2.12.1 \(\left\{\begin{array}{l}\text { ASL_zbhesl } \\ \text { ASL_cbhesl }\end{array}\right\}\) or 2.12.4 \(\left\{\begin{array}{l}\text { ASL_zbhels } \\ \text { ASL_cbhels }\end{array}\right\}\) function. Therefore, the coefficient matrix \(A\) after being decomposed by the 2.12.3 \(\left\{\begin{array}{l}\text { ASL_zbheuc } \\ \text { ASL_cbheuc }\end{array}\right\}\), 2.12.1 \(\left\{\begin{array}{l}\text { ASL_zbhesl } \\ \text { ASL_cbhesl }\end{array}\right\}\) or 2.12.2 \(\left\{\begin{array}{l}\text { ASL_zbheud } \\ \text { ASL_cbheud }\end{array}\right\}\) 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.

\subsection*{2.13 REAL BAND MATRIX (BAND TYPE)}

\subsection*{2.13.1 ASL_dbbdsl, ASL_rbbdsl}

\section*{Simultaneous Linear Equations (Real Band Matrix)}
(1) Function

ASL_dbbdsl or ASL_rbbdsl uses the Gauss method to solve the simultaneous linear equations \(\boldsymbol{A} \boldsymbol{x}=\boldsymbol{b}\) 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:
\[
\text { ierr }=\text { ASL_rbbdsl }(\mathrm{a}, \text { lma, } \mathrm{n}, \mathrm{mu}, \mathrm{ml}, \mathrm{~b}, \mathrm{ipvt}) ;
\]
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline \multirow[t]{2}{*}{1} & \multirow[t]{2}{*}{a} & \multirow[t]{2}{*}{\[
\left\{\begin{array}{l}
\mathrm{D} * \\
\mathrm{R} *
\end{array}\right\}
\]} & \multirow[t]{2}{*}{\(\operatorname{lma} \times \mathrm{n}\)} & Input & Coefficient matrix \(A\) (real band matrix, band type) (See Appendix B) \\
\hline & & & & Output & Upper triangular matrix \(U\) and unit lower triangular matrix \(L\) when \(A\) is decomposed into \(A=L U\) (See Note (b)) \\
\hline 2 & lma & I & 1 & Input & Adjustable dimension of array a \\
\hline 3 & n & I & 1 & Input & Order of matrix \(A\) \\
\hline 4 & mu & I & 1 & Input & Upper band width of matrix \(A\) \\
\hline 5 & ml & I & 1 & Input & Lower band width of matrix \(A\) \\
\hline 6 & b & \{ \(\mathrm{D} *\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline & & \{ \(\mathrm{R} *\}\) & & Output & Solution \(\boldsymbol{x}\) \\
\hline 7 & ipvt & I* & n & Output & \begin{tabular}{l}
Pivoting information \\
ipvt[i-1]: Number of row exchanged with row \(i\) in the i-th processing step \\
(See Note (b))
\end{tabular} \\
\hline 8 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) \(\mathrm{n}>0\)
(b) \(0 \leq m u \leq n-1\)
\(0 \leq \mathrm{ml} \leq \mathrm{n}-1\)
(c) \(\min (2 \times \mathrm{ml}+\mathrm{mu}+1, \mathrm{n}+\mathrm{ml}) \leq \mathrm{lma}\)

\section*{(5) Error indicator (Return Value)}
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1. & \begin{tabular}{l} 
Contents of array a are not changed. \\
\(\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]\) is performed.
\end{tabular} \\
\hline 3000 & \begin{tabular}{l} 
Restriction (a), (b) or (c) was not \\
satisfied.
\end{tabular} & Processing is aborted. \\
\hline \(4000+i\) & \begin{tabular}{l} 
The pivot became 0.0 in the \(i\)-th process- \\
ing step of the LU decomposition of coef- \\
ficient matrix \(A\). \\
\(A\) is nearly singular.
\end{tabular} & \\
\hline
\end{tabular}
(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 \(\mathrm{j}(\mathrm{i} \leq \mathrm{j})\), then j is stored in ipvt \([\mathrm{i}-1]\). 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 \(\mathrm{ml} \times \mathrm{n}\). Therefore, if \(\mathrm{n}<2 \mathrm{ml}+\mathrm{mu}+1\), less memory is required to use the function for real matrices.

\(\Downarrow\)
Storage status within array a \([\mathrm{lma} \times \mathrm{k}]\)


\section*{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. \(\quad \operatorname{lma} \geq 2 \times \mathrm{ml}+\mathrm{mu}+1\) and \(\mathrm{k} \geq \mathrm{n}\) must hold.

Figure 2-13 Storage Status of Array a before and after LU Decomposition

\section*{(7) Example}
(a) Problem

Solve the following simultaneous linear equations.
\[
\left[\begin{array}{rrrr}
1 & -2 & 0 & 0 \\
-1 & 3 & 2 & 0 \\
1 & -1 & 4 & -2 \\
0 & 1 & -1 & 7
\end{array}\right]\left[\begin{array}{l}
X_{1} \\
X_{2} \\
X_{3} \\
X_{4}
\end{array}\right]=\left[\begin{array}{r}
3 \\
-7 \\
1 \\
13
\end{array}\right]
\]
(b) Input data

Coefficient matrix \(\mathrm{a}, \operatorname{lma}=11, \mathrm{n}=4, \mathrm{mu}=1, \mathrm{ml}=2\) 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 SPP=, ,;
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" );
}
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" );
}
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( bpvi )
    return 0;
    }
(d) Output results

```
```

*** ASL_dbbdsl ***

```
*** ASL_dbbdsl ***
    ** Input **
    ** Input **
n = 4
Upper Band Width = 1
Lower Band Width = 2
Coefficient Matrix
```



```
    M
Constant Vector
    3
    ** Output **
ierr = 0
Solution
\begin{tabular}{lll}
\(\mathrm{x}\left[\begin{array}{ll}{[ } & 0]\end{array}\right.\) & \(=\) & -29 \\
\(\mathrm{x}\left[\begin{array}{ll}{[ } & 1\end{array}\right]\) & \(=\) & -16 \\
\(\mathrm{x}[\) & \(2]\) & \(=\) \\
\(\mathrm{x}[\) & \(3]\) & \(=\)
\end{tabular}
```


### 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 |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64 bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | 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 $A=L U$ (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) $n>0$
(b) $0 \leq m u \leq n-1$
$0 \leq \mathrm{ml} \leq \mathrm{n}-1$
(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 <br> satisfied. | Processing is aborted. |
| $4000+i$ | The pivot became 0.0 in the $i$-th process- <br> ing step. <br> $A$ 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, then j is stored in ipvt $[\mathrm{i}-1]$. In addition, since columns ithrough 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 $\mathrm{ml} \times \mathrm{n}$. Therefore, if $\mathrm{n}<2 \mathrm{ml}+\mathrm{mu}+1$, 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 |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64 bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | 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 $A=L U$ (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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 1 | Output | Reciprocal of the condition number |
| 8 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 9 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $n>0$
(b) $0 \leq m u \leq n-1$
$0 \leq \mathrm{ml} \leq \mathrm{n}-1$
(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 <br> satisfied. | Processing is aborted. |
| $4000+i$ | The pivot became 0.0 in the $i$-th process- <br> ing step. <br> $A$ is nearly singular. | Processing is aborted. <br> 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 $\mathrm{j}(\mathrm{i} \leq \mathrm{j})$, then j is stored in ipvt[ $\mathrm{i}-1]$. 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 $\mathrm{ml} \times \mathrm{n}$. Therefore, if $\mathrm{n}<2 \mathrm{ml}+\mathrm{mu}+1$, 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 $\|A\| \cdot\left\|A^{-1}\right\|$, 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 $L U \boldsymbol{x}=\boldsymbol{b}$ 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 ( $\mathrm{a}, \mathrm{lma}, \mathrm{n}, \mathrm{mu}, \mathrm{ml}, \mathrm{b}, \mathrm{ipvt})$;
Single precision:

$$
\text { ierr }=\text { ASL_rbbdls }(\mathrm{a}, \operatorname{lma}, \mathrm{n}, \mathrm{mu}, \mathrm{ml}, \mathrm{~b}, \mathrm{ipvt}) ;
$$

(3) Arguments and Return Value

| D:Double precision real | Z:Double precision complex |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | 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 | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  |  |  | Output | Solution $\boldsymbol{x}$ |
| 7 | ipvt | $\mathrm{I}^{*}$ | n | Input | Pivoting information <br> ipvt[i-1]: Number of row exchanged with row i in the i-th processing step <br> (See Note (c)) |
| 8 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $\mathrm{n}>0$
(b) $0 \leq m u \leq n-1$
$0 \leq \mathrm{ml} \leq \mathrm{n}-1$
(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. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]$ is performed. |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | Processing is aborted. |
| $4000+i$ | $L$ has a 0.0 diagonal element. <br> $i$ is the number of the first 0.0 diagonal <br> 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:

$$
\text { ierr }=\text { ASL_dbbddi }(\mathrm{a}, \mathrm{lma}, \mathrm{n}, \mathrm{mu}, \mathrm{ml}, \mathrm{ipvt}, \text { det }) ;
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbbddi }(\mathrm{a}, \mathrm{lma}, \mathrm{n}, \mathrm{mu}, \mathrm{ml}, \mathrm{ipvt}, \text { det }) ;
$$

(3) Arguments and Return Value

| D:Double precision real | Z:Double precision complex |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 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) $0 \leq m u \leq n-1$

$$
0 \leq \mathrm{ml} \leq \mathrm{n}-1
$$

(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. | $\operatorname{det}[0] \leftarrow \mathrm{a}[0]$ <br> $\operatorname{det}[1] \leftarrow 0.0$ (See Note (d)) |
| 1000 | n was equal to 1. | Processing is aborted. |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. |  |

(6) Notes
(a) The coefficient matrix $A$ must be LU decomposed before using this function. Use any of 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 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:

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

Scaling is performed at this time so that:

$$
1.0 \leq|\operatorname{det}[0]|<10.0
$$

(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 R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | Input | Coefficient matrix $A$ after LU decomposition (See Note (a)) |
| 7 | b | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Input | Constant vector b |
| 8 | x | \{ $\mathrm{D} *\}$ | n | Input | Approximate solution $\boldsymbol{x}$ |
|  |  | R** |  | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 13 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $\mathrm{n}>0$
(b) $0 \leq m u \leq n-1$
$0 \leq \mathrm{ml} \leq \mathrm{n}-1$
(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. | The solution is not improved. |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | Processing is aborted. |
| $4000+i$ | The $i$-th diagonal element of alu was equal <br> to 0.0. | The solution did not converge within the <br> maximum number of iterations. |
| 5000 | Processing is aborted after calculating the <br> itol output value. |  |

(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 $\left\{\begin{array}{l}\text { ASL_dbbdlu } \\ \text { ASL_rbbdlu }\end{array}\right\}$ or 2.13.3 $\left\{\begin{array}{l}\text { ASL_dbbdlc } \\ \text { ASL_rbbdlc }\end{array}\right\}$ 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.

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

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.

$$
\left[\begin{array}{rrrrrrrrrr}
10 & 9 & 8 & 7 & 6 & 0 & 0 & 0 & 0 & 0 \\
9 & 9 & 8 & 7 & 6 & 5 & 0 & 0 & 0 & 0 \\
8 & 8 & 8 & 7 & 6 & 5 & 4 & 0 & 0 & 0 \\
7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 0 & 0 \\
6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 0 \\
0 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\
0 & 0 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\
0 & 0 & 0 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\
0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{array}\right]\left[\begin{array}{c}
X_{1} \\
X_{2} \\
X_{3} \\
X_{4} \\
X_{5} \\
X_{6} \\
X_{7} \\
X_{8} \\
X_{9} \\
X_{10}
\end{array}\right]=\left[\begin{array}{r}
8 \\
7 \\
2 \\
2 \\
4 \\
-2 \\
-2 \\
2 \\
2 \\
0
\end{array}\right]
$$

(b) Input data

Coefficient matrix $\mathrm{a}, \operatorname{lna}=21, \mathrm{n}=10, \mathrm{mu}=4, \mathrm{ml}=4$ 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;
    fp = fopen( "dbbdlx.dat", "r" );
    if( fp == NULL )
    if(
        printf( "file open error\n" );
    }
    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" );
```

```
}
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;
            salj
            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%%.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 **
$\begin{array}{lr}\mathrm{n}= & 10 \\ \mathrm{mu}= & 4 \\ \mathrm{ml}= & 4\end{array}$
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 | $=$ | $-3.78 \mathrm{e}$ |  |
| x | 6 | $=$ | - |  |
| x | 7 |  | -6.29e |  |
| x | 8 | $=$ |  | 1 |
| x [ | $9]$ | $=$ | $4.88 \mathrm{e}-1$ |  |

** Output **
ierr $=0$
Improved Solution


### 2.14 POSITIVE SYMMETRIC BAND MATRIX (SYMMETRIC BAND TYPE)

### 2.14.1 ASL_dbbpsl, ASL_rbbpsl <br> 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:

$$
\text { ierr }=\text { ASL_dbbpsl } \quad(\mathrm{a}, \mathrm{lma}, \mathrm{n}, \mathrm{mb}, \mathrm{~b}) ;
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbbpsl } \quad(\mathrm{a}, \operatorname{lma}, \mathrm{n}, \mathrm{mb}, \mathrm{~b}) ;
$$

(3) Arguments and Return Value

| D:Double precision real R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | Input | Positive symmetric band matrix $A$ (symmetric band type) (See Appendix B) |
|  |  |  |  | Output | Upper triangular matrix $L^{T}$ when $A$ is decomposed into $A=L L^{T}$ (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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  |  |  | Output | Solution $\boldsymbol{x}$ |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $n>0$
(b) $0 \leq m b \leq n-1$
(c) $\mathrm{mb}+1 \leq \operatorname{lma}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :---: | :---: |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1 . | $\begin{aligned} & \mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]} \text { and } \\ & \mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0] \text { are performed } . \end{aligned}$ |
| 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 $L^{T}$ decomposition of coefficient matrix $A$. <br> $A$ 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 $\left\{\begin{array}{c}\text { ASL_dbbpls } \\ \text { ASL_rbbpls }\end{array}\right\}$ the required number of times varying only the contents of $b$. This enables you to eliminate unnecessary calculations by performing the $L^{T}$ 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.

\[

\]

Storage status within array a[lma $\times \mathrm{k}$ ]


## Remarks

a. Input time values of elements indicated by asterisks (*) are guaranteed.
b. mb is the band width.
c. $\quad \operatorname{lma} \geq \mathrm{mb}+1$ and $\mathrm{k} \geq \mathrm{n}$ must hold.

Figure 2-14 Storage Status of Matrix $L^{T}$

## (7) Example

(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
10 & -2 & 1 & 0 \\
-2 & 9 & -1 & 2 \\
1 & -1 & 8 & -3 \\
0 & 2 & -3 & 7
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
72 \\
9 \\
62 \\
-4
\end{array}\right]
$$

(b) Input data

Coefficient matrix $\mathrm{a}, \operatorname{lma}=11, \mathrm{n}=4, \mathrm{mb}=2$ 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, ch j ;, ,
    char \(\mathrm{SP}=\) '
    \(\mathrm{fp}=\) fopen( "dbbpsl.dat", "r");
    if ( \(f p==\) NULL \()\)
    \{
        printf( "file open error\n" );
    \}
    printf( " \(\quad * * *\) ASL_dbbpsl \(* * * \backslash \mathrm{n} ")\);
    printf ( "\n \(\quad * *\) Input \(* * \backslash \mathrm{n} \backslash \mathrm{n} ")\);
    fscanf( fp, "\%d", \&ma );
    fscanf( fp, "\%d", \&n );
    fscanf( fp, "\%d", \&m );
    \(\mathrm{a}=(\) double \(*)\) malloc \(((\) size_t) \((\operatorname{sizeof}(\) double) \(*(\operatorname{ma*})))\);
    \(\operatorname{if}(\mathrm{a}==\) NULL \()\)
        printf( "no enough memory for array \(a \backslash n "\) )
        return -1 ;
    \}
    wa \(=(\) double \(*)\) malloc \(((\) size_t) \((\operatorname{sizeof}(\operatorname{double}) *(m a * n)))\);
    if (wa == NULL )
        printf( "no enough memory for array wa\n" );
        return -1
    \}
    \(\mathrm{b}=(\) double \(*)\) malloc \(((\) size_t) ( sizeof (double) \(* \mathrm{n}))\);
    \(b=(\) double *
\(\operatorname{if}(\mathrm{b}==\) NULL \()\)
        printf( "no enough memory for array b\n" );
        return -1 ;
    \}
    printf( "\t \(n=\% 6 d \backslash n \backslash t\) Band Width \(=\% 6 d \backslash n ", n, m)\);
    for \((i=0 \quad ; i<n \quad ; i++)\)
    for \((j=0 ; i<n ; i++)\)
        fscanf( fp, "\%lf", \&wa[i+ma*j]);
    for ( \(j=0\); \(j<n ; j++\) ) \{
    if (i-m<=0) \{
    \(\operatorname{lf}(i-m<=0)\)
\(\operatorname{for}(i=0 ; i<=j ; i++)\)
        \(a[i+(m-j)+m a * j]=w a[i+m a * j] ;\)
    \} else \{
    for ( i=j-m ; i<=j ; i++ ) \{
    if \((i>=0) a[i+(m-j)+m a * j]=w a[i+m a * j]\);
    \}
    \}
```

```
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%%.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
```

Coefficient Matrix
$10 \quad \begin{array}{rr} & -2 \\ 9\end{array}$
$\begin{array}{rr}1 & 2 \\ -1 & -3 \\ 8 & 7\end{array}$
Constant Vector
72
9
62
-4
** Output **
ierr $=0$
Solution

| $\mathrm{x}[$ | $0]$ | $=$ | 7 |
| :--- | :--- | :--- | :--- |
| $\mathrm{x}[$ | $1]$ | $=$ | 3 |
| $\mathrm{x}[$ | $2]$ | $=$ | 8 |
| $\mathrm{x}[$ | $3]$ | $=$ | 2 |

### 2.14.2 ASL_dbbpuu, ASL_rbbpuu

## $L^{\mathrm{T}}$ Decomposition of a Positive Symmetric Band Matrix

(1) Function

ASL_dbbpuu or ASL_rbbpuu uses the Cholesky method to perform an $L L^{T}$ decomposition of the positive symmetric band matrix $A$ (symmetric band type).
(2) Usage

Double precision:
ierr $=$ ASL_dbbpuu (a, lma, n, mb);
Single precision:

$$
\text { ierr }=\text { ASL_rbbpuu (a, lma, n, mb); }
$$

(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |
| :--- |
|  <br> Z:Double precision complex <br> C:Single precision complex |
| No. $\mathrm{I}:\left\{\begin{array}{l}\text { Argument and } \\ \text { Return Value }\end{array}\right.$ |
| 1 |

(4) Restrictions
(a) $n>0$
(b) $0 \leq \mathrm{mb} \leq \mathrm{n}-1$
(c) $\mathrm{mb}+1 \leq \operatorname{lma}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |  |  |  |
| :---: | :--- | :--- | :---: | :---: | :---: |
| 0 | Normal termination. |  |  |  |  |
| 1000 | n was equal to 1. | $\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}$ <br> is performed. |  |  |  |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | Processing is aborted. |  |  |  |
| $4000+i$ | A diagonal element became less than or <br> 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

## $L^{T}$ Decomposition and Condition Number of a Positive Symmetric Band Matrix

(1) Function

ASL_dbbpuc or ASL_rbbpuc uses the Cholesky method to perform an $L L^{T}$ 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 R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | Input | Positive symmetric band matrix $A$ (symmetric band type) (See Appendix B) |
|  |  |  |  | Output | Upper triangular matrix $L^{T}$ when $A$ is decomposed into $A=L L^{T}$ (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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 1 | Output | Reciprocal of the condition number |
| 6 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 7 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $\mathrm{n}>0$
(b) $0 \leq m b \leq n-1$
(c) $\mathrm{mb}+1 \leq \mathrm{lma}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. | $\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}$ and <br> cond $\leftarrow 1.0$ are performed. |
| 1000 | n was equal to 1. | Processing is aborted. |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | A diagonal element became less than or <br> 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 $\|A\| \cdot\left\|A^{-1}\right\|$, an approximate value is obtained by this function.

### 2.14.4 ASL_dbbpls, ASL_rbbpls <br> Simultaneous Linear Equations (LL ${ }^{T}$-Decomposed Positive Symmetric Band Matrix)

(1) Function

ASL_dbbpls or ASL_rbbpls solves the simultaneous linear equations $L L^{T} \boldsymbol{x}=\boldsymbol{b}$ having the positive symmetric band matrix $A$ (symmetric band type) which has been $L L^{T}$ decomposed by the Cholesky method as coefficient matrix.
(2) Usage

Double precision:

$$
\text { ierr }=\text { ASL_dbbpls }(\mathrm{a}, \operatorname{lma}, \mathrm{n}, \mathrm{mb}, \mathrm{~b}) ;
$$

Single precision:
ierr $=$ ASL_rbbpls (a, lma, n, mb, b);
(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |
| :--- |
| Z:Double precision complex <br> C:Single precision complex |
| No. |
| Argument and <br> Return Value |
| 1 |

(4) Restrictions
(a) $\mathrm{n}>0$
(b) $0 \leq \mathrm{mb} \leq \mathrm{n}-1$
(c) $\mathrm{mb}+1 \leq \operatorname{lma}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]^{2}$ is performed. |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | Processing is aborted. |
| $4000+i$ | $L^{T}$ has a diagonal element that is less <br> than or equal to 0.0. <br> $i$ is the number of the first diagonal ele- <br> ment that is less than or equal to 0.0. |  |

(6) Notes
(a) The coefficient matrix $A$ must be $L^{T}$ decomposed before using this function. Normally, you should decompose matrix $A$ by calling the 2.14.2 $\left\{\begin{array}{c}\text { ASL_dbbpuu } \\ \text { ASL_rbbpuu }\end{array}\right\}$ function. However, if you also want to obtain the condition number, you should use 2.14.3 $\left\{\begin{array}{l}\text { ASL_dbbpuc } \\ \text { ASL_rbbpuc }\end{array}\right\}$. In addition, if you have already used 2.14.1 $\left\{\begin{array}{c}\text { ASL_dbbpsl } \\ \text { ASL_rbbpsl }\end{array}\right\}$ to solve simultaneous linear equations having the same coefficient matrix $A$, you can use the $\mathrm{LL}^{\mathrm{T}}$ 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 $L L^{\mathrm{T}}$ decomposed by the Cholesky method.
(2) Usage

Double precision:
ierr = ASL_dbbpdi (a, lma, n, mb, det);
Single precision:

$$
\text { ierr }=\text { ASL_rbbpdi }(\mathrm{a}, \operatorname{lma}, \mathrm{n}, \mathrm{mb}, \text { det }) ;
$$

(3) Arguments and Return Value

| D:Double precision real | Z:Double precision complex |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lma} \times \mathrm{n}$ | Input | Upper triangular matrix $L^{T}$ after $\mathrm{LL}^{T}$ 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 2 | Output | Determinant of matrix $A$ (See Note (c)) |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

## (4) Restrictions

(a) $n>0$
(b) $0 \leq m b \leq n-1$
(c) $\mathrm{mb}+1 \leq \operatorname{lma}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\operatorname{det}[0] \leftarrow \mathrm{a}[0]$ <br> $\operatorname{det}[1] \leftarrow 0.0$ (See Note (c)) |
| 3000 | Restriction (a), (b) or (c) was not <br> satisfied. | Processing is aborted. |

## (6) Notes

(a) The coefficient matrix $A$ must be $L^{T}$ decomposed before using this function. Use any of the 2.14.1 $\left\{\begin{array}{c}\text { ASL_dbbpsl } \\ \text { ASL_rbbpsl }\end{array}\right\}, 2.14 .2\left\{\begin{array}{c}\text { ASL_dbbpuu } \\ \text { ASL_rbbpuu }\end{array}\right\}, 2.14 .3\left\{\begin{array}{c}\text { ASL_dbbpuc } \\ \text { ASL_rbbpuc }\end{array}\right\}$ 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:

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

Scaling is performed at this time so that:

$$
1.0 \leq|\operatorname{det}[0]|<10.0
$$

(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 <br> R:Single precision real |
| :--- |
| Z:Double precision complex <br> C:Single precision complex |
| No.Argument and <br> Return Value |
| 1 |

(4) Restrictions
(a) $\mathrm{n}>0$
(b) $0 \leq m b \leq n-1$
(c) $\mathrm{mb}+1 \leq \operatorname{lma}$
(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 <br> satisfied. | Processing is aborted. |
| $4000+i$ | The $i$-th diagonal element of array all was <br> less than or equal to 0.0. |  |
| 5000 | The solution did not converge within the <br> maximum number of iterations. | Processing is aborted after calculating the <br> 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 $\left\{\begin{array}{l}\text { ASL_dbbpuc } \\ \text { ASL_rbbpuc }\end{array}\right\}$ 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.

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

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 <br> 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 |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | sdl | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | 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 | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | 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 | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  |  |  | Output | Solution $\boldsymbol{x}$ |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $n>0$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]$ is performed. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| $4000+i$ | The pivot became 0.0 in the $i$-th process- <br> ing step. <br> $A$ is nearly singular. |  |

(6) Notes
(a) This function performs partial pivoting.
(7) Example
(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{llll}
2 & 3 & 0 & 0 \\
1 & 2 & 3 & 0 \\
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 2
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
8 \\
14 \\
20 \\
11
\end{array}\right]
$$

(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 SPP=, ,;
    FILE *fp;
    fp = fopen( "dbtdsl.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
    }
    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 \() * \mathrm{nn}))\);
if ( a3 == NULL )
    printf( "no enough memory for array a3\n" );
    return -1;
\}
\(\mathrm{b}=\left(\right.\) double \(\left.{ }^{*}\right)\) malloc ( \((\) size_t) \((\) sizeof (double) \(* \mathrm{nn}))\);
if \((b==\) NULL \()\)
        printf( "no enough memory for array b\n" );
\}
printf( "\tn = \%6d ", nn );
for ( i=0 ; i<nn ; i++ )
    for ( \(\mathrm{j}=0\); \(\mathrm{j}<\mathrm{nn}\); \(\mathrm{j}++\) )
    \{
        fscanf( fp, "\%lf", \&wa[i+nn*j] );
    \}
\}
for ( i=0 ; i<nn ; i++ )
    a2[i]=wa[i+(nn)*i];
\(\underset{\{ }{\}}\) or ( \(i=0\); \(i<n n-1\); i++ \()\)
    \(\mathrm{a} 1[\mathrm{i}+1]=\mathrm{wa}[\mathrm{i}+1+(\mathrm{nn}) * \mathrm{i}] ;\)
\(\mathrm{a} 3[\mathrm{i}]=\mathrm{wa}[\mathrm{i}+(\mathrm{nn}) *(\mathrm{i}+1)] ;\)
\}
printf( "\n\n\tCoefficient Matrix \(\backslash n \backslash n ")\);
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 ( " \(\backslash t \% 8.3 \mathrm{~g} \% 8.3 \mathrm{~g} \% 8.3 \mathrm{~g} \backslash \mathrm{n} ", \quad \mathrm{a}\) [0], a3[1], a3[2] );
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_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 \(x[\% 6 d]=\% 8.3 g \backslash n ", i, b[i]) ;\)
\}
free( a1 );
free( wa )
free( a2 )
free( a3 )
free( b );
return 0;
```

