ADVANCED SCIENTIFIC LIBRARY

ASL C INTERFACE

User's Guide

<Shared Memory Parallel Functions>

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.

Copyright 2023
NEC Corporation

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 shared memory parallel functions.

Basic Functions Volume 1

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

Basic Functions Volume 2

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

Basic Functions Volume 3

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

Basic Functions Volume 4

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

Basic Functions Volume 5

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

Basic Functions Volume 6

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

Shared Memory Parallel Functions

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

Document Version 3.0.0-230301 for ASL, March 2023

Remarks

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

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

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

Contents

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 ASL C INTERFACE SHARED MEMORY PARALLEL FUNCTIONS . . . 9

1.5.1 Overview of Shared Memory Parallel Functions . . . 9

1.5.2 Performance Improvement Due to Parallel Functions . . . 9

1.5.3 General Notes Concerning the Use of shared memory Parallel Functions . . . 9

1.6 NOTES . . . 11

2 BASIC MATRIX ALGEBRA **13**

2.1 INTRODUCTION . . . 13

2.1.1 Notes . . . 13

2.1.2 Algorithms Used . . . 13

2.1.2.1 Matrix Multiplication . . . 13

2.2 BASIC MATRIX ALGEBRA . . . 14

2.2.1 ASL_qam1mu, ASL_pam1mu
Multiplying Real Matrices (Two-Dimensional Array Type) . . . 14

2.2.2 ASL_qam1mm, ASL_pam1mm
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm AB$) . . . 18

2.2.3 ASL_qam1mt, ASL_pam1mt
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm AB^T$) . . . 22

2.2.4 ASL_qam1tm, ASL_pam1tm
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B$) . . . 26

2.2.5 ASL_qam1tt, ASL_pam1tt
Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B^T$) . . . 30

2.2.6 ASL_ham1mm, ASL_gam1mm
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm AB$) . . . 34

2.2.7 ASL_ham1mh, ASL_gam1mh
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm AB^*$) . . . 39

2.2.8 ASL_ham1hm, ASL_gam1hm
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm A^* B$) . . . 44

2.2.9 ASL_ham1hh, ASL_gam1hh
Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) ($C = C \pm A^* B^*$) . . . 49

2.2.10 ASL_han1mm, ASL_gan1mm
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB$) . . . 54
2.2.11 ASL_han1mh, ASL_gan1mh
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB^*$) . . . 58
2.2.12 ASL_han1hm, ASL_gan1hm
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm A^*B$) . . . 62
2.2.13 ASL_han1hh, ASL_gan1hh
Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm A^*B^*$) . . . 66

3 SIMULTANEOUS LINEAR EQUATIONS (DIRECT METHOD) 70
3.1 INTRODUCTION . . . 70
3.1.1 Methods of using functions . . . 71
3.1.2 Notes . . . 73
3.1.3 Algorithms Used . . . 74
3.1.3.1 Solution of Simultaneous Linear Equations . . . 74
3.1.3.2 LU Decomposition (Gauss Method) . . . 74
3.1.4 Reference Bibliography . . . 77
3.2 REAL MATRIX (TWO-DIMENSIONAL ARRAY TYPE) . . . 78
3.2.1 ASL_qbgmsm
Simultaneous Linear Equations with Multiple Right-Hand Sides (Real Matrix) . . . 78
3.2.2 ASL_qbgmsl
Simultaneous Linear Equations (Real Matrix) . . . 83
3.2.3 ASL_qbgmlu
LU Decomposition of a Real Matrix . . . 88
3.2.4 ASL_qbgmlc
LU Decomposition and Condition Number of a Real Matrix . . . 90
3.3 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (REAL ARGUMENT TYPE) . 92
3.3.1 ASL_hbgmsm
Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix) . . . 92
3.3.2 ASL_hbgmsl
Simultaneous Linear Equation (Complex Matrix) . . . 98
3.3.3 ASL_hbgmlu
LU Decomposition of a Complex Matrix . . . 103
3.3.4 ASL_hbgmlc
LU Decomposition and Condition Number of a Complex Matrix . . . 105
3.4 COMPLEX MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (COMPLEX ARGUMENT TYPE) 107
3.4.1 ASL_hbgnsm
Simultaneous Linear Equations with Multiple Right-Hand Sides (Complex Matrix) . . . 107
3.4.2 ASL_hbgnsl
Simultaneous Linear Equations (Complex Matrix) . . . 111
3.4.3 ASL_hbgnlu
LU Decomposition of a Complex Matrix . . . 115
3.4.4 ASL_hbgnlc
LU Decomposition and Condition Number of a Complex Matrix . . . 117
3.5 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) . . . 119
3.5.1 ASL_qbspsl, ASL_pbspsl
Simultaneous Linear Equations (Real Symmetric Matrix) . . . 119
3.5.2 ASL_qbspud, ASL_pbspud
LDL^T Decomposition of a Real Symmetric Matrix . . . 124

3.6 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE, LOWER TRIANGULAR TYPE) (NO PIVOTING) 126
3.6.1 ASL_qbsnsl, ASL_pbsnsl
Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting) 126
3.6.2 ASL_qbsnud, ASL_pbsnud
$U^T DU$ Decomposition of a Real Symmetric Matrix (No Pivoting) 131
3.7 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) 133
3.7.1 ASL_hbhpsl, ASL_gbhpsl
Simultaneous Linear Equations (Hermitian Matrix) 133
3.7.2 ASL_hbhpud, ASL_gbhpud
LDL^* Decomposition of a Hermitian Matrix 139
3.8 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) (NO PIVOTING) 141
3.8.1 ASL_hbhrsl, ASL_gbhrsl
Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) 141
3.8.2 ASL_hbhrud, ASL_gbhrud
LDL^* Decomposition of a Hermitian Matrix (No Pivoting) 147
3.9 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) 149
3.9.1 ASL_hbhfsl, ASL_gbhfsl
Simultaneous Linear Equations (Hermitian Matrix) 149
3.9.2 ASL_hbhfud, ASL_gbhfud
LDL^* Decomposition of a Hermitian Matrix 154
3.10 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) (NO PIVOTING) 156
3.10.1 ASL_hbhesl, ASL_gbhesl
Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting) 156
3.10.2 ASL_hbheud, ASL_gbheud
LDL^* Decomposition of a Hermitian Matrix (No Pivoting) 161

4 SIMULTANEOUS LINEAR EQUATIONS (ITERATIVE METHOD) 163
4.1 INTRODUCTION 163
4.1.1 Notes 164
4.1.2 Algorithms Used 166
4.1.2.1 Nonstationary iterative method (for Symmetric Matrix only) 166
4.1.2.2 Nonstationary iterative method (for Asymmetric Matrix) 166
4.1.2.3 Preconditioned Iterative Method 168
4.1.2.4 Preconditioning Methods 170
4.1.2.5 Advanced Techniques for Improving Performance 171
4.1.3 Reference Bibliography 173
4.2 SPARSE MATRIX—NONSTATIONARY ITERATIVE METHODS (BASIC ITERATION METHOD FUNCTIONS) 174
4.2.1 ASL_qxe010, ASL_pxe010
Positive Definite Symmetric Sparse Matrix (ELLPACK Format) (CG method) 174
4.2.2 ASL_qxe020, ASL_pxe020
Asymmetric Sparse Matrix (ELLPACK Format) (CGS method) 183
4.2.3 ASL_qxe030, ASL_pxe030
Asymmetric Sparse Matrix (ELLPACK Format) (BiCGSTAB method) 192
4.2.4 ASL_qxe040, ASL_pxe040
Asymmetric Sparse Matrix (ELLPACK Format) (GMRES(m) method) 202

5 EIGENVALUES AND EIGENVECTORS **211**
5.1 INTRODUCTION . . . 211
5.1.1 Notes . . . 212
5.1.2 Algorithms Used . . . 213
5.1.2.1 Transforming a real symmetric matrix to a real symmetric tridiagonal matrix . . . 213
5.1.2.2 Transforming a Hermitian matrix to a real symmetric tridiagonal matrix . . . 213
5.1.2.3 The Householder transformation by block algorithm . . . 213
5.1.2.4 QR method . . . 214
5.1.2.5 root-free QR method . . . 214
5.1.2.6 Bisection method . . . 215
5.1.2.7 Accumulation of similarity (unitary) transformation by block algorithm . . . 216
5.1.2.8 Inverse iteration method . . . 217
5.1.2.9 Generalized eigenvalue problem . . . 217
5.1.3 Reference Bibliography . . . 219
5.2 REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) . . . 220
5.2.1 ASL_qcsmaa, ASL_pcsmaa
All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix . . . 220
5.2.2 ASL_qcsman, ASL_pcsman
All Eigenvalues of a Real Symmetric Matrix . . . 224
5.2.3 ASL_qcsmss, ASL_pcsmss
Eigenvalues and Eigenvectors of a Real Symmetric Matrix . . . 226
5.2.4 ASL_qcsmsn, ASL_pcsmsn
Eigenvalues of a Real Symmetric Matrix . . . 231
5.3 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) . . . 233
5.3.1 ASL_hchraa, ASL_gchraa
All Eigenvalues and All Eigenvectors of a Hermitian Matrix . . . 233
5.3.2 ASL_hchran, ASL_gchran
All Eigenvalues of a Hermitian Matrix . . . 238
5.3.3 ASL_hchrss, ASL_gchrss
Eigenvalues and Eigenvectors of a Hermitian Matrix . . . 240
5.3.4 ASL_hchrsn, ASL_gchrsn
Eigenvalues of a Hermitian Matrix . . . 246
5.4 HERMITIAN MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (COMPLEX ARGUMENT TYPE) . . . 249
5.4.1 ASL_hcheaa, ASL_gcheaa
All Eigenvalues and All Eigenvectors of a Hermitian Matrix . . . 249
5.4.2 ASL_hchean, ASL_gchean
All Eigenvalues of a Hermitian Matrix . . . 253
5.4.3 ASL_hchess, ASL_gchess
Eigenvalues and Eigenvectors of a Hermitian Matrix . . . 255
5.4.4 ASL_hchesn, ASL_gchesn
Eigenvalues of a Hermitian Matrix . . . 261
5.5 GENERALIZED EIGENVALUE PROBLEM FOR A REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($Ax = \lambda Bx$) . . . 264
5.5.1 ASL_qcgsaa, ASL_pcgsaa
All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive) . . . 264
5.5.2 ASL_qcgsan, ASL_pcgsan
All Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive) . . . 270
5.5.3 ASL_qcgsss, ASL_pcgsss
Eigenvalues and Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive) . . . 272

5.5.4 ASL_qcgssn, ASL_pcgssn
Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive) . . . 279
5.6 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($ABx = \lambda x$) . . . 282
5.6.1 ASL_qcgjaa, ASL_pcgjaa
All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive) . . . 282
5.6.2 ASL_qcgjan, ASL_pcgjan
All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive) . . . 286
5.7 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($BAx = \lambda x$) . . . 288
5.7.1 ASL_qcgkaa, ASL_pcgkaa
All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive) . . . 288
5.7.2 ASL_qcgkan, ASL_pcgkan
All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive) . . . 292
5.8 GENERALIZED EIGENVALUE PROBLEM ($Az = \lambda Bz$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) 294
5.8.1 ASL_hcgraa, ASL_gcgraa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive) . . . 294
5.8.2 ASL_hcgran, ASL_gcgran
All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive) . . . 299
5.9 GENERALIZED EIGENVALUE PROBLEM ($ABz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) 302
5.9.1 ASL_hcgjaa, ASL_gcgjaa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive) . . . 302
5.9.2 ASL_hcgjan, ASL_gcgjan
All Eigenvalues of Hermitian Matrices
(Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive) . . . 307
5.10 GENERALIZED EIGENVALUE PROBLEM ($BAz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE) 309
5.10.1 ASL_hcgkaa, ASL_gcgkaa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive) . . . 309
5.10.2 ASL_hcgkan, ASL_gcgkan
All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive) . . . 314

6 FOURIER TRANSFORMS AND THEIR APPLICATIONS 317
6.1 INTRODUCTION . . . 317
6.1.1 Notes . . . 318
6.1.2 Algorithms Used . . . 319
6.1.2.1 Two-dimensional complex Fourier transform . . . 319
6.1.2.2 Two-dimensional real Fourier transform . . . 319
6.1.2.3 Three-dimensional complex Fourier transform . . . 320
6.1.2.4 Three-dimensional real Fourier transform . . . 320
6.1.3 Reference Bibliography . . . 321
6.2 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE) . . . 322

6.2.1 [DEPRECATED]ASL_qfcmfb, ASL_pfcmfb
Multiple One-Dimensional Complex Fourier Transforms (Include Initialization) 322
6.2.2 [DEPRECATED]ASL_qfcmbf, ASL_pfcmbf
Multiple One-Dimensional Complex Fourier Transforms (After Initialization) 326
6.3 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE) 334
6.3.1 [DEPRECATED]ASL_hfcmfb, ASL_gfcmfb
Multiple One-Dimensional Complex Fourier Transforms (Include Initialization) 334
6.3.2 [DEPRECATED]ASL_hfcmbf, ASL_gfcmbf
Multiple One-Dimensional Complex Fourier Transforms (After Initialization) 338
6.4 MULTIPLE ONE-DIMENSIONAL REAL FOURIER TRANSFORM 346
6.4.1 [DEPRECATED]ASL_qfrmfb, ASL_pfrmfb
Multiple One-Dimensional Real Fourier Transforms (Including Initialization) 346
6.4.2 [DEPRECATED]ASL_qfrmbf, ASL_pfrmbf
Multiple One-Dimensional Real Fourier Transforms (After Initialization) 350
6.5 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE) . . . 358
6.5.1 [DEPRECATED]ASL_qfc2fb, ASL_pfc2fb
Two-Dimensional Complex Fourier Transform (Including Initialization) 358
6.5.2 [DEPRECATED]ASL_qfc2bf, ASL_pfc2bf
Two-Dimensional Complex Fourier Transform (After Initialization) 362
6.6 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE) 367
6.6.1 [DEPRECATED]ASL_hfc2fb, ASL_gfc2fb
Two-Dimensional Complex Fourier Transform (Including Initialization) 367
6.6.2 [DEPRECATED]ASL_hfc2bf, ASL_gfc2bf
Two-Dimensional Complex Fourier Transform (After Initialization) 371
6.7 TWO-DIMENSIONAL REAL FOURIER TRANSFORM 376
6.7.1 [DEPRECATED]ASL_qfr2fb, ASL_pfr2fb
Two-Dimensional Real Fourier Transform (Including Initialization) 376
6.7.2 [DEPRECATED]ASL_qfr2bf, ASL_pfr2bf
Two-Dimensional Real Fourier Transform (After Initialization) 380
6.8 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE) . . 386
6.8.1 [DEPRECATED]ASL_qfc3fb, ASL_pfc3fb
Three-Dimensional Complex Fourier Transform (Including Initialization) 386
6.8.2 [DEPRECATED]ASL_qfc3bf, ASL_pfc3bf
Three-Dimensional Complex Fourier Transform (After Initialization) 390
6.9 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)397
6.9.1 [DEPRECATED]ASL_hfc3fb, ASL_gfc3fb
Three-Dimensional Complex Fourier Transform (Including Initialization) 397
6.9.2 [DEPRECATED]ASL_hfc3bf, ASL_gfc3bf
Three-Dimensional Complex Fourier Transform (After Initialization) 401
6.10 THREE-DIMENSIONAL REAL FOURIER TRANSFORM 408
6.10.1 [DEPRECATED]ASL_qfr3fb, ASL_pfr3fb
Three-Dimensional Real Fourier Transform (Including Initialization) 408
6.10.2 [DEPRECATED]ASL_qfr3bf, ASL_pfr3bf
Three-Dimensional Real Fourier Transform (After Initialization) 412
6.11 CONVOLUTIONS 419
6.11.1 ASL_qfcn2d, ASL_pfcn2d
Two-Dimensional Convolutions 419
6.11.2 ASL_qfcn3d, ASL_pfcn3d
Three-Dimensional Convolutions 427
6.12 CORRELATIONS 437
6.12.1 ASL_qfcr2d, ASL_pfcr2d
Two-Dimensional Correlations 437
6.12.2 ASL_qfcr3d, ASL_pfcr3d
Three-Dimensional Correlations 445

6.13 POWER SPECTRUM ANALYSIS 456
6.13.1 ASL_qfps2d, ASL_pfps2d
Two-Dimensional Fourier Periodograms 456
6.13.2 ASL_qfps3d, ASL_pfps3d
Three-Dimensional Fourier Periodograms 465

7 SORTING 480
7.1 INTRODUCTION 480
7.1.1 Notes 480
7.1.2 Algorithms Used 481
7.1.3 Reference Bibliography 483
7.2 SORTING 484
7.2.1 ASL_qssta1, ASL_pssta1
Sorting a List of Data 484
7.2.2 ASL_qssta2, ASL_pssta2
Sorting a List of Pairwise Data 488

A METHODS OF HANDLING ARRAY DATA 493
A.1 Methods of handling array data corresponding to matrix 493
A.2 Data storage modes 495
A.2.1 Real matrix (two-dimensional array type) 495
A.2.2 Complex matrix 496
A.2.3 Real symmetric matrix and positive symmetric matrix 498
A.2.4 Hermitian matrix 500
A.2.5 Random sparse matrix (For symmetric matrix only) 502
A.2.6 Random sparse matrix 503

B MACHINE CONSTANTS USED IN ASL C INTERFACE 506
B.1 Units for Determining Error 506
B.2 Maximum and Minimum Values of Floating Point Data 506

Chapter 1

INTRODUCTION

1.1 OVERVIEW

1.1.1 Introduction to The Advanced Scientific Library ASL C interface

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

Table 1−1 Classification of functions included in ASL

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

1.1.2 Distinctive Characteristics of ASL C interface

ASL C interface has the following distinctive characteristics.