(d) Output results

```
*** ASL_dbtdsl ***
    ** Input **
n = 4
Coefficient Matrix
\begin{tabular}{llll} 
& 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 &
\end{tabular}
```

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 <br> Simultaneous Linear Equations (Positive Symmetric Tridiagonal Matrix)

## (1) Function

ASL_dbtpsl or ASL_rbtpsl uses the Gauss method to solve the simultaneous linear equations $\boldsymbol{A x}=\boldsymbol{b}$ having a positive symmetric tridiagonal matrix $A$ (vector type) as coefficient matrix.
(2) Usage

Double precision:

$$
\text { ierr }=\text { ASL_dbtpsl }(\mathrm{d}, \mathrm{sd}, \mathrm{n}, \mathrm{~b}) ;
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbtpsl }(\mathrm{d}, \mathrm{sd}, \mathrm{n}, \mathrm{~b}) ;
$$

(3) Arguments and Return Value

| D:Double precision real R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | d | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 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 | $\underline{\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  |  |  | Output | Solution $\boldsymbol{x}$ |
| 5 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $\mathrm{n}>0$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |  |  |
| :---: | :--- | :--- | :---: | :---: |
| 0 | Normal termination. |  |  |  |
| 1000 | n was equal to 1. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]$ is performed. |  |  |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |  |  |
| 4000 | The diagonal component became 0.0 dur- <br> ing processing. <br> $A$ 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.

$$
\left[\begin{array}{rrrr}
-2 & 1 & 0 & 0 \\
1 & -2 & 1 & 0 \\
0 & 1 & -2 & 1 \\
0 & 0 & 1 & -2
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
-1 \\
0 \\
0 \\
0
\end{array}\right]
$$

(b) Input data

Diagonal component d, subdiagonal component $\mathrm{sd}, \mathrm{n}=4$ 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;
$\mathrm{fp}=\mathrm{fopen}($ "dbtpsl.dat", "r");
if ( $f p==$ NULL $)$
\{
printf( "file open error\n" );

```
}
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;
}
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++ )
a 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
    lr-2 -2 -2 
Constant Vector
    -1
** Output **
ierr = 0
Solution
\begin{tabular}{llll}
\(\mathrm{x}[\) & \(0]\) & \(=\) & 0.8 \\
\(\mathrm{x}[\) & \(1]\) & \(=\) & 0.6 \\
\(\mathrm{x}[\) & \(2]\) & \(=\) & 0.4 \\
\(\mathrm{x}[\) & \(3]\) & \(=\) & 0.2
\end{tabular}
```


### 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 <br> R:Single precision real | Z:Double precision complex <br> C:Single precision complex |
| :--- | :--- | I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long 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 $A$ (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 <br> Contents | Work | Work area (See Note (a)) <br> Size: $3 \times\left\lfloor\log _{2}(\mathrm{n})\right\rfloor+1$ |
| 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 |  |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | $\mathrm{n}=1$ | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]$ |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| 4000 | a is nearly singular. |  |

(6) Notes
(a) $\left\lfloor\log _{2}(\mathrm{n})\right\rfloor$ 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

$$
\left[\begin{array}{llll}
6 & 2 & 0 & 0 \\
1 & 6 & 2 & 0 \\
0 & 1 & 6 & 2 \\
0 & 0 & 1 & 6
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{l}
10 \\
19 \\
28 \\
27
\end{array}\right]
$$

(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;
    ouble *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" );
    }
    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
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" );
}
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] );
}
{
    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( " % %.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 **
$\mathrm{n}=$
Coefficient Matrix

|  | 1 | 1 | 1 |
| :--- | :--- | :--- | :--- |
| 6 | 6 | 6 | 6 |
| 2 | 2 | 2 |  |

Constant Vector
10
19
28 18
27
** Output **
ierr $=0$
Solution

| $\mathrm{x}[$ | 0 | $]=$ | 1 |
| :--- | :--- | :--- | :--- |
| $\mathrm{x}[$ | 1 | $=$ | 2 |
| $\mathrm{x}[$ | 2 | $=$ | 3 |
| $\mathrm{x}[$ | 3 | $=$ | 4 |

### 2.16.2 ASL_wbtdls <br> 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}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> 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 <br> Contents | Input | Reduction operation information (See Notes (a) and (b)) <br> Size: $3 \times\left\lfloor\log _{2}(\mathrm{n})\right\rfloor+1$ |
| 7 | w1 | D* | $4 \times \mathrm{n}$ | Input | Reduction operation information (See Note (a)) |
| 8 | 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 | $\mathrm{n}=1$ | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]$ |
| 3000 | Restriction (a) 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 2.16.1 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) $\left\lfloor\log _{2}(\mathrm{n})\right\rfloor$ 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 simultaneous linear equations $A \boldsymbol{x}=\boldsymbol{b}_{1}$ and $A \boldsymbol{y}=\boldsymbol{b}_{2}$ with unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$ where,

$$
A=\left[\begin{array}{llll}
6 & 2 & 0 & 0 \\
1 & 6 & 2 & 0 \\
0 & 1 & 6 & 2 \\
0 & 0 & 1 & 6
\end{array}\right], \boldsymbol{b}_{1}=\left[\begin{array}{l}
10 \\
19 \\
28 \\
27
\end{array}\right], \boldsymbol{b}_{2}=\left[\begin{array}{c}
30 \\
26 \\
17 \\
8
\end{array}\right]
$$

(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>
{nt 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" );
}
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" );
}
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] );
}
{
    fscanf( fp, "%lf", &a2[i] );
}
for( i=0 ; i<n-1 ; i++ )
    fscanf( fp, "%lf", &a3[i] );
printf( "\t%8c %%.3g %8.3g %8.3g\n", SP,a1[1],a1[2],a1[3] );
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\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
```

| 10 | 30 |
| ---: | ---: |
| 19 | 26 |
| 28 | 17 |
| 27 | 8 |


| $y[$ | 0 | $]$ | $=$ |
| :--- | :--- | :--- | :--- |
| $y[$ | 1 | $]$ | 4 |
| $y[$ | 2 | $]$ | 3 |
| $y[$ | 3 | $=$ | 2 |

```
```

*** ASL_wbtdls ***

```
*** ASL_wbtdls ***
** Input **
** Input **
n= 4
n= 4
Coefficient Matrix
Coefficient Matrix
    llll
    llll
Constant Vector
Constant Vector
** Output **
** Output **
ierr = 0
ierr = 0
Solution x
Solution x
    < [ [ 0 ] = 
    < [ [ 0 ] = 
Solution y
```

Solution y

```

\subsection*{2.17 FIXED COEFFICIENT REAL TRIDIAGONAL MATRIX (SCALAR TYPE)}

\subsection*{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 \(\boldsymbol{A} \boldsymbol{x}=\boldsymbol{b}\) having the fixed coefficient real tridiagonal matrix a (scalar type) as coefficient matrix.
(2) Usage

Double precision:
\[
\text { ierr }=\text { ASL_wbtcsl }(\& d, \& s d, \text { n, b, isw, iw, w1); }
\]

Single precision:
Nothing
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \\
\begin{tabular}{|c|c|c|l|c|l|} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} \\
\hline No. \\
\begin{tabular}{c} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(n>0\)
(b) isw \(\in\{1,2,3,4\}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c|}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & \(\mathrm{n}=1\) & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]\) \\
\hline 3000 & Restriction (a) or (b) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 4000 & \(A\) is nearly singular. & \\
\hline
\end{tabular}
(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 \(=1\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & & \mathrm{sd} & & & & 0 \\
\mathrm{sd} & \mathrm{~d} & \text { sd } & & & \\
& & \mathrm{sd} & \mathrm{~d} & \text { sd } & & \\
& & & \cdot & \cdot & \cdot & \\
& & & & \cdot & \cdot & \\
& 0 & & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & & \text { sd } & \mathrm{d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=2\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & \mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{~d} & \text { sd } & & & \\
& \mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & \\
& & \cdot & \cdot & \cdot & \\
& & & \cdot & \cdot & \cdot \\
& 0 & & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & 2 \times \mathrm{sd} & \mathrm{~d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=3\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & 2 \times \mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & & \\
& \mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & \\
& & \cdot & \cdot & \cdot & \\
& & & \cdot & \cdot & \cdot \\
& 0 & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & \text { sd } & \mathrm{d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=4\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & 2 \times \mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & & \\
& \mathrm{sd} & \mathrm{~d} & \text { sd } & & \\
& & \cdot & \cdot & \cdot & \\
& & & \cdot & \cdot & \cdot \\
& 0 & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & 2 \times \mathrm{sd} & \mathrm{~d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

Coefficient matrices of the types shown above appear when discretizing the Dirichlet or Neumann boundary value problem.
(b) \(\left\lfloor\log _{2}(\mathrm{n})\right\rfloor\) 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
\[
\left[\begin{array}{llll}
6 & 2 & 0 & 0 \\
2 & 6 & 2 & 0 \\
0 & 2 & 6 & 2 \\
0 & 0 & 2 & 6
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
8 \\
10 \\
10 \\
8
\end{array}\right]
\]
(b) Input data

Diagonal components d, subdiagonal components sd , \(\mathrm{n}=4\), 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
double *Wk
int i,nn;,log2n,nwk,niwk;
int i,nn,log2n
Char SP='
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" );
}
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" );
}
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],a2[3] );
printf( "\n\tConstant Vector\n\n" );
for( i=0 ; i<n ; i++ )
fscanf( fp, "%lf", \&b[i] );
printf( "\t%%.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;

```

\section*{\}}
(d) Output results
```

*** ASL_wbtcsl ***
** Input **
n=

```
Coefficient Matrix
    \(\begin{array}{llll} & 2 & 2 & 2 \\ 6 & 6 & 6 & 6 \\ 2 & 2 & 2 & \end{array}\)
Constant Vector
    8
10
10
8
    ** Output **
ierr \(=0\)
Solution
    \(\begin{array}{lll}\mathrm{x}[ & 0 & ]= \\ \mathrm{x}[ & 1 & = \\ \mathrm{x}[ & 2 & 1 \\ \mathrm{x}[ & 3 & = \\ \end{array}\)

\subsection*{2.17.2 ASL_wbtcls}

\section*{Simultaneous Linear Equations (Fixed Coefficient Real Tridiagonal Matrix after Reduction Operations)}

\section*{(1) Function}

ASL_wbtcls uses the cyclic reduction method to solve the simultaneous linear equations \(\boldsymbol{A} \boldsymbol{x}=\boldsymbol{b}\) having the fixed coefficient real tridiagonal matrix \(A\) (scalar type) after reduction operations have been performed as coefficient matrix.

\section*{(2) Usage}

Double precision:
ierr \(=\) ASL_wbtcls (d, sd, n, b, isw, iw, w1);
Single precision:
Nothing

\section*{(3) Arguments and Return Value}
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular} \\
\(\left.\begin{array}{|c|c|c|c|c|l|}\text { I: }\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right.\end{array}\right\}\) \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}

\section*{(4) Restrictions}
(a) \(\mathrm{n}>0\)
(b) isw \(\in\{1,2,3,4\}\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & Processing \\
\hline 0 & Normal termination. & \\
\hline 1000 & \(\mathrm{n}=1\) & \(\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{d}[0]\) \\
\hline 3000 & Restriction (a) or (b) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 4000 & \(A\) is nearly singular. (Only when \(\mathrm{n}=1\) ) & \\
\hline
\end{tabular}
(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 \(=1\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & & \mathrm{sd} & & & & 0 \\
\text { sd } & \mathrm{d} & \text { sd } & & & \\
& \mathrm{sd} & \mathrm{~d} & \text { sd } & & \\
& & & \cdot & \cdot & \cdot & \\
& & & & \cdot & \cdot & \cdot \\
& 0 & & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & & \text { sd } & \mathrm{d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=2\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & \mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{~d} & \text { sd } & & & \\
& \mathrm{sd} & \mathrm{~d} & \text { sd } & & \\
& & & \cdot & \cdot & \cdot & \\
& & & & \cdot & \cdot & \cdot \\
& 0 & & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & 2 \times \mathrm{sd} & \mathrm{~d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=3\)
\[
\left[\begin{array}{lllllll}
\mathrm{d} & 2 \times \mathrm{sd} & & & & 0 & \\
\mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & & \\
& \mathrm{sd} & \mathrm{~d} & \mathrm{sd} & & \\
& & \cdot & \cdot & \cdot & \\
& & & \cdot & \cdot & \cdot \\
& 0 & & \cdot & \mathrm{~d} & \mathrm{sd} \\
& & & & \mathrm{sd} & \mathrm{~d}
\end{array}\right], \mathrm{d} \neq 0, \mathrm{sd} \neq 0
\]

For isw \(=4\)


Coefficient matrices of the types shown above appear when discretizing the Dirichlet or Neumann boundary value problem.
(c) \(\left\lfloor\log _{2}(\mathrm{n})\right\rfloor\) 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.

\section*{(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,
\[
A=\left[\begin{array}{cccc}
6 & 2 & 0 & 0 \\
2 & 6 & 2 & 0 \\
0 & 2 & 6 & 2 \\
0 & 0 & 2 & 6
\end{array}\right], \boldsymbol{b}_{1}=\left[\begin{array}{c}
8 \\
10 \\
10 \\
8
\end{array}\right], \boldsymbol{b}_{2}=\left[\begin{array}{l}
10 \\
20 \\
30 \\
30
\end{array}\right]
\]
(b) Input data

Diagonal components d, subdiagonal components sd, \(\mathrm{n}=4\), 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 *isw
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 \(=\left(\right.\) double \(\left.{ }^{*}\right)\) malloc \(((\) size_t)( sizeof(double) * n ));
    b1 \((\mathrm{b} 1 \mathrm{double}\) ( \(=\) 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 \(=n+3 * \log 2 n+2\);
    wk = ( double * ) malloc((size_t) ( sizeof(double) * nwk ));
    if ( wk == NULL )
        printf( "no enough memory for array wk\n" );
    \(\}\)
    niwk=3*log2n+1;
    iw \(=(\) int *) malloc ((size_t) ( sizeof(int) * niwk ));
    \(\operatorname{if}^{f}(\) iw \(==\) NULL \()\)
        printf( "no enough memory for array iw in " );
        return -1;
\}
printf( "\n\tn \(=\% 6 d \backslash n ", n)\);
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 \(\backslash n \backslash n ")\);
for ( i=0 ; i<n ; i++ )
        fscanf( fp, "\%lf", \&b1[i] );
\(\underset{\{ }{\text { for }(i=0 ~ ; ~ i<n ~ ; ~ i++~) ~}\)
    fscanf( fp, "\%lf", \&b2[i] );
\}
\(\underset{\{ }{\text { for }(i=0 ~ ; ~ i<n ~ ; ~ i++~) ~}\)
    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 \(x \backslash n \backslash n ")\);
for ( i=0 ; i<n ; i++ )
    printf( "\t \(x[\% 6 d \quad]=\% 8.3 \mathrm{~g} \backslash \mathrm{n} ", \mathrm{i}, \mathrm{b} 1[\mathrm{i}])\);
\}
printf( "\n\tSolution \(y \backslash n \backslash n ")\);
for ( i=0 ; i<n ; i++ )
    printf( "\t y[\%6d ] = \%8.3g\n", i,b2[i] );
\}
free ( b1 );
free ( b2 );
free ( b2 );
free( iw );
free( wk );
return 0;
\(\}\)
(d) Output results
```

*** ASL_wbtcls ***
** Input **
n = 4
Coefficient Matrix
6

```

2


\subsection*{2.18 VANDERMONDE MATRIX AND TOEPLITZ MATRIX}

\subsection*{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.
\[
R=\left[\begin{array}{cccccc}
r_{0} & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\
r_{1} & r_{0} & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\
\vdots & \vdots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \vdots & \vdots \\
r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_{0} & r_{-1} \\
r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_{1} & r_{0}
\end{array}\right]
\]

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 ( \(\mathrm{r}, \mathrm{n}, \mathrm{b}, \mathrm{x}, \mathrm{w}, \mathrm{isw})\);
Single precision:
ierr \(=\) ASL_rbtosl ( \(\mathrm{r}, \mathrm{n}, \mathrm{b}, \mathrm{x}, \mathrm{w}\), isw);
(3) Arguments and Return Value
\(\begin{array}{ll}\text { D:Double precision real } & \text { Z:Double precision complex } \\ \text { R:Single precision real } & \text { C:Single precision complex }\end{array} \quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|c|c|c|}
\hline No. & Argument and Return Value & Type & Size & \begin{tabular}{l}
Input/ \\
Output
\end{tabular} & Contents \\
\hline 1 & r & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times \mathrm{n}-1\) & Input & Components \(r_{k}(k=-n+1,-n+2, \cdots, n-1)\) of Toeplitz matrix \(R\) \\
\hline 2 & n & I & 1 & Input & Order of matrix \(R\) \\
\hline 3 & b & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline 4 & x & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & n & Output & Solution vector \(\boldsymbol{x}\) \\
\hline 5 & w & \(\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}\) & \(2 \times \mathrm{n}\) & Work & Work area \\
\hline 6 & isw & I & 1 & Input & \begin{tabular}{l}
Processing switch \\
1: Solve \(R \boldsymbol{x}=\boldsymbol{b}\) \\
2: Solve \(R^{T} \boldsymbol{x}=\boldsymbol{b}\)
\end{tabular} \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) isw \(\in\{1,2\}\)
(b) \(\mathrm{n}>0\)
(c) \(\mathrm{r}[\mathrm{n}-1] \neq 0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{2}{c}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \(\mathrm{x}[0] \leftarrow \mathrm{b}[0] / \mathrm{r}[\mathrm{n}-1]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 3010 & Restriction (b) was not satisfied. & \multirow{3}{*}{} \\
\hline 3020 & Restriction (c) was not satisfied. & \multirow{3}{*}{} \\
\hline 4000 & The divisor \(x^{(d e)}\) was zero. & \\
\hline 4010 & The divisor \(g^{(d e)}\) was zero. & \\
\hline
\end{tabular}
(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 and calculation efficiency. However, the solution may not be obtained theoretically even if the matrix is regular. In particular, if \(x^{(d e)}\) or \(g^{(d e)}\), 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".)

\section*{(7) Example}
(a) ProblemSolve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
r_{0} & r_{-1} & r_{-2} & r_{-3} \\
r_{1} & r_{0} & r_{-1} & r_{-2} \\
r_{2} & r_{1} & r_{0} & r_{-1} \\
r_{3} & r_{2} & r_{1} & r_{0}
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{c}
b_{1} \\
b_{2} \\
b_{3} \\
b_{4}
\end{array}\right]
\]
(b) Input data

Array \(\mathrm{r}=\left\{\mathrm{r}_{-3}, \mathrm{r}_{-2}, \mathrm{r}_{-1}, \mathrm{r}_{0}, \mathrm{r}_{1}, \mathrm{r}_{2}, \mathrm{r}_{3}\right\}\) in which matrix \(R\) components are stored, \(\mathrm{n}=4\), isw \(=1\) and constant vector \(\boldsymbol{b}\).
Note The same problem can be solved by storing matrix \(R\) components as \(\mathrm{r}=\left\{\mathrm{r}_{3}, \mathrm{r}_{2}, \mathrm{r}_{1}, \mathrm{r}_{0}, \mathrm{r}_{-1}, \mathrm{r}_{-2}, \mathrm{r}_{-3}\right\}\) 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 isw;
int i,j;
int i,j;
intLE *fp;
fp = fopen( "dbtosl.dat", "r" );
if( fp == NULL )
{
printf( "file open error\n" );
}
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}=(\mathrm{ 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 );
ree( b )
free( x );
return 0;

```
(d) Output results
```

*** ASL_dbtosl ***
** Input **
isw = 1 n = 4
Coefficient Matrix
1
Constant Vector
** Output **
ierr = 0
Solution

| 1 | 1 | 1 | 1 |
| :--- | :--- | :--- | :--- |

```

\subsection*{2.18.2 ASL_dbtssl, ASL_rbtssl}

Simultaneous Linear Equations (Symmetric Toeplitz Matrix)

\section*{(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.
\[
R=\left[\begin{array}{cccccc}
r_{0} & r_{1} & r_{2} & \cdots & r_{n-2} & r_{n-1} \\
r_{1} & r_{0} & r_{1} & \cdots & r_{n-3} & r_{n-2} \\
\vdots & \vdots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \vdots & \vdots \\
r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_{0} & r_{1} \\
r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_{1} & r_{0}
\end{array}\right]
\]

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:
\[
\text { ierr }=\text { ASL_dbtssl }(\mathrm{r}, \mathrm{n}, \mathrm{~b}, \mathrm{x}, \mathrm{w}) ;
\]

Single precision:
ierr \(=\) ASL_rbtssl (r, n, b, x, w);
(3) Arguments and Return Value
\begin{tabular}{l}
\begin{tabular}{l} 
D:Double precision real \\
R:Single precision real
\end{tabular} \\
\begin{tabular}{|c|c|c|l|l|l|}
\multicolumn{1}{l}{\begin{tabular}{l} 
Z:Double precision complex \\
C:Single precision complex
\end{tabular}} & I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\end{tabular} \\
\hline No. \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} \\
\hline 1
\end{tabular}
(4) Restrictions
(a) \(\mathrm{n}>0\)
(b) \(r[0] \neq 0\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{1}{c}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & n was equal to 1 & \(\mathrm{x}[0] \leftarrow \mathrm{b}[0] / \mathrm{r}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 3010 & Restriction (b) was not satisfied. & \\
\hline 4000 & The divisor \(x^{(d e)}\) was zero. & \\
\hline
\end{tabular}
(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 \(x^{(d e)}\), 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").

\section*{(7) Example}
(a) ProblemSolve the following simultaneous linear equations.
\[
\left[\begin{array}{llll}
r_{0} & r_{1} & r_{2} & r_{3} \\
r_{1} & r_{0} & r_{1} & r_{2} \\
r_{2} & r_{1} & r_{0} & r_{1} \\
r_{3} & r_{2} & r_{1} & r_{0}
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{l}
b_{1} \\
b_{2} \\
b_{3} \\
b_{4}
\end{array}\right]
\]
(b) Input data

Array \(\mathrm{r}=\left\{\mathrm{r}_{0}, \mathrm{r}_{1}, \mathrm{r}_{2}, \mathrm{r}_{3}\right\}\) in which matrix \(R\) components are stored, \(\mathrm{n}=4\) 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" );
}
printf( " *** ASL_dbtssl ***\n" );
printf( "\n ** Input **\n\n" );
fscanf( fp, "%d", \&n );

```
```

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

```
printf( "\t n = %6d\n", n );
r = ( double * )malloc((size_t)( sizeof(double) * lna ));
r = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( r == NULL )
    if( r == NULL )
    printf( "no enough memory for array r\n" );
    printf( "no enough memory for array r\n" );
    return -1;
    return -1;
}
}
if( b == NULL )
if( b == NULL )
if( b == NULL )
if( b == NULL )
    printf( "no enough memory for array b\n" );
    printf( "no enough memory for array b\n" );
    return -1;
    return -1;
}
}
x = ( double *) malloc((size_t)( sizeof(double) * lna ));
x = ( double *) malloc((size_t)( sizeof(double) * lna ));
if( x == NULL )
if( x == NULL )
    printf( "no enough memory for array x\n" );
    printf( "no enough memory for array x\n" );
    return -1;
    return -1;
}
}
w = ( double *)malloc((size_t)( sizeof(double) * lna ));
w = ( double *)malloc((size_t)( sizeof(double) * lna ));
if( w == NULL )
if( w == NULL )
    printf( "no enough memory for array w\n" );
    printf( "no enough memory for array w\n" );
    return -1;
    return -1;
f for( i=0 ; i<n ; i++ )
f for( i=0 ; i<n ; i++ )
    fscanf( fp, "%lf", &r[i] );
    fscanf( fp, "%lf", &r[i] );
}
}
printf( "\n\tCoefficient Matrix\n\n");
printf( "\n\tCoefficient Matrix\n\n");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    printf( "\t" );
    printf( "\t" );
    for( j=0 ; j<n ; j++ )
    for( j=0 ; j<n ; j++ )
    {
    {
            printf( "%8.3g ", r[abs(i-j)] );
            printf( "%8.3g ", r[abs(i-j)] );
    }
    }
    printf( "\n" );
    printf( "\n" );
}
}
printf( "\n\tConstant Vector\n\n");
printf( "\n\tConstant Vector\n\n");
printf( "\t");
printf( "\t");
for( i=0 ; i<n ; i++ )
for( i=0 ; i<n ; i++ )
    fscanf( fp, "%lf", &b[i] );
    fscanf( fp, "%lf", &b[i] );
    printf( "%8.3g ", b[i] );
    printf( "%8.3g ", b[i] );
}
}
fclose( fp );
fclose( fp );
ierr = ASL_dbtssl(r, n, b, x, w);
ierr = ASL_dbtssl(r, n, b, x, w);
printf( "\n ** Output **\n\n" );
printf( "\n ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\tierr = %6d\n", ierr );
printf( "\n\tSolution\n\n" );
printf( "\n\tSolution\n\n" );
printf( "\t");
printf( "\t");
{or( i=0 ; i<n ; i++ )
{or( i=0 ; i<n ; i++ )
    printf( "%8.3g ", x[i] );
    printf( "%8.3g ", x[i] );
}
}
printf( "\n" );
printf( "\n" );
free( r );
free( r );
free( r );
free( r );
free( b );
free( b );
free( x );
```

free( x );

```
```

return 0;

```
```

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
** Output ** 8 8 8
ierr = 0
Solution
1 1 1
1

```

\subsection*{2.18.3 ASL_dbvmsl, ASL_rbvmsl Simultaneous Linear Equations (Vandermonde Matrix)}

\section*{(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.
\[
V=\left[\begin{array}{cccccc}
1 & v_{1} & v_{1}^{2} & \cdots & v_{1}^{n-2} & v_{1}^{n-1} \\
1 & v_{2} & v_{2}^{2} & \cdots & v_{2}^{n-2} & v_{2}^{n-1} \\
\vdots & \vdots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \vdots & \vdots \\
1 & v_{n-1} & v_{n-1}^{2} & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\
1 & v_{n} & v_{n}^{2} & \cdots & v_{n}^{n-2} & v_{n}^{n-1}
\end{array}\right]
\]

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:
\[
\text { ierr }=\text { ASL_dbvmsl }(\mathrm{v}, \mathrm{n}, \mathrm{~b}, \mathrm{x}, \mathrm{w}, \text { isw })
\]

Single precision:
ierr \(=\) ASL_rbvmsl ( \(\mathrm{v}, \mathrm{n}, \mathrm{b}, \mathrm{x}, \mathrm{w}, \mathrm{isw})\);
(3) Arguments and Return Value
\begin{tabular}{ll} 
D:Double precision real & Z:Double precision complex \\
R:Single precision real & C:Single precision complex
\end{tabular}\(\quad\) I: \(\left\{\begin{array}{l}\text { int as for 32bit Integer } \\
\text { long as for 64bit Integer }\end{array}\right\}\)
\begin{tabular}{|c|c|c|l|c|l|}
\hline No. & \begin{tabular}{l} 
Argument and \\
Return Value
\end{tabular} & Type & Size & \begin{tabular}{l} 
Input/ \\
Output
\end{tabular} & \multicolumn{1}{|c|}{ Contents } \\
\hline 1 & v & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Input & \begin{tabular}{l} 
Components \(v_{k}(k=1,2, \cdots, n)\) of Vander- \\
monde matrix \(V\)
\end{tabular} \\
\hline 2 & n & I & 1 & Input & Order \(n\) of matrix \(V\) \\
\hline 3 & b & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Input & Constant vector \(\boldsymbol{b}\) \\
\hline 4 & x & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Output & Solution vector \(\boldsymbol{x}\) \\
\hline 5 & w & \(\left\{\begin{array}{l}\mathrm{D} * \\
\mathrm{R} *\end{array}\right\}\) & n & Work & Work area (See Note (b)) \\
\hline 6 & isw & I & 1 & Input & \begin{tabular}{l} 
Processing switch \\
\(1:\) Solve \(V \boldsymbol{x}=\boldsymbol{b}\) \\
2: Solve \(V^{T} \boldsymbol{x}=\boldsymbol{b}\)
\end{tabular} \\
\hline 7 & ierr & I & 1 & Output & Error indicator (Return Value) \\
\hline
\end{tabular}
(4) Restrictions
(a) isw \(\in\{1,2\}\)
(b) \(\mathrm{n}>0\)
(c) \(\mathrm{v}[\mathrm{i}-1] \neq 0 \quad(\mathrm{i}=1, \ldots, \mathrm{n})\)
(5) Error indicator (Return Value)
\begin{tabular}{|c|l|l|}
\hline ierr value & \multicolumn{1}{|c|}{ Meaning } & \multicolumn{2}{c}{ Processing } \\
\hline 0 & Normal termination. & \\
\hline 1000 & \(\mathrm{n}=1\) is specified. & \(\mathrm{x}[0] \leftarrow \mathrm{b}[0]\) is performed. \\
\hline 3000 & Restriction (a) was not satisfied. & \multirow{2}{*}{ Processing is aborted. } \\
\hline 3010 & Restriction (b) was not satisfied. & \multirow{3}{*}{} \\
\hline 3020 & Restriction (c) was not satisfied. & \\
\hline 4000 & \begin{tabular}{l} 
A division by zero occurred during an \\
operation.
\end{tabular} & \multicolumn{1}{|l}{} \\
\hline
\end{tabular}
(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.
\[
P(x)=\prod_{k=1}^{n}\left(x-v_{k}\right)=x^{n}+w_{1} x^{n-1}+\cdots+w_{n-1} x+w_{n}
\]

\section*{(7) Example}
(a) ProblemSolve the following simultaneous linear equations.
\[
\left[\begin{array}{cccc}
1 & v_{1} & v_{1}^{2} & v_{1}^{3} \\
1 & v_{2} & v_{2}^{2} & v_{2}^{3} \\
1 & v_{3} & v_{3}^{2} & v_{3}^{3} \\
1 & v_{4} & v_{4}^{2} & v_{4}^{3}
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{l}
b_{1} \\
b_{2} \\
b_{3} \\
b_{4}
\end{array}\right]
\]
(b) Input data

Array \(\mathrm{v}=\left\{\mathrm{v}_{1}, \mathrm{v}_{2}, \mathrm{v}_{3}, \mathrm{v}_{4}\right\}\) in which matrix \(V\) components are stored, \(\mathrm{n}=4\), isw \(=1\) and constant vector 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" );
}
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}=(\mathrm{ double * )malloc((size_t)( sizeof(double) * lna ));
if( v == NULL )
printf( "no enough memory for array v\n" );
return -1;
}
b}=(\mathrm{ 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 ));

```
```

    {
        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 );
    }
(d) Output results

```
```

*** ASL_dbvmsl ***

```
*** ASL_dbvmsl ***
** Input **
** Input **
isw = 1 n = 4
isw = 1 n = 4
Coefficient Matrix
\begin{tabular}{llrr}
1 & 2 & 4 & 8 \\
1 & 3 & 9 & 27 \\
1 & 4 & 16 & 64 \\
1 & 5 & 25 & 125
\end{tabular}
Constant Vector
    ** 15 Output ** 40 85 156
** Output **
ierr = 0
Solution
\begin{tabular}{llll}
1 & 1 & 1 & 1
\end{tabular}
```


### 2.19 REAL UPPER TRIANGULAR MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

### 2.19.1 ASL_dbtusl, ASL_rbtusl <br> Simultaneous Linear Equations (Real Upper Triangular Matrix)

(1) Function

ASL_dbtusl or ASL_rbtusl solves the simultaneous linear equations $A \boldsymbol{x}=\boldsymbol{b}$ having a real upper triangular matrix $A$ (two-dimensional array type) as coefficient matrix.
(2) Usage

Double precision:

$$
\text { ierr }=\text { ASL_dbtusl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbtusl (a, lna, n, b); }
$$

(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{c} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lna\times n}$ | Input | Coefficient matrix $A$ (real upper triangular matrix, two-dimensional array type) |
| 2 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of array a |
| 3 | n | I | 1 | Input | Order of matrix $A$ |
| 4 | b | \{ $\mathrm{D} *\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  | R* $\}$ |  | Output | Solution $\boldsymbol{x}$ |
| 5 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\mathrm{b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]$ is performed. |
| 2100 | There existed the diagonal element which <br> was close to zero in the coefficient matrix | Processing continues. |
|  The result may not be obtained with <br> a good accuracy.  |  |  |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| $4000+i$ | The coefficient matrix $A$ has a 0.0 diago- <br> nal element. <br> $A$ is singular. |  |

(6) Notes

None
(7) Example
(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
1 & 2 & -3 & 4 \\
0 & 4 & -1 & 1 \\
0 & 0 & 5 & -1 \\
0 & 0 & 0 & 8
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
-10 \\
-9 \\
-3 \\
-16
\end{array}\right]
$$

(b) Input data

Coefficient matrix $A$, 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;
    nt 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" );
    }
```

```
b = ( double * )malloc((size_t)( sizeof(double) * nn ));
    \(\mathrm{b}=\left(\mathrm{double}{ }^{*}\right)\)
\(\operatorname{iff}^{*}(\mathrm{~b}==\mathrm{NULL})\)
        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 ( \(\mathrm{j}=0\); \(\mathrm{j}<\mathrm{nn}\); \(\mathrm{j}++\)
            fscanf( fp, "\%lf", \&a[i+na*j]) ;
            printf( "\t\%\%.3g", a[i+na*j] );
        \}
        printf( "\n" )
\}
printf( "\n\tConstant Vector\n" );
for ( \(i=0\); \(i<n n\); 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 \(* * \backslash \mathrm{n} \backslash \mathrm{n} ")\);
printf( "\tierr = \%6d\n", ierr );
printf( "\n\tSolution\n" );
for ( i=0 ; i<nn ; i++ )
    printf( "\t \(x[\% 6 d]=\% 8.3 g \backslash n ", i, b[i]) ;\)
\}
free( a );
return 0 ;
(d) Output results
```

```
*** ASL_dbtusl ***
```

*** ASL_dbtusl ***
** Input **
** Input **
$\mathrm{n}=\quad 4$
$\mathrm{n}=\quad 4$
Coefficient Matrix
Coefficient Matrix
$\begin{array}{rrrr}1 & 2 & -3 & 4 \\ 0 & 4 & -1 & 1 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 8\end{array}$
$\begin{array}{rrrr}1 & 2 & -3 & 4 \\ 0 & 4 & -1 & 1 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 8\end{array}$
Constant Vector
Constant Vector
-10
-9
-10
-9
-9
-3
-16
-9
-3
-16
-16
-16
** Output **
** Output **
ierr $=0$
ierr $=0$
$\begin{array}{rlll}\text { Solution } & & \\ \mathrm{x}\left[\begin{array}{ll}\text { S }\end{array}\right. & 0 & \\ \mathrm{x}\left[\begin{array}{ll}1\end{array}\right. & = & -1 \\ \mathrm{x}[ & 2] & = & -1 \\ \mathrm{x}[ & 3] & = & -2\end{array}$

```
\(\begin{array}{rlll}\text { Solution } & & \\ \mathrm{x}\left[\begin{array}{ll}\text { S }\end{array}\right. & 0 & \\ \mathrm{x}\left[\begin{array}{ll}1\end{array}\right. & = & -1 \\ \mathrm{x}[ & 2] & = & -1 \\ \mathrm{x}[ & 3] & = & -2\end{array}\)
```

\}

### 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 $A$ (twodimensional array type).
(2) Usage

Double precision:

$$
\text { ierr }=\text { ASL_dbtuco }(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \& \operatorname{cond}, \mathrm{w} 1)
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbtuco (a, lna, n, \&cond, w1); }
$$

(3) Arguments and Return Value

| D:Double precision real <br> R:Single precision real |
| :--- |
| Z:Double precision complex <br> C:Single precision complex |
| $\left.\begin{array}{\|c\|c\|c\|c\|}\hline \text { No. } & \begin{array}{l}\text { Argument and } \\ \text { Return Value }\end{array} & \text { Type } & \text { Size } \\ \text { int as for 32bit Integer } \\ \text { long as for 64bit Integer }\end{array}\right\}$ |
| 1 |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | cond $\leftarrow 1.0$ is performed. |
| 2100 | There existed the diagonal element which <br> was close to zero in the coefficient matrix <br> $A$. The result may not be obtained with <br> a good accuracy. | Processing continues. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| $4000+i$ | Matrix $A$ has a 0.0 diagonal element. <br> $i$ is the number of the first 0.0 diagonal <br> element. |  |

(6) Notes
(a) Although the condition number is defined by $\|A\| \cdot\left\|A^{-1}\right\|$, 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:

$$
\text { ierr }=\text { ASL_dbtudi }(\mathrm{a}, \ln \mathrm{a}, \mathrm{n}, \text { det, isw }) ;
$$

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 |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | a | $\left\{\begin{array}{l} \mathrm{D} * \\ \mathrm{R} * \end{array}\right\}$ | $\operatorname{lna} \times \mathrm{n}$ | Input | Real upper triangular matrix $A$ (two-dimensional array type) |
|  |  |  |  | Output | inverse matrix of matrix $A$ (See Note (a)) |
| 2 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of array a |
| 3 | n | I | 1 | Input | Order of matrix $A$ |
| 4 | det | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 2 | Output | Determinant of matrix $A($ (See Note (b)) |
| 5 | isw | I | 1 | Input | Processing switch <br> isw $>0$ :Obtain determinant. <br> isw $=0$ :Obtain determinant and inverse matrix. <br> isw $<0$ :Obtain inverse matrix. |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\operatorname{det}[0] \leftarrow \mathrm{a}[0]$ |
|  |  | $\operatorname{det}[1] \leftarrow 0.0$ |
|  |  | $\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]$ 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 $A^{-1}$
$\left[\begin{array}{ccccc}\tilde{a}_{1,1} & \tilde{a}_{1,2} & \tilde{a}_{1,3} & \cdots & \tilde{a}_{1,5} \\ 0.0 & \tilde{a}_{2,2} & \tilde{a}_{2,3} & \cdots & \tilde{a}_{2,5} \\ 0.0 & 0.0 & \tilde{a}_{3,3} & \cdots & \tilde{a}_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & \tilde{a}_{5,5} \\ & & \Downarrow & & \end{array}\right.$


## Remarks

a. $\quad \operatorname{lna} \geq \mathrm{n}$ and $\mathrm{n} \leq \mathrm{k}$ 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:

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

Scaling is performed at this time so that:

$$
1.0 \leq|\operatorname{det}[0]|<10.0
$$

(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 $A X=B$ 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:

$$
\text { ierr }=\text { ASL_dbtlsl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbtlsl }(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \mathrm{~b}) ;
$$

(3) Arguments and Return Value

| D:Double precision real R:Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\operatorname{lna\times n}$ | Input | Coefficient matrix $A$ (real lower triangular matrix, two-dimensional array type) |
| 2 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of array a |
| 3 | n | I | 1 | Input | Order of matrix $A$ |
| 4 | b | \{ $\mathrm{D} *\}$ | n | Input | Constant vector $\boldsymbol{b}$ |
|  |  | (R* $\}$ |  | Output | Solution $\boldsymbol{x}$ |
| 5 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$

## (5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\mathrm{~b}[0] \leftarrow \mathrm{b}[0] / \mathrm{a}[0]$ is performed. |
| 2100 | There existed the diagonal element which <br> was close to zero in the coefficient matrix <br> $A$. The result may not be obtained with <br> a good accuracy. | Processing continues. <br> 3000 |
| $4000+i$ | Restriction (a) was not satisfied. <br> The coefficient matrix $A$ has a 0.0 diago- <br> nal element. <br> $i$ is the number of the first 0.0 diagonal <br> element. <br> The matrix $A$ is singular. |  |

(6) Notes

None
(7) Example
(a) Problem

Solve the following simultaneous linear equations.