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

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

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

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

1.2 KINDS OF LIBRARIES

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

Table 1−2 Kinds of libraries providing ASL C interface

Size of variable(byte)		Declaration of arguments	Kind	Kind of library
integer	real			
4	8	int double	32bit integer Double-precision function	32bit integer library (link option: -lasl_openmp)
4	4	int float	32bit integer Single-precision function	
8	8	long double	64bit integer Double-precision function	64bit integer library (link option: -lasl_openmp_i64)
8	4	long float	64bit integer Single-precision function	

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

(∗2) For compiling the program with functions in the 64-bit integer library, the option "`-DASL_LIB_INT64`" must be specified (See the Note (2) in 1.6).

1.3 ORGANIZATION

This section describes the organization of Chapters 2 and later.

1.3.1 Introduction

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

1.3.2 Organization of Function Description

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

(1) Function

(2) Usage

(3) Arguments and return value

(4) Restrictions

(5) Error indicator (Return Value)

(6) Notes

(7) Example

Each item is described according to the following principles.

1.3.3 Contents of Each Item

(1) **Function**
Function briefly describes the purpose of the ASL C interface function.

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

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

isw is an input argument for specifying the processing procedure. ierr is a return value. In some cases, input/output arguments precede input arguments. The following general principles also apply.

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

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

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

(a) Arguments and return value

Arguments and return value are explained in the order they are designated in the Usage paragraph.

(b) Type

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

I : Integer type

D : Double precision real

R : Real

Z : Double precision complex

C : Complex

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

(c) Size

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

1 : Indicates that argument is a variable.

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

(d) Input/Output

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

i. When only "Input" appears

When the control returns to the program using this function, information when the argument is input is preserved. The user must assign input-time information unless specifically instructed otherwise. When the argument is a variable, the variable value must be passed.

ii. When only "Output" appears

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

iii. When both "Input" and "Output" appear

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

iv. When "Work" appears

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

(e) Contents

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

- A sample Argument description follows.

  Example

  The statement of the function (ASL_dbgmlc, ASL_rbgmlc) that obtains the LU decomposition and the condition number of a real matrix is as follows.

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

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

Table 1−3 Sample Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*} Note	lna×n	Input	Real matrix A(two-dimensional array)
				Output	The matrix A decomposed into the matrix LU where U is a unit upper triangular matrix and L is a lower triangular matrix.
2	lna	I	1	Input	Adjustable dimension size of array a
3	n	I	1	Input	Order n of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i−1]: Number of the row exchanged with row i in the i-th step.
5	cond	{D / R}	1	Output	Reciprocal of the condition number
6	w1	{D* / R*}	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 {double-precision / single-precision}Note real array of size [lna × n] , ipvt is an integer array of size n and w1 is a {double-precision / single-precision} real array of size n.
When the 64-bit integer version is used, all integer-type arguments (lna, n, ipvt and ierr) must be declared by using `long`, not `int`.

Note The entries enclosed in brace { } mean that the array should be declared double precision type when using function ASL_dbgmlc and real type when using function ASL_rbgmlc. Braces are used in this manner throughout the remainder of the text unless specifically stated otherwise.

Data must be stored in a, lna and n before this function is called. The LU decomposition and condition number of the assigned matrix are calculated with in the function, and the results are stored in array a and variable cond. In addition, pivoting information is stored in ipvt for use by subsequent functions.

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

Since w1 is a work area used only within the function, its contents at input and output time have no special meaning.

(4) **Restrictions**

Restrictions indicate limiting ranges for function arguments.

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

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

Table 1−4 Classification of Return Values

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

(6) **Notes**

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

(7) **Example**

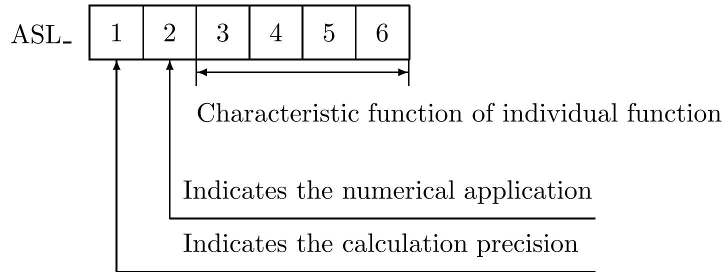
Here gives an example of how to use the function. Note that in some cases, multiple functions are combined in a single example. The output results are given in the 32-bit integer version, and may differ within the range of rounding error if the compiler or intrinsic functions are different.

In addition, when the 64-bit integer version library is used, the `long`-type conversion specification to be given to `printf` or `scanf` must be `%ld`. The source codes of examples in this document are included in User's Guide. Input data, if required, is also included in it. To build up an executable files by compiling these example source codes, they should be linked with this product library.

1.4 FUNCTION NAMES

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

Figure 1−1 Function Name Components



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

q	Parallel function double precision real-type calculation
p	Parallel function single precision real-type calculation
h	Parallel function double precision complex-type calculation
g	Parallel function single precision complex-type calculation

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

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

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

"2" in Figure 1−1 **:** Currently, the following letters 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

Letter	Application Field	Volume
k	Ordinary differential equation (initial value problems)	4
l	Roots of equations	5
m	Extremum problems and optimization	5
n	Approximation and regression analysis	4, 6
o	Ordinary differential equations (boundary value problems), integral equations and partial differential equations	4
p	Interpolation	4
q	Numerical differentials	4
s	Sorting and ranking	5, 7
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 ASL C INTERFACE SHARED MEMORY PARALLEL FUNCTIONS

1.5.1 Overview of Shared Memory Parallel Functions

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

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

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

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

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

1.5.2 Performance Improvement Due to Parallel Functions

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

(1) Execution environment

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

(2) Problem scale

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

1.5.3 General Notes Concerning the Use of shared memory Parallel Functions

(1) Execution environment

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

(2) Thread parameter (nt)

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

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

- C shell:

  setenv OMP_NUM_THREADS 2 [RETURN]

- Bourne shell:

  OMP_NUM_THREADS=2 [RETURN]
  export OMP_NUM_THREADS [RETURN]

1.6 NOTES

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

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

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

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

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

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

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

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

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

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

(11) Depending on the problem being dealt with, there may be cases when there are multiple solutions, and the execution result differs in appearance according to the compiler used or the computer or OS under which

the program is executed. For example, when an eigenvalue problem is solved, the eigenvectors that are obtained may differ in appearance in this way.

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

Chapter 2

BASIC MATRIX ALGEBRA

2.1 INTRODUCTION

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

2.1.1 Notes

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

2.1.2 Algorithms Used

2.1.2.1 Matrix Multiplication

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

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

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

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

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

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

$$\begin{array}{ccccc} \text{Matrix } C & & \text{Matrix } A & & \text{Matrix } B \\ \left[\begin{array}{c|c} C_1 & C_2 \end{array}\right] & = & [A] & \cdot & \left[\begin{array}{c|c} B_1 & B_2 \end{array}\right] \end{array}$$

Figure 2−1 Parallelization of Matrix Multiplication for Two CPUs

2.2 BASIC MATRIX ALGEBRA

2.2.1 ASL_qam1mu, ASL_pam1mu Multiplying Real Matrices (Two-Dimensional Array Type)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qam1mu (a, lma, nm, nn, b, lnb, nl, c, lmc, nt);

Single precision:

ierr = ASL_pam1mu (a, lma, nm, nn, b, lnb, nl, c, lmc, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lma×nn	Input	Real matrix A (two-dimensional array type).
2	lma	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A (Number of rows in matrix C).
4	nn	I	1	Input	Number of columns in matrix A (Number of rows in matrix B).
5	b	{D* / R*}	lnb×nl	Input	Real matrix B (two-dimensional array type).
6	lnb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B (Number of columns in matrix C).
8	c	{D* / R*}	lmc×nl	Output	Product $A \cdot B$ of matrices A, B (two-dimensional array type).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	nt	I	1	Input	Number of tasks to be generated.
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nn} \leq \mathrm{lnb}$

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

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

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, lma = 11, lnb = 11, lmc = 11, nm = 4, nn = 4, nl = 4 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_qam1mu */

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

int main()
{
    double *a;
    int ma=4;
    int mm=4;
    int nn=4;
    double *b;
    int nb=4;
    int nt=2;
    int ll=4;
    double *c;
    int mc=4;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "qam1mu.dat", "r" );
    if( fp == NULL )
    {
```

```
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_qam1mu ***\n" );
    printf( "\n     ** Input **\n\n" );

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

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

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

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

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

    fclose( fp );

    ierr = ASL_qam1mu(a, ma, mm, nn, b, nb, ll, c, mc, nt);

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

    printf( "\n\tMatrix c\n\n" );
    for( i=0 ; i<mc ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", c[i+mc*j] );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
    *** ASL_qam1mu ***

     ** Input **

    Matrix a

          1        2        0       -1
         -3       -5        1        2
          1        3        2       -2
```

```
         0         2         1        -1

Matrix b

        -3        -1         1        -1
        -3        -1         0         1
        -4        -1         1         0
       -10        -3         1         1

 ** Output **

ierr =      0

Matrix c

         1         0         0         0
         0         1         0         0
         0         0         1         0
         0         0         0         1
```

2.2.2 ASL_qam1mm, ASL_pam1mm Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm AB$)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qam1mm (a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_pam1mm (a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lma×nn	Input	Real matrix A (two-dimensional array type).
2	lma	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D*, R*}	lnb×nl	Input	Real matrix B (two-dimensional array type).
6	lnb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{D*, R*}	lmc×nl	Input	Initial real matrix C (If isw $= \pm 1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB$).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB$ isw = 0: Obtain $C = AB$ isw = −1: Obtain $C = C - AB$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{nm} \leq \text{lma}, \text{lmc}$

(b) $0 < \text{nn} \leq \text{lnb}$

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

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

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

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB$.

(b) Input data

Matrices A and B, lma = 11, lnb = 11, lnc = 11, nm = 4, nn = 5, nl = 6, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_qam1mm */

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

int main()
{
    double *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double *b;
    int lnb=11;
    int nl=6;
    double *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_qam1mm ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

```
    }

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

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

    printf( "\tMatrix A\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", a[i+lma*j]=i+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tMatrix B\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", b[i+lnb*j]=i+1 );
        }
        printf( "\n" );
    }


    ierr = ASL_qam1mm(a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

    printf( "\n\tMatrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", c[i+lmc*j] );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_qam1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 6

isw= 0   nt = 2

Matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4

Matrix B
       1        1        1        1        1        1
       2        2        2        2        2        2
       3        3        3        3        3        3
       4        4        4        4        4        4
       5        5        5        5        5        5

 ** Output **
```

```
ierr =      0

Matrix C
      15        15        15        15        15        15
      30        30        30        30        30        30
      45        45        45        45        45        45
      60        60        60        60        60        60
```

2.2.3 ASL_qam1mt, ASL_pam1mt Multiplying Real Matrices (Two-Dimensional Array Type) $(C = C \pm AB^T)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qam1mt (a, lma, nm, nn, b, llb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_pam1mt (a, lma, nm, nn, b, llb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D∗ / R∗}	lma×nn	Input	Real matrix A (two-dimensional array type).
2	lma	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D∗ / R∗}	llb×nn	Input	Real transposed matrix B (two-dimensional array type).
6	llb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{D∗ / R∗}	lmc×nl	Input	Initial real matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of real matrices $(C = [C\pm]AB^T)$.
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB^T$ isw = 0: Obtain $C = AB^T$ isw = −1: Obtain $C = C - AB^T$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nl} \leq \mathrm{llb}$

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

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

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

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB^T$.

(b) Input data

Matrices A and B, lma = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 6, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_qam1mt */

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

int main()
{
    double *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double *b;
    int llb=11;
    int nl=5;
    double *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_qam1mt ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   llb=%2d   lmc=%2d\n\n", lma, llb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

```
    }

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

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

    printf( "\tMatrix A\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", a[i+lma*j]=i+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tMatrix B(Transposed Storage)\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", b[j+llb*i]=j+1 );
        }
        printf( "\n" );
    }


    ierr = ASL_qam1mt(a, lma, nm, nn, b, llb, nl, c, lmc, isw, nt);


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

    printf( "\n\tMatrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", c[i+lmc*j] );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_qam1mt ***

 ** Input **

lma=11   llb=11   lmc=11

nm = 4   nn = 5   nl = 5

isw= 0   nt = 2

Matrix A
      1        1        1        1        1
      2        2        2        2        2
      3        3        3        3        3
      4        4        4        4        4

Matrix B(Transposed Storage)
      1        2        3        4        5
      1        2        3        4        5
      1        2        3        4        5
      1        2        3        4        5
      1        2        3        4        5

 ** Output **
```

```
ierr =       0

Matrix C
       5        10        15        20        25
      10        20        30        40        50
      15        30        45        60        75
      20        40        60        80       100
```

2.2.4 ASL_qam1tm, ASL_pam1tm Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B$)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qam1tm (a, lna, nm, nn, b, lnb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_pam1tm (a, lna, nm, nn, b, lnb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×nm	Input	Real transposed matrix A (two-dimensional array type).
2	lna	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D*, R*}	lnb×nl	Input	Real matrix B (two-dimensional array type).
6	lnb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{D*, R*}	lmc×nl	Input	Initial real matrix C (If isw $= \pm1$) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]A^T B$).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^T B$ isw = 0: Obtain $C = A^T B$ isw = −1: Obtain $C = C - A^T B$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \text{nm} \leq \text{lmc}$

(b) $0 < \text{nn} \leq \text{lna}, \text{lnb}$

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

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

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

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

Obtain $C = A^T B$.

(b) Input data

Matrices A and B, lna = 11, lnb = 11, lnc = 11, nm = 4, nn = 5, nl = 6, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_qam1tm */

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

int main()
{
    double *a;
    int lna=11;
    int nm=5;
    int nn=5;
    double *b;
    int lnb=11;
    int nl=4;
    double *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_qam1tm ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlna=%2d   lnb=%2d   lmc=%2d\n\n", lna, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

```
    }

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

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

    printf( "\tMatrix A(Transposed Storage)\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", a[j+lna*i]=j+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tMatrix B\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", b[i+lnb*j]=i+1 );
        }
        printf( "\n" );
    }


    ierr = ASL_qam1tm(a, lna, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

    printf( "\n\tMatrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", c[i+lmc*j] );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_qam1tm ***

 ** Input **

lna=11   lnb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt = 2

Matrix A(Transposed Storage)
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Matrix B
       1        1        1        1
       2        2        2        2
       3        3        3        3
       4        4        4        4
       5        5        5        5

 ** Output **
```

```
ierr =       0

Matrix C
       55        55        55        55
       55        55        55        55
       55        55        55        55
       55        55        55        55
       55        55        55        55
```

2.2.5 ASL_qam1tt, ASL_pam1tt Multiplying Real Matrices (Two-Dimensional Array Type) ($C = C \pm A^T B^T$)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qam1tt (a, lna, nm, nn, b, llb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_pam1tt (a, lna, nm, nn, b, llb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×nm	Input	Real transposed matrix A (two-dimensional array type).
2	lna	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D* / R*}	llb×nn	Input	Real transposed matrix B (two-dimensional array type).
6	llb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{D* / R*}	lmc×nl	Input	Initial real matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm] A^T B^T$).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^T B^T$ isw = 0: Obtain $C = A^T B^T$ isw = −1: Obtain $C = C - A^T B^T$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 <$ nm $\leq$ lmc

(b) $0 <$ nn $\leq$ lna

(c) $0 <$ nl $\leq$ lnb

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

(e) nt ≥ 1

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

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

(b) Input data

Matrices A and B, lna = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 6, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_qam1tt */

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

int main()
{
    double *a;
    int lna=11;
    int nm=5;
    int nn=5;
    double *b;
    int llb=11;
    int nl=4;
    double *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_qam1tt ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlna=%2d   llb=%2d   lmc=%2d\n\n", lna, llb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

```
    }

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

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

    printf( "\tMatrix A(Transposed Storage)\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", a[j+lna*i]=j+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tMatrix B(Transposed Storage)\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", b[j+llb*i]=j+1 );
        }
        printf( "\n" );
    }


    ierr = ASL_qam1tt(a, lna, nm, nn, b, llb, nl, c, lmc, isw, nt);


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

    printf( "\n\tMatrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", c[i+lmc*j] );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_qam1tt ***

 ** Input **

lna=11   llb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt = 2

Matrix A(Transposed Storage)
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Matrix B(Transposed Storage)
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4

 ** Output **
```

```
ierr =      0

Matrix C
      15        30        45        60
      15        30        45        60
      15        30        45        60
      15        30        45        60
      15        30        45        60
```

2.2.6 ASL_ham1mm, ASL_gam1mm Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) $(C = C \pm AB)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_ham1mm (ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);

Single precision:

ierr = ASL_gam1mm (ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* / R*}	lma×nn	Input	Real part of complex matrix A (two-dimensional array type).
2	ai	{D* / R*}	lma×nn	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	lma	I	1	Input	Adjustable dimension of array ar and ai.
4	nm	I	1	Input	Number of rows in matrix A.
5	nn	I	1	Input	Number of columns in matrix A.
6	br	{D* / R*}	lnb×nl	Input	Real part of complex matrix B (two-dimensional array type).
7	bi	{D* / R*}	lnb×nl	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	lnb	I	1	Input	Adjustable dimension of array br and bi.
9	nl	I	1	Input	Number of columns in matrix B.
10	cr	{D* / R*}	lmc×nl	Input	Real part of Initial complex matrix C (If isw $= \pm 1$) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C \pm] AB$).
11	ci	{D* / R*}	lmc×nl	Input	Imaginary part of initial complex matrix C (If isw $= \pm 1$) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C \pm] AB$)
12	lmc	I	1	Input	Adjustable dimension of array cr and ci.
13	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB$ isw = 0: Obtain $C = AB$ isw = −1: Obtain $C = C - AB$
14	nt	I	1	Input	Number of tasks to be generated.
15	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nn} \leq \mathrm{lnb}$

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

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

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

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, lma = 11, lnb = 11, lnc = 11, nm = 4, nn = 5, nl = 4, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_ham1mm */

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

int main()
{
    double *ar;
    double *ai;
    int lma=11;
    int nm=4;
    int nn=5;
    double *br;
    double *bi;
    int lnb=11;
    int nl=4;
    double *cr;
    double *ci;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_ham1mm ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt=%2d\n\n", isw, nt );

    ar = ( double * )malloc((size_t)( sizeof(double) * (lma*nn) ));
    if( ar == NULL )
```

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

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

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

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", ar[i+lma*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", ai[i+lma*j]=j+1 );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", br[i+lnb*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", bi[i+lnb*j]=j+1 );
    }
    printf( "\n" );
}


ierr = ASL_ham1mm(ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);


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

```
        printf( "\n\tReal part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", cr[i+lmc*j] );
            }
            printf( "\n" );
        }
        printf( "\n\tImaginary part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", ci[i+lmc*j] );
            }
            printf( "\n" );
        }

        free( ar );
        free( ai );
        free( br );
        free( bi );
        free( cr );
        free( ci );

        return 0;
    }
```

(d) Output results

```
*** ASL_ham1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt= 2

Real part of matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4
Imaginary part of matrix A
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Real part of matrix B
       1        1        1        1
       2        2        2        2
       3        3        3        3
       4        4        4        4
       5        5        5        5

Imaginary part of matrix B
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4

 ** Output **

ierr =      0

Real part of matrix C
       0      -15      -30      -45
      15        0      -15      -30
      30       15        0      -15
      45       30       15        0

Imaginary part of matrix C
      60       65       70       75
      65       75       85       95
      70       85      100      115
      75       95      115      135
```

2.2.7 ASL_ham1mh, ASL_gam1mh Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) $(C = C \pm AB^*)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_ham1mh (ar, ai, lma, nm, nn, br, bi, llb, nl, cr, ci, lmc, isw, nt);

Single precision:

ierr = ASL_gam1mh (ar, ai, lma, nm, nn, br, bi, llb, nl, cr, ci, lmc, isw, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D∗ / R∗}	lma×nn	Input	Real part of complex matrix A (two-dimensional array type).
2	ai	{D∗ / R∗}	lma×nn	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	lma	I	1	Input	Adjustable dimension of array ar and ai.
4	nm	I	1	Input	Number of rows in matrix A.
5	nn	I	1	Input	Number of columns in matrix A.
6	br	{D∗ / R∗}	llb×nn	Input	Real part of complex matrix B (two-dimensional array type).
7	bi	{D∗ / R∗}	llb×nn	Input	Real part of complex matrix B (two-dimensional array type).
8	llb	I	1	Input	Adjustable dimension of array br and bi.
9	nl	I	1	Input	Number of rows in matrix B.
10	cr	{D∗ / R∗}	lmc×nl	Input	Real part of initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB^*$).
11	ci	{D∗ / R∗}	lmc×nl	Input	Imaginary part of initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]AB^*$).
12	lmc	I	1	Input	Adjustable dimension of array cr and ci.
13	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB^*$ isw = 0: Obtain $C = AB^*$ isw = −1: Obtain $C = C - AB^*$
14	nt	I	1	Input	Number of tasks to be generated.
15	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nl} \leq \mathrm{llb}$

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

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

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

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

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

(6) **Notes**

None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB^*$.

(b) Input data

Matrix A, matrix B, lma = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 4, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_ham1mh */

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

int main()
{
    double *ar;
    double *ai;
    int lma=11;
    int nm=4;
    int nn=5;
    double *br;
    double *bi;
    int lnb=11;
    int nl=4;
    double *cr;
    double *ci;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_ham1mh ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt=%2d\n\n", isw, nt );

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

```
        return -1;
    }

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

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

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

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

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

    printf( "\tReal part of matrix A\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", ar[i+lma*j]=i+1 );
        }
        printf( "\n" );
    }
    printf( "\tImaginary part of matrix A\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", ai[i+lma*j]=j+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tReal part of matrix B\n" );
    for( i=0 ; i<nl ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", br[i+lnb*j]=i+1 );
        }
        printf( "\n" );
    }
    printf( "\n\tImaginary part of matrix B\n" );
    for( i=0 ; i<nl ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", bi[i+lnb*j]=j+1 );
        }
        printf( "\n" );
    }


    ierr = ASL_ham1mh(ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);


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

    printf( "\n\tReal part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
```

```
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", cr[i+lmc*j] );
            }
            printf( "\n" );
        }
        printf( "\n\tImaginary part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", ci[i+lmc*j] );
            }
            printf( "\n" );
        }

        free( ar );
        free( ai );
        free( br );
        free( bi );
        free( cr );
        free( ci );

        return 0;
    }
```

(d) Output results

```
*** ASL_ham1mh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt= 2

Real part of matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4
Imaginary part of matrix A
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Real part of matrix B
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4

Imaginary part of matrix B
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

 ** Output **

ierr =      0

Real part of matrix C
      60       65       70       75
      65       75       85       95
      70       85      100      115
      75       95      115      135

Imaginary part of matrix C
       0       15       30       45
     -15        0       15       30
     -30      -15        0       15
     -45      -30      -15        0
```

2.2.8 ASL_ham1hm, ASL_gam1hm Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) $(C = C \pm A^*B)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_ham1hm (ar, ai, lna, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);

Single precision:

ierr = ASL_gam1hm (ar, ai, lna, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lna×nm	Input	Real part of complex matrix A (two-dimensional array type).
2	ai	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lna×nm	Input	Imaginary part of complex matrix A (two-dimensional array type).
3	lna	I	1	Input	Adjustable dimension of array ar and ai.
4	nm	I	1	Input	Number of rows in matrices AR and AI.
5	nn	I	1	Input	Number of columns in matrices AR and AI.
6	br	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lnb×nl	Input	Real part of complex matrix B (two-dimensional array type).
7	bi	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lnb×nl	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	lnb	I	1	Input	Adjustable dimension of array br and bi.
9	nl	I	1	Input	Number of columns in matrix B.
10	cr	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lmc×nl	Input	Real part of initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Real part of product ($C = [C\pm]A^* B$).
11	ci	$\left\{ \begin{array}{l} D* \\ R* \end{array} \right\}$	lmc×nl	Input	Imaginary part of initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Imaginary part of product ($C = [C\pm]A^* B$).
12	lmc	I	1	Input	Adjustable dimension of array cr and ci.
13	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^* B$ isw = 0: Obtain $C = A^* B$ isw = −1: Obtain $C = C - A^* B$
14	nt	I	1	Input	Number of tasks to be generated.
15	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $0 < \text{nm} \leq \text{lmc}$

(b) $0 < \text{nn} \leq \text{lna}, \text{lnb}$

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

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

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

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

Obtain $C = A^*B$.

(b) Input data
Matrix A, matrix B, lna = 11, lnb = 11, lnc = 11, nm = 4, nn = 5, nl = 6 and isw = 0.

(c) Main program

```
/*      C interface example for ASL_ham1hm */

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

int main()
{
    double *ar;
    double *ai;
    int lma=11;
    int nm=5;
    int nn=4;
    double *br;
    double *bi;
    int lnb=11;
    int nl=4;
    double *cr;
    double *ci;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_ham1hm ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt=%2d\n\n", isw, nt );

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

```
    return -1;
}

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

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

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        printf( "%8.3g ", ar[i+lma*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        printf( "%8.3g ", ai[i+lma*j]=j+1 );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", br[i+lnb*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", bi[i+lnb*j]=j+1 );
    }
    printf( "\n" );
}


ierr = ASL_ham1hm(ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);


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

printf( "\n\tReal part of matrix C\n" );
for( i=0 ; i<nm ; i++ )
```

```
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", cr[i+lmc*j] );
            }
            printf( "\n" );
        }
        printf( "\n\tImaginary part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", ci[i+lmc*j] );
            }
            printf( "\n" );
        }

        free( ar );
        free( ai );
        free( br );
        free( bi );
        free( cr );
        free( ci );

        return 0;
    }
```

(d) Output results

```
*** ASL_ham1hm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 5   nn = 4   nl = 4

isw= 0   nt= 2

Real part of matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4
Imaginary part of matrix A
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Real part of matrix B
       1        1        1        1
       2        2        2        2
       3        3        3        3
       4        4        4        4

Imaginary part of matrix B
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4

 ** Output **

ierr =      0

Real part of matrix C
      34       38       42       46
      38       46       54       62
      42       54       66       78
      46       62       78       94
      50       70       90      110

Imaginary part of matrix C
       0       10       20       30
     -10        0       10       20
     -20      -10        0       10
     -30      -20      -10        0
     -40      -30      -20      -10
```

2.2.9 ASL_ham1hh, ASL_gam1hh Multiplying Complex Matrices (Two-Dimensional Array Type) (Real Argument Type) $(C = C \pm A^* B^*)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_ham1hh (ar, ai, lna, nm, nn, br, bi, llb, nl, cr, ci, lmc, isw, nt);

Single precision:

ierr = ASL_gam1hh (ar, ai, lna, nm, nn, br, bi, llb, nl, cr, ci, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna×nm	Input	Real part of complex matrix A (two-dimensional array).
2	ai	{D*, R*}	lna×nm	Input	Imaginary part of complex matrix A (two-dimensional array).
3	lna	I	1	Input	Adjustable dimension of array ar and ai.
4	nm	I	1	Input	Number of rows in matrix A.
5	nn	I	1	Input	Number of columns in matrix A.
6	br	{D*, R*}	llb×nn	Input	Real part of complex matrix A (two-dimensional array type).
7	bi	{D*, R*}	llb×nn	Input	Imaginary part of complex matrix B (two-dimensional array type).
8	llb	I	1	Input	Adjustable dimension of array br and bi.
9	nl	I	1	Input	Number of columns in matrix B.
10	cr	{D*, R*}	lmc×nl	Input	Real part of initial complex matrix C (If isw $= \pm 1$) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]A^*B^*$).
11	ci	{D*, R*}	lmc×nl	Input	Imaginary part of initial complex matrix C (If isw $= \pm 1$) (two-dimensional array type).
				Output	Product of complex matrices ($C = [C\pm]A^*B^*$).
12	lmc	I	1	Input	Adjustable dimension of array cr and ci.
13	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^*B^*$ isw = 0: Obtain $C = A^*B^*$ isw = −1: Obtain $C = C - A^*B^*$
14	nt	I	1	Input	Number of tasks to be generated.
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 <$ nm $\leq$ lmc

(b) $0 <$ nn $\leq$ lna

(c) $0 <$ nl $\leq$ lnb

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

(e) nt ≥ 1

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

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

(b) Input data
Matrix A, matrix B, lna = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 6 and isw = 0.

(c) Main program

```
/*      C interface example for ASL_ham1hh */

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

int main()
{
    double *ar;
    double *ai;
    int lma=11;
    int nm=5;
    int nn=5;
    double *br;
    double *bi;
    int lnb=11;
    int nl=4;
    double *cr;
    double *ci;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


    printf( "    *** ASL_ham1hh ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt=%2d\n\n", isw, nt );

    ar = ( double * )malloc((size_t)( sizeof(double) * (lma*nm) ));
    if( ar == NULL )
```

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

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

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

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        printf( "%8.3g ", ar[i+lma*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        printf( "%8.3g ", ai[i+lma*j]=j+1 );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nl ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", br[i+lnb*j]=i+1 );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nl ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", bi[i+lnb*j]=j+1 );
    }
    printf( "\n" );
}


ierr = ASL_ham1hh(ar, ai, lma, nm, nn, br, bi, lnb, nl, cr, ci, lmc, isw, nt);


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

```
    printf( "\n\tReal part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", cr[i+lmc*j] );
        }
        printf( "\n" );
    }
    printf( "\n\tImaginary part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", ci[i+lmc*j] );
        }
        printf( "\n" );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( cr );
    free( ci );

    return 0;
}
```

(d) Output results

```
    *** ASL_ham1hh ***

     ** Input **

    lma=11   lnb=11   lmc=11

    nm = 5   nn = 5   nl = 4

    isw= 0   nt= 2

    Real part of matrix A
           1        1        1        1        1
           2        2        2        2        2
           3        3        3        3        3
           4        4        4        4        4
           5        5        5        5        5
    Imaginary part of matrix A
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5

    Real part of matrix B
           1        1        1        1        1
           2        2        2        2        2
           3        3        3        3        3
           4        4        4        4        4

    Imaginary part of matrix B
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5

     ** Output **

    ierr =      0

    Real part of matrix C
           0       15       30       45
         -15        0       15       30
         -30      -15        0       15
         -45      -30      -15        0
         -60      -45      -30      -15

    Imaginary part of matrix C
         -60      -65      -70      -75
         -65      -75      -85      -95
         -70      -85     -100     -115
         -75      -95     -115     -135
         -80     -105     -130     -155
```

2.2.10 ASL_han1mm, ASL_gan1mm Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB$)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_han1mm (a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_gan1mm (a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lma×nn	Input	Complex matrix A (two-dimensional array type).
2	lma	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D* / R*}	lnb×nl	Input	Complex matrix B (two-dimensional array type).
6	lnb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{D* / R*}	lmc×nl	Input	Initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB$).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB$ isw = 0: Obtain $C = AB$ isw = −1: Obtain $C = C - AB$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nn} \leq \mathrm{lnb}$

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

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

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

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB$.

(b) Input data

Matrix A, matrix B, lma = 11, lnb = 11, lnc = 11, nm = 4, nn = 5, nl = 4, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_han1mm */

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

int main()
{
    double _Complex *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double _Complex *b;
    int lnb=11;
    int nl=4;
    double _Complex *c;
    int lmc=11;
```

```
int isw=0;
int nt=2;
int ierr;
int i,j;

printf( "    *** ASL_han1mm ***\n" );
printf( "\n     ** Input **\n\n" );

printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        a[i+lma*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", cimag(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        b[i+lnb*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(b[i+lnb*j]) );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", cimag(b[i+lnb*j]) );
    }
    printf( "\n" );
}


ierr = ASL_han1mm(a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

printf( "\n\tReal part of matrix C\n" );
for( i=0 ; i<nm ; i++ )
```

```
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", creal(c[i+lmc*j]) );
        }
        printf( "\n" );
    }
    printf( "\n\tImaginary part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", cimag(c[i+lmc*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_han1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt = 2

Real part of matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4
Imaginary part of matrix A
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Real part of matrix B
       1        1        1        1
       2        2        2        2
       3        3        3        3
       4        4        4        4
       5        5        5        5

Imaginary part of matrix B
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4
       1        2        3        4

 ** Output **

ierr =      0

Real part of matrix C
       0      -15      -30      -45
      15        0      -15      -30
      30       15        0      -15
      45       30       15        0

Imaginary part of matrix C
      60       65       70       75
      65       75       85       95
      70       85      100      115
      75       95      115      135
```

2.2.11 ASL_han1mh, ASL_gan1mh Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) ($C = C \pm AB^*$)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_han1mh (a, lma, nm, nn, b, llb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_gan1mh (a, lma, nm, nn, b, llb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lma×nn	Input	Complex matrix A (two-dimensional array type).
2	lma	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{D* / R*}	llb×nn	Input	Complex matrix B (two-dimensional array type).
6	llb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of rows in matrix B.
8	c	{D* / R*}	lmc×nl	Input	Initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of real matrices ($C = [C\pm]AB^*$).
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + AB^*$ isw = 0: Obtain $C = AB^*$ isw = −1: Obtain $C = C - AB^*$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

(b) $0 < \mathrm{nl} \leq \mathrm{llb}$

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

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

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

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

Obtain $C = AB^*$.

(b) Input data
Matrix A, matrix B, lma = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 4, isw = 0 and nt = 2.