$$
\left[\begin{array}{rrrr}
5 & 0 & 0 & 0 \\
-1 & 4 & 0 & 0 \\
2 & 1 & 2 & 0 \\
3 & 2 & 7 & 10
\end{array}\right]\left[\begin{array}{l}
x_{1} \\
x_{2} \\
x_{3} \\
x_{4}
\end{array}\right]=\left[\begin{array}{r}
5 \\
3 \\
5 \\
22
\end{array}\right]
$$

(b) Input data

Coefficient matrix $\mathrm{a}, \operatorname{lna}=11, \mathrm{n}=4$ and constant vector b .
(c) Main program

```
/* C interface example for ASL_dbtlsl */
```

\#include <stdio.h>
\#include <stdlib.h>
\#include <stdlib.
int main()
double *a;
int na;
int nn;
double *b;
int ierr;
int i,j;
FILE *f
fp $=$ fopen( "dbtlsl.dat", "r" );
if ( $\mathrm{fp}==$ NULL $)$
\{
printf( "file open error\n" );
\}
printf( " $\quad * * *$ ASL_dbtlsl $* * * \backslash \mathrm{n} ")$;
printf ( "\n $* *$ Input $* * \backslash \mathrm{n} \backslash \mathrm{n} ")$;
fscanf( fp, "\%d", \&na);
fscanf( fp, "\%d", \&nn );
$\mathrm{a}=($ double $*)$ malloc $(($ size_t) ( sizeof(double) * (na*nn) ));
if ( $\mathrm{a}==\mathrm{NULL}$ )
printf( "no enough memory for array a\n" );

## \}

$\mathrm{b}=($ double * $)$ malloc ( (size_t) ( sizeof(double) * nn ));
if ( $\mathrm{b}==$ NULL $)$
printf( "no enough memory for array b\n" );
return -1;
\}
printf( "\t $n=\% 6 d \backslash n ", n n)$;
printf( "\n\tCoefficient Matrix\n\n" );
for ( i=0 ; i<nn ; i++ )
for ( $j=0 ; j<n n \quad ; j++$ )
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, "\%1f", \&b[i] );
printf( "\t\%8.3g\n", b[i]);
\}
fclose( fp );
ierr = ASL_dbtlsl(a, na, nn, b);
printf( "\n ** Output $* * \backslash \mathrm{n} \backslash \mathrm{n} ")$;
printf( "\tierr = \%6d\n", ierr );
printf( "\n\tSolution\n\n" );
for ( $i=0$; i<nn ; i++ )
printf( "\t $x[\% 6 d]=\% 8.3 g \backslash n ", i, b[i])$;
\}
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

| $\mathrm{x}[$ | $0]=$ | 1 |
| :--- | :--- | :--- |
| $\mathrm{x}[$ | $1]=$ | 1 |
| $\mathrm{x}[$ | $2]=$ | 1 |
| $\mathrm{x}[$ | $3]=$ | 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:

$$
\text { ierr }=\text { ASL_rbtlco } \quad(\mathrm{a}, \operatorname{lna}, \mathrm{n}, \& c o n d, \mathrm{w} 1) ;
$$

(3) Arguments and Return Value

| D:Double precision real R :Single precision real |  | Z:Double precision complex C:Single precision complex |  |  | $\text { I: }\left\{\begin{array}{l} \text { int as for } 32 \text { bit Integer } \\ \text { long as for } 64 \text { bit Integer } \end{array}\right\}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| No. | Argument and Return Value | Type | Size | Input/ <br> Output | Contents |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | $\ln a \times n$ | Input | Real lower triangular matrix $A$ (two-dimensional array type) |
| 2 | $\operatorname{lna}$ | I | 1 | Input | Adjustable dimension of array a |
| 3 | n | I | 1 | Input | Order of matrix $A$ |
| 4 | cond | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 1 | Output | Reciprocal of the condition number |
| 5 | w1 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | n | Work | Work area |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | cond $\leftarrow 1.0$ is performed. |
| 2100 | There existed the diagonal element which <br> was close to zero in the coefficient matrix <br> $A$. The result may not be obtained with <br> a good accuracy. | Processing continues. |
| 3000 | Restriction (a) was not satisfied. | Processing is aborted. |
| $4000+i$ | Matrix $A$ has a 0.0 diagonal element. <br> $i$ is the number of the first 0.0 diagonal <br> element. |  |

(6) Notes
(a) Although the condition number is defined by $\|A\| \cdot\left\|A^{-1}\right\|$, 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:

$$
\text { ierr }=\text { ASL_dbtldi (a, lna, n, det, isw); }
$$

Single precision:

$$
\text { ierr }=\text { ASL_rbtldi (a, lna, n, det, isw); }
$$

(3) Arguments and Return Value

| D:Double precision real | Z:Double precision complex |
| :--- | :--- |
| R:Single precision real | C:Single precision complex |$\quad$ I: \(\left\{\begin{array}{l}int as for 32bit Integer <br>

long as for 64bit Integer\end{array}\right\}\)

| No. | Argument and <br> Return Value | Type | Size | Input/ <br> Output | Contents |
| :---: | :---: | :---: | :---: | :---: | :--- |
| 1 | a | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | lna $\times \mathrm{n}$ | Input | Real lower triangular matrix $A$ <br> (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 | $\left\{\begin{array}{l}\mathrm{D} * \\ \mathrm{R} *\end{array}\right\}$ | 2 | Output | Determinant of matrix $A(($ See Note (b)) |
| 5 | isw | I | 1 | Input | Processing switch <br> isw $>0:$ Obtain determinant <br> isw=0:Obtain determinant and inverse ma- <br> trix <br> isw $<0:$ Obtain inverse matrix |
| 6 | ierr | I | 1 | Output | Error indicator (Return Value) |

(4) Restrictions
(a) $0<\mathrm{n} \leq \ln \mathrm{a}$
(5) Error indicator (Return Value)

| ierr value | Meaning | Processing |
| :---: | :--- | :--- |
| 0 | Normal termination. |  |
| 1000 | n was equal to 1. | $\operatorname{det}[0] \leftarrow \mathrm{a}[0]$ |
|  |  | $\operatorname{det}[1] \leftarrow 0.0$ |
|  |  | $\mathrm{a}[0] \leftarrow 1.0 / \mathrm{a}[0]$ 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.

\[

\]



Remarks
a. $\quad \operatorname{lna} \geq \mathrm{n}$ and $\mathrm{n} \leq \mathrm{k}$ 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:

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

Scaling is performed at this time so that:

$$
1.0 \leq|\operatorname{det}[0]|<10.0
$$

(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 $A X=B$ 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.

$$
\left[\begin{array}{cccc}
a_{1,1} & a_{1,2} & \cdots & a_{1, n} \\
a_{2,1} & a_{2,2} & \cdots & a_{2, n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m, 1} & a_{m, 2} & \cdots & a_{m, n}
\end{array}\right]
$$

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 $\left(a_{i j}\right)$. In this manual, $\left(a_{i, j}\right)$ is used for distinguishing between the row subscript $i$ and column subscript $j$ as necessary.
(2) (Number) vector
$1 \times n$ matrix is called a row vector of size $n$, and an $m \times 1$ 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 $A B=\left(c_{i, l}\right)$ of the two matrices $A=\left(a_{i, j}\right)$ and $B=\left(b_{k, l}\right)$ 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 $A B$ is a column vector $\boldsymbol{x}$, then the product $A \boldsymbol{x}$ is called the matrixvector product.

## (5) Transpose of matrix

The matrix $A^{\prime}=\left(a_{j, i}\right)$ formed by interchanging the rows and columns in $m \times n$ matrix $A=\left(a_{i, j}\right)(i=$ $1,2, \cdots, m ; j=1,2, \cdots, n)$ is called the transpose of matrix $A$ and is represented by $A^{T}$. The transpose may be also represented as ${ }^{t} A$.
(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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A E=E A=A$ for any matrix $A$.
(8) Inverse matrix

For a square matrix $A$, if a square matrix $B$ exist that satisfies $A B=B A=E$ (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 $A^{\dagger}$.

- $A X A=A$
- $X A X=X$
- $(A X)^{T}=A X$
- $(X A)^{T}=X A$


## (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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $U U^{*}=I$ ( $I$ is the unit matrix) holds is called the unitary matrix.

## (15) Orthogonal matrix

The real square matrix $A$ for which $A A^{T}=I$ ( $I$ 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$ 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 $A=L U$ 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 $P A=L U$ is satisfied are obtained, respectively.
(24) $\mathrm{U}^{\mathrm{T}} \mathrm{DU}$ 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 $\mathrm{U}^{\mathrm{T}} \mathrm{DU}$ decomposition.

## (25) $\mathrm{U}^{*} \mathrm{DU}$ 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 $\mathrm{U}^{*} \mathrm{DU}$ decomposition.

## (26) Positive definite

If a real symmetric matrix or Hermitian matrix $A$ satisfies $\boldsymbol{x}^{*} A \boldsymbol{x}>0$ for an arbitrary vector $\boldsymbol{x}(\boldsymbol{x} \neq \mathbf{0})$, 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 $A=\left(a_{i, j}\right)(i, j=1,2, \cdots, n)$

$$
\left|a_{i, i}\right|>\sum_{\substack{j=1 \\ j \neq i}}^{n}\left|a_{i, j}\right| \quad(i=1,2, \cdots, n)
$$

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 $\mathrm{U}^{\mathrm{T}} \mathrm{DU}$ 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 $\mathrm{U}^{\mathrm{T}} \mathrm{DU}$ 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. $(\boldsymbol{a}+\boldsymbol{b})+\boldsymbol{c}=\boldsymbol{a}+(\boldsymbol{b}+\boldsymbol{c}) \quad$ (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. $(c d) \boldsymbol{a}=c(d \boldsymbol{a})$
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 $\left\{\boldsymbol{a}_{1}, \cdots, \boldsymbol{a}_{k}\right\}$ 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 $\left\{\boldsymbol{a}_{1}, \cdots, \boldsymbol{a}_{k}\right\}$. For an arbitrary vector $\boldsymbol{b}$ of $S$, if $\left\{\boldsymbol{a}_{1}, \cdots, \boldsymbol{a}_{k}, \boldsymbol{b}\right\}$ is linearly dependent, $\left\{\boldsymbol{a}_{1}, \cdots, \boldsymbol{a}_{k}\right\}$ 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 $\left\{\boldsymbol{u}_{1}, \cdots, \boldsymbol{u}_{n}\right\}$, then an arbitrary vector $\boldsymbol{a}$ of $V_{n}$ is represented uniquely as a linear combination of $\left\{\boldsymbol{u}_{1}, \cdots, \boldsymbol{u}_{n}\right\}$.
(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 $\boldsymbol{A}: V_{n} \rightarrow V_{m}$ 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) $\boldsymbol{A}\left(\boldsymbol{x}_{1}+\boldsymbol{x}_{2}\right)=\boldsymbol{A}\left(\boldsymbol{x}_{1}\right)+\boldsymbol{A}\left(\boldsymbol{x}_{1}\right) \quad \boldsymbol{x}_{1}, \boldsymbol{x}_{2} \in V_{n}$
(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 $\left\{\boldsymbol{u}_{1}, \cdots, \boldsymbol{u}_{n}\right\}$ and $\left\{\boldsymbol{v}_{1}, \cdots, \boldsymbol{v}_{m}\right\}$, then $\boldsymbol{A}(\boldsymbol{x})$ is determined for an arbitrary $\boldsymbol{x} \in V_{n}$ according to the coefficient matrix $A=\left(a_{i, j}\right)$ of

$$
\boldsymbol{A}\left(\boldsymbol{u}_{j}\right)=\sum_{i=1}^{m} a_{i, j} \boldsymbol{v}_{i} \quad(j=1, \cdots, n)
$$

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 $\boldsymbol{E}: V_{n} \rightarrow V_{n}$, 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 $\lambda$ and a vector $\boldsymbol{x}(\boldsymbol{x} \neq \mathbf{0})$ such that

$$
\boldsymbol{A}(\boldsymbol{x})=\lambda \boldsymbol{x}, \text { that is, }(\boldsymbol{A}-\lambda \boldsymbol{E})(\boldsymbol{x})=\mathbf{0}
$$

is satisfied, then $\lambda$ is called an eigenvalue of $\boldsymbol{A}$ and $\boldsymbol{x}$ is called the eigenvector belonging to the eigenvalue $\lambda$. 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 $\lambda$ 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

$$
\hat{\boldsymbol{x}}=\left[\begin{array}{c}
x_{1} \\
\vdots \\
x_{n}
\end{array}\right]
$$

Normally, $\lambda$ 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 $\lambda$ of the linear transformation $\boldsymbol{A}: V_{n} \rightarrow V_{n}$ together with the zero vector $\mathbf{0}$ form a single vector space, this is called the eigenvector space belonging to the eigenvalue $\lambda$ 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 $\boldsymbol{A} \boldsymbol{x} \in U$ 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.

$$
S_{k l}(\theta)=\left[\begin{array}{ccc}
E_{1: k-1} & O_{1: k-1, k: l} & O_{1: k-1, l: n} \\
O_{k: l, 1: k-1} & T_{k: l}(\theta) & O_{k: l, l: n} \\
O_{l: n, 1: k-1} & O_{l: n, k: l} & E_{l: n}
\end{array}\right]
$$

Here, $T_{k: l}(\theta)$ is defined as follows:

$$
T_{k: l}(\theta)=\left[\begin{array}{ccc}
\cos \theta & O_{k: k, k+1: l-1} & -\sin \theta \\
O_{k+1: l-1, k: k} & E_{k+1: l-1} & O_{k+1: l-1, l: l} \\
\sin \theta & O_{l: l, k+1: l-1} & \cos \theta
\end{array}\right]
$$

$E_{p: q}$ is the $q-p+1$-dimensional unit matrix shown below:

$$
E_{p: q}=\left[\begin{array}{llll}
1 & & & 0 \\
& 1 & & \\
& & \ddots & \\
0 & & & 1
\end{array}\right] \begin{aligned}
& (p \\
& (p+1 \\
& \vdots \\
& (q
\end{aligned}
$$

and $O_{p: r, q: s}$ is the $r-p+1 \times s-q+1$-dimensional zero matrix shown below:

$$
\begin{aligned}
\underbrace{}_{p: r, q: s} & \underbrace{q+1} \\
\cdots & \underbrace{s} \\
{\left[\begin{array}{cccc}
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{array}\right] } & \begin{array}{l}
(p \\
(p+1 \\
\vdots \\
(r
\end{array}
\end{aligned}
$$

Now, if the submatrix $A_{p: r, q: s}$ of $A=\left(a_{i, j}\right)(i=1,2, \cdots, n ; j=1,2, \cdots, n)$ is defined as follows:

$$
A_{p: r, q: s}=\left[\begin{array}{cccc}
a_{p, q} & a_{p, q+1} & \cdots & a_{p, s} \\
a_{p+1, q} & a_{p+1, q+1} & \cdots & a_{p+1, s} \\
\vdots & \vdots & \ddots & \vdots \\
a_{r, q} & a_{r, q+1} & \cdots & a_{r, s}
\end{array}\right]
$$

the matrix $A$ is represented as follows:

$$
A=\left[\begin{array}{ccc}
A_{1: k-1,1: k-1} & A_{1: k-1, k: l} & A_{1: k-1, l+1: n} \\
A_{k: l, 1: k-1} & A_{k: l, k: l} & A_{k: l, l+1: n} \\
A_{l+1: n, 1: k-1} & A_{l+1: n, k: l} & A_{l+1: n, l+1: n}
\end{array}\right]
$$

At this time, since $S_{k: l}(\theta) A$ and $T_{k: l}(\theta) A_{k: l, q: s}$ are as follows:

$$
\begin{gathered}
S_{k: l}(\theta) A=\left[\begin{array}{ccc}
A_{1: k-1,1: k-1} & A_{1: k-1, k: l} & A_{1: k-1, l+1: n} \\
T_{k: l}(\theta) A_{k: l, 1: k-1} & T_{k: l}(\theta) A_{k: l, k: l} & T_{k: l}(\theta) A_{k: l, l+1: n} \\
A_{l+1: n, 1: k-1} & A_{l+1: n, k: l} & A_{l+1: n, l+1: n}
\end{array}\right] \\
T_{k: l}(\theta) A_{k: l, q: s}=\left[\begin{array}{ccc}
\cos \theta a_{k, q}-\sin \theta a_{l, q} & \cdots & \cos \theta a_{k, r}-\sin \theta a_{l, s} \\
a_{k+1, q} & \cdots & a_{k+1, r} \\
\vdots & \cdots & \vdots \\
a_{l-1, q} & \cdots & a_{l-1, r} \\
\sin \theta a_{k, q}+\cos \theta a_{l, q} & \cdots & \sin \theta a_{k, r}+\cos \theta a_{l, s}
\end{array}\right]
\end{gathered}
$$

if $\theta$ 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:

$$
\begin{aligned}
& A S_{k: l}(-\theta)=\left[\begin{array}{cccc}
A_{1: k-1,1: k-1} & A_{1: k-1, k: l} T_{k: l}(-\theta) & A_{1: k-1, l+1: n} \\
A_{k: l, 1: k-1} & A_{k: l, k: l} T_{k: l}(-\theta) & A_{k: l, l+1: n} \\
A_{l+1: n, 1: k-1} & A_{l+1: n, k: l} T_{k: l}(-\theta) & A_{l+1: n, l+1: n}
\end{array}\right] \\
& A_{p: r, k: l} T_{k: l}(-\theta)=\left[\begin{array}{ccccc}
\cos \theta a_{p, k}-\sin \theta a_{p, l} & a_{p, k+1} & \cdots & a_{p, l-1} & \sin \theta a_{p, k}+\cos \theta a_{p, l} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\cos \theta a_{r, k}-\sin \theta a_{r, l} & a_{r, k+1} & \cdots & a_{r, l-1} & \sin \theta a_{r, k}+\cos \theta a_{r, l}
\end{array}\right]
\end{aligned}
$$

if $\theta$ 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 $A S_{k: l}(-\theta)$ can be set to zero. Now, since the following relationship holds:

$$
S_{k: l}(-\theta)=S_{k: l}(\theta)^{T}
$$

and since $\tilde{A}=S_{k: l}(\theta) A S_{k: l}(-\theta)$ is as follows:

$$
\tilde{A}=S_{k: l}(\theta) A S_{k: l}(-\theta)=\left[\begin{array}{ccc}
A_{1: k-1,1: k-1} & A_{1: k-1, k: l} T_{k: l}(-\theta) & A_{1: k-1, l+1: n} \\
T_{k: l}(\theta) A_{k: l, 1: k-1} & T_{k: l}(\theta) A_{k: l, k: l} T_{k: l}(-\theta) & T_{k: l}(\theta) A_{k: l, l+1: n} \\
A_{l+1: n, 1: k-1} & A_{l+1: n, k: l} T_{k: l}(-\theta) & A_{l+1: n, l+1: n}
\end{array}\right]
$$

if matrix $A$ is a symmetric matrix, then by adjusting $\theta$, 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 $\tilde{A}=S_{k: l}(\theta) A S_{k: l}(-\theta)$ are represented by $\left(\tilde{a}_{i, j}\right)$.

## 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 $\times \mathrm{k}]$, then $n \times n$ matrix $A=\left(a_{i, j}\right)(i=1,2, \cdots, n ; j=1,2, \cdots, n)$ is stored in array a as shown in the figure below.

Matrix Storage Mode Within an Array a[lna $\times \mathrm{k}$ ]


## Remarks

a. $\quad \operatorname{lna} \geq n$ and $\mathrm{k} \geq n$ must hold.
b. Matrix element $a_{i, j}$ corresponds to the array element $\mathrm{a}[(i-1)+\ln \mathrm{a} \times(j-1)]$.

Figure B-1 Matrix Storage Mode Within an Array a[lna $\times \mathrm{k}]$
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 $a_{i, j}(i=$ $1,2, \cdots, \operatorname{lna} ; j=1,2, \cdots, \mathrm{k})$ must correspond to the array element $\mathrm{a}[\ell](\ell=0,1,2, \cdots, \ln \mathrm{a} \times \mathrm{k}-1)$, as follows on the main memory.

$$
\begin{array}{ccccccc}
a_{1,1} & a_{2,1} & \cdots & a_{\operatorname{lna}, 1} & a_{1,2} & a_{2,2} & \cdots \\
\mathfrak{\imath} & \uparrow & \cdots & \downarrow & \downarrow & \uparrow & \cdots \\
\mathrm{a}[0] & \mathrm{a}[1] & \cdots & \mathrm{a}[\ln \mathrm{l}-1] & \mathrm{a}[\ln \mathrm{ln}] & \mathrm{a}[\ln \mathrm{ln}+1] & \cdots
\end{array}
$$

Example ASL_dam1ad (Real matrix addition)
Add $3 \times 2$ 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 $\times \mathrm{k}]$
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)

| Matrix to be stored |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ | $a_{1,4}$ | $a_{1,5}$ |
| $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ | $a_{2,5}$ |
| $a_{3,1}$ | $a_{3,2}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ |
| $a_{4,1}$ | $a_{4,2}$ | $a_{4,3}$ | $a_{4,4}$ | $a_{4,5}$ |
| $a_{5,1}$ | $a_{5,2}$ | $a_{5,3}$ | $a_{5,4}$ | $a_{5,5}$ |
| $\Downarrow$ |  |  |  |  |
|  |  |  |  |  |



Remarks
a. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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.

$$
\begin{gathered}
\text { Matrix to be stored } \\
\begin{array}{|lll|}
\hline a_{1,1}+b_{1,1} i & a_{1,2}+b_{1,2} i & a_{1,3}+b_{1,3} i \\
a_{2,1}+b_{2,1} i & a_{2,2}+b_{2,2} i & a_{2,3}+b_{2,3} i \\
a_{3,1}+b_{3,1} i & a_{3,2}+b_{3,2} i & a_{3,3}+b_{3,3} i \\
\hline
\end{array}
\end{gathered}
$$

$\Downarrow$


Storage status within array ai $[\ln \mathrm{a} \times \mathrm{k}$ ]


## Remarks

a. $\quad \ln a \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ must hold.