(c) Main program

```
/*      C interface example for ASL_han1mh */

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

int main()
{
    double _Complex *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double _Complex *b;
    int lnb=11;
    int nl=4;
    double _Complex *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
```

```
int i,j;

printf( "    *** ASL_han1mh ***\n" );
printf( "\n     ** Input **\n\n" );

printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        a[i+lma*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", cimag(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nl ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        b[i+lnb*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(b[i+lnb*j]) );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nl ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "%8.3g ", cimag(b[i+lnb*j]) );
    }
    printf( "\n" );
}


ierr = ASL_han1mh(a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

printf( "\n\tReal part of matrix C\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
```

```
        {
            printf( "%8.3g ", creal(c[i+lmc*j]) );
        }
        printf( "\n" );
    }
    printf( "\n\tImaginary part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nl ; j++ )
        {
            printf( "%8.3g ", cimag(c[i+lmc*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( c );

    return 0;
}
```

(d) Output results

```
*** ASL_han1mh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt = 2

Real part of matrix A
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4
Imaginary part of matrix A
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

Real part of matrix B
       1        1        1        1        1
       2        2        2        2        2
       3        3        3        3        3
       4        4        4        4        4

Imaginary part of matrix B
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5
       1        2        3        4        5

 ** Output **

ierr =      0

Real part of matrix C
      60       65       70       75
      65       75       85       95
      70       85      100      115
      75       95      115      135

Imaginary part of matrix C
       0       15       30       45
     -15        0       15       30
     -30      -15        0       15
     -45      -30      -15        0
```

2.2.12 ASL_han1hm, ASL_gan1hm Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) $(C = C \pm A^* B)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_han1hm (a, lna, nm, nn, b, lnb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_gan1hm (a, lna, nm, nn, b, lnb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z∗ / C∗}	lna×nm	Input	Complex Matrix A (two-dimensional array type).
2	lna	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns of matrix A.
5	b	{Z∗ / C∗}	lnb×nl	Input	Complex matrix B (two-dimensional array type).
6	lnb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns of matrix B.
8	c	{Z∗ / C∗}	lmc×nl	Input	Initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of complex matrices $(C = [C\pm] A^* B)$.
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^* B$ isw = 0: Obtain $C = A^* B$ isw = −1: Obtain $C = C - A^* B$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < \mathrm{nm} \leq \mathrm{lmc}$

(b) $0 < \mathrm{nn} \leq \mathrm{lna}, \mathrm{lnb}$

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

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

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

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

Obtain $C = A^*B$.

(b) Input data
Matrix A, Matrix B, lna $= 11$, lnb $= 11$, lnc $= 11$, nm $= 4$, nn $= 5$, nl $= 6$ and isw $= 0$.

(c) Main program

```
/*      C interface example for ASL_han1hm */

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

int main()
{
    double _Complex *a;
    int lma=11;
    int nm=5;
    int nn=4;
    double _Complex *b;
    int lnb=11;
    int nl=4;
    double _Complex *c;
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
```

```
int i,j;

printf( "    *** ASL_han1hm ***\n" );
printf( "\n     ** Input **\n\n" );

printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

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

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

printf( "\tReal part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        a[i+lma*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\tImaginary part of matrix A\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nm ; j++ )
    {
        printf( "%8.3g ", cimag(a[i+lma*j]) );
    }
    printf( "\n" );
}
printf( "\n\tReal part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        b[i+lnb*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
        printf( "%8.3g ", creal(b[i+lnb*j]) );
    }
    printf( "\n" );
}
printf( "\n\tImaginary part of matrix B\n" );
for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
    {
        printf( "%8.3g ", cimag(b[i+lnb*j]) );
    }
    printf( "\n" );
}


ierr = ASL_han1hm(a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

printf( "\n\tReal part of matrix C\n" );
for( i=0 ; i<nm ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<nl ; j++ )
```

```
            {
                printf( "%8.3g ", creal(c[i+lmc*j]) );
            }
            printf( "\n" );
        }
        printf( "\n\tImaginary part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", cimag(c[i+lmc*j]) );
            }
            printf( "\n" );
        }

        free( a );
        free( b );
        free( c );

        return 0;
    }
```

(d) Output results

```
    *** ASL_han1hm ***

     ** Input **

    lma=11   lnb=11   lmc=11

    nm = 5   nn = 4   nl = 4

    isw= 0   nt = 2

    Real part of matrix A
           1        1        1        1        1
           2        2        2        2        2
           3        3        3        3        3
           4        4        4        4        4
    Imaginary part of matrix A
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5

    Real part of matrix B
           1        1        1        1
           2        2        2        2
           3        3        3        3
           4        4        4        4

    Imaginary part of matrix B
           1        2        3        4
           1        2        3        4
           1        2        3        4
           1        2        3        4

     ** Output **

    ierr =      0

    Real part of matrix C
          34       38       42       46
          38       46       54       62
          42       54       66       78
          46       62       78       94
          50       70       90      110

    Imaginary part of matrix C
           0       10       20       30
         -10        0       10       20
         -20      -10        0       10
         -30      -20      -10        0
         -40      -30      -20      -10
```

2.2.13 ASL_han1hh, ASL_gan1hh Multiplying Complex Matrices (Two-Dimensional Array Type) (Complex Argument Type) $(C = C \pm A^*B^*)$

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_han1hh (a, lna, nm, nn, b, llb, nl, c, lmc, isw, nt);

Single precision:

ierr = ASL_gan1hh (a, lna, nm, nn, b, llb, nl, c, lmc, isw, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z∗ / C∗}	lna×nm	Input	Complex matrix A (two-dimensional array type).
2	lna	I	1	Input	Adjustable dimension of array a.
3	nm	I	1	Input	Number of rows in matrix A.
4	nn	I	1	Input	Number of columns in matrix A.
5	b	{Z∗ / C∗}	llb×nn	Input	Complex matrix B (two-dimensional array type).
6	llb	I	1	Input	Adjustable dimension of array b.
7	nl	I	1	Input	Number of columns in matrix B.
8	c	{Z∗ / C∗}	lmc×nl	Input	Initial complex matrix C (If isw = ±1) (two-dimensional array type).
				Output	Product of real matrices $(C = [C\pm]A^*B^*)$.
9	lmc	I	1	Input	Adjustable dimension of array c.
10	isw	I	1	Input	Processing switch. isw = 1: Obtain $C = C + A^*B^*$ isw = 0: Obtain $C = A^*B^*$ isw = −1: Obtain $C = C - A^*B^*$
11	nt	I	1	Input	Number of tasks to be generated.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 <$ nm $\leq$ lmc

(b) $0 <$ nn $\leq$ lna

(c) $0 <$ nl $\leq$ lnb

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

(e) nt ≥ 1

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

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

(6) **Notes**
None

(7) **Example**

(a) Problem

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

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

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

(b) Input data

Matrix A, matrix B, lna = 11, llb = 11, lnc = 11, nm = 4, nn = 5, nl = 6 and isw = 0.

(c) Main program

```
/*      C interface example for ASL_han1hh */

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

int main()
{
    double _Complex *a;
    int lma=11;
    int nm=5;
    int nn=5;
    double _Complex *b;
    int lnb=11;
    int nl=4;
    double _Complex *c;
    int lmc=11;
```

```
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;

    printf( "    *** ASL_han1hh ***\n" );
    printf( "\n     ** Input **\n\n" );

    printf( "\tlma=%2d   lnb=%2d   lmc=%2d\n\n", lma, lnb, lmc );
    printf( "\tnm =%2d   nn =%2d   nl =%2d\n\n", nm,  nn,  nl  );
    printf( "\tisw=%2d   nt =%2d\n\n", isw, nt );

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

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

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

    printf( "\tReal part of matrix A\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nm ; j++ )
        {
            a[i+lma*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
            printf( "%8.3g ", creal(a[i+lma*j]) );
        }
        printf( "\n" );
    }
    printf( "\tImaginary part of matrix A\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nm ; j++ )
        {
            printf( "%8.3g ", cimag(a[i+lma*j]) );
        }
        printf( "\n" );
    }
    printf( "\n\tReal part of matrix B\n" );
    for( i=0 ; i<nl ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            b[i+lnb*j] = (double)(i+1) + (double)(j+1) * _Complex_I;
            printf( "%8.3g ", creal(b[i+lnb*j]) );
        }
        printf( "\n" );
    }
    printf( "\n\tImaginary part of matrix B\n" );
    for( i=0 ; i<nl ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "%8.3g ", cimag(b[i+lnb*j]) );
        }
        printf( "\n" );
    }


    ierr = ASL_han1hh(a, lma, nm, nn, b, lnb, nl, c, lmc, isw, nt);


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

    printf( "\n\tReal part of matrix C\n" );
    for( i=0 ; i<nm ; i++ )
```

```
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", creal(c[i+lmc*j]) );
            }
            printf( "\n" );
        }
        printf( "\n\tImaginary part of matrix C\n" );
        for( i=0 ; i<nm ; i++ )
        {
            printf( "\t" );
            for( j=0 ; j<nl ; j++ )
            {
                printf( "%8.3g ", cimag(c[i+lmc*j]) );
            }
            printf( "\n" );
        }

        free( a );
        free( b );
        free( c );

        return 0;
    }
```

(d) Output results

```
    *** ASL_han1hh ***

     ** Input **

    lma=11   lnb=11   lmc=11

    nm = 5   nn = 5   nl = 4

    isw= 0   nt = 2

    Real part of matrix A
           1        1        1        1        1
           2        2        2        2        2
           3        3        3        3        3
           4        4        4        4        4
           5        5        5        5        5
    Imaginary part of matrix A
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5

    Real part of matrix B
           1        1        1        1        1
           2        2        2        2        2
           3        3        3        3        3
           4        4        4        4        4

    Imaginary part of matrix B
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5
           1        2        3        4        5

     ** Output **

    ierr =      0

    Real part of matrix C
           0       15       30       45
         -15        0       15       30
         -30      -15        0       15
         -45      -30      -15        0
         -60      -45      -30      -15

    Imaginary part of matrix C
         -60      -65      -70      -75
         -65      -75      -85      -95
         -70      -85     -100     -115
         -75      -95     -115     -135
         -80     -105     -130     -155
```

Chapter 3

SIMULTANEOUS LINEAR EQUATIONS (DIRECT METHOD)

3.1 INTRODUCTION

This chapter describes functions that solve simultaneous linear equations and obtain the determinant value and inverse matrix of a matrix.
Function described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

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.
For explanations of functions (4) and (5) (See Chapter 2 of <Basic Functions Vol. 2> .)

3.1.1 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

(a) Using ASL_qbgmsl

ierr = ASL_qbgmsl $(A, \cdots, \boldsymbol{b}, \cdots)$;

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

(b) Using ASL_qbgmlu and $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$

ierr = ASL_qbgmlu $(A, \cdots)$;

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

ASL_qbgmlu performs a triangular decomposition of coefficient matrix A, and $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$.

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

ierr = ASL_qbgmlc $(A, \cdots,$ &cond, $\cdots)$;

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

ASL_qbgmlc calculates the condition number and performs a triangular decomposition of coefficient matrix A, and $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$.

(2) Determinant and inverse matrix

ierr = ASL_qbgmlu $(A, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmdi}\\ \text{ASL_rbgmdi}\end{array}\right\}$ $(A, \cdots,$ det, $\cdots)$;

ASL_qbgmlu performs a triangular decomposition of matrix A, and $\left\{\begin{array}{l}\text{ASL_dbgmdi}\\ \text{ASL_rbgmdi}\end{array}\right\}$ (See Section 2.2.7 of <Basic Functions Vol. 2>) obtains the determinant and inverse matrix.

(3) Improving the solution

(a) Using ASL_qbgmsl

$A_2 \leftarrow A$

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

ierr = ASL_qbgmsl $(A_2, \cdots, \boldsymbol{b}_2, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmlx}\\ \text{ASL_rbgmlx}\end{array}\right\}$ $(A, \cdots, A_2, \cdots, \boldsymbol{b}, \cdots, \boldsymbol{b}_2, \cdots)$;

The function shown above improves the solution obtained by using ASL_qbgmsl.

(b) Using ASL_qbgmlu and $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$

$A_2 \leftarrow A$

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

ierr = ASL_qbgmlu $(A_2, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ $(A_2, \cdots, \boldsymbol{b}_2, \cdots)$;

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

ASL_qbgmlu performs a triangular decomposition of matrix A, $\begin{Bmatrix} \text{ASL_dbgmls} \\ \text{ASL_rbgmls} \end{Bmatrix}$ (See Section 2.2.5 of <Basic Functions Vol. 2>) solves $A\boldsymbol{x} = \boldsymbol{b}$, and $\begin{Bmatrix} \text{ASL_dbgmlx} \\ \text{ASL_rbgmlx} \end{Bmatrix}$ (See Section 2.2.8 of <Basic Functions Vol. 2>) improves the solution.

3.1.2 Notes

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

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

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

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

execute either:

ierr = ASL_qbgmsl $(A, \cdots, \boldsymbol{b}_1, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ $(A, \cdots, \boldsymbol{b}_2, \cdots)$;

or

ierr = ASL_qbgmlu $(A, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ $(A, \cdots, \boldsymbol{b}_1, \cdots)$;

ierr = $\left\{\begin{array}{l}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{array}\right\}$ $(A, \cdots, \boldsymbol{b}_2, \cdots)$;

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

(4) 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 a matrix of order n, 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.

3.1.3 Algorithms Used

3.1.3.1 Solution of Simultaneous Linear Equations

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

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

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

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

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

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

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

3.1.3.2 LU Decomposition (Gauss Method)

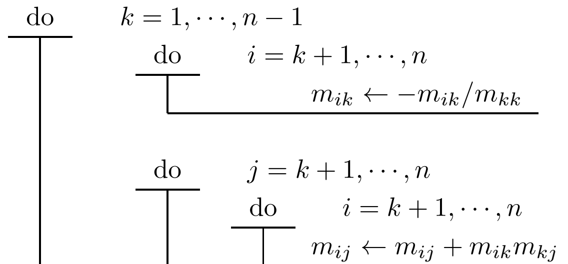
Consider the decomposition of the regular matrix $A = (a_{ij})$ for $(i, j = 1, 2, \cdots, n)$ into $PA = LU$ as the product of the lower triangular matrix $L = (l_{ij})$ for $(i, j = 1, 2, \cdots, n)$ and upper triangular matrix $U = (u_{ij})$ for $(i, j = 1, 2, \cdots, n)$. This kind of decomposition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

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

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

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

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

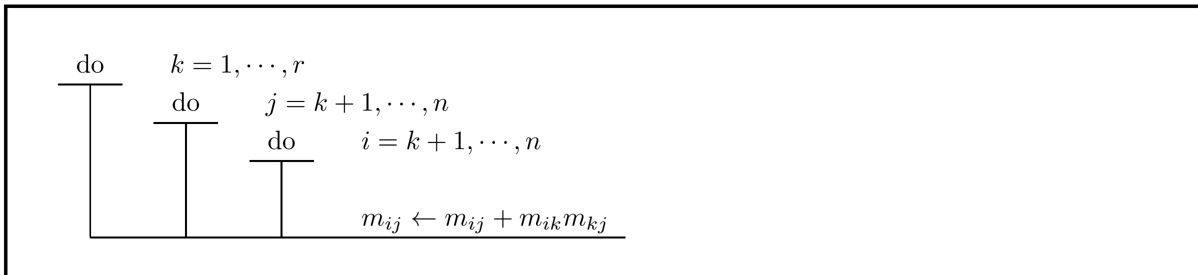


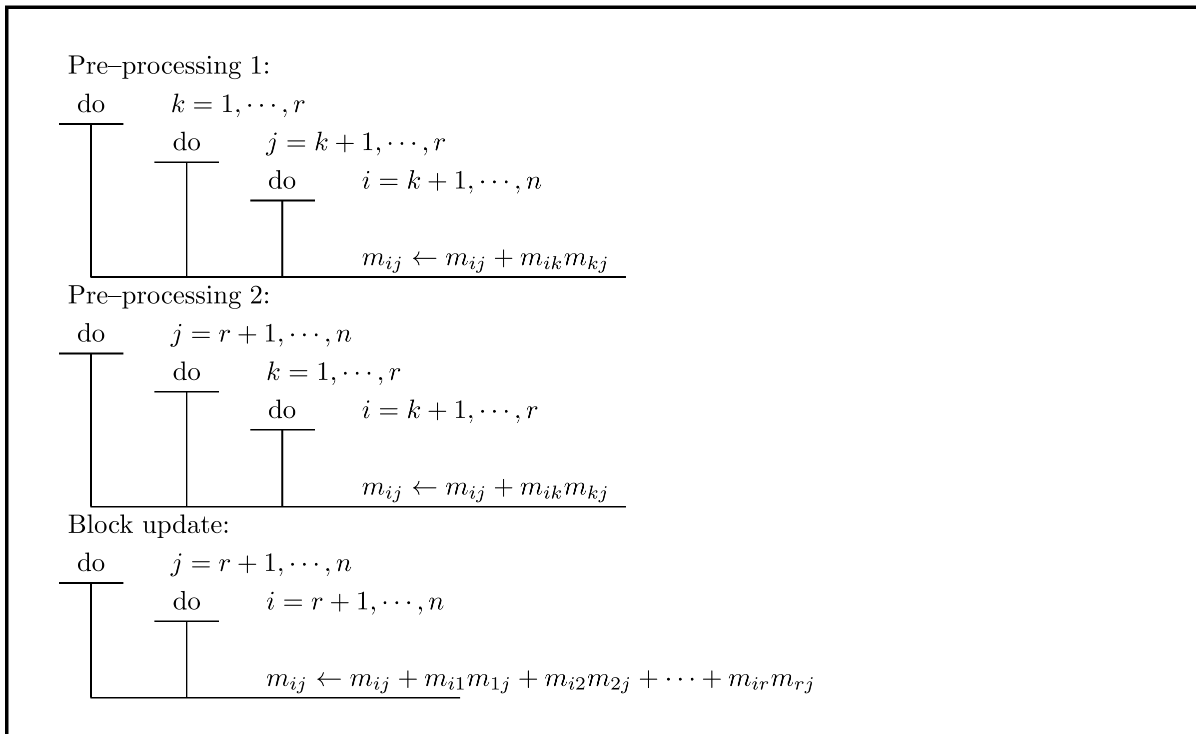
To prevent division by an element $m_{kk} \simeq 0$ when implementing the algorithm shown above, select the element m_{pk} for which the following equation holds:

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

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

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





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

Parallelization of this algorithm is described below.

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

3.1.4 Reference Bibliography

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

3.2 REAL MATRIX (TWO-DIMENSIONAL ARRAY TYPE)

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

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qbgmsm (ab, lna, n, m, ipvt, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

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

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

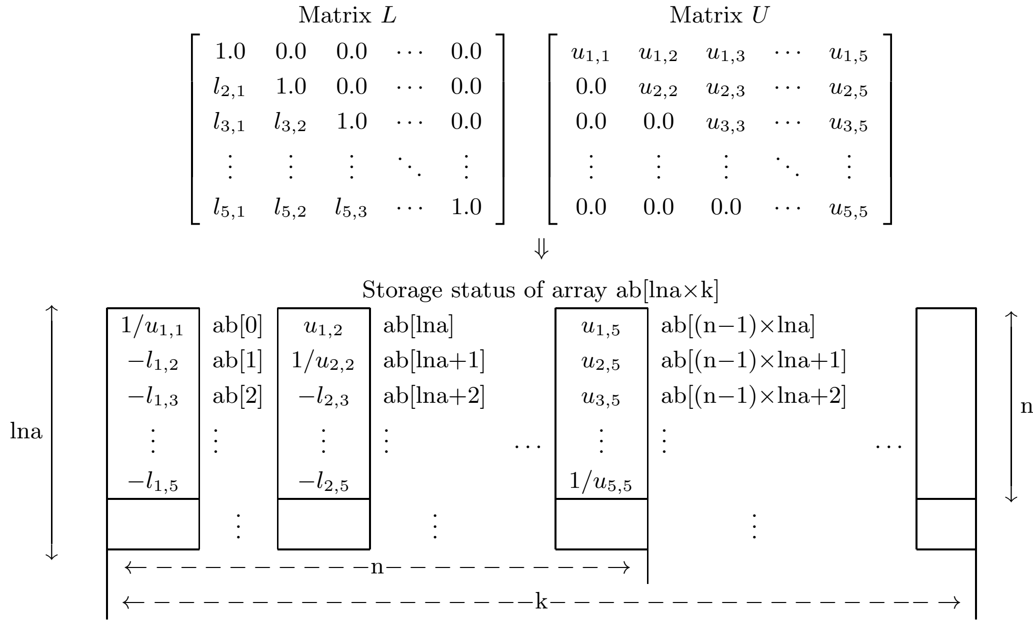
ierr value	Meaning	Processing
0	Normal termination.	
1000	n = 1	$\mathrm{ab[lna} * (\mathrm{n} + \mathrm{i} - 1)] \leftarrow \mathrm{ab[lna} * (\mathrm{n} + \mathrm{i} - 1)]/\mathrm{ab}[0]$ ($i = 1, 2, \cdots, \mathrm{m}$) is performed.
2100	There existed the diagonal element which was close to zero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

(6) **Notes**

(a) This function perform partial pivoting when obtaining the LU decomposition of coefficient matrix A. If the pivot row in the i-th step is row j ($\mathrm{i} \leq \mathrm{j}$), then j is stored in ipvt[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 3−1 Storage Status of Matrices L and U



Remarks

a. lna ≥ n and n+m ≤ k must be hold.

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

(7) **Example**

(a) ProblemSolve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main program

```
/*      C interface example for ASL_qbgmsm */

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

int main()
{
    double *ab;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    int nt=2;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_qbgmsm ***\n" );
```

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

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

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

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

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

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

    fclose( fp );


    ierr = ASL_qbgmsm(ab, lna, n, m, ipvt, nt);

    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_qbgmsm ***

 ** Input **

Coefficient Matrix

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

Constant Vectors

      36        11
      15         0
      22         7
      -6         4
```

```
 ** Output **

ierr =      0

Solution

       1         1
       2         1
       4         1
       5         1
```

3.2.2 ASL_qbgmsl Simultaneous Linear Equations (Real Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qbgmsl (a, lna, n, b, ipvt, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

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

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

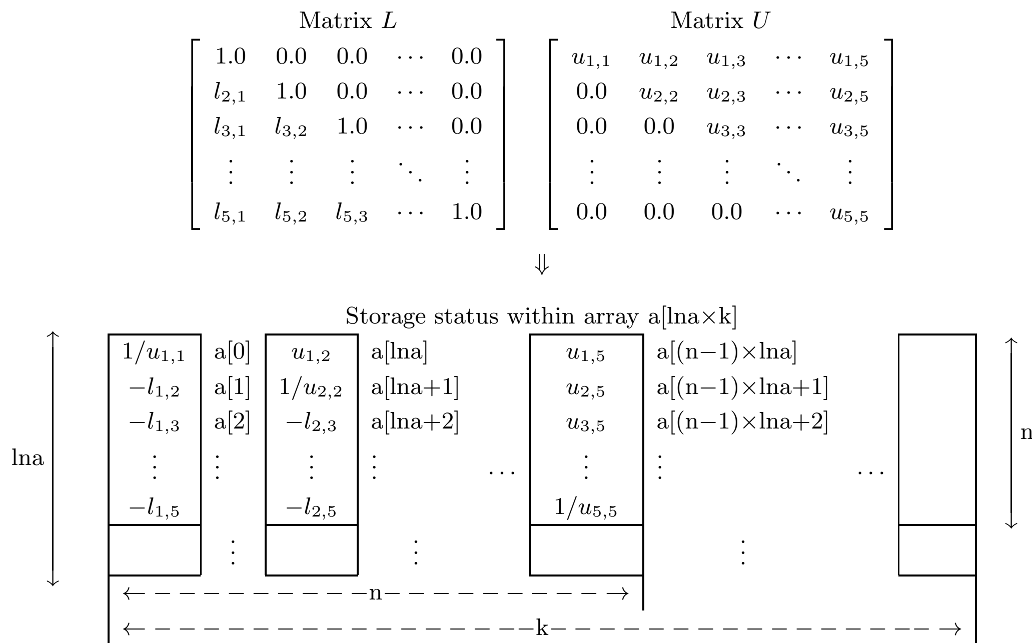
(6) **Notes**

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

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

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

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



Remarks

a. lna ≥ n and n ≤ k must hold.

Figure 3−2 Storage Status of Matrices L and U

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, lna=11, n=4, constant vector $\boldsymbol{b}$ and nt=2.

(c) Main Program

```
/*      C interface example for ASL_qbgmsl */

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

int main()
{
    double *a;
    int lna;
    int n;
    double *b;
    int *ipvt;
    int ierr;
    int nt = 2;
    int i,j;
    FILE *fp;

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

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

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

        fclose( fp );


        ierr = ASL_qbgmsl(a, lna, n, b, ipvt, nt);


        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_qbgmsl ***

         ** Input **

         n  =      4
         nt =      2

        Coefficient Matrix

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

        Constant Vector

              36
              15
              22
              -6
```

```
 ** Output **

ierr =      0

Solution

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

3.2.3 ASL_qbgmlu LU Decomposition of a Real Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qbgmlu (a, lna, n, ipvt, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	D*	lna×n	Input	Real matrix A (two-dimensional array type)
				Output	Upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (b))
5	nt	I	1	Input	Number of tasks to be generated
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

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

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

(6) **Notes**

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

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

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

3.2.4 ASL_qbgmlc LU Decomposition and Condition Number of a Real Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_qbgmlc (a, lna, n, ipvt, &cond, w1, nt);
Single precision:
Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

(b) $\mathrm{nt} \ge 1$

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

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

(6) **Notes**

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

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

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

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

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

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

(1) **Function**

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

(2) **Usage**

Double precision:

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

Single precision:

Nothing

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	abr	D*	lna×(n + m)	Input	Real part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Real part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
2	abi	D*	lna×(n + m)	Input	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of coefficient matrix A and right-hand side vectors $\boldsymbol{b_i}$ $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				Output	Imaginary part of matrix (complex matrix, two-dimensional array type) consisting of the factored matrix A' of coefficient matrix A and solution vectors $\boldsymbol{x_i}$ $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays abr and abi
4	n	I	1	Input	Order of matrix A
5	m	I	1	Input	Number of right-hand side vectors, m
6	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step. (See Note (a))
7	w1	D*	n	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

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

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

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

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1	abr[lna∗(n+i-1)] ← (abr[lna∗(n+i−1)] × abr[0] + abi[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2) , abi[lna∗(n+i−1)] ← (abi[lna∗(n+i−1)] × abr[0] − abr[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2) (i=1, 2, $\cdots$, m)
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is singular.	

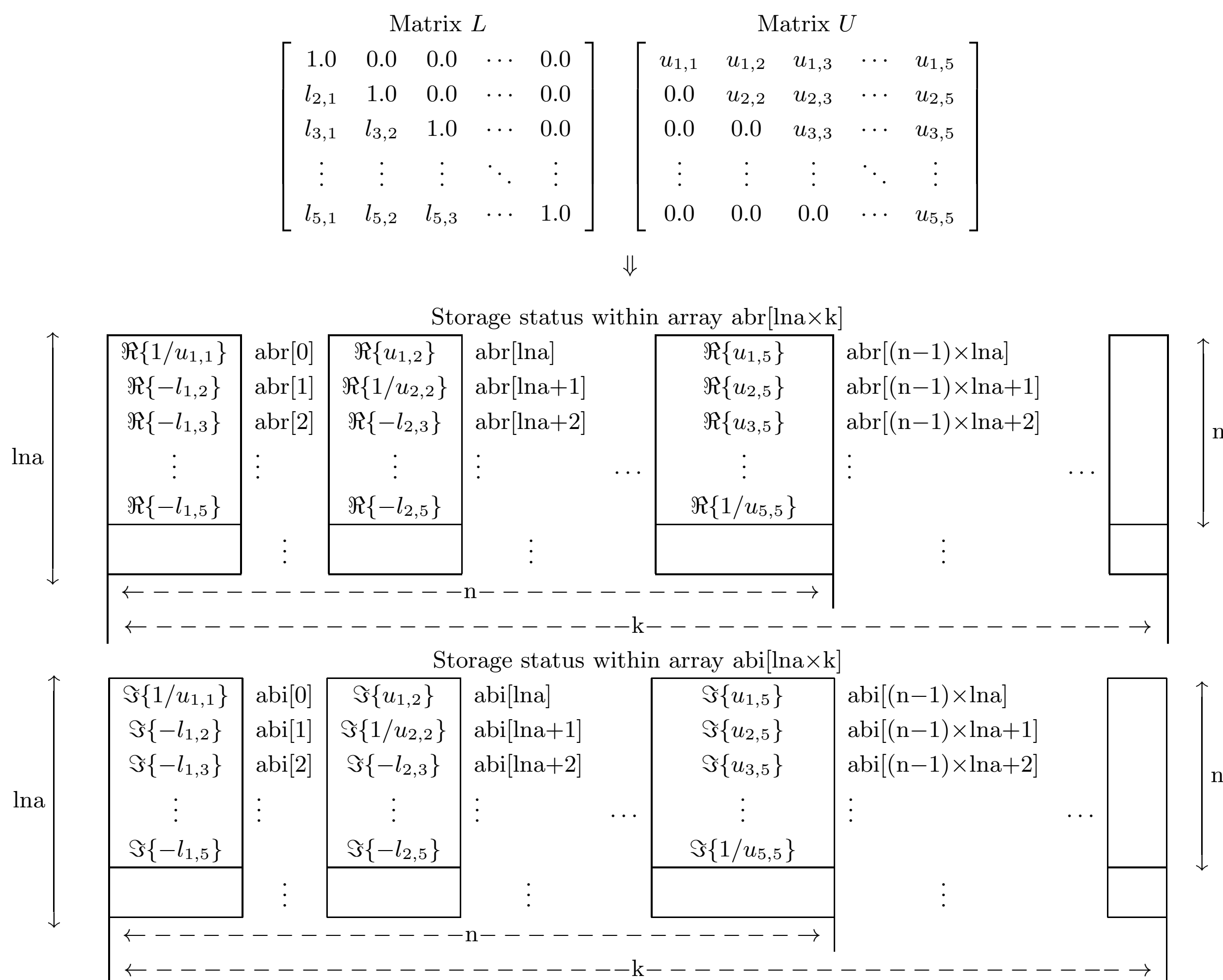
(6) **Notes**

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

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

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

Figure 3−3 Storage Status of Matrices L and U



Remarks

a. lna ≥ n and n+m ≤ k must be hold.

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main program

```
/*      C interface example for ASL_hbgmsm */

#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;
    int nt=2;
    FILE *fp;

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

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

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

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

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

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

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

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

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

    fclose( fp );


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


    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_hbgmsm ***

 ** Input **

 n =      4 m =      4

Coefficient Matrix

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

Constant Vectors

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

 ** Output **

ierr =      0

Solution

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

3.3.2 ASL_hbgmsl Simultaneous Linear Equation (Complex Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_hbgmsl (ar, ai, lna, n, br, bi, ipvt, w, nt);
Single precision:
  Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

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

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

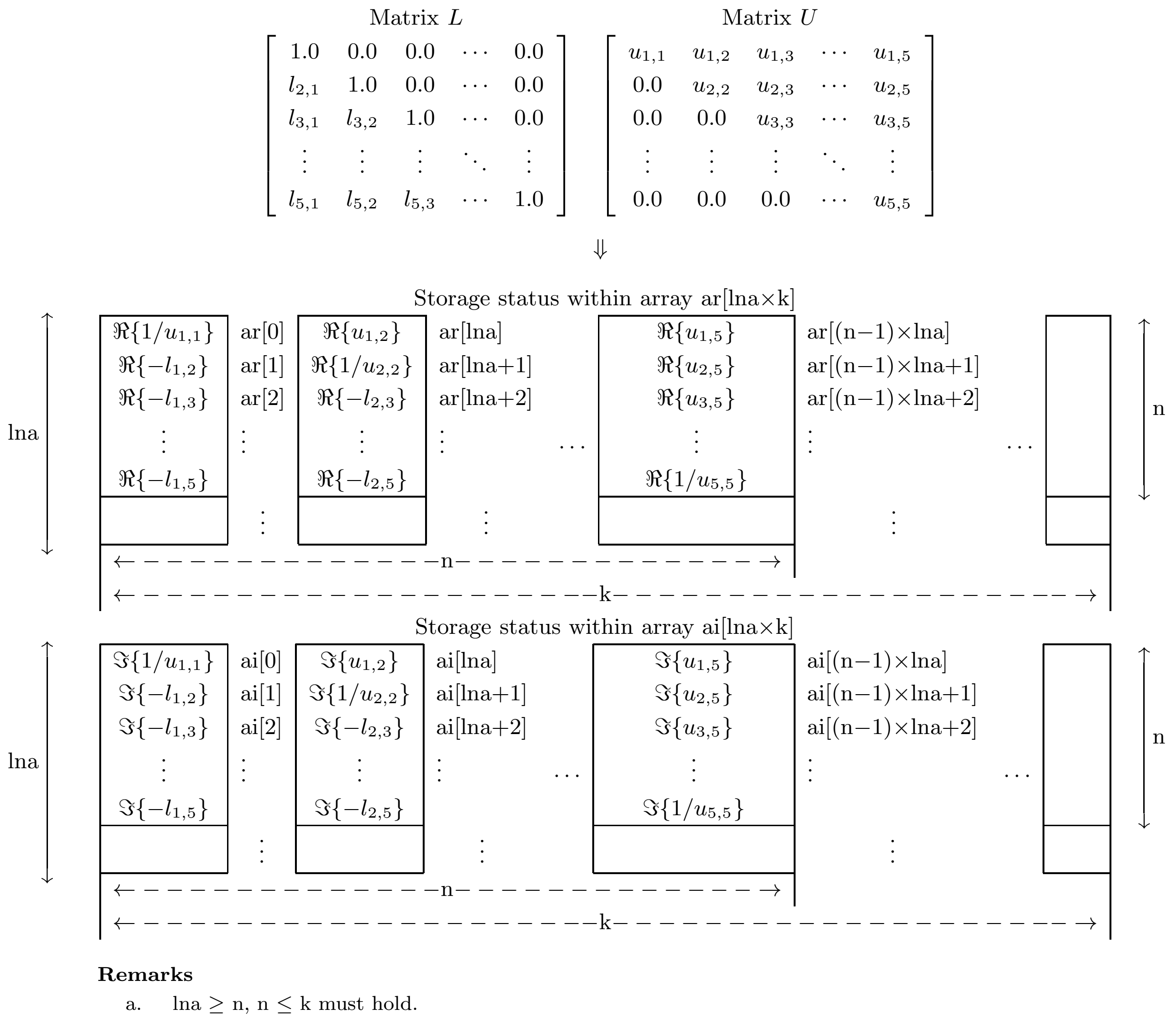
(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, the solution is obtained more efficiently by directly using the function3.3.1 ASL_hbgmsm to perform the calculations. However, when 3.3.1 ASL_hbgmsm cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this function only once and then call function 2.3.5 {ASL_zbgmls, ASL_cbgmls} (in <Basic Functions Vol. 2>) the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculations by performing the LU decomposition of matrix A only once.

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

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

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



Remarks

a. lna ≥ n, n ≤ k must hold.

Figure 3−4 Storage Status of Matrices L and U

(7) **Example**

(a) Problem

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

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

(b) Input data

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

(c) Main program

```
/*      C interface example for ASL_hbgmsl */

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

int main()
{
```

```
double *ar;
double *ai;
int na;
int n;
int nt = 2;
double *br;
double *bi;
int *kpvt;
double *w;
int ierr;
int i,j;
FILE *fp;

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

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

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

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

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

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

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

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

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

printf( "\t n  = %6d\n", n );
printf( "\t nt = %6d\n", nt );
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_hbgmsl(ar, ai, na, n, br, bi, kpvt, w, nt);

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

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

        return 0;
    }
```

(d) Output results

```
        *** ASL_hbgmsl ***

         ** Input **

         n  =      4
         nt =      2

        Coefficient Matrix (Real, Imaginary)

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

        Constant Vector (Real, Imaginary)

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

         ** Output **

        ierr =      0

        Solution (Real, Imaginary)

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

3.3.3 ASL_hbgmlu LU Decomposition of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbgmlu (ar, ai, lna, n, ipvt, w, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	D*	lna×n	Input	Real part of complex matrix A (two-dimensional array type)
				Output	Real part of upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	ai	D*	lna×n	Input	Imaginary part of complex matrix A (two-dimensional array type)
				Output	Imaginary part of upper triangular matrix U and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i int the i-th processing step. (See Note (b))
6	w	D*	n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < n \leq lna$

(b) $nt \geq 1$

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

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

(6) **Notes**

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

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

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

3.3.4 ASL_hbgmlc LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbgmlc (ar, ai, lna, n, ipvt, &cond, w, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	D*	lna×n	Input	Real part of complex matrix A (two-dimensional array type)
				Output	Real part of upper triangular matrix L and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
2	ai	D*	lna×n	Input	Imaginary part of complex matrix A (two-dimensional array type)
				Output	Imaginary part of upper triangular matrix L and lower triangular matrix L when A is decomposed into $A = LU$. (See Notes (a) and (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of row exchanged with row i in the i-th processing step (See Note (b))
6	cond	D*	1	Output	Reciprocal of the condition number
7	w	D*	$2 \times n$	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

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

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

(6) **Notes**

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

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

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

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

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

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

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbgnsm (ab, lna, n, m, ipvt, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

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

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

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

(6) **Notes**

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

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

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

(7) **Example**

(a) Problem
Solve the following simultaneous linear equations.