Figure B-4 Complex Matrix (Two-dimensional Array Type) (Real Argument Type) Storage Mode
(2) Two-dimensional array type, complex argument type

| Matrix to be stored |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ | $a_{1,4}$ | $a_{1,5}$ |
| $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ | $a_{2,5}$ |
| $a_{3,1}$ | $a_{3,2}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ |
| $a_{4,1}$ | $a_{4,2}$ | $a_{4,3}$ | $a_{4,4}$ | $a_{4,5}$ |
| $a_{5,1}$ | $a_{5,2}$ | $a_{5,3}$ | $a_{5,4}$ | $a_{5,5}$ |



## Remarks

a. $\quad \ln a \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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


## Remarks

a. The asterisk $(*)$ indicates an arbitrary value.
b. $\quad \operatorname{lna} \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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 |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ | $a_{1,4}$ | $a_{1,5}$ |
| $a_{1,2}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ | $a_{2,5}$ |
| $a_{1,3}$ | $a_{2,3}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ |
| $a_{1,4}$ | $a_{2,4}$ | $a_{3,4}$ | $a_{4,4}$ | $a_{4,5}$ |
| $a_{1,5}$ | $a_{2,5}$ | $a_{3,5}$ | $a_{4,5}$ | $a_{5,5}$ |
|  |  |  |  |  |
|  | $\Downarrow$ |  |  |  |



## Remarks

a. The asterisk (*) indicates an arbitrary value.
b. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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.

$$
\begin{gathered}
\text { Matrix to be stored } \\
\end{gathered}
$$



Storage status within array ai[lna $\times \mathrm{k}$ ]


## Remarks

a. The asterisk (*) indicates an arbitrary value.
b. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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

| $c$ |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| Matrix to be stored |  |  |  |  |
| $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$ <br> $\overline{a_{1,2}}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$ <br> $\overline{a_{1,3}}$ $\overline{a_{2,3}}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$ <br> $\overline{a_{1,4}}$ $\overline{a_{2,4}}$ $\overline{a_{3,4}}$ $a_{4,4}$ $a_{4,5}$ <br> $\overline{a_{1,5}}$ $\overline{a_{2,5}}$ $\overline{a_{3,5}}$ $\overline{a_{4,5}}$ $a_{5,5}$ |  |  |  |  |



## Remarks

a. The $\bar{x}$ indicates the complex conjugate of $x$.
b. The asterisk $*$ indicates an arbitrary value.
c. $\quad \ln \mathrm{a} \geq \mathrm{n}$ and $\mathrm{k} \geq \mathrm{n}$ 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

| Matrix to be stored |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ |  | 0 |
| $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ |  |
|  | $a_{3,2}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ |
| 0 |  | $a_{4,3}$ | $a_{4,4}$ | $a_{4,5}$ |
|  |  |  | $a_{5,4}$ | $a_{5,5}$ |

$\Downarrow$


## 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. $\quad \operatorname{lna} \geq 2 \times \mathrm{ml}+\mathrm{mu}+1$ and $\mathrm{k} \geq \mathrm{n}$ must hold. (However, if no LU decomposition is to be performed, lna $\geq$ $\mathrm{ml}+\mathrm{mu}+1$ and $\mathrm{k} \geq \mathrm{n}$ 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 |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ |  | 0 |  |
| $a_{1,2}$ | $a_{2,2}$ | $a_{2,3}$ | $a_{2,4}$ |  |  |
| $a_{1,3}$ | $a_{2,3}$ | $a_{3,3}$ | $a_{3,4}$ | $a_{3,5}$ |  |
| 0 | $a_{2,4}$ | $a_{3,4}$ | $a_{4,4}$ | $a_{4,5}$ |  |
|  |  | $a_{3,5}$ | $a_{4,5}$ | $a_{5,5}$ |  |

$\Downarrow$


## Remarks

[^0]Figure B-11 Real Symmetric Band Matrix (Symmetric Band Type) Storage Mode

## B.2.7 Real tridiagonal matrix (vector type)

| Matrix to be stored |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ $a_{1,2}$   0 <br> $a_{2,1}$ $a_{2,2}$ $a_{2,3}$   <br>  $a_{3,2}$ $a_{3,3}$ $a_{3,4}$  <br> 0  $a_{4,3}$ $a_{4,4}$ $a_{4,5}$ <br>    $a_{5,4}$ $a_{5,5}$ |  |  |  |  |



## Remarks

a. The asterisk $*$ indicates an arbitrary value.
b. na $\geq \mathrm{n}$ 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 |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | $a_{1,2}$ |  |  | 0 |
| $a_{1,2}$ | $a_{2,2}$ | $a_{2,3}$ |  |  |
|  | $a_{2,3}$ | $a_{3,3}$ | $a_{3,4}$ |  |
| 0 |  | $a_{3,4}$ | $a_{4,4}$ | $a_{4,5}$ |
|  |  |  | $a_{4,5}$ | $a_{5,5}$ |

$\Downarrow$

Storage status within arrays d[na] (diagonal component) and sd[na] (subdiagonal component)


## Remarks

a. The asterisk * indicates an arbitrary value.
b. na $\geq \mathrm{n}$ must hold.

Figure B-13 Real Symmetric Tridiagonal Matrix (Vector Type) Storage Mode

## B.2.9 Fixed coefficient real tridiagonal matrix (scalar type)


$\Downarrow$

> 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)


## 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 jen 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. $n \leq m<n a$ 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 |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| $a_{1,1}$ | 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 | $a_{6,5}$ | $a_{6,6}$ |

$\Downarrow$
Storage status of arrays aval[na], jen[na] and ia[n]

$\mathrm{n} \xlongequal[{\begin{array}{|l|l}1 & \mathrm{ia}[0] \\ 5 & \mathrm{ia}[1] \\ 8 & \mathrm{ia}[2] \\ 11 & \mathrm{ia}[3] \\ 15 \\ 19 \\ \hline\end{array}}\end{array}]{ }$

## 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 |
| :---: | :---: |
| $2^{-23}\left(\simeq 1.19 \times 10^{-7}\right)$ | $2^{-52}\left(\simeq 2.22 \times 10^{-16}\right)$ |

Remark: The unit for determining error $\varepsilon$, which is also called the machine $\varepsilon$, is usually defined as the smallest positive constant for which the calculation result of $1+\varepsilon$ differs from 1 in the corresponding floating point mode. Therefore, seeing the unit for determining error enables you to know the maximum number of significant digits of an operation (on the mantissa) in that floating point mode.

## 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 | $2^{127}\left(2-2^{-23}\right)\left(\simeq 3.40 \times 10^{38}\right)$ | $2^{1023}\left(2-2^{-52}\right)\left(\simeq 1.80 \times 10^{308}\right)$ |
| Positive <br> minimum value | $2^{-126}\left(\simeq 1.17 \times 10^{-38}\right)$ | $2^{-1022}\left(\simeq 2.23 \times 10^{-308}\right)$ |
| Negative <br> maximum value | $-2^{-126}\left(\simeq-1.17 \times 10^{-38}\right)$ | $-2^{-1022}\left(\simeq-2.23 \times 10^{-308}\right)$ |
| Minimum value | $-2^{127}\left(2-2^{-23}\right)\left(\simeq-3.40 \times 10^{38}\right)$ | $-2^{1023}\left(2-2^{-52}\right)\left(\simeq-1.80 \times 10^{308}\right)$ |

## Index

ASL_cam1hh : Vol.1, 106
ASL_cam1hm : Vol.1, 101
ASL_cam1mh : Vol.1, 96
ASL_cam1mm : Vol.1, 91
ASL_can1hh : Vol.1, 123
ASL_can1hm : Vol.1, 119
ASL_can1mh : Vol.1, 115
ASL_can1mm : Vol.1, 111
ASL_canvj1 : Vol.1, 155
ASL_cargjm : Vol.1, 44
ASL_carsjd : Vol.1, 38
ASL_cbgmdi : Vol.2, 80
ASL_cbgmlc : Vol.2, 72
ASL_cbgmls : Vol.2, 74
ASL_cbgmlu : Vol.2, 70
ASL_cbgmlx : Vol.2, 82
ASL_cbgmms : Vol.2, 76
ASL_cbgmsl : Vol.2, 64
ASL_cbgmsm : Vol.2, 59
ASL_cbgndi : Vol.2, 102
ASL_cbgnlc : Vol.2, 94
ASL_cbgnls : Vol.2, 96
ASL_cbgnlu : Vol.2, 92
ASL_cbgnlx : Vol.2, 104
ASL_cbgnms : Vol.2, 98
ASL_cbgnsl : Vol.2, 88
ASL_cbgnsm : Vol.2, 84
ASL_cbhedi : Vol.2, 239
ASL_cbhels : Vol.2, 233
ASL_cbhelx : Vol.2, 241
ASL_cbhems : Vol.2, 235
ASL_cbhesl : Vol.2, 224
ASL_cbheuc : Vol.2, 231
ASL_cbheud : Vol.2, 229
ASL_cbhfdi : Vol.2, 220
ASL_cbhfls : Vol.2, 214
ASL_cbhflx : Vol.2, 222
ASL_cbhfms : Vol.2, 216
ASL_cbhfsl : Vol.2, 205
ASL_cbhfuc : Vol.2, 212
ASL_cbhfud : Vol.2, 210
ASL_cbhpdi : Vol.2, 182
ASL_cbhpls : Vol.2, 176
ASL_cbhplx : Vol.2, 184
ASL_cbhpms : Vol.2, 178

ASL_cam1hm : Vol.1, 101
ASL_cam1mh : Vol.1, 96
_cam1mm : Vol.1, 91

ASL_can1hm : Vol.1, 119
ASL_can1mh : Vol.1, 115
ASL_can1mm : Vol.1, 111

ASL_carsjd : Vol.1, 38
ASL_cbgmdi : Vol.2, 80
ASL_cbgmlc : Vol.2, 72
ASL_cbgmls : Vol.2, 74

ASL_cbgmlx : Vol.2, 82
ASL_cbgmms : Vol.2, 76
ASL_cbgmsl : Vol.2, 64
ASL_cbgmsm : Vol.2, 59
ASL_cbgnlc : Vol.2, 94
ASL_cbgnls : Vol.2, 96
ASL_cbgnlu : Vol.2, 92
ASL_cbgnlx : Vol.2, 104
ASL_Cbgnms : Vol.2, 98
ASL_cbgnsm : Vol.2, 84
ASL_cbhedi : Vol.2, 239
ASL_cbhels : Vol.2, 233
ASL_cbhelx : Vol.2, 241
Vol.2, 235

ASL_cbheuc : Vol.2, 231
ASL_cbheud : Vol.2, 229
ASL_cbhfdi : Vol.2, 220
ASL_cbhfls : Vol.2, 214
ASL_cbhflx : Vol.2, 222

ASL_cbhfsl : Vol.2, 205
ASL_cbhfuc : Vol.2, 212
ASL_cbhfud : Vol.2, 210
ASL_cbhpdi : Vol.2, 182
ASL_cbhplx : Vol.2, 184
ASL_cbhpms : Vol.2, 178

| ASL_cbhpsl : Vol.2, 167 |
| :--- |
| ASL_cbhpuc : Vol.2, 174 |
| ASL_cbhpud : Vol.2, 172 |
| ASL_cbhrdi : Vol.2, 201 |
| ASL_cbhrls : Vol.2, 195 |
| ASL_cbhrlx : Vol.2, 203 |
| ASL_cbhrms : Vol.2, 197 |
| ASL_cbhrsl : Vol.2, 186 |
| ASL_cbhruc : Vol.2, 193 |
| ASL_cbhrud : Vol.2, 191 |
| ASL_ccgeaa : Vol.1, 191 |
| ASL_ccgean : Vol.1, 196 |
| ASL_ccghaa : Vol.1, 379 |
| ASL_ccghan : Vol.1, 384 |
| ASL_ccgjaa : Vol.1, 386 |
| ASL_ccgjan : Vol.1, 391 |
| ASL_ccgkaa : Vol.1, 393 |
| ASL_ccgkan : Vol.1, 398 |
| ASL_ccgnaa : Vol.1, 198 |
| ASL_ccgnan : Vol.1, 202 |
| ASL_ccgraa : Vol.1, 372 |
| ASL_ccgran : Vol.1, 377 |
| ASL_ccheaa : Vol.1, 244 |
| ASL_cchean : Vol.1, 248 |
| ASL_ccheee : Vol.1, 257 |
| ASL_ccheen : Vol.1, 262 |
| ASL_cchesn : Vol.1, 255 |
| ASL_cchess : Vol.1, 250 |
| ASL_cchjss : Vol.1, 320 |
| ASL_cchraa : Vol.1, 224 |
| ASL_cchran : Vol.1, 228 |
| ASL_cchree : Vol.1, 237 |
| ASL_cchren : Vol.1, 242 |
| ASL_cchrsn : Vol.1, 235 |
| ASL_cchrss : Vol.1, 230 |
| ASL_cfc1bf : Vol.3, 61 |
| ASL_cfc1fb : Vol.3, 57 |
| ASL_cfc2bf : Vol.3, 127 |
| ASL_cfc2fb : Vol.3, 123 |
| ASL_cfc3bf : Vol.3, 157 |
| ASL_cfc3fb : Vol.3, 153 |
| ASL_cfcmbf : Vol.3, 93 |
| ASL_cfcmfb : Vol.3, 89 |
| ASL_cibh1n : Vol.5, 159 |
| ASL_cibh2n : Vol.5, 162 |

ASL_cbhpuc : Vol.2, 174
ASL_cbhpud : Vol.2, 172
ASL_cbhrdi : Vol.2, 201
ASL_cbhrls : Vol.2, 103
ASL_cbhrms : Vol.2, 197
ASL_cbhrsl : Vol.2, 186
ASL_cbhruc : Vol.2, 193
ASL_cbhrud : Vol.2, 191
ASL_ccgeaa : Vol.1, 191
ASL_ccgean : Vol.1, 196
ASL_ccghaa : Vol.1, 379
ccghan:Vol.1, 384
ASL_ccgjan : Vol.1, 391
ASL_ccgkaa : Vol.1, 393
ASL_ccgkan : Vol.1, 398
ASL_ccgnaa : Vol.1, 198
ASL_ccgnan : Vol.1, 202
ASL_ccgraa : Vol.1, 372
ASL_ccheaa : Vol.1, 244
ASL_cchean : Vol.1, 248
ASL_ccheee : Vol.1, 257
ASL_ccheen : Vol.1, 262

ASL_cchess : Vol.1, 250
ASL_cchjss : Vol.1, 320
ASL_cchraa : Vol.1, 224
ASL_cchran : Vol.1, 228
ASL_cchren : Vol.1, 242
ASL_cchrsn : Vol.1, 235
ASL_cchrss : Vol.1, 230
Cfclbf:Vol.3, 61

ASL_cfc2bf : Vol.3, 127
ASL_cfc2fb : Vol.3, 123
ASL_cfc3bf : Vol.3, 157
ASL_cfc3fb : Vol.3, 153
ASL_cfcmbf : Vol.3, 93

ASL_cibh1n : Vol.5, 159
ASL_cibh2n : Vol.5, 162
ASL_cibinz : Vol.5, 141
ASL_cibjnz : Vol.5, 96
ASL_cibknz : Vol.5, 144
ASL_cibynz : Vol.5, 99
ASL_cigamz : Vol.5, 205
ASL_ciglgz : Vol.5, 207
ASL_clacha : Vol.5, 392
ASL_clncis : Vol.5, 410
ASL_d1cdbn : Vol.6, 79
ASL_d1cdbt : Vol.6, 123
ASL_d1cdcc : Vol.6, 160
ASL_d1cdch : Vol.6, 83
ASL_d1cdex : Vol.6, 145
ASL_d1cdfb : Vol.6, 109
ASL_d1cdgm : Vol.6, 116
ASL_d1cdgu : Vol.6, 148
ASL_d1cdib : Vol.6, 127
ASL_d1cdic : Vol.6, 86
ASL_d1cdif : Vol.6, 113
ASL_d1cdig : Vol.6, 120
ASL_d1cdin : Vol.6, 76
ASL_d1cdis : Vol.6, 106
ASL_d1cdit : Vol.6, 99
ASL_d1cdix : Vol.6, 93
ASL_d1cdld : Vol.6, 151
ASL_d1cdlg : Vol.6, 157
ASL_d1cdln : Vol.6, 154
ASL_d1cdnc : Vol.6, 89
ASL_d1cdno : Vol.6, 73
ASL_d1cdnt : Vol.6, 102
ASL_d1cdpa : Vol.6, 137
ASL_d1cdtb : Vol.6, 96
ASL_d1cdtr : Vol.6, 134
ASL_d1cduf : Vol.6, 131
ASL_d1cdwe : Vol.6, 141
ASL_d1ddbp : Vol.6, 164
ASL_d1ddgo : Vol.6, 168
ASL_d1ddhg : Vol.6, 174
ASL_d1ddhn : Vol.6, 177
ASL_d1ddpo : Vol.6, 171
ASL_d2ba1t : Vol.6, 188
ASL_d2ba2s : Vol.6, 195
ASL_d2bagm : Vol.6, 210
ASL_d2bahm : Vol.6, 219
ASL_d2bamo : Vol.6, 215
ASL_d2bams : Vol.6, 204
ASL_d2basm : Vol.6, 223
ASL_d2ccma : Vol.6, 249
ASL_d2ccmt : Vol.6, 243
ASL_d2ccpr : Vol.6, 256
ASL_d2vcgr : Vol.6, 233
ASL_d2vcmt : Vol.6, 227
ASL_d3iecd : Vol.6, 337
ASL_d3ieme : Vol.6, 322

ASL_cibjnz : Vol.5, 96
ASL_cibknz : Vol.5, 144
ASL_cibynz : Vol.5, 99
ASL_cigamz : Vol.5, 205
ASL_ciglgz : Vol.5, 207
ASL_Clacha : Vol.5, 392
ASL_clncis : Vol.5, 410

ASL_d1cdbt : Vol.6, 123
ASL_d1cdcc : Vol.6, 160
ASL_d1cdch : Vol.6, 83
ASL_d1cdex : Vol.6, 145

ASL_d1cdgm : Vol.6, 116
ASL_d1cdgu : Vol.6, 148
ASL_d1cdib : Vol.6, 127
Vol.6, 86

ASL_d1cdig : Vol.6, 120
ASL_d1cdin : Vol.6, 76
ASL_d1cdis : Vol.6, 106
ASL_d1cdit : Vol.6, 99
ASL_d1cdix : Vol.6, 93

ASL_d1cdlg : Vol.6, 157
ASL_d1cdln : Vol.6, 154
ASL_d1cdnc : Vol.6, 89
ASL_d1cdno : Vol.6, 73
ASL_d1cdnt : Vol.6, 102

ASL_d1cdtb : Vol.6, 96
ASL_d1cdtr : Vol.6, 134
ASL_d1cduf : Vol.6, 131
ASL_d1cdwe : Vol.6, 141

ASL_d1ddgo : Vol.6, 168
ASL_d1ddhg : Vol.6, 174
ASL_d1ddhn : Vol.6, 177
ASL_d1ddpo : Vol.6, 171
_d2ba1t : Vol.6, 188

ASL_d2bagm : Vol.6, 210
ASL_d2bahm : Vol.6, 219
ASL_d2bamo : Vol.6, 215
ASL_d2bams : Vol.6, 204

ASL_d2ccma : Vol.6, 249
ASL_d2ccmt : Vol.6, 243
ASL_d2ccpr : Vol.6, 256
ASL_d2vcgr : Vol.6, 233

ASL_d3iecd : Vol.6, 337
ASL_d3ieme : Vol.6, 322

ASL_d3iera : Vol.6, 319
ASL_d3iesr : Vol.6, 342
ASL_d3iesu : Vol.6, 326
ASL_d3ietc : Vol.6, 333
ASL_d3ieva : Vol.6, 330
ASL_d3tscd : Vol.6, 380
ASL_d3tsme : Vol.6, 357
ASL_d3tsra : Vol.6, 348
ASL_d3tsrd : Vol.6, 352
ASL_d3tssr : Vol.6, 383
ASL_d3tssu : Vol.6, 362
ASL_d3tstc : Vol.6, 373
ASL_d3tsva : Vol.6, 369
ASL_d41wr1 : Vol.6, 397
ASL_d42wr1 : Vol.6, 417
ASL_d42wrm : Vol.6, 409
ASL_d42wrn : Vol.6, 403
ASL_d4bi01 : Vol.6, 477
ASL_d4gl01 : Vol.6, 472
ASL_d4mu01 : Vol.6, 452
ASL_d4mwrf : Vol.6, 426
ASL_d4mwrm : Vol.6, 439
ASL_d4rb01 : Vol.6, 468
ASL_d5chef : Vol.6, 486
ASL_d5chmd : Vol.6, 497
ASL_d5chmn : Vol.6, 493
ASL_d5chtt : Vol.6, 490
ASL_d5temh : Vol.6, 509
ASL_d5tesg : Vol.6, 501
ASL_d5tesp : Vol.6, 513
ASL_d5tewl : Vol.6, 505
ASL_d6clan : Vol.6, 571
ASL_d6clda : Vol.6, 576
ASL_d6clds : Vol.6, 565
ASL_d6cpcc : Vol.6, 526
ASL_d6cpsc : Vol.6, 528
ASL_d6cvan : Vol.6, 543
ASL_d6cvsc : Vol.6, 546
ASL_d6dafn : Vol.6, 553
ASL_d6dasc : Vol.6, 557
ASL_d6fald : Vol.6, 535
ASL_d6favr : Vol.6, 537
ASL_dabmcs : Vol.1, 13
ASL_dabmel : Vol.1, 17
ASL_dam1ad : Vol.1, 55
ASL_dam1mm : Vol.1, 75
ASL_dam1ms : Vol.1, 65
ASL_dam1mt : Vol.1, 79
ASL_dam1mu : Vol.1, 61
ASL_dam1sb : Vol.1, 58
ASL_dam1tm : Vol.1, 83
ASL_dam1tp : Vol.1, 136
ASL_dam1tt : Vol.1, 87
ASL_dam1vm : Vol.1, 127