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

(b) Input data
Array ab in which coefficient matrix A and constant vectors $\boldsymbol{b_1}$, $\boldsymbol{b_2}$, $\boldsymbol{b_3}$ and $\boldsymbol{b_4}$ are stored, lna=11, n=4, m=4 and nt=2.

(c) Main program

```
/*      C interface example for ASL_hbgnsm */

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

int main()
{
    double _Complex *ab;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    int nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

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

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

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

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

    fclose( fp );

    ierr = ASL_hbgnsm(ab, lna, n, m, ipvt, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
```

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

    free( ab );
    free( ipvt );

    return 0;
}
```

(d) Output results

```
*** ASL_hbgnsm ***

 ** Input **

 n =      4 m =      4

Coefficient Matrix

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

Constant Vectors

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

 ** Output **

ierr =      0

Solution

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

3.4.2 ASL_hbgnsl Simultaneous Linear Equations (Complex Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:
 ierr = ASL_hbgnsl (a, lna, n, b, ipvt, nt);
Single precision:
 Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$ and $\mathrm{nt} \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	b[0] ← b[0]/a[0] is performed.
2100	There existed the diagonal element which was close to zero in the LU decomposition of the coefficient matrix A. The result may not be obtained with a good accuracy.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
4000+i	The pivot became 0.0 in the i-th processing step of the LU decomposition of coefficient matrix A. A is 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 function3.4.1 ASL_hbgnsm to perform the calculations. However, when 3.4.1 ASL_hbgnsm cannot be used such as when all of the right-hand side vectors $\boldsymbol{b}$ are not known in advance, call this function only once and then call function 2.4.5 $\left\{\begin{array}{l}\text{ASL_zbgnls}\\ \text{ASL_cbgnls}\end{array}\right\}$ the required number of times varying only the contents of b. This enables you to eliminate unnecessary calculation by performing the LU decomposition of matrix A only once.

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

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

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

(7) **Example**

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

Coefficient matrix A, lna = 11, n = 4, constant vector $\boldsymbol{b}$ and nt=2.

(c) Main program

```
/*      C interface example for ASL_hbgnsl */

#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 nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

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

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

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

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

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

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

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

    fclose( fp );

    ierr = ASL_hbgnsl(a, lna, n, b, ipvt, nt);

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

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

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

    return 0;
}
```

(d) Output results

```
*** ASL_hbgnsl ***

 ** Input **

 n =      4

Coefficient Matrix   (Real, Imaginary)

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

Constant Vector  (Real, Imaginary)

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

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

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

3.4.3 ASL_hbgnlu LU Decomposition of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbgnlu (a, lna, n, ipvt, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

(a) $0 < n \leq lna$ and $nt \geq 1$

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

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

(6) **Notes**

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

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

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

3.4.4 ASL_hbgnlc LU Decomposition and Condition Number of a Complex Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbgnlc (a, lna, n, ipvt, &cond, w1, nt);

Single precision:

Nothing

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

(a) $0 < \mathrm{n} \leq \mathrm{lna}$ and $\mathrm{nt} \geq 1$

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

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

(6) **Notes**

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

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

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

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

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

3.5.1 ASL_qbspsl, ASL_pbspsl Simultaneous Linear Equations (Real Symmetric Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_qbspsl (a, lna, n, b, ipvt, wk, nt);

Single precision:

ierr = ASL_pbspsl (a, lna, n, b, ipvt, wk, nt);

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

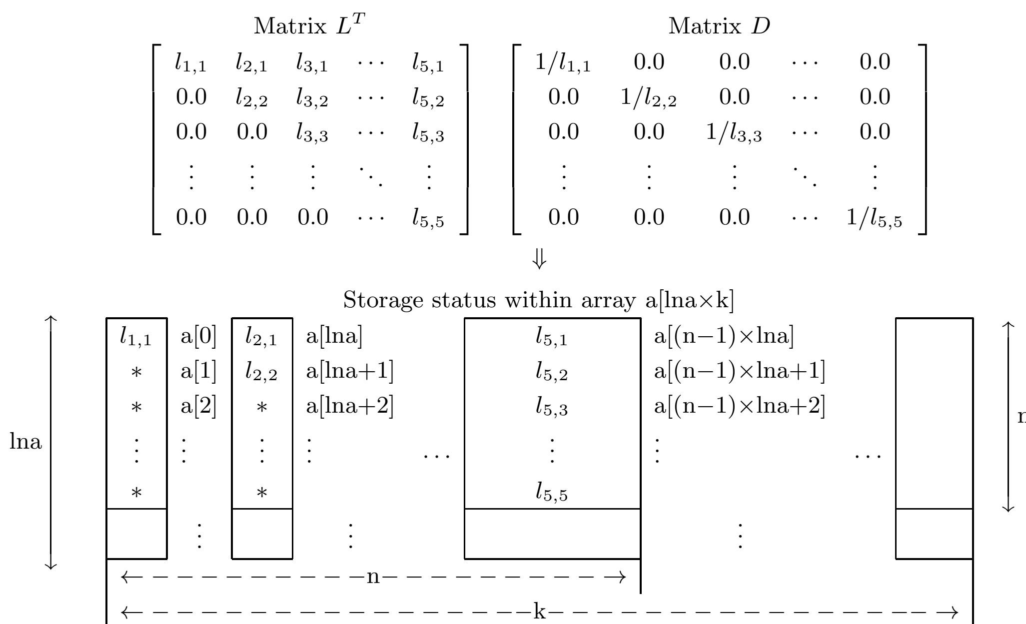
(5) **Error indicator (Return Value)**

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

(6) **Notes**

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

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



Remarks

a. lna ≥ n and n ≤ k must hold.

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

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

(c) This function performs partial pivoting when obtaining the $\mathrm{LDL^T}$ decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*      C interface example for ASL_qbspsl */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int *ipvt;
    double *wk;
    int nt = 2;
    int ierr;
```

```
int i,j;

FILE *fp;

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

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

fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );
a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( a == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}

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

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

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

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

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


fclose( fp );


ierr = ASL_qbspsl(a, na, n, b, ipvt, wk, nt);


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


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

free( a );
free( b );
free( ipvt );
free( wk );
```

```
    return 0;
}
```

(d) Output results

```
*** ASL_qbspsl ***

 ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

 ** Output **

ierr =      0

Solution

  x[     0] =        1
  x[     1] =       -1
  x[     2] =        2
  x[     3] =       -2
```

3.5.2 ASL_qbspud, ASL_pbspud LDL^T Decomposition of a Real Symmetric Matrix

(1) **Function**

ASL_qbspud or ASL_pbspud uses the modified Cholesky method to perform an LDL^T decomposition of the real symmetric matrix A (two-dimensional array type).

(2) **Usage**

Double precision:

ierr = ASL_qbspud (a, lna, n, ipvt, wk, nt);

Single precision:

ierr = ASL_pbspud (a, lna, n, ipvt, wk, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Upper triangular matrix L^T when A is decomposed into $A = LDL^T$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	N	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	wk	{D* / R*}	n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

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

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

(6) **Notes**

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

(b) This function performs partial pivoting when obtaining the LDL^{T} decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

3.6.1 ASL_qbsnsl, ASL_pbsnsl
Simultaneous Linear Equations (Real Symmetric Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_qbsnsl (a, lna, n, b, nt);
Single precision:
  ierr = ASL_pbsnsl (a, lna, n, b, nt);

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

(b) $\text{nt} \ge 1$

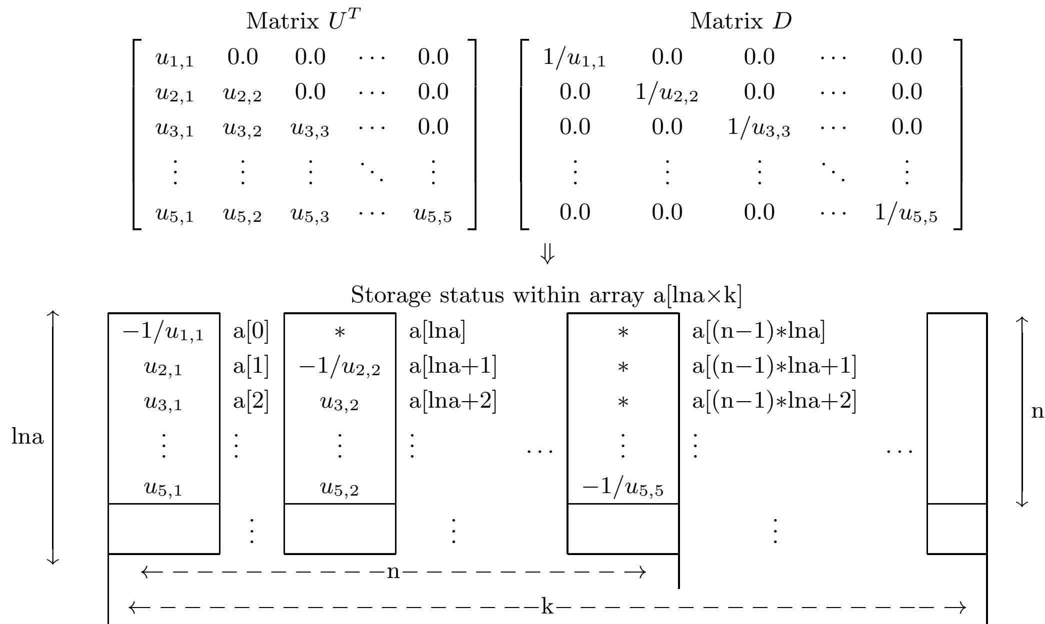
(5) **Error indicator (Return Value)**

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

(6) **Notes**

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

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



Remarks

a. lna ≥ n and n ≤ k must hold.

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

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

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*    C interface example for ASL_qbsnsl */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int ierr;

    int i,j;
    int nt = 2;

    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

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

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

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

    fclose( fp );


    ierr = ASL_qbsnsl(a, na, n, b, nt);

    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_qbsnsl ***

  ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

  ** Output **

ierr =      0

Solution
```

```
x[     0] =         1
x[     1] =        -1
x[     2] =         2
x[     3] =        -2
```

3.6.2 ASL_qbsnud, ASL_pbsnud $\mathrm{U^TDU}$ Decomposition of a Real Symmetric Matrix (No Pivoting)

(1) **Function**

ASL_qbsnud or ASL_pbsnud uses the modified Cholesky method to perform an $\mathrm{U^TDU}$ decomposition of the real symmetric matrix A (two-dimensional array type) (lower triangular type).

(2) **Usage**

Double precision:
ierr = ASL_qbsnud (a, lna, n, nt);
Single precision:
ierr = ASL_pbsnud (a, lna, n, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (lower triangular type)
				Output	Lower triangular matrix U^T when A is decomposed into $A = U^T DU$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	nt	I	1	Input	Number of tasks to be generated
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Contents of array a are not changed.
2100	There existed the diagonal element which was close to zero in the $\mathrm{U^T DU}$ 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 $\mathrm{U}^T\mathrm{DU}$ decomposition of coefficient matrix A. A is singular.	

(6) **Notes**

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

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

3.7.1 ASL_hbhpsl, ASL_gbhpsl Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbhpsl (ar, ai, lna, n, br, bi, ipvt, w1, nt);

Single precision:

ierr = ASL_gbhpsl (ar, ai, lna, n, br, bi, ipvt, w1, nt);

(3) Arguments and Return Value

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

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

(4) Restrictions

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

(b) $\text{nt} \geq 1$

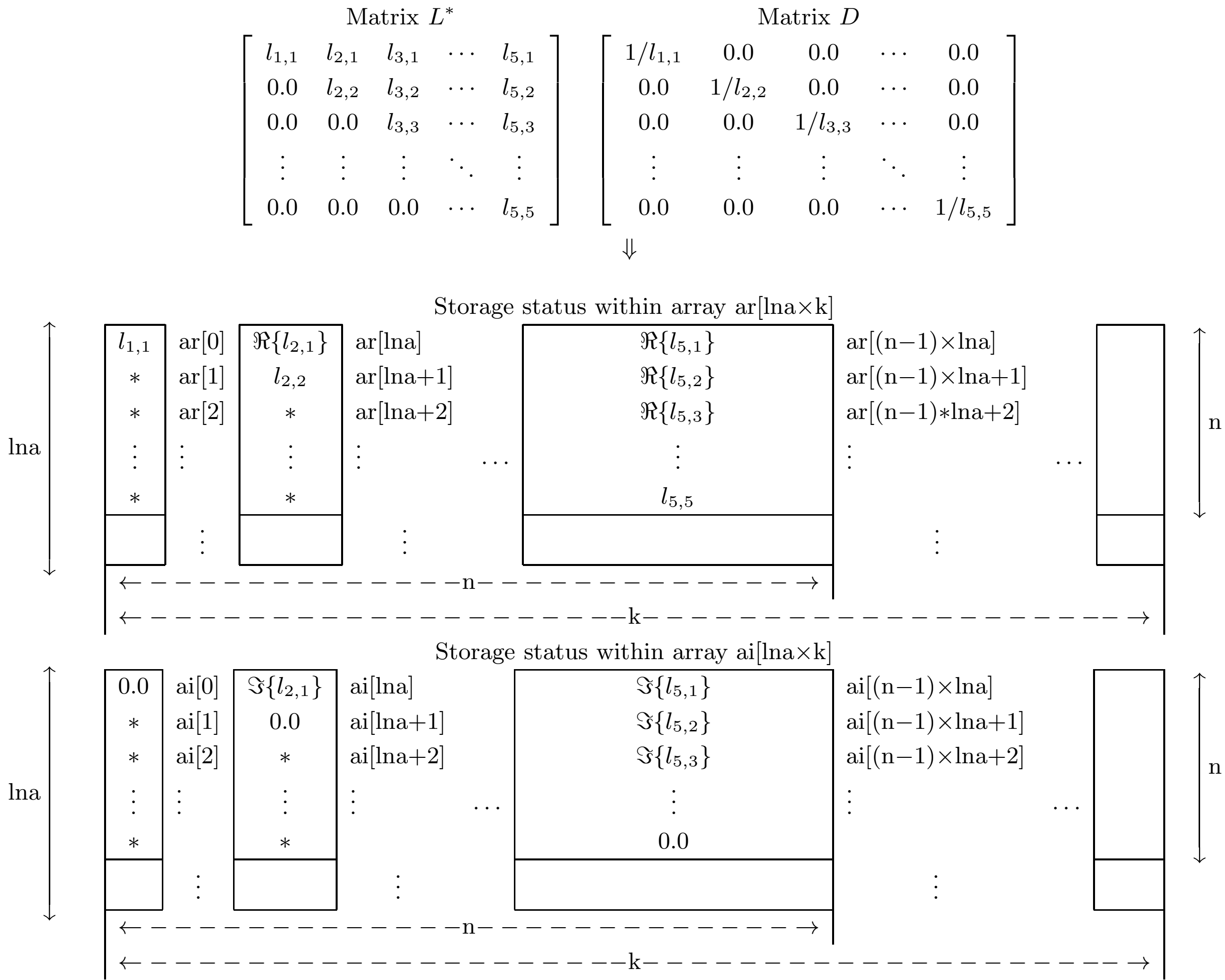
(5) **Error indicator (Return Value)**

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

(6) **Notes**

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

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



Remarks

a. lna ≥ n and n ≤ k must hold.

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

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

(c) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*      C interface example for ASL_hbhpsl */

#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 nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_hbhpsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );

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

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

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

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

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

    w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

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

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

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

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
    }

    fclose( fp );


    ierr = ASL_hbhpsl(ar, ai, na, n, br, bi, ipvt, w1, nt);


    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_hbhpsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector  (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 ,        0)
  x[     2] = (-6.63e-17 ,        1)
  x[     3] = (2.09e-17 ,        1)
```

3.7.2 ASL_hbhpud, ASL_gbhpud
 LDL* Decomposition of a Hermitian Matrix

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbhpud (ar, ai, lna, n, ipvt, w1, nt);

Single precision:

ierr = ASL_gbhpud (ar, ai, lna, n, ipvt, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	{D*, R*}	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
6	w1	{D*, R*}	$2 \times$ n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

(b) $\text{nt} \geq 1$

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

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

(6) **Notes**

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

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

3.8.1 ASL_hbhrsl, ASL_gbhrsl Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbhrsl (ar, ai, lna, n, br, bi, w1, nt);

Single precision:

ierr = ASL_gbhrsl (ar, ai, lna, n, br, bi, w1, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna×n	Input	Real part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	ai	{D*, R*}	lna×n	Input	Imaginary part of coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	br	{D*, R*}	n	Input	Real part of constant vector $\boldsymbol{b}$
				Output	Real part of solution $\boldsymbol{x}$
6	bi	{D*, R*}	n	Input	Imaginary part of constant vector $\boldsymbol{b}$
				Output	Imaginary part of solution $\boldsymbol{x}$
7	w1	{D*, R*}	n	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

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

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

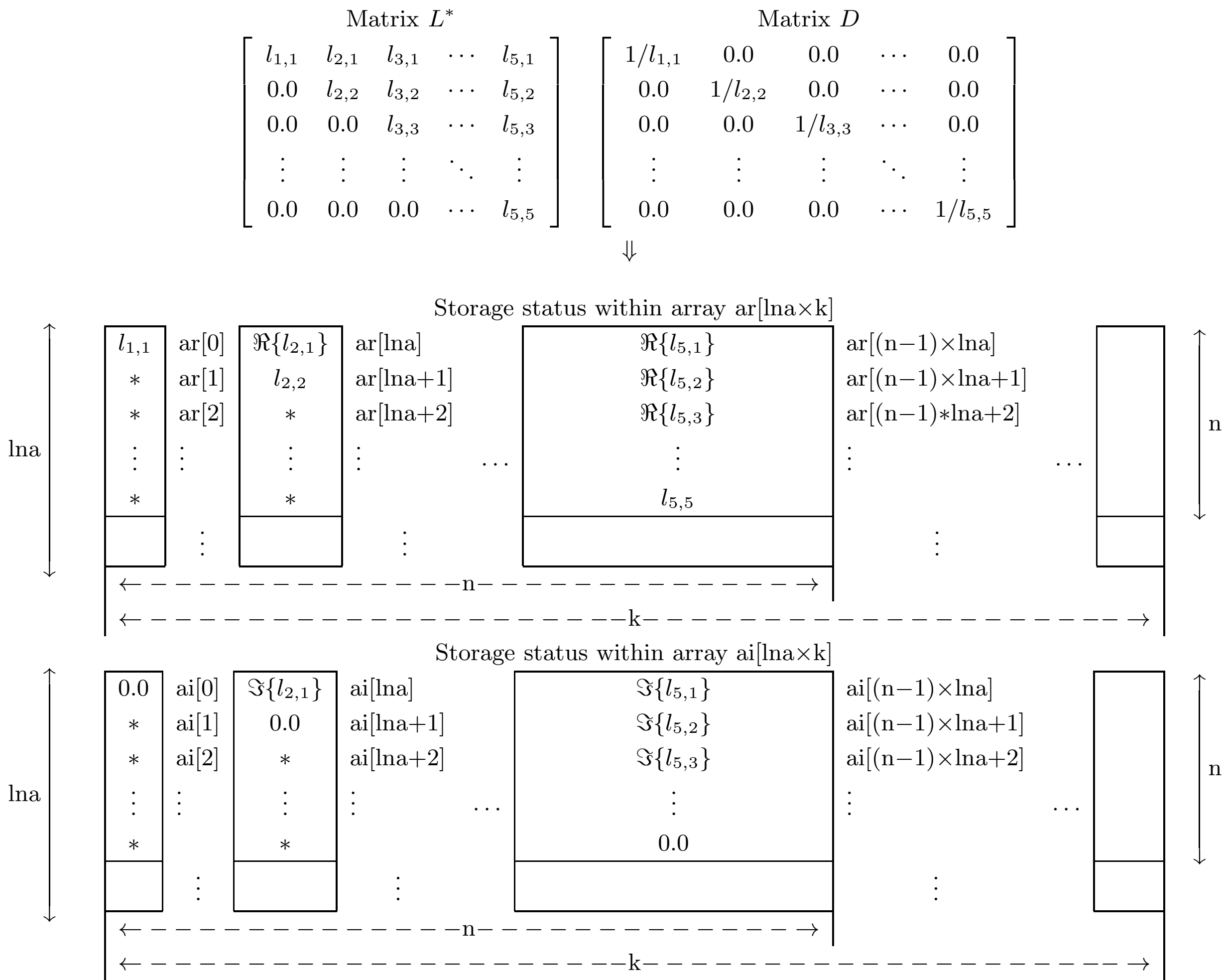
(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function <Basic Functions Vol. 2> 2.10.4 $\left\{\begin{array}{c}\text{ASL_zbhrls}\\ \text{ASL_cbhrls}\end{array}\right\}$ 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. 3−8.)



Remarks

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

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

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

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*      C interface example for ASL_hbhrsl */

#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 nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_hbhrsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );

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

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

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

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

    w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

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

    printf( "\n\tConstant Vector  (Real, Imaginary)\n\n");
```

```
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &br[i] );
        }
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &bi[i] );
        }

        for( i=0 ; i<n ; i++ )
        {
            printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
        }

        fclose( fp );


        ierr = ASL_hbhrsl(ar, ai, na, n, br, bi, w1, nt);


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

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

        free( ar );
        free( ai );
        free( br );
        free( bi );
        free( w1 );

        return 0;
    }
```

(d) Output results

```
        *** ASL_hbhrsl ***

         ** Input **

         n =      4

        Coefficient Matrix  (Real, Imaginary)

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

        Constant Vector  (Real, Imaginary)

        (      10 ,        6)
        (      11 ,        2)
        (       4 ,        6)
        (       4 ,        6)

         ** Output **

        ierr =      0

        Solution   (Real, Imaginary)

          x[     0] = (       1 , 1.48e-16)
          x[     1] = (       1 , -3.9e-17)
          x[     2] = (-1.02e-16 ,        1)
          x[     3] = (8.34e-17 ,        1)
```

3.8.2 ASL_hbhrud, ASL_gbhrud LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbhrud (ar, ai, lna, n, w1, nt);

Single precision:

ierr = ASL_gbhrud (ar, ai, lna, n, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* / R*}	lna×n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Real part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	ai	{D* / R*}	lna×n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Imaginary part of upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
3	lna	I	1	Input	Adjustable dimension of array ar and ai
4	n	I	1	Input	Order of matrix A
5	w1	{D* / R*}	n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

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

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

(6) **Notes**

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

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

3.9.1 ASL_hbhfsl, ASL_gbhfsl Simultaneous Linear Equations (Hermitian Matrix)

(1) **Function**

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

(2) **Usage**

Double precision:

ierr = ASL_hbhfsl (a, lna, n, b, ipvt, w1, nt);

Single precision:

ierr = ASL_gbhfsl (a, lna, n, b, ipvt, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z*, C*}	lna × n	Input	Coefficient matrix A (Hermitian matrix, two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (b))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	b	{Z*, C*}	n	Input	Constant vector $\boldsymbol{b}$
				Output	Solution $\boldsymbol{x}$
5	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (c))
6	w1	{D*, R*}	n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

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

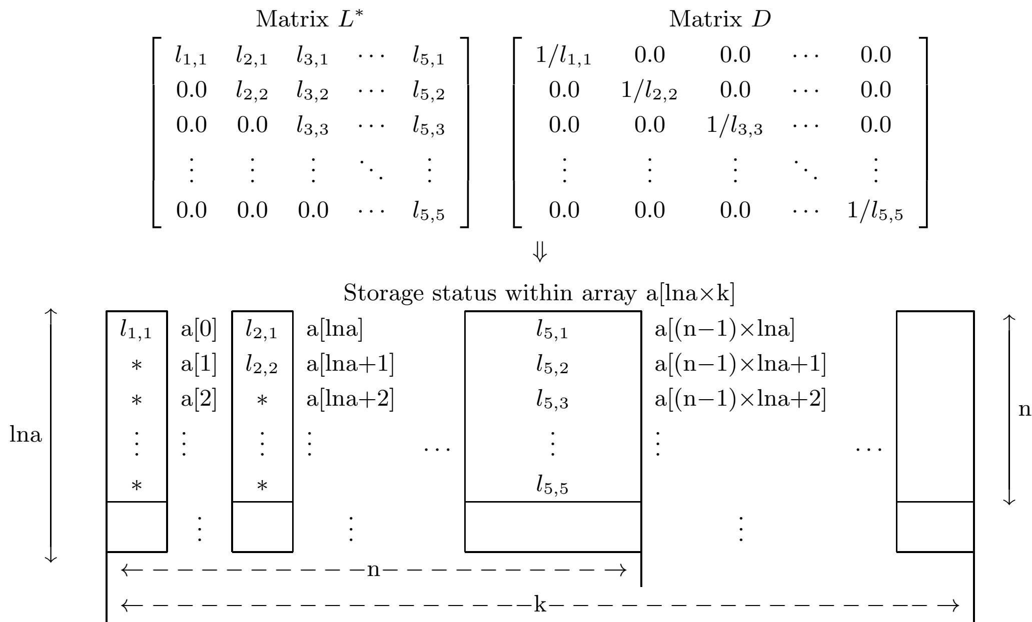
(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function <Basic Functions Vol. 2> 2.11.4 {ASL_zbhfls, ASL_cbhfls} 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.



Remarks

a. lna ≥ n and n ≤ k must hold.

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

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

(c) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*      C interface example for ASL_hbhfsl */

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

int main()
{
    double _Complex *a;
    int na;
    int n;
    double _Complex *b;
    int *ipvt;
    double *w1;
    int nt=2;
```

```
int ierr;
int i,j;
FILE *fp;

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

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

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

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

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

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

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

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

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

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    double tmp_re;
    fscanf( fp, "%lf", &tmp_re );
    b[i] = tmp_re;
}
for( i=0 ; i<n ; i++ )
{
```

```
            double tmp_im;
            fscanf( fp, "%lf", &tmp_im );
            b[i] = b[i] + tmp_im * _Complex_I;
        }
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
        }
        fclose( fp );

        ierr = ASL_hbhfsl(a, na, n, b, ipvt, w1, nt);

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

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

        free( a );
        free( b );
        free( ipvt );
        free( w1 );

        return 0;
    }
```

(d) Output results

```
        *** ASL_hbhfsl ***

         ** Input **

         n =      4

        Coefficient Matrix  (Real, Imaginary)

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

        Constant Vector (Real, Imaginary)

        (      10 ,        6)
        (      11 ,        2)
        (       4 ,        6)
        (       4 ,        6)

         ** Output **

        ierr =      0

        Solution  (Real, Imaginary)

          x[     0] = (       1 ,        0)
          x[     1] = (       1 ,        0)
          x[     2] = (-6.63e-17 ,        1)
          x[     3] = (2.09e-17 ,        1)
```

3.9.2 ASL_hbhfud, ASL_gbhfud
LDL* Decomposition of a Hermitian Matrix

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_hbhfud (a, lna, n, ipvt, w1, nt);
Single precision:
ierr = ASL_gbhfud (a, lna, n, ipvt, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z* / C*}	lna × n	Input	Hermitian matrix A (two-dimensional array type, upper triangular type)
				Output	Upper triangular matrix L^* when A is decomposed into $A = LDL^*$ (See Note (a))
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i − 1]: Number of the row(column) exchanged with row(column) i in the i-th processing step. (See Note (b))
5	w1	{D* / R*}	n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < n \leq lna$

(b) $nt \geq 1$

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

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

(6) **Notes**

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

(b) This function performs partial pivoting when obtaining the LDL* decomposition of coefficient matrix A. The permutation of rows and columns is symmetrical for row and column. If the pivot row(column) in the i-th step is row(column) j (i < j), then j is stored in ipvt[i-1]. In addition, among the column(row) elements corresponding to row(column) i and row(column) j of matrix A, elements from column(row) i to column(row) n actually are exchanged at this time.

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

3.10.1 ASL_hbhesl, ASL_gbhesl Simultaneous Linear Equations (Hermitian Matrix) (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
  ierr = ASL_hbhesl (a, lna, n, b, nt);
Single precision:
  ierr = ASL_gbhesl (a, lna, n, b, nt);

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

(b) $\mathrm{nt} \ge 1$

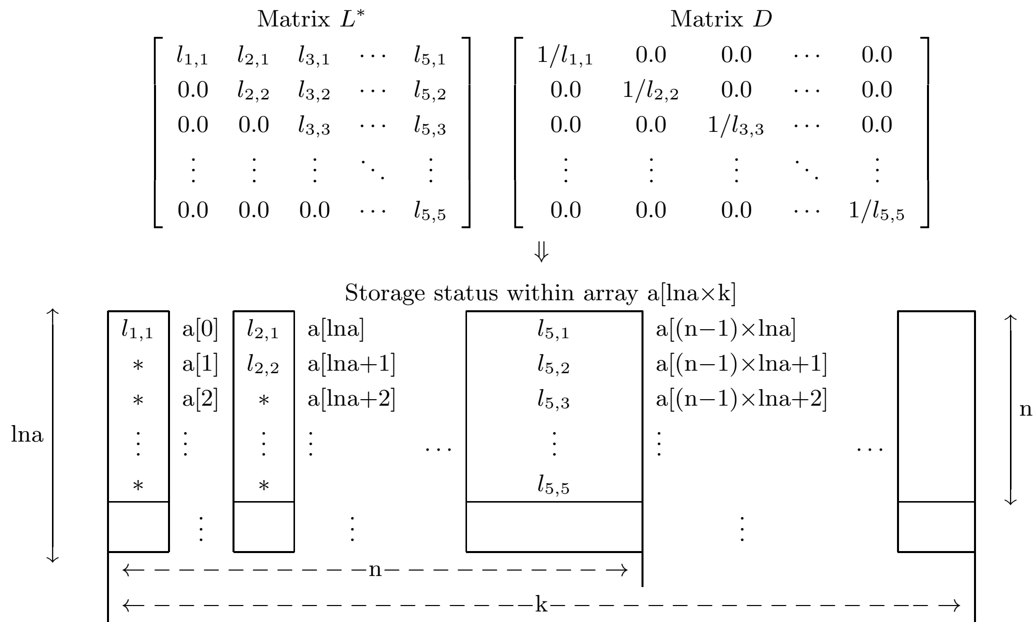
(5) **Error indicator (Return Value)**

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

(6) **Notes**

(a) To solve multiple sets of simultaneous linear equations where only the constant vector $\boldsymbol{b}$ differs, call this function only once and then call function <Basic Functions Vol. 2> 2.12.4 {ASL_zbhels, ASL_cbhels} 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.



Remarks

a. lna ≥ n and n ≤ k must hold.

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

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

(7) Example

(a) Problem

Solve the following simultaneous linear equations.

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

(b) Input data

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

(c) Main Program

```
/*      C interface example for ASL_hbhesl */

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

int main()
{
    double _Complex *a;
    int na;
    int n;
    double _Complex *b;
    int nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_hbhesl ***\n" );
```

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

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

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

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

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

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

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    double tmp_re;
    fscanf( fp, "%lf", &tmp_re );
    b[i] = tmp_re;
}
for( i=0 ; i<n ; i++ )
{
    double tmp_im;
    fscanf( fp, "%lf", &tmp_im );
    b[i] = b[i] + tmp_im * _Complex_I;
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
}
fclose( fp );


ierr = ASL_hbhesl(a, na, n, b, nt);


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

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

```
    free( a );
    free( b );

    return 0;
}
```

(d) Output results

```
*** ASL_hbhesl ***

 ** Input **

 n =       4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector (Real, Imaginary)

(       10 ,         6)
(       11 ,         2)
(        4 ,         6)
(        4 ,         6)

 ** Output **

ierr =       0

Solution  (Real, Imaginary)

  x[      0] = (        1 , -9.87e-17)
  x[      1] = (        1 , -6.25e-17)
  x[      2] = (5.11e-17 ,         1)
  x[      3] = (        0 ,         1)
```

3.10.2 ASL_hbheud, ASL_gbheud LDL* Decomposition of a Hermitian Matrix (No Pivoting)

(1) **Function**

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

(2) **Usage**

Double precision:
ierr = ASL_hbheud (a, lna, n, nt);
Single precision:
ierr = ASL_gbheud (a, lna, n, nt);

(3) **Arguments and Return Value**

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

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

(4) **Restrictions**

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

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

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

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

(6) **Notes**

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

Chapter 4

SIMULTANEOUS LINEAR EQUATIONS (ITERATIVE METHOD)

4.1 INTRODUCTION

This chapter describes functions that use iterative methods to solve simultaneous linear equations having a large dimensional sparse matrix as coefficient matrix.
Function described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

This library having the functions listed below use either the nonstationary iterative methods as the basic iterative algorithms. In the nonstationary iterative methods are included the methods such as the CG iterative method or the GMRES method.
For the preconditioner, the Scaling method are supported.
Among these functions, one matrix storage format for and two matrix storage formats for random sparse matrices (ELLPACK format) are available.
These functions use the same user interface regardless of the iterative basic method. The users can only have to change the functions name which is called in the user's program to try different iterative methods under the the same preconditioning and the same storage format.