ASL_dam3tp : Vol.1, 139
ASL_dam3vm : Vol.1, 130
ASL_dam4vm : Vol.1, 133
ASL_damt1m : Vol.1, 69
ASL_damvj1 : Vol.1, 143
ASL_damvj3 : Vol.1, 147
ASL_damvj4 : Vol.1, 151
ASL_dargjm : Vol.1, 32
ASL_darsjd : Vol.1, 26
ASL_dasbcs : Vol.1, 20
ASL_dasbel : Vol.1, 23
ASL_datm1m : Vol.1, 72
ASL_dbbddi : Vol.2, 255
ASL_dbbdlc : Vol.2, 250
ASL_dbbdls : Vol.2, 253
ASL_dbbdlu : Vol.2, 248
ASL_dbbdlx : Vol.2, 257
ASL_dbbdsl : Vol.2, 243
ASL_dbbpdi : Vol.2, 272
ASL_dbbpls : Vol.2, 270
ASL_dbbplx : Vol.2, 274
ASL_dbbpsl : Vol.2, 262
ASL_dbbpuc : Vol.2, 268
ASL_dbbpuu : Vol.2, 266
ASL_dbgmdi : Vol.2, 52
ASL_dbgmlc : Vol.2, 44
ASL_dbgmls : Vol.2, 46
ASL_dbgmlu : Vol.2, 42
ASL_dbgmlx : Vol.2, 54
ASL_dbgmms : Vol.2, 48
ASL_dbgmsl : Vol.2, 37
ASL_dbgmsm : Vol.2, 32
ASL_dbpddi : Vol.2, 116
ASL_dbpdls : Vol.2, 114
ASL_dbpdlx : Vol.2, 118
ASL_dbpdsl : Vol.2, 106
ASL_dbpduc : Vol.2, 112
ASL_dbpduu : Vol.2, 110
ASL_dbsmdi : Vol.2, 154
ASL_dbsmls : Vol.2, 148
ASL_dbsmlx : Vol.2, 156
ASL_dbsmms : Vol.2, 150
ASL_dbsmsl : Vol.2, 139
ASL_dbsmuc : Vol.2, 146
ASL_dbsmud : Vol.2, 144
ASL_dbsnls : Vol.2, 165
ASL_dbsnsl : Vol.2, 158
ASL_dbsnud : Vol.2, 163
ASL_dbspdi : Vol.2, 135
ASL_dbspls : Vol.2, 129
ASL_dbsplx : Vol.2, 137
ASL_dbspms : Vol.2, 131
ASL_dbspsl : Vol.2, 120
ASL_dbspuc : Vol.2, 127

ASL_dbspud : Vol.2, 125
ASL_dbtdsl : Vol.2, 276
ASL_dbtlco : Vol.2, 324
ASL_dbtldi : Vol.2, 326
ASL_dbtlsl : Vol.2, 321
ASL_dbtosl : Vol.2, 302
ASL_dbtpsl : Vol.2, 280
ASL_dbtssl : Vol.2, 306
ASL_dbtuco : Vol.2, 317
ASL_dbtudi : Vol.2, 319
ASL_dbtusl : Vol.2, 314
ASL_dbvmsl : Vol.2, 310
ASL_dcgbff : Vol.1, 400
ASL_dcgeaa : Vol.1, 177
ASL_dcgean : Vol.1, 183
ASL_dcggaa : Vol.1, 328
ASL_dcggan : Vol.1, 335
ASL_dcgjaa : Vol.1, 360
ASL_dcgjan : Vol.1, 364
ASL_dcgkaa : Vol.1, 366
ASL_dcgkan : Vol.1, 370
ASL_dcgnaa : Vol.1, 185
ASL_dcgnan : Vol.1, 189
ASL_dcgsaa : Vol.1, 337
ASL_dcgsan : Vol.1, 342
ASL_dcgsee : Vol.1, 352
ASL_dcgsen : Vol.1, 358
ASL_dcgssn : Vol.1, 350
ASL_dcgsss : Vol.1, 344
ASL_dcsbaa : Vol.1, 264
ASL_dcsban : Vol.1, 268
ASL_dcsbff : Vol.1, 277
ASL_dcsbsn : Vol.1, 275
ASL_dcsbss : Vol.1, 270
ASL_dcsjss : Vol.1, 311
ASL_dcsmaa : Vol.1, 204
ASL_dcsman : Vol.1, 208
ASL_dcsmee : Vol.1, 217
ASL_dcsmen : Vol.1, 222
ASL_dcsmsn : Vol.1, 215
ASL_dcsmss : Vol.1, 210
ASL_dcsrss : Vol.1, 303
ASL_dcstaa : Vol.1, 283
ASL_dcstan : Vol.1, 287
ASL_dcstee : Vol.1, 296
ASL_dcsten : Vol.1, 301
ASL_dcstsn : Vol.1, 294
ASL_dcstss : Vol.1, 289
ASL_dfasma : Vol.6, 285
ASL_dfc1bf : Vol.3, 50
ASL_dfc1fb : Vol.3, 46
ASL_dfc2bf : Vol.3, 117
ASL_dfc2fb : Vol.3, 113
ASL_dfc3bf : Vol.3, 146

|  | ol.3, 14 |
| :---: | :---: |
| df | 3, 81 |
| df | 3, 77 |
| ASL_dfcn | , |
| ASL_df cn | 3, 187 |
| _df | 3, 195 |
| df | ol.3, |
| df | 3, 216 |
| ASL_dfc | ol.3, 224 |
| ASL dff | l.6, 283 |
| ASL dfc | 6, 281 |
| ASL df | 1.6, 279 |
| ASL_df | 1.6, 274 |
|  | 1.6, 269 |
| L_dfdp | 6, |
| L_dfdp | l.6, |
| L_dfdp | 1.6, 294 |
| 12 | ol.3, 273 |
| ASL_dfla | ol.3, 267 |
| L_dfps | 1.3, 235 |
| L_dfp | ol.3, 24 |
| _dfps | ol.3, 252 |
| ASL_dfr | l.3, 71 |
| ASL_df | l.3, 67 |
| ASL_dfr | l.3, 136 |
| ASL_df | l.3, 132 |
| df | l.3, 169 |
| _df | l.3, 164 |
| ASL_df | 3, 106 |
| L_dfr | ol.3, 101 |
| df | 1.3, 306 |
| _dfw | 1.3, 308 |
| df | -1.3, 277 |
| ASL_df | 1.3, 289 |
| ASL_d | 1.3, 296 |
| ASL_df | 1.3, 280 |
| df | ol.3, 284 |
| df | l.3, 292 |
| df | 1.3, 301 |
| df | l.3, 303 |
| L_dgic | ol.4, 513 |
| L_dgic | ol.4, 537 |
| _dgi | ol.4, 483 |
| dgi | -1.4, 486 |
| _dgic | 1.4, 478 |
| L_dgic | Vol.4, 469 |
| _dgi | 4, 471 |
| L_dgi | Vol.4, 473 |
| _dgic | 1.4, 475 |
| L_dgic | ol.4, 480 |
| SL_dgid | Vol.4, 517 |
| dgid | 1.4, 492 |
| _dgidm | Vol.4, 445 |
|  |  |

ASL_dfc3fb : Vol.3, 142
ASL_dfcmbf : Vol.3, 81
Vol.3, 77
ASL_dfcn2d : Vol.3, 187
ASL_dfcn3d : Vol.3, 195
AsL_dfcr1d:Vol.3, 206
ASL_dfcr3d : Vol.3, 224
ASL_dfcrcs : Vol.6, 283
ASL_dfcrcz : Vol.6, 281
ASL_dfcrsc : Vol.6, 279
Vol.6, 274
ASL_dfdped : Vol.6, 291
ASL_dfdpes : Vol.6, 289
ASL_dfdpet : Vol.6, 294
ASL_dflage : Vol.3, 273
ASL_dfps1d : Vol.3, 235
ASL_dfps2d : Vol.3, 243
ASL_dfps3d : Vol.3, 252
ASL_dfr1bf : Vol.3, 71
ASL_dfr2bf : Vol.3, 136
ASL_dfr2fb : Vol.3, 132
ASL_dfr3bf : Vol.3, 169
ASL_dfr3fb : Vol.3, 164
ASL_dfrmbf :Vol.3, 106
ASL_dfwtff : Vol.3, 306
ASL_dfwtft : Vol.3, 308
ASL_dfwth1 : Vol.3, 277
ASL_dfwth2 : Vol.3, 289
ASL_dfwthr : Vol.3, 280
ASL_dfwths : Vol.3, 284
ASL_dfwtht : Vol.3, 292
ASL_dfwtmf : Vol.3, 301
ASL_dgicbp: Vol.4, 513
ASL_dgicbs : Vol.4, 537
ASL_dgiccm : Vol.4, 483
ASL_dgiccn : Vol.4, 486
ASL_dgicco : Vol.4, 478
ASL_dgiccq: Vol.4, 471
ASL_dgiccr : Vol.4, 473
ASL_dgiccs : Vol.4, 475
ASL_dgicct : Vol.4, 480
ASL_dgidby : Vol.4, 517
ASL_dgidmc : Vol.4, 445
ASL_dgidpc : Vol.4, 432

ASL_dgidsc : Vol.4, 438
ASL_dgidyb : Vol.4, 503
ASL_dgiibz : Vol.4, 519
ASL_dgiicz: Vol.4, 495
ASL_dgiimc : Vol.4, 463
ASL_dgiipc : Vol.4, 452
ASL_dgiisc : Vol.4, 457
ASL_dgiizb : Vol.4, 509
ASL_dgisbx : Vol.4, 515
ASL_dgiscx : Vol.4, 490
ASL_dgisi1 : Vol.4, 540
ASL_dgisi2 : Vol.4, 545
ASL_dgisi3 : Vol.4, 554
ASL_dgismc : Vol.4, 426
ASL_dgispc : Vol.4, 416
ASL_dgispo : Vol.4, 521
ASL_dgispr : Vol.4, 525
ASL_dgiss1 : Vol.4, 561
ASL_dgiss2 : Vol.4, 566
ASL_dgiss3 : Vol.4, 574
ASL_dgissc : Vol.4, 420
ASL_dgisso : Vol.4, 529
ASL_dgissr : Vol.4, 533
ASL_dgisxb : Vol.4, 497
ASL_dh2int : Vol.4, 299
ASL_dhbdfs : Vol.4, 264
ASL_dhbsfc : Vol.4, 267
ASL_dhemnh : Vol.4, 270
ASL_dhemni : Vol.4, 287
ASL_dhemnl : Vol.4, 223
ASL_dhnanl : Vol.4, 259
ASL_dhnefl : Vol.4, 235
ASL_dhnenh : Vol.4, 279
ASL_dhnenl : Vol.4, 250
ASL_dhnfml : Vol.4, 317
ASL_dhnfnm : Vol.4, 307
ASL_dhnifl : Vol.4, 240
ASL_dhninh : Vol.4, 283
ASL_dhnini : Vol.4, 295
ASL_dhninl : Vol.4, 255
ASL_dhnofh : Vol.4, 274
ASL_dhnofi : Vol.4, 291
ASL_dhnofl : Vol.4, 230
ASL_dhnpnl : Vol.4, 245
ASL_dhnrml : Vol.4, 312
ASL_dhnrnm : Vol.4, 302
ASL_dhnsnl : Vol.4, 227
ASL_dibaid : Vol.5, 189
ASL_dibaix : Vol.5, 185
ASL_dibbei : Vol.5, 167
ASL_dibber : Vol.5, 165
ASL_dibbid : Vol.5, 191
ASL_dibbix : Vol.5, 187
ASL_dibimx : Vol.5, 135

|  | Vol.5, 129 |
| :---: | :---: |
| ASL_dibjmx | , 90 |
| ASL_dibjn | Vol.5, 84 |
| ASL_dibkei | Vol.5, 171 |
|  | Vol.5, 169 |
| m | Vol.5, 138 |
| ASL_dibknx | Vol.5, 132 |
| ASL_dibs | Vol.5, 153 |
| ASL_dibsjn | Vol.5, 147 |
| SL_dibskn | Vol.5, 156 |
|  | 0 |
| bym | Vol.5, 93 |
| ibyn | Vol.5, 87 |
| ASL_dieii1 | Vol.5, 221 |
| ASL_dieii2 | Vol |
| SL_dieii3 | Vol.5, 226 |
| SL_dieii4 | Vol.5, 228 |
| ASL_digig1 | Vo |
| igig | Vol |
| SL_diicos | Vol.5, 261 |
| ASL_diierf | Vol.5, |
| ASL_diisin | Vol.5, 259 |
| , | Vol.5, 285 |
| L_dileg2 | Vol.5, 288 |
| mtce | Vol.5, 306 |
| dimtse | Vol.5, 309 |
| _diopc2 | Vol.5, 302 |
| SL_diopch | Vol.5, 300 |
| - | Vol.5, |
| diop | Vol.5, 298 |
| _diopla | Vol.5, 296 |
| SL_diople | Vol.5, 291 |
| ixeps | Vol.5, 324 |
| SL_dizbs0 | Vol.5, 102 |
| SL_dizbs1 | Vol.5, 105 |
| izbsl | Vol.5, 11 |
| izbsn | Vol.5, 108 |
| L_dizbyn | Vol.5, 111 |
| _dizglw | Vol.5, 293 |
| _djtecc | Vol.6, 33 |
| SL_djteex | Vol.6, 29 |
| SL_djtegm | Vol.6, 45 |
| SL_djtegu | Vol.6, 37 |
| djtelg | Vol.6, 49 |
| L_djteno | Vol.6, 25 |
| SL_djteun | Vol.6, 20 |
| SL_djtewe | Vol.6, 41 |
| L_dkfncs | Vol.4, 72 |
| L_dkhncs | Vol.4, 78 |
| L_dkinct | Vol.4, 55 |
| SL_dkmncn | Vol.4, 84 |
| SL_dksnca | Vol.4, 49 |
| SL_dksncs | Vol.4, 43 |
| SL_dkssca | Vol.4, 65 |

ASL_dibjmx : Vol.5, 90
ASL_dibjnx : Vol.5, 84
ASL_dibkei : Vol.5, 171
ASL_dibker : Vol.5, 169
ASL_dibkmx : Vol.5, 138
ASL_dibknx : Vol.5, 132
SL_dibsin : Vol.5, 153

ASL_dibskn : Vol.5, 156
ASL_dibsyn : Vol.5, 150
ASL_dibymx : Vol.5, 93
ASL_dibynx : Vol.5, 87

ASL_dieii2 : Vol.5, 223
ASL_dieii3 : Vol.5, 226
ASL_dieii4 : Vol.5, 228
ASL_digig1 : Vol.5, 199
SL_digig2 : Vol.5, 202

ASL_diierf : Vol.5, 281
ASL_diisin : Vol.5, 259
ASL_dileg1 : Vol.5, 285
SL_dileg2 : Vol.5, 288
ASL_dimtce : Vol.5, 306

ASL_diopc2 : Vol.5, 302
ASL_diopch : Vol.5, 300
Vol.5, 304

ASL_diopla : Vol.5, 296
ASL_diople : Vol.5, 291
ASL_dixeps : Vol.5, 324
ASL_dizbs0 : Vol.5, 102
ASL_dizbs1 : Vol.5, 105
. Vol.5, 114

ASL_dizbsn : Vol.5, 108
ASL_dizglw : Vol.5, 293
ASL_djtecc : Vol.6, 33
_djteex : Vol.6, 29

ASL_djtegu : Vol.6, 37
ASL_djtelg : Vol.6, 49
ASL_djteno : Vol.6, 25
ASL_djteun : Vol.6, 20

ASL_dkfncs : Vol.4, 72
ASL_dkhncs : Vol.4, 78
ASL_dkinct : Vol.4, 55
Vol.4, 84

ASL_dksncs : Vol.4, 43
ASL_dkssca : Vol.4, 65

|  | , |
| :---: | :---: |
| ASL_dlnrds | 96 |
| ASL_dlnris | Vol.5, 400 |
|  | Vol.5, |
| ASL_dlnrss | 403 |
| ASL_dlsrds | Vol |
| ASL_dlsris |  |
| ASL_dmclaf | Vol.5, 490 |
|  | Vol.5, 517 |
|  | Vol |
| ASL_dmclmz | Vol.5, 502 |
| ASL_dmclsn | 483 |
|  | 4 |
| - | 4 |
| dmcql | Vol.5, 538 |
| , | 1 |
| ASL_dmcusn | Vol.5, 479 |
| Sp11 | Vol.5, 567 |
| Spl | Vol.5, 558 |
| _dmspm | Vol.5, 563 |
| ASL_dmsqpm | Vo |
| ¢ | Vol.5, 469 |
|  | Vol.5, 465 |
| ASL_dmussn | O1.5, |
| ASL_dmuusn | Vol.5, 462 |
| cbpo | Vol.4, |
| SL_dndaao | Vol.4, 362 |
| ASL_dndanl | 372 |
| SL_dndapo | Vol.4, 367 |
| L_dngap | Vol.4, 386 |
| ASL_dnlnma | Vol. 6 |
| __dnlnrg | Vol.6, 592 |
| _d | Vol.6, 598 |
| SL_dnnlgf | Vol.6, 6 |
| SL_dnnlp | Vol.6, 611 |
| SL_dnrap | Vol.4, 379 |
| SL_dofnnf | Vol.4, |
| SL_dofnnv | Vol.4, 108 |
| v | Vol.4, 136 |
| SL_dohnnf | Vol.4, 129 |
| L_dohn | Vol.4, 122 |
| _doief2 | Vol.4, 149 |
| _doiev1 | Vol.4, 153 |
| L_dolnlv | Vol.4, 143 |
| SL_dopdh2 | Vol.4, 157 |
| SL_dopdh3 | Vol.4, 165 |
| L_dosnnf | Vol.4, 100 |
| _dosnnv | Vol.4, 91 |
| SL_dpdapn | Vol.4, 347 |
| dpdopl | Vol.4, 343 |
| SL_dpgopl | Vol.4, 358 |
| SL_dplopl | Vol.4, 351 |
| L_dqfodx | Vol.4, 182 |
| SL_dqmog | Vol.4, 186 |

ASL_dlnrds : Vol.5, 396
ASL_dlnris : Vol.5, 400
ASL_dlnrsa : Vol.5, 406
ASL_dlnrss : Vol.5, 403
ASL_dlsrds : Vol.5, 414
ASL_dlsris : Vol.5, 421

ASL_dmclcp : Vol.5, 517
ASL_dmclmc : Vol.5, 511
ASL_dmclmz : Vol.5, 502
ASL_dmclsn : Vol.5, 483
ASL_dmcltp : Vol.5, 524
ASL_dmcqaz . Vol.5, 544

ASL_dmcqsn : Vol.5, 531
ASL_dmcusn : Vol.5, 479
ASL_dmsp11 : Vol.5, 567
L_dmsp1m : Vol.5, 558

ASL_dmsqpm : Vol.5, 551
ASL_dmumqg : Vol.5, 469
ASL_dmumqn : Vol.5, 465
ASL_dmussn : Vol.5, 474
ASL_dmuusn : Vol.5, 462

ASL_dndaao : Vol.4, 362
ASL_dndanl : Vol.4, 372
ASL_dndapo : Vol.4, 367
Ledngapl :Vol.4, 386

ASL_dnlnrg : Vol.6, 592
ASL_dnlnrr : Vol.6, 598
ASL_dnnlgf : Vol.6, 617
ASL_dnnlpo : Vol.6, 611

ASL_dofnnf : Vol.4, 115
ASL_dofnnv : Vol.4, 108
ASL_dohnlv : Vol.4, 136
ASL_dohnnf : Vol.4, 129
Vol.4, 122

ASL_doiev1 : Vol.4, 153
ASL_dolnlv : Vol.4, 143
ASL_dopdh2 : Vol.4, 157
ASL_dopdh3 : Vol.4, 165

ASL_dosnnv : Vol.4, 91
ASL_dpdapn : Vol.4, 347
ASL_dpdopl : Vol.4, 343
ASL_dpgopl : Vol.4, 358
Vol.4, 351

ASL_dqmogx : Vol.4, 186

| ASL_dqmo | Vol.4, 190 |
| :---: | :---: |
| ASL_dqmoj | Vol.4, 194 |
| ASL_dsmgo | Vol.5, 348 |
| ASL_dsmgp | Vol.5, 352 |
| ASL_dssta | Vol.5, 331 |
| ASL_dssta | Vol.5, 335 |
| ASL_dsstp | Vol.5, 344 |
| ASL_dsstr | Vol.5, 340 |
| ASL_dxa00 | Vol.1, 47 |
| ASL_gam1 | SMP Functions ${ }^{(*)}$ |
| ASL_gam1 | SMP Functions, 44 |
| ASL_gam1m | SMP Functions, 39 |
| ASL_gam1m | SMP Functions, 34 |
| ASL_gan1 | SMP Functions, 66 |
| ASL_gan1 | SMP Functions, 62 |
| ASL_gan1m | SMP Functions, 58 |
| ASL_gan1m | SMP Functions, 54 |
| ASL_gbhes | SMP Functions, 156 |
| ASL_gbheu | SMP Functions, 161 |
| ASL_gbhfs | SMP Functions, 149 |
| ASL_gbhfu | SMP Functions, 154 |
| ASL_gbhps | SMP Functions, 133 |
| ASL_gbhpu | SMP Functions, 139 |
| ASL_gbhrs | SMP Functions, 141 |
| ASL_gbhru | SMP Functions, 147 |
| ASL_gcgj | SMP Functions, 302 |
| ASL_gcgj | SMP Functions, 307 |
| ASL_gcgka | SMP Functions, 309 |
| ASL_gcgk | SMP Functions, 314 |
| ASL_gcgr | SMP Functions, 294 |
| ASL_gcgran | SMP Functions, 299 |
| ASL_gchea | SMP Functions, 249 |
| ASL_gchea | SMP Functions, 253 |
| ASL_gches | SMP Functions, 261 |
| ASL_gches | SMP Functions, 255 |
| ASL_gchra | SMP Functions, 233 |
| ASL_gchra | SMP Functions, 238 |
| ASL_gchrs | SMP Functions, 246 |
| ASL_gchrs | SMP Functions, 240 |
| ASL_gfc2b | SMP Functions, 371 |
| ASL_gfc2f | SMP Functions, 367 |
| ASL_gfc3b | SMP Functions, 401 |
| ASL_gfc3 | SMP Functions, 397 |
| ASL_gf cmb | SMP Functions, 338 |
| ASL_gf cmf | SMP Functions, 334 |
| ASL_ham1h | SMP Functions, 49 |
| ASL_ham1h | SMP Functions, 44 |
| ASL_ham1m | SMP Functions, 39 |
| ASL_ham1m | SMP Functions, 34 |
| ASL_han1h | SMP Functions, 66 |
| ASL_han1h | SMP Functions, 62 |

[^1]ASL_han1mh : SMP Functions, 58
ASL_han1mm : SMP Functions, 54
ASL_hbgmlc : SMP Functions, 105
ASL_hbgmlu : SMP Functions, 103
ASL_hbgmsl : SMP Functions, 98
ASL_hbgmsm : SMP Functions, 92
ASL_hbgnlc : SMP Functions, 117
ASL_hbgnlu : SMP Functions, 115
ASL_hbgnsl : SMP Functions, 111
ASL_hbgnsm : SMP Functions, 107
ASL_hbhesl : SMP Functions, 156
ASL_hbheud : SMP Functions, 161
ASL_hbhfsl : SMP Functions, 149
ASL_hbhfud : SMP Functions, 154
ASL_hbhpsl : SMP Functions, 133
ASL_hbhpud : SMP Functions, 139
ASL_hbhrsl : SMP Functions, 141
ASL_hbhrud : SMP Functions, 147
ASL_hcgjaa : SMP Functions, 302
ASL_hcgjan : SMP Functions, 307
ASL_hcgkaa : SMP Functions, 309
ASL_hcgkan : SMP Functions, 314
ASL_hcgraa : SMP Functions, 294
ASL_hcgran : SMP Functions, 299
ASL_hcheaa : SMP Functions, 249
ASL_hchean : SMP Functions, 253
ASL_hchesn : SMP Functions, 261
ASL_hchess : SMP Functions, 255
ASL_hchraa : SMP Functions, 233
ASL_hchran : SMP Functions, 238
ASL_hchrsn : SMP Functions, 246
ASL_hchrss : SMP Functions, 240
ASL_hfc2bf : SMP Functions, 371
ASL_hfc2fb : SMP Functions, 367
ASL_hfc3bf : SMP Functions, 401
ASL_hfc3fb : SMP Functions, 397
ASL_hfcmbf : SMP Functions, 338
ASL_hfcmfb : SMP Functions, 334
ASL_iiierf : Vol.5, 283
ASL_jiierf : Vol.5, 283
ASL_pam1mm : SMP Functions, 18
ASL_pam1mt : SMP Functions, 22
ASL_pam1mu : SMP Functions, 14
ASL_pam1tm : SMP Functions, 26
ASL_pam1tt : SMP Functions, 30
ASL_pbsnsl : SMP Functions, 126
ASL_pbsnud : SMP Functions, 131
ASL_pbspsl : SMP Functions, 119
ASL_pbspud : SMP Functions, 124
ASL_pcgjaa : SMP Functions, 282
ASL_pcgjan : SMP Functions, 286
ASL_pcgkaa : SMP Functions, 288
ASL_pcgkan : SMP Functions, 292
ASL_pcgsaa : SMP Functions, 264

| ASL_pcgsan : SMP Functions, 270 |
| :--- |
| ASL_pcgssn : SMP Functions, 279 |
| ASL_pcgsss : SMP Functions, 272 |
| ASL_pcsmaa : SMP Functions, 220 |
| ASL_pcsman : SMP Functions, 224 |
| ASL_pcsmsn : SMP Functions, 231 |
| ASL_pcsmss : SMP Functions, 226 |
| ASL_pfc2bf : SMP Functions, 362 |
| ASL_pfc2fb : SMP Functions, 358 |
| ASL_pfc3bf : SMP Functions, 390 |
| ASL_pfc3fb : SMP Functions, 386 |
| ASL_pfcmbf : SMP Functions, 326 |
| ASL_pfcmfb : SMP Functions, 322 |
| ASL_pfcn2d : SMP Functions, 419 |
| ASL_pfcn3d : SMP Functions, 427 |
| ASL_pfcr2d : SMP Functions, 437 |
| ASL_pfcr3d : SMP Functions, 445 |
| ASL_pfps2d : SMP Functions, 456 |
| ASL_pfps3d : SMP Functions, 465 |
| ASL_pfr2bf : SMP Functions, 380 |
| ASL_pfr2fb : SMP Functions, 376 |
| ASL_pfr3bf : SMP Functions, 412 |
| ASL_pfr3fb : SMP Functions, 408 |
| ASL_pfrmbf : SMP Functions, 350 |
| ASL_pfrmfb : SMP Functions, 346 |
| ASL_pssta1 : SMP Functions, 484 |
| ASL_pssta2 : SMP Functions, 488 |
| ASL_pxe010 : SMP Functions, 174 |
| ASL_pxe020 : SMP Functions, 183 |
| ASL_pxe030 : SMP Functions, 192 |
| ASL_pxe040 : SMP Functions, 202 |

ASL_pcgssn : SMP Functions, 279
ASt_pcgsss SMP Functions, 272

ASL_pcsman : SMP Functions, 224
ASL_pcsmsn : SMP Functions, 231
ASL_pcsmss : SMP Functions, 226
ASL_pfc2bf : SMP Functions, 362

ASL_pfc3bf : SMP Functions, 390
ASL_pfc3fb : SMP Functions, 386
ASL_pfcmbf : SMP Functions, 326
ASL_pfcmfb : SMP Functions, 322
ASL_pfen2d : SMP Functions, 41
ASL_pfcr2d : SMP Functions, 437
ASL_pfcr3d : SMP Functions, 445
ASL_pfps2d : SMP Functions, 456
ASL_pfps3d: SMP Functions, 465

ASL_pfr2fb : SMP Functions, 376
ASL_pfr3bf : SMP Functions, 412
ASL_pfr3fb : SMP Functions, 408
SMP Functions, 350

ASL_pssta1 : SMP Functions, 484
ASL_pssta2 : SMP Functions, 488
ASL_pxe010 : SMP Functions, 174
ASL_pxe020 : SMP Functions, 183
SMP Functions, 192

ASL_qam1mm : SMP Functions, 18
ASL_qam1mt : SMP Functions, 22
ASL_qam1mu : SMP Functions, 14

ASL_qbgmlc : SMP Functions, 90
ASL_qbgmlu : SMP Functions, 88
ASL_qbgmsl : SMP Functions, 83
_qbgms

ASL_qbsnud : SMP Functions, 131
ASL_qbspsl : SMP Functions, 119
ASL_qbspud : SMP Functions, 124
ASL_qcgjaa : SMP Functions, 282
ASL_qcgjan : SMP Functions, 286
ASL_qcgkaa : SMP Functions, 288
ASL_qcgsaa : SMP Functions, 264
ASL_qcgsan : SMP Functions, 270
ASL_qcgssn : SMP Functions, 279
ASL_qcsmaa : SMP Functions, 220
ASL_qcsman : SMP Functions, 224

ASL_qcsmsn : SMP Functions, 231
ASL_qcsmss : SMP Functions, 226
ASL_qfc2bf : SMP Functions, 362
ASL_qfc2fb : SMP Functions, 358
ASL_qfc3bf : SMP Functions, 390
ASL_qfc3fb : SMP Functions, 386
ASL_qfcmbf : SMP Functions, 326
ASL_qfcmfb : SMP Functions, 322
ASL_qfcn2d : SMP Functions, 419
ASL_qfcn3d : SMP Functions, 427
ASL_qfcr2d : SMP Functions, 437
ASL_qfcr3d : SMP Functions, 445
ASL_qfps2d : SMP Functions, 456
ASL_qfps3d : SMP Functions, 465
ASL_qfr2bf : SMP Functions, 380
ASL_qfr2fb : SMP Functions, 376
ASL_qfr3bf : SMP Functions, 412
ASL_qfr3fb : SMP Functions, 408
ASL_qfrmbf : SMP Functions, 350
ASL_qfrmfb : SMP Functions, 346
ASL_qssta1 : SMP Functions, 484
ASL_qssta2 : SMP Functions, 488
ASL_qxe010 : SMP Functions, 174
ASL_qxe020 : SMP Functions, 183
ASL_qxe030 : SMP Functions, 192
ASL_qxe040 : SMP Functions, 202
ASL_r1cdbn : Vol.6, 79
ASL_r1cdbt : Vol.6, 123
ASL_r1cdcc : Vol.6, 160
ASL_r1cdch : Vol.6, 83
ASL_r1cdex : Vol.6, 145
ASL_r1cdfb : Vol.6, 109
ASL_r1cdgm : Vol.6, 116
ASL_r1cdgu : Vol.6, 148
ASL_r1cdib : Vol.6, 127
ASL_r1cdic : Vol.6, 86
ASL_r1cdif : Vol.6, 113
ASL_r1cdig : Vol.6, 120
ASL_r1cdin : Vol.6, 76
ASL_r1cdis : Vol.6, 106
ASL_r1cdit : Vol.6, 99
ASL_r1cdix : Vol.6, 93
ASL_r1cdld : Vol.6, 151
ASL_r1cdlg : Vol.6, 157
ASL_r1cdln : Vol.6, 154
ASL_r1cdnc : Vol.6, 89
ASL_r1cdno : Vol.6, 73
ASL_r1cdnt : Vol.6, 102
ASL_r1cdpa : Vol.6, 137
ASL_r1cdtb : Vol.6, 96
ASL_r1cdtr : Vol.6, 134
ASL_r1cduf : Vol.6, 131
ASL_r1cdwe : Vol.6, 141
ASL_r1ddbp : Vol.6, 164

|  |  |
| :---: | :---: |
| ASL_r1ddhg | Vol.6, 174 |
|  | Vol.6, 177 |
| ASL_r1ddpo | Vol.6, 171 |
| ASL_r2ba1t | Vol.6, 188 |
| ba | Vol.6, 195 |
|  | , |
|  | Vol.6, 219 |
| SL_r2bamo | Vol.6, 215 |
| ams | Vo |
| basm | Vol.6, 22 |
| SL_r2ccma | Vol.6, 249 |
| SL_r2ccmt | Vol |
|  | Vol.6, |
| SL_r2vcgr | Vo |
| SL_r2vcmt | Vo |
| -_3i | Vol.6, 337 |
| SL_r3ieme | Vol.6, |
| SL_r3iera | Vo |
| L_r3iesr | Vol.6, |
| $3 i e s u$ | Vol.6, |
| SL_r3ietc | Vol.6, |
| SL_r3ieva | Vol.6, |
| ASL_r3tscd | Vol.6, |
| t | Vol.6, |
| ASL_r3tsra | Vol.6, |
| ASL_r3tsrd | Vol.6, 352 |
| L_r3ts | Vol.6, |
| t | Vol.6, |
| c | Vol.6, 373 |
| t | Vol.6, 369 |
| SLr41 | Vol.6, 3 |
| L_r42wr1 | Vol.6, |
| L_r42 | Vol.6, 40 |
| L_r42w | Vol.6, 403 |
| ASL_r4bi01 | Vol.6, 4 |
| , | Vol.6, |
| SL_r4mu01 | Vol.6, 452 |
| ASL_r4mwrf | Vol.6, 426 |
| ASL_r4mwrm | Vol.6, 439 |
| _r4rb01 | Vol.6, 468 |
| r5chef | Vol.6, 48 |
| d | Vol.6, 49 |
| c | Vol.6, 493 |
| L_r5ch | Vol.6, 490 |
| L_r5tem | Vol.6, 509 |
| 5tesg | Vol.6, 501 |
| L_r5tesp | Vol.6, 513 |
| L_r5t | Vol.6, 505 |
| r6clan | Vol.6, 57 |
| _r6cld | Vol.6, 576 |
| L_r6clds | Vol.6, 565 |
| SL_r6cpcc | Vol.6, 526 |
| c | Vol.6, 52 |

ASL_r1ddhg : Vol.6, 174
ASL_r1ddhn : Vol.6, 177
ASL_r1ddpo : Vol.6, 171
ASL_r2ba1t : Vol.6, 188
ASL_r2ba2s : Vol.6, 195
ASL_r2bagm : Vol.6, 210
ASL_r2bahm : Vol.6, 219

ASL_r2bams : Vol.6, 204
ASL_r2basm : Vol.6, 223
ASL_r2ccma : Vol.6, 249
r2ccmt Vol.6, 243

ASL_r2vcgr : Vol.6, 233
ASL_r2vcmt : Vol.6, 227
ASL_r3iecd : Vol.6, 337
ASL_r3ieme : Vol.6, 322
ASL_r3iera : Vol.6, 319

ASL_r3iesu : Vol.6, 326
ASL_r3ietc : Vol.6, 333
ASL_r3ieva : Vol.6, 330
ASL_r3tscd : Vol.6, 380
ASL_r3tsra : Vol.6, 348
ASL_r3tsrd : Vol.6, 352
ASL_r3tssr : Vol.6, 383
ASL_r3tssu : Vol.6, 362
ASL_r3tstc : Vol.6, 373

ASL_r41wr1 : Vol.6, 397
ASL_r42wr1 : Vol.6, 417
ASL_r42wrm : Vol.6, 409
ASL_r42wrn : Vol.6, 403

ASL_r4gl01 : Vol.6, 472
ASL_r4mu01 : Vol.6, 452
ASL_r4mwrf : Vol.6, 426
r4mwrm :Vol.6, 439

ASL_r5chef : Vol.6, 486
ASL_r5chmd : Vol.6, 497
ASL_r5chmn : Vol.6, 493
ASL_r5chtt : Vol.6, 490
ASL_r5temh : Vol.6, 509

ASL_r5tesp : Vol.6, 513
ASL_r5tewl : Vol.6, 505
ASL_r6clan : Vol.6, 571
ASL_r6clda : Vol.6, 576

ASL_r6cpcc : Vol.6, 526
ASL_r6cpsc : Vol.6, 528

ASL_r6cvan : Vol.6, 543
ASL_r6cvsc : Vol.6, 546
ASL_r6dafn : Vol.6, 553
ASL_r6dasc : Vol.6, 557
ASL_r6fald : Vol.6, 535
ASL_r6favr : Vol.6, 537
ASL_rabmcs : Vol.1, 13
ASL_rabmel : Vol.1, 17
ASL_ram1ad : Vol.1, 55
ASL_ram1mm : Vol.1, 75
ASL_ram1ms : Vol.1, 65
ASL_ram1mt : Vol.1, 79
ASL_ram1mu : Vol.1, 61
ASL_ram1sb : Vol.1, 58
ASL_ram1tm : Vol.1, 83
ASL_ram1tp : Vol.1, 136
ASL_ram1tt : Vol.1, 87
ASL_ram1vm : Vol.1, 127
ASL_ram3tp : Vol.1, 139
ASL_ram3vm : Vol.1, 130
ASL_ram4vm : Vol.1, 133
ASL_ramt1m : Vol.1, 69
ASL_ramvj1 : Vol.1, 143
ASL_ramvj3 : Vol.1, 147
ASL_ramvj4 : Vol.1, 151
ASL_rargjm : Vol.1, 32
ASL_rarsjd : Vol.1, 26
ASL_rasbcs : Vol.1, 20
ASL_rasbel : Vol.1, 23
ASL_ratm1m : Vol.1, 72
ASL_rbbddi : Vol.2, 255
ASL_rbbdlc : Vol.2, 250
ASL_rbbdls : Vol.2, 253
ASL_rbbdlu : Vol.2, 248
ASL_rbbdlx : Vol.2, 257
ASL_rbbdsl : Vol.2, 243
ASL_rbbpdi : Vol.2, 272
ASL_rbbpls : Vol.2, 270
ASL_rbbplx : Vol.2, 274
ASL_rbbpsl : Vol.2, 262
ASL_rbbpuc : Vol.2, 268
ASL_rbbpuu : Vol.2, 266
ASL_rbgmdi : Vol.2, 52
ASL_rbgmlc : Vol.2, 44
ASL_rbgmls : Vol.2, 46
ASL_rbgmlu : Vol.2, 42
ASL_rbgmlx : Vol.2, 54
ASL_rbgmms : Vol.2, 48
ASL_rbgmsl : Vol.2, 37
ASL_rbgmsm : Vol.2, 32
ASL_rbpddi : Vol.2, 116
ASL_rbpdls : Vol.2, 114
ASL_rbpdlx : Vol.2, 118
ASL_rbpdsl : Vol.2, 106

| ASL_rbpduc : Vol.2, 112 |
| :--- |
| ASL_rbpduu : Vol.2, 110 |
| ASL_rbsmdi : Vol.2, 154 |
| ASL_rbsmls : Vol.2, 148 |
| ASL_rbsmlx : Vol.2, 156 |
| ASL_rbsmms : Vol.2, 150 |
| ASL_rbsmsl : Vol.2, 139 |
| ASL_rbsmuc : Vol.2, 146 |
| ASL_rbsmud : Vol.2, 144 |
| ASL_rbsnls : Vol.2, 165 |
| ASL_rbsnsl : Vol.2, 158 |
| ASL_rbsnud : Vol.2, 163 |
| ASL_rbspdi : Vol.2, 135 |
| ASL_rbspls : Vol.2, 129 |
| ASL_rbsplx : Vol.2, 137 |
| ASL_rbspms : Vol.2, 131 |
| ASL_rbspsl : Vol.2, 120 |
| ASL_rbspuc : Vol.2, 127 |
| ASL_rbspud : Vol.2, 125 |
| ASL_rbtdsl : Vol.2, 276 |
| ASL_rbtlco : Vol.2, 324 |
| ASL_rbtldi : Vol.2, 326 |
| ASL_rbtlsl : Vol.2, 321 |
| ASL_rbtosl : Vol.2, 302 |
| ASL_rbtpsl : Vol.2, 280 |
| ASL_rbtssl : Vol.2, 306 |
| ASL_rbtuco : Vol.2, 317 |
| ASL_rbtudi : Vol.2, 319 |
| ASL_rbtusl : Vol.2, 314 |
| ASL_rbvmsl : Vol.2, 310 |
| ASL_rcgbff : Vol.1, 400 |
| ASL_rcgeaa : Vol.1, 177 |
| ASL_rcgean : Vol.1, 183 |
| ASL_rcggaa : Vol.1, 328 |
| ASL_rcggan : Vol.1, 335 |
| ASL_rcgjaa : Vol.1, 360 |
| ASL_rcgjan : Vol.1, 364 |
| ASL_rcgkaa : Vol.1, 366 |
| ASL_rcgkan : Vol.1, 370 |
| ASL_rcgnaa : Vol.1, 185 |
| ASL_rcgnan : Vol.1, 189 |
| ASL_rcgsaa : Vol.1, 337 |
| ASL_rcgsan : Vol.1, 342 |
| ASL_rcgsee : Vol.1, 352 |
| ASL_rcgsen : Vol.1, 358 |
| ASL_rcgssn : Vol.1, 350 |
| ASL_rcgsss : Vol.1, 344 |
| ASL_rcsbaa : Vol.1, 264 |
| ASL_rcsban : Vol.1, 268 |
| ASL_rcsbff : Vol.1, 277 |
| ASL_rcsbsn : Vol.1, 275 |
| ASL_rcsbss : Vol.1, 270 |
| ASLe Vol.1, 311 |

ASL_rbpduu : Vol.2, 110
ASL_rbsmdi : Vol.2, 154
ASL_rbsmls : Vol.2, 148
ASL_rbsmlx : Vol.2, 156
ASL_rbsmms : Vol.2, 150
ASL_rbsmsl : Vol.2, 139
AS_rbsmuc : Vol.2, 146

ASL_rbsnls : Vol.2, 165
ASL_rbsnsl : Vol.2, 158
ASL_rbsnud : Vol.2, 163
ASL_rbspdi : Vol.2, 135
ASL_rbspls : Vol.2, 129
ASL_rbspms : Vol.2, 131
ASL_rbspsl : Vol.2, 120
ASL_rbspuc : Vol.2, 127
ASL_rbspud : Vol.2, 125
AS_rbtasl: Vol.2, 276
ASL_rbtldi : Vol.2, 326
ASL_rbtlsl : Vol.2, 321
ASL_rbtosl : Vol.2, 302

ASL_rbtssl : Vol.2, 306
ASL_rbtuco : Vol.2, 317
ASL_rbtudi : Vol.2, 319
ASL_rbtusl : Vol.2, 314
ASLrbvms1 : Vol.2, 310

ASL_rcgeaa : Vol.1, 177
ASL_rcgean : Vol.1, 183
ASL_rcggaa : Vol.1, 328
ASL_rcggan : Vol.1, 335
ASL_rcgjaa : Vol.1, 360
ASL_rcgkaa : Vol.1, 366
ASL_rcgkan : Vol.1, 370
Vol.1, 185

ASL_rcgsaa : Vol.1, 337
ASL_rcgsan : Vol.1, 342
ASL_rcgsee : Vol.1, 352
ASL_rcgsen : Vol.1, 358
ASL_rcgssn : Vol.1, 350
ASL_rcgsss : Vol.1, 344
ASL_rcsban : Vol.1, 268
ASL_rcsbff : Vol.1, 277
ASL_rcsbsn : Vol.1, 275

ASL_rcsjss : Vol.1, 311
ASL_rcsmaa : Vol.1, 204

| $\begin{aligned} & \text { ASL_rcsme } \\ & \text { ASL_rcsme } \end{aligned}$ |  |
| :---: | :---: |
| rcs | 222 |
| L_resi | 1, 215 |
| L | 1, 210 |
| L_rcsr | 1, 303 |
| L_rc | 283 |
| L_rcs | 1, 287 |
| L_ | 1, 296 |
| L_ | l.1, 301 |
| L_rcst | ol.1, 294 |
| L_rcst | l.1, 289 |
| rf | l.6, 285 |
| ASL_rfc | 1.3, 50 |
| L_rf | 1.3, 46 |
| L_rfc2b | ol.3, 117 |
| L_rfc2f | ol.3, 113 |
| L_rf | 1.3, |
| L_r | .1.3, 142 |
| L_rfcmb | ol.3, 81 |
| L_rfcmf | Vol.3, 77 |
| L_rfcn | ol.3, 177 |
| rf | l.3, 187 |
| L_rfon | ol.3, 195 |
| L_rfcr | ol.3, 206 |
| SL_rfcr2d | -1.3, 216 |
| SL_rfcr | ol.3, 224 |
| L_r | l.6, 283 |
| L_r | ol.6, 281 |
| L_rfcr | ol.6, 279 |
| L_rfcv | ol.6, 274 |
| L_ | , 6, 269 |
| SL_rfdped | l.6, 291 |
| SL_rfdpe | ol.6, 289 |
| SL_rfdpe | ol.6, 294 |
| SL_rflag | l.3, 273 |
| L_rfla | 1.3, |
| SL_rfps1 | ol.3, 235 |
| SL_rfps2 | ol.3, 243 |
| SL_rfps3 | ol.3, 252 |
| L_rfr1 | ol.3, 71 |
| L_rfr1f | Vol.3, 67 |
| SL_rfr2b | Vol.3, 136 |
| L_rfr2 | Vol.3, 132 |
| L_rfr3 | Vol.3, 169 |
| SL_rfr3 | l.3, 164 |
| SL_rfrmb | Vol.3, 106 |
| SL_rfrmf | ol.3, 101 |
| SL_rfwtf | Vol.3, 306 |
| S_rfwt | Vol.3, 308 |
| SL_rfwth | ol.3, 277 |
| ASL_rfwth | Vol.3, 289 |
| _rfw | Vol.3, 296 |
| L_rfw | .3, 280 |


| ASL_rfwths : Vol.3, 284 |
| :--- |
| ASL_rfwtht : Vol.3, 292 |
| ASL_rfwtmf : Vol.3, 301 |
| ASL_rfwtmt : Vol.3, 303 |
| ASL_rgicbp : Vol.4, 513 |
| ASL_rgicbs : Vol.4, 537 |
| ASL_rgiccm : Vol.4, 483 |
| ASL_rgiccn : Vol.4, 486 |
| ASL_rgicco : Vol.4, 478 |
| ASL_rgiccp : Vol.4, 469 |
| ASL_rgiccq : Vol.4, 471 |
| ASL_rgiccr : Vol.4, 473 |
| ASL_rgiccs : Vol.4, 475 |
| ASL_rgicct : Vol.4, 480 |
| ASL_rgidby : Vol.4, 517 |
| ASL_rgidcy : Vol.4, 492 |
| ASL_rgidmc : Vol.4, 445 |
| ASL_rgidpc : Vol.4, 432 |
| ASL_rgidsc : Vol.4, 438 |
| ASL_rgidyb : Vol.4, 503 |
| ASL_rgiibz : Vol.4, 519 |
| ASL_rgiicz : Vol.4, 495 |
| ASL_rgiimc : Vol.4, 463 |
| ASL_rgiipc : Vol.4, 452 |
| ASL_rgiisc : Vol.4, 457 |
| ASL_rgiizb : Vol.4, 509 |
| ASL_rgisbx : Vol.4, 515 |
| ASL_rgiscx : Vol.4, 490 |
| ASL_rgisi1 : Vol.4, 540 |
| ASL_rgisi2 : Vol.4, 545 |
| ASL_rgisi3 : Vol.4, 554 |
| ASL_rgismc : Vol.4, 426 |
| ASL_rgispc : Vol.4, 416 |
| ASL_rgispo : Vol.4, 521 |
| ASL_rgispr : Vol.4, 525 |
| ASL_rgiss1 : Vol.4, 561 |
| ASL_rgiss2 : Vol.4, 566 |
| ASL_rgiss3 : Vol.4, 574 |
| ASL_rgissc : Vol.4, 420 |
| ASL_rgisso : Vol.4, 529 |
| ASL_rgissr : Vol.4, 533 |
| ASL_rgisxb : Vol.4, 497 |
| ASL_rh2int : Vol.4, 299 |
| ASL_rhbdfs : Vol.4, 264 |
| ASL_rhbsfc : Vol.4, 267 |
| ASL_rhemnh : Vol.4, 270 |
| ASL_rhemni : Vol.4, 287 |
| ASL_rhemnl : Vol.4, 223 |
| ASL_rhnanl : Vol.4, 259 |
| ASL_rhnefl : Vol.4, 235 |
| ASL_rhnenh : Vol.4, 279 |
| AShnenl : Vol.4, 250 |

ASL_rfwtht : Vol.3, 292
ASL_rfwtmf : Vol.3, 301
ASL_rfwtmt : Vol.3, 303
ASL_rgicbp : Vol.4, 513
ASL_rgicbs : Vol.4, 537
ASL_rgiccm : Vol.4, 483
ASL_rgiccn : Vol.4, 486

ASL_rgiccp : Vol.4, 469
ASL_rgiccq : Vol.4, 471
ASL_rgiccr : Vol.4, 473
ASL_rgiccs : Vol.4, 475
ASL_rgicct : Vol.4, 480
ASL_rgidcy : Vol.4, 492
ASL_rgidmc : Vol.4, 445
ASL_rgidpc : Vol.4, 432
ASL_rgidsc: Vol.4, 438

ASL_rgiibz : Vol.4, 519
ASL_rgiicz : Vol.4, 495
ASL_rgiimc : Vol.4, 463
ASL_rgiipc : Vol.4, 452
ASL_rgiizb : Vol.4, 509
ASL_rgisbx : Vol.4, 515
ASL_rgiscx : Vol.4, 490
ASL_rgisi1 : Vol.4, 540
ASL_rgisi2 : Vol.4, 545

ASL_rgismc : Vol.4, 426
ASL_rgispc : Vol.4, 416
ASL_rgispo : Vol.4, 521
ASL_rgispr : Vol.4, 525
ASL_rgiss1 : Vol.4, 561
ASL_rgiss3 : Vol.4, 574
ASL_rgissc : Vol.4, 420
ASLrgisso Vol.4, 529

ASL_rgisxb : Vol.4, 497
ASL_rh2int : Vol.4, 299
ASL_rhbdfs : Vol.4, 264
ASL_rhbsfc : Vol.4, 267
Vol.4, 270

ASL_rhemnl : Vol.4, 223
ASL_rhnanl : Vol.4, 259
ASL_rhnefl : Vol.4, 235
ASL_rhnenh : Vol.4, 279
ASL_rhnfml : Vol.4, 317
ASL_rhnfnm : Vol.4, 307

ASL_rhnifl : Vol.4, 240
ASL_rhninh : Vol.4, 283
ASL_rhnini : Vol.4, 295
ASL_rhninl : Vol.4, 255
ASL_rhnofh : Vol.4, 274
ASL_rhnofi : Vol.4, 291
ASL_rhnofl : Vol.4, 230
ASL_rhnpnl : Vol.4, 245
ASL_rhnrml : Vol.4, 312
ASL_rhnrnm : Vol.4, 302
ASL_rhnsnl : Vol.4, 227
ASL_ribaid : Vol.5, 189
ASL_ribaix : Vol.5, 185
ASL_ribbei : Vol.5, 167
ASL_ribber : Vol.5, 165
ASL_ribbid : Vol.5, 191
ASL_ribbix : Vol.5, 187
ASL_ribimx : Vol.5, 135
ASL_ribinx : Vol.5, 129
ASL_ribjmx : Vol.5, 90
ASL_ribjnx : Vol.5, 84
ASL_ribkei : Vol.5, 171
ASL_ribker : Vol.5, 169
ASL_ribkmx : Vol.5, 138
ASL_ribknx : Vol.5, 132
ASL_ribsin : Vol.5, 153
ASL_ribsjn : Vol.5, 147
ASL_ribskn : Vol.5, 156
ASL_ribsyn : Vol.5, 150
ASL_ribymx : Vol.5, 93
ASL_ribynx : Vol.5, 87
ASL_rieii1 : Vol.5, 221
ASL_rieii2 : Vol.5, 223
ASL_rieii3 : Vol.5, 226
ASL_rieii4 : Vol.5, 228
ASL_rigig1 : Vol.5, 199
ASL_rigig2 : Vol.5, 202
ASL_riicos : Vol.5, 261
ASL_riierf : Vol.5, 281
ASL_riisin : Vol.5, 259
ASL_rileg1: Vol.5, 285
ASL_rileg2 : Vol.5, 288
ASL_rimtce : Vol.5, 306
ASL_rimtse : Vol.5, 309
ASL_riopc2 : Vol.5, 302
ASL_riopch : Vol.5, 300
ASL_riopgl : Vol.5, 304
ASL_riophe : Vol.5, 298
ASL_riopla : Vol.5, 296
ASL_riople : Vol.5, 291
ASL_rixeps : Vol.5, 324
ASL_rizbs0 : Vol.5, 102
ASL_rizbs1 : Vol.5, 105
ASL_rizbsl : Vol.5, 114
ASL_rizbsn : Vol.5, 108
ASL_rizbyn : Vol.5, 111
ASL_rizglw : Vol.5, 293
ASL_rjtebi : Vol.6, 53
ASL_rjtecc : Vol.6, 33
ASL_rjteex : Vol.6, 29
ASL_rjtegm : Vol.6, 45
ASL_rjtegu : Vol.6, 37
ASL_rjtelg : Vol.6, 49
ASL_rjteng : Vol.6, 57
ASL_rjteno : Vol.6, 25
ASL_rjtepo : Vol.6, 61
ASL_rjteun : Vol.6, 20
ASL_rjtewe : Vol.6, 41
ASL_rkfncs : Vol.4, 72
ASL_rkhncs : Vol.4, 78
ASL_rkinct : Vol.4, 55
ASL_rkmncn : Vol.4, 84
ASL_rksnca : Vol.4, 49
ASL_rksncs : Vol.4, 43
ASL_rkssca : Vol.4, 65
ASL_rlarha : Vol.5, 388
ASL_rlnrds : Vol.5, 396
ASL_rlnris : Vol.5, 400
ASL_rlnrsa : Vol.5, 406
ASL_rlnrss : Vol.5, 403
ASL_rlsrds : Vol.5, 414
ASL_rlsris : Vol.5, 421
ASL_rmclaf : Vol.5, 490
ASL_rmclcp : Vol.5, 517
ASL_rmclmc : Vol.5, 511
ASL_rmclmz : Vol.5, 502
ASL_rmclsn : Vol.5, 483
ASL_rmcltp : Vol.5, 524
ASL_rmcqaz : Vol.5, 544
ASL_rmcqlm : Vol.5, 538
ASL_rmcqsn : Vol.5, 531
ASL_rmcusn : Vol.5, 479
ASL_rmsp11 : Vol.5, 567
ASL_rmsp1m : Vol.5, 558
ASL_rmspmm : Vol.5, 563
ASL_rmsqpm : Vol.5, 551
ASL_rmumqg : Vol.5, 469
ASL_rmumqn : Vol.5, 465
ASL_rmussn : Vol.5, 474
ASL_rmuusn : Vol.5, 462
ASL_rncbpo : Vol.4, 392
ASL_rndaao : Vol.4, 362
ASL_rndanl : Vol.4, 372
ASL_rndapo : Vol.4, 367
ASL_rngapl : Vol.4, 386
ASL_rnlnma : Vol.6, 605
ASL_rnlnrg : Vol.6, 592
ASL_rnlnrr : Vol.6, 598
ASL_rizbyn : Vol.5, 111
ASL_rizglw : Vol.5, 293
ASL_rjtebi : Vol.6, 53
ASL_rjtecc : Vol.6, 33
ASL_rjteex : Vol.6, 29
ASL_rjtegm : Vol.6, 45
ASL_rjtelg : Vol.6, 49
ASL_rjteng : Vol.6, 57
ASL_rjteno : Vol.6, 25
ASL_rjtepo : Vol.6, 61
ASL_rjtewe : Vol.6, 41
ASL_rkfncs : Vol.4, 72
ASL_rkhncs : Vol.4, 78
ASL_rkinct : Vol.4, 55
Vol.4, 84
ASL_rksncs : Vol.4, 43
ASL_rkssca : Vol.4, 65
ASL_rlarha : Vol.5, 388
ASL_rlnrds : Vol.5, 396
ASL_rlnris : Vol.5, 400
ASL_rlnrss : Vol.5, 403
ASL_rlsrds : Vol.5, 414
ASL_rlsris : Vol.5, 421
ASL_rmclcp : Vol.5, 517
ASL_rmclmc : Vol.5, 511
ASL_rmclmz : Vol.5, 502
ASL_rmclsn : Vol.5, 483
ASL_rmcltp : Vol.5, 524
ASL_rmcqlm : Vol.5, 538
ASL_rmcqsn : Vol.5, 531
ASL_rmcusn : Vol.5, 479
ASL_rmsp11 : Vol.5, 567
ASL_rmspmm : Vol.5, 563
ASL_rmsqpm : Vol.5, 551
ASL_rmumqg : Vol.5, 469
ASL_rmumqn : Vol.5, 465
ASL_rmussn : Vol.5, 474
ASL_rmuusn : Vol.5, 462
ASL_rncbpo : Vol.4, 392
ASL_rndaao : Vol.4, 362
ASL_rndanl : Vol.4, 372
ASL_rndapo : Vol.4, 367
ASL_rngapl : Vol.4, 386
ASL_rnlnrg : Vol.6, 592
ASL_rnlnrr : Vol.6, 598

| ASL_rnn | Vol.6, 617 |
| :---: | :---: |
| SL | 4, 379 |
| L | Vol.4, 115 |
| ASL_rof | 8 |
| ASL_roh | Vol.4, 136 |
| ASL_roh | Vol.4, 129 |
| ASL_roh | ,ol |
| ASL_ro |  |
| ASL_ro | O |
| ASL_rol | Vol.4, 143 |
| ASL_rop | Vol.4, 157 |
| _rop | Vol |
| L_rosn | Vol |
|  |  |
| _rpd | Vol.4, |
| rpdo | Vo |
| L_rpgo | Vo |
| ASL_rplop | Vo |
| rqf | Vo |
| L_rqmo | Vo |
| _rqm | Vol. |
| rqmo | Vol. 4 |
| ASL_rsmg | Vol.5, |
| SL_rsmg | Vol.5, |
| ASL rss | Vol |
| ASL_rss | Vol.5, |
| ASL_rss | Vol.5, |
| ASL_rss | Vol.5, |
| ASL_rx | Vol |
| ASL_vib | Vol. 5 |
| ASL vib | Vol.5, |
| ASL_vibh | Vol.5, |
| ASL_vibh | Vol.5, |
|  | Vol.5, |
| ASL_ | Vol.5, |
| , | l.5, |
| ASL_vib | Vol.5, |
|  | Vol.5, |
| ASL_vibk | Vol.5, |
| _vib | Vol.5, |
| ASL_viby | Vol.5, |
| _vid | Vol.5, |
| ASL_vie | Vol.5, |
| ASL_vieci | Vol.5, |
| ASL_vie | Vol.5, |
| ASL_viej | Vol.5, |
| ASL_viej | Vol |
| _vie | Vol.5, |
| SL_vie | Vol.5, |
| ASL_viep | Vol.5, |
| ASL_vie | Vol.5, |
| _vie | Vol.5, |
| L_vie | Vol.5, 238 |
|  |  |


|  |  |
| :---: | :---: |
| ASL_vigdig |  |
| ASL_viglgx | Vol.5, 196 |
| ASL_viicnc | , |
|  | Vol.5, 270 |
| aw | Vol.5, 268 |
|  | Vol.5, 253 |
|  | Vol.5, 265 |
|  | Vol.5, 263 |
| ASL_viilog | Vol.5, 256 |
|  | Vol.5, 316 |
| SL_vixsla | Vol.5, 319 |
| ASL_vixsps | Vol.5, 312 |
|  | Vol.5, |
| SL_wbtcls | Vol.2, 297 |
| SL_wbtcsl | Vol.2, 292 |
| wbtdls | Vol.2, 288 |
| L_wbtdsl | Vol.2, |
| SL_wibh0x | Vol.5, |
|  | Vol.5, |
| _-wibhyo | Vol.5, |
| SL_wibhy1 | Vol.5, |
| 0x | Vol.5, |
|  | Vol.5, |
| SL_wibj0x | Vol |
| SL_wibj1x | 8 |
| ASL_wibk0x | Vol.5, 120 |
| SL_wibk1x | Vo |
| ASL_wiby0x | Vol.5, 75 |
| ASL_wiby1x | 5, 81 |
| dbey | Vol.5, |
| SL_wieci1 | Vol.5, |
| SL_wieci2 | , 218 |
| c | Vol |
| ASL_wiejep | Vol.5, 244 |
| 尤 | Vol.5, 247 |
| SL_wiejzt | Vol.5, |
| ASL_wienmq | Vol.5, 234 |
| ep | Vol.5, 250 |
| L_wierfc | Vol.5, 278 |
| wier | Vol.5, 275 |
| ie | Vol.5, 238 |
| L_wigamx | Vol.5, 193 |
| igbet | Vol.5, 212 |
| L_wigdig | Vol.5, 209 |
| L_wiglgx | Vol.5, 196 |
| ASL_wiicnc | Vol.5, 272 |
|  | Vol.5, 27 |
| L_Wiidaw | Vol.5, 268 |
| _Wilexp | Vol.5, 253 |
| ASL_wiifco | Vol.5, 265 |
| ASL_wiifsi | Vol.5, 263 |
| L_wiilog | Vol.5, 256 |
| SLWinplg | Vol.5, 316 |

ASL_vigdig : Vol.5, 209
ASL_viglgx : Vol.5, 196
ASL_viicnc : Vol.5, 272
ASL_viicnd : Vol.5, 270
ASL_viidaw : Vol.5, 268
ASL_viiexp : Vol.5, 253
ASL_Viifco:Vol.5, 265

ASL_viilog : Vol.5, 256
ASL_vinplg : Vol.5, 316
ASL_vixsla : Vol.5, 319
ASL_vixsps : Vol.5, 312
ASL_Vixzta . Vol.5, 321
ASL_wbtcsl : Vol.2, 292
ASL_wbtdls : Vol.2, 288
ASL_wbtdsl : Vol.2, 284
ASL_Wibh0x : Vol.5, 173

ASL_wibhy0 : Vol.5, 179
ASL_wibhy1 : Vol.5, 182
ASL_wibiOx : Vol.5, 117
ASL_Wibilx : Vol.5, 123

ASL_wibj1x : Vol.5, 78
ASL_wibk0x : Vol.5, 120
ASL_wibk1x : Vol.5, 126
: Vol.5, 75

ASL_widbey : Vol.5, 314
ASL_wieci1 : Vol.5, 215
ASL_wieci2 : Vol.5, 218
ASL_wiejac : Vol.5, 230

ASL_wiejzt : Vol.5, 241
ASL_wienmq : Vol.5, 234
ASL_wiepai : Vol.5, 250
Vol.5, 278

ASL_wiethe : Vol.5, 238
ASL_wigamx : Vol.5, 193
ASL_wigbet : Vol.5, 212
ASL_wigdig : Vol.5, 209
ASL_wiglgx : Vol.5, 196

ASL_wiicnd : Vol.5, 270
ASL_wiidaw : Vol.5, 268
ASL_wiiexp : Vol.5, 253
ASL_wiifco : Vol.5, 265

ASL_wiilog : Vol.5, 256
ASL_winplg : Vol.5, 316

ASL_wixsla : Vol.5, 319
ASL_wixsps : Vol.5, 312
ASL_wixzta : Vol.5, 321
ASL_zam1hh : Vol.1, 106
ASL_zam1hm : Vol.1, 101
ASL_zam1mh : Vol.1, 96
ASL_zam1mm : Vol.1, 91
ASL_zan1hh : Vol.1, 123
ASL_zan1hm : Vol.1, 119
ASL_zan1mh : Vol.1, 115
ASL_zan1mm : Vol.1, 111
ASL_zanvj1 : Vol.1, 155
ASL_zargjm : Vol.1, 44
ASL_zarsjd : Vol.1, 38
ASL_zbgmdi : Vol.2, 80
ASL_zbgmlc : Vol.2, 72
ASL_zbgmls : Vol.2, 74
ASL_zbgmlu : Vol.2, 70
ASL_zbgmlx : Vol.2, 82
ASL_zbgmms : Vol.2, 76
ASL_zbgmsl : Vol.2, 64
ASL_zbgmsm : Vol.2, 59
ASL_zbgndi : Vol.2, 102
ASL_zbgnlc : Vol.2, 94
ASL_zbgnls : Vol.2, 96
ASL_zbgnlu : Vol.2, 92
ASL_zbgnlx : Vol. 2,104
ASL_zbgnms : Vol.2, 98
ASL_zbgnsl : Vol.2, 88
ASL_zbgnsm : Vol.2, 84
ASL_zbhedi : Vol.2, 239
ASL_zbhels : Vol.2, 233
ASL_zbhelx : Vol.2, 241
ASL_zbhems : Vol.2, 235
ASL_zbhesl : Vol.2, 224
ASL_zbheuc : Vol.2, 231
ASL_zbheud : Vol.2, 229
ASL_zbhfdi : Vol.2, 220
ASL_zbhfls : Vol.2, 214
ASL_zbhflx : Vol.2, 222
ASL_zbhfms : Vol. 2,216
ASL_zbhfsl : Vol.2, 205
ASL_zbhfuc : Vol.2, 212
ASL_zbhfud : Vol.2, 210
ASL_zbhpdi : Vol.2, 182
ASL_zbhpls : Vol.2, 176
ASL_zbhplx : Vol.2, 184
ASL_zbhpms : Vol.2, 178
ASL_zbhpsl : Vol.2, 167
ASL_zbhpuc : Vol.2, 174
ASL_zbhpud : Vol.2, 172
ASL_zbhrdi : Vol.2, 201
ASL_zbhrls : Vol.2, 195
ASL_zbhrlx : Vol.2, 203

```
ASL_zbhrms : Vol.2, 197
ASL_zbhrsl : Vol.2, 186
ASL_zbhruc : Vol.2, 193
ASL_zbhrud : Vol.2, 191
ASL_zcgeaa : Vol.1, 191
ASL_zcgean : Vol.1, 196
ASL_zcghaa : Vol.1, 379
ASL_zcghan : Vol.1, 384
ASL_zcgjaa : Vol.1, 386
ASL_zcgjan : Vol.1, 391
ASL_zcgkaa : Vol.1, 393
ASL_zcgkan : Vol.1, 398
ASL_zcgnaa : Vol.1, 198
ASL_zcgnan : Vol.1, 202
ASL_zcgraa : Vol.1, 372
ASL_zcgran : Vol.1, 377
ASL_zcheaa : Vol.1, 244
ASL_zchean : Vol.1, 248
ASL_zcheee : Vol.1, 257
ASL_zcheen : Vol.1, 262
ASL_zchesn : Vol.1, 255
ASL_zchess : Vol.1, 250
ASL_zchjss : Vol.1, 320
ASL_zchraa : Vol.1, 224
ASL_zchran : Vol.1, 228
ASL_zchree : Vol.1, 237
ASL_zchren : Vol.1, 242
ASL_zchrsn : Vol.1, 235
ASL_zchrss : Vol.1, 230
ASL_zfc1bf : Vol.3, 61
ASL_zfc1fb : Vol.3, 57
ASL_zfc2bf : Vol.3, 127
ASL_zfc2fb : Vol.3, 123
ASL_zfc3bf : Vol.3, 157
ASL_zfc3fb : Vol.3, 153
ASL_zfcmbf : Vol.3, 93
ASL_zfcmfb : Vol.3, }8
ASL_zibh1n : Vol.5, 159
ASL_zibh2n : Vol.5, 162
ASL_zibinz : Vol.5, 141
ASL_zibjnz : Vol.5, 96
ASL_zibknz : Vol.5, 144
ASL_zibynz : Vol.5, 99
ASL_zigamz : Vol.5, 205
ASL_ziglgz : Vol.5, 207
ASL_zlacha : Vol.5, 392
ASL_zlncis : Vol.5, 410
```


[^0]:    a. The asterisk * indicates an arbitrary value.
    b. mb is the band width.
    c. $\quad \ln \mathrm{a} \geq \mathrm{mb}+1$ and $\mathrm{k} \geq \mathrm{n}$ must hold.

[^1]:    $\overline{(*)}$ SMP Functions $=$ Shared Memory Parallel Processing Functions