Table 4−1 Function for each basic iterative method

BASIC ITERATIVE METHOD	FUNCTION NAME
CG (For Symmetric Matrices only) Conjugate Gradient method	ASL_qxe010 ASL_pxe010
CGS Conjugate Gradient Squared method	ASL_qxe020 ASL_pxe020
BiCGSTAB Bi-Conjugate Gradient Stabilized method	ASL_qxe030 ASL_pxe030
GMRES General Minimal Residual method	ASL_qxe040 ASL_pxe040

4.1.1 Notes

(1) Proper use of iterative methods

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

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

(2) Matrix storage format

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

However, the coefficient matrix of simultaneous linear equations generated by using the finite element method to discretize a randomly shaped area generally has no regularity other than the fact that the diagonal elements are nonzero elements. This type of matrix is called an random (irregular) sparse matrix. The functions described in this chapter support a storage format for random (irregular) sparse matrices.

Each function has a common user interface, with a single exception that the GMRES method functions have an additional input argument (gmitr). Therefore, the users can only have to replace the function name in their programs if they want to try different iterative methods with the input values unchanged.(As for trying GMRES method, an input argument (gmitr) should also be added to the user interface.) However, attention must be paid to the fact that the CG method functions cannot be applied to problems for which the coefficient matrix is asymmetric, since the CG method is for solving positive definite symmetric simultaneous linear equations.

(3) Preconditioning methods

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

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

(4) Conditions for terminating iterations

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

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

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

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

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

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

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

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

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

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

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

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

4.1.2 Algorithms Used

4.1.2.1 Nonstationary iterative method (for Symmetric Matrix only)

(1) **the CG method**

Consider the following simultaneous linear equations of order n:

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

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

$$\boldsymbol{u} = \boldsymbol{u}_1 \qquad (\boldsymbol{u}_1 \text{ is the initial solution}),$$
$$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1,$$
$$\boldsymbol{p}_1 = \boldsymbol{r}_1.$$

for $i = 1, 2, \cdots, n$

$$a_i = \frac{(\boldsymbol{r}_i, \boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)},$$
$$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i,$$
$$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_iA\boldsymbol{p}_i,$$
$$b_i = \frac{(\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, \boldsymbol{r}_i)},$$
$$\boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i.$$

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

4.1.2.2 Nonstationary iterative method (for Asymmetric Matrix)

(1) **the CGS and BiCGSTAB methods**

To explain the CGS and BiCGSTAB methods, we will first explain the BiCG method. The BiCG method considers the simultaneous linear equations of order $2n$ formed by combining the equations shown in (1) with the dual equations as follows:

$$\tilde{A}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{b}},$$
$$\tilde{A} = \begin{bmatrix} A & O \\ O & A^T \end{bmatrix}, \tilde{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{u}^* \end{bmatrix}, \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}^* \end{bmatrix}$$

and uses the conjugate gradient method to obtain the stationary value of:

$$F(\tilde{\boldsymbol{u}}) =< (\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}), \quad A(\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}) > H \quad (\hat{\boldsymbol{u}}; \text{Truesolution})$$

where,

$$< \boldsymbol{u}, \boldsymbol{v} > H \equiv (\boldsymbol{u}, H\boldsymbol{v}) = (H\boldsymbol{u}, \boldsymbol{v}), \quad H = \begin{bmatrix} O & I \\ I & O \end{bmatrix}.$$

Using the initial residual vector $\boldsymbol{r}_0$ the residual vector after k iterations $\boldsymbol{r}_k$ and direction vector after k iterations $\boldsymbol{p}_k$ are represented by:

$$\boldsymbol{r}_k = R_k(A)\boldsymbol{r}_0,$$
$$\boldsymbol{p}_k = P_k(A)\boldsymbol{r}_0.$$

Here, $R_k(A)$, $P_k(A)$ are polynomials generated from A. At this time, the CGS method uses this $R_k(A)$ and $P_k(A)$ to calculate new vectors $\boldsymbol{r}'_k$ and $\boldsymbol{p}'_k$ as follows.

$$\begin{aligned}\boldsymbol{r}'_k &= R_k^2(A)\boldsymbol{r}_0,\\ \boldsymbol{p}'_k &= P_k^2(A)\boldsymbol{r}_0\end{aligned}$$

Using this calculation method, as

$$\|\boldsymbol{r}_k\| = \|R_k(A)\boldsymbol{r}_0\|$$

gets smaller,

$$\|\boldsymbol{r}'_k\| = \|R_k^2(A)\boldsymbol{r}_0\|$$

is expected to get even smaller, and convergence is expected to be faster than with the BCG method.

The method in which the iterations proceed so that the residual becomes $\boldsymbol{r}'_k$ is the CGS method.

However, the CGS method may exhibit extremely irregular convergence characteristics. In particular, the residual may be extremely large at the beginning of the iterations such as when the iteration initial value is close to the solution, and accurate iteration calculations may not be able to be performed due to the effect of rounding error. The BiCGSTAB method lets $Q_k(A)$ represent the following:

$$Q_k(A) = (1-\omega_1 A)(1-\omega_2 A)\cdots(1-\omega_k A)$$

and obtains the approximate solution $\boldsymbol{u}_k$ so that the residual

$$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{u}_k$$

in each iteration is given by:

$$\boldsymbol{r}_k = Q_k(A)R_k(A)\boldsymbol{r}_0$$

Here, the parameters $\omega_k$that minimize $(\boldsymbol{r}_k, \boldsymbol{r}_k)$ are selected so that the residuals are reduced stably.

(2) **the GMRES method (GMRES(m))**

The GMRES method is an iterative method that for each $j = 1, 2, \cdots$ searches for the vector $\boldsymbol{z}^{(j)}$ that minimizes the L^2norm of

$$\boldsymbol{b} - A(\boldsymbol{u}_0^{(j)} + \boldsymbol{z}^{(j)}) = \boldsymbol{r}^{(j)} - A\boldsymbol{z}^{(j)}$$

from the space spanned by

$$\boldsymbol{r}_0, \quad A\boldsymbol{r}_0, \quad A^2\boldsymbol{r}_0, \quad \cdots, \quad A^i\boldsymbol{r}_0 \quad (i = 1, 2, \cdots, m)$$

when the initial approximate solution vector of the j-th step is $\boldsymbol{u}_0^{(j)}$ and the initial residual vector of the j-th step is

$$\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)}$$

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0.$
for $j = 1, 2, \cdots$
  $\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)},$
  $\boldsymbol{v}_1 = \boldsymbol{r}^{(j)} / \|\boldsymbol{r}^{(j)}\|_2.$
  for $i = 1, 2, \cdots m$
    $\boldsymbol{w} = A\boldsymbol{v}_i.$
    for $k = 1, 2, \cdots i$
      $\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k,$
    $\boldsymbol{v}_{i+1} = \boldsymbol{w} / \|\boldsymbol{w}\|_2,$
    Determine the parameters $\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$
    so that $\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ takes the least value.
  $\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}.$

As shown in this algorithm, the GMRES method consists of an external iterative calculation (loop related to j) and an internal iterative calculation (loop related to i). Since the GMRES method is an iterative method that depends on the parameter m representing the upper bound on the number of inner iterations, it is referred to, strictly speaking, as the GMRES(m) method.
In this library, the number of iterations in the GMRES(m) algorithm is defined by
(number of internal iterations i) +(number of external iterations j)× m .

4.1.2.3 Preconditioned Iterative Method

Since error is introduced in actual iterative method calculations, convergence characteristics become worse. In particular, if the difference mesh is fine in the difference scheme for partial differential equations, if the coefficient (dispersion or thermal conductivity) values severely fluctuate spatially or if the difference mesh widths or the coefficients described above are anisotropic, then the eigenvalues of matrix A are dispersed, and the iterative method has difficulty in converging under these circumstances. Conversely, convergence will be fast if the eigenvalues are clustered together.
To overcome the difficulties described above, preconditioned iterative methods were developed. A preconditioned iterative method is a technique for improving convergence by preconditioning the original matrix to convert it to a well-conditioned matrix before performing the iterative method calculations. This is represented as follows.
Assume that the matrix $M = M_1M_2$, which approximates $A = (a_{ij})$ in some way, is given.

$$A \sim M = M_1M_2$$

At this time, if the following equation, which is equivalent to equation (1), is created:

$$M_1^{-1}AM_2^{-1}\boldsymbol{u}' = \boldsymbol{b}' \qquad (2)$$
$$(\quad \boldsymbol{u}' = M_2\boldsymbol{u} \quad)$$
$$(\quad \boldsymbol{b}' = M_1^{-1}\boldsymbol{b} \quad)$$

the coefficient matrix of this equation is easier to solve since it is closer to the unit matrix than the original matrix was. Equation (2) is called the preconditioning equation, and matrix M is called the preconditioning matrix. The preconditioned iterative method for equation (1) is defined/derived as the (basic) iterative method for equation (1). The preconditioned iterative method must execute a matrix inversion on M_1 and M_2 or M (left operation on a column vector according to the inverse matrix). This is what is called the preconditioning operation (or simply preconditioning). Usually, a matrix for which the preconditioning operation can easily be performed is used as preconditioning matrix $M = M_1M_2$. The more closely the preconditioning matrix M approximates the original

matrix A, the more the convergence of the preconditioned iterative method will improve. However, there generally is a tendency for the amount of time required for a single preconditioning operation to increase in proportion to the quality of convergence of the preconditioning matrix.

See 4.1.2.4 for the detailed explanation of the preconditioning algorithms that ASL supports.

The algorithms for the preconditioned CG method (PCG method), preconditioned CGS method (PCGS method), preconditioned BiCGSTAB (PBiCGSTAB method), and preconditioned GMRES(m) method (PGMRES(m) method) are shown below.

(1) **the PCG method**

$\boldsymbol{u} = \boldsymbol{u}_1$ ($\boldsymbol{u}_1$ is the initial solution),

$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1$,

$\boldsymbol{p}_1 = M^{-1}\boldsymbol{r}_1$.

for $i = 1, 2, \cdots, n$

$$a_i = \frac{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)},$$

$$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i,$$

$$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_iA\boldsymbol{p}_i,$$

$$b_i = \frac{(\boldsymbol{r}_{i+1}, M^{-1}\boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)},$$

$$\boldsymbol{p}_i = M^{-1}\boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i.$$

(2) **the PCGS method** (Bibliography(1))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0$,

$\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$,

$\boldsymbol{e}_0 = \boldsymbol{r}_0$.

for $k = 0, 1, 2, \cdots$

$$\alpha_{k+1} = \frac{(\boldsymbol{r}_0, \boldsymbol{r}_k)}{(\boldsymbol{r}_0, A\boldsymbol{p}_k)},$$

$$\boldsymbol{h}_{k+1} = \boldsymbol{e}_k - \alpha_{k+1}A\boldsymbol{p}_k,$$

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha_{k+1}M^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1}),$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_{k+1}AM^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1}),$$

$$\beta_{k+1} = \frac{(\boldsymbol{r}_0, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_0, \boldsymbol{r}_k)},$$

$$\boldsymbol{e}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1},$$

$$\boldsymbol{p}_{k+1} = M^{-1}(\boldsymbol{e}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1}) + \beta_{k+1}^2\boldsymbol{p}_k.$$

(3) **the PBiCGSTAB method** (Bibliography(2))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0,$

for $k = 0, 1, 2, \cdots$

- $\rho_{k+1} = (\boldsymbol{r}_0, \boldsymbol{r}_k),$
- if $k = 0$ then
  - $\boldsymbol{p}_1 = \boldsymbol{r}_0.$
- else
  - $\beta = (\frac{\rho_{k+1}}{\rho_k})(\frac{\alpha}{\omega_k}),$
  - $\boldsymbol{p}_{k+1} = \boldsymbol{r}_k + \beta(\boldsymbol{p}_k - \omega_k \boldsymbol{v}_k).$
- end if
- $\boldsymbol{y} = M^{-1}\boldsymbol{p}_{k+1},$
- $\boldsymbol{v}_{k+1} = A\boldsymbol{y},$
- $\alpha = \frac{\rho_{k+1}}{(\boldsymbol{r}_0, \boldsymbol{v}_{k+1})},$
- $\boldsymbol{s} = \boldsymbol{r}_k - \alpha \boldsymbol{v}_{k+1},$
- $\boldsymbol{z} = M^{-1}\boldsymbol{s},$
- $\boldsymbol{t} = A\boldsymbol{z},$
- $\omega_{k+1} = \frac{(\boldsymbol{t}, \boldsymbol{s})}{(\boldsymbol{t}, \boldsymbol{t})},$
- $\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha \boldsymbol{y} + \omega_{k+1}\boldsymbol{z},$
- $\boldsymbol{r}_{k+1} = \boldsymbol{s} - \omega_{k+1}\boldsymbol{t}.$

(4) **the GMRES(m) method** (Bibliography(3))

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0.$

for $j = 1, 2, \cdots$

- $\boldsymbol{r}^{(j)} = M^{-1}(\boldsymbol{b} - A\boldsymbol{u}_0^{(j)}),$
- $\boldsymbol{v}_1 = \frac{\boldsymbol{r}^{(j)}}{\|\boldsymbol{r}^{(j)}\|_2}.$
- for $i = 1, 2, \cdots m$
  - $\boldsymbol{w} = M^{-1}A\boldsymbol{v}_i.$
  - for $k = 1, 2, \cdots i$
    - $\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k.$
  - $\boldsymbol{v}_{i+1} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2},$
  - Determine the parameters $\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$ so that $\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ takes the least value.
- $\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}$

Note: the number of iterations in the GMRES(m) algorithm is defined by

(number of internal iterations i) +(number of external iterations j)× m .

4.1.2.4 Preconditioning Methods

In the following are described the details of those preconditioning algorithms that are implemented in functions in this chapter, synopses of which are explained 4.1.2.3.

(1) **Scaling preconditioning**

Scaling preconditioning is a technique that uses the following matrix consisting of the diagonal components of matrix A as the preconditioning matrix M.

$$D = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn}),$$

In particular, the scaling preconditioned CG method is known as the SCG method. On an Vector Engine, the scaling method often is faster than other preconditioning techniques.

4.1.2.5 Advanced Techniques for Improving Performance

(1) **Iterative Improvement Method**

An iterative improvement method is a technique for constructing a higher precision solution based on an approximate solution obtained by some solution method. If a single-precision function was used, then a solution on the order of a double-precision solution will be obtained, and if a double-precision function was used, then a solution on the order of a quadruple-precision solution will be obtained. (However, the calculation time will be approximately two to four times longer.) Another benefit of an iterative improvement method is that the error of the obtained solution can be estimated. If an iterative improvement method is not used, then the error is difficult to estimate even if the residual is known. A problem with the PCG method is that the error cannot be improved beyond a certain degree no matter how close the residual gets to zero. Therefore, if the precision of the solution is particularly important or if a single-precision function has been used, then an iterative improvement of the solution should be performed.

The iterative improvement method can be intuitively derived from the following kind of transformation, where $\boldsymbol{u}$ is the true solution of the equations $A\boldsymbol{u} = \boldsymbol{b}$ and $\boldsymbol{u}'$ is the approximation solution:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{u}' + \boldsymbol{u} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}\boldsymbol{b} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}(\boldsymbol{b} - A\boldsymbol{u}') \end{aligned}$$

From this transformation the following algorithm is obtained:

(a) Use a function from this chapter to solve the given equations $A\boldsymbol{u} = \boldsymbol{b}$ and let $\boldsymbol{u}'$ be the solution that was obtained.

(b) Obtain the residual of the solution with high precision. That is, if a single-precision function was used, calculate $\boldsymbol{r}' = \boldsymbol{b} - A\boldsymbol{u}'$ with double precision, and if a double-precision function was used, calculate $\boldsymbol{r}'$ with quadruple precision.

(c) Return $\boldsymbol{r}'$ to the original precision and assign it to $\boldsymbol{r}$. That is, for single-precision function, assign $\boldsymbol{r}'$ to a single-precision $\boldsymbol{r}$, and for a double-precision function, assign $\boldsymbol{r}'$ to a double-precision $\boldsymbol{r}$.

(d) Use a function from this chapter to solve the equations $A\boldsymbol{v} = \boldsymbol{r}$ and let $\boldsymbol{v}'$ be the solution that was obtained.

(e) Calculate $\boldsymbol{u}' + \boldsymbol{v}'$ and let it be called $\boldsymbol{u}'$.

(f) $\boldsymbol{v}'$ is the estimated error. If $\boldsymbol{v}'$ is sufficiently small, then stop. If not, then return to (b).

The most important part of this algorithm is step (b). The error is improved by calculating the residual with high precision. The iterative improvement method generally can obtain a sufficient degree of precision in several iterations.

(2) **Node numbering in the finite element method**

Although it it known that odd-even ordering or multicolor ordering, for example, should be performed to obtain long vectors, this kind of tricky numbering may significantly degrade convergence in a preconditioned

iterative method. Therefore, care is required(Bibliography(4)). The most stable numbering is numbering that reduces the matrix band width. To automatically generate this kind of numbering, you should use a technique such as the Cuthill-McKee method(Bibliography(5)).

4.1.3 Reference Bibliography

(1) Sonneveld, P. , "CGS, a Fast Lanczos–type Solver for Nonsymmetric Linear Systems", Delft University of Technology, Report No. 84–16, Delft The Netherland (1984).

(2) Van Der Vorst, H, A. , "Bi–CGSTAB:A more smoothly converging variant of CGS for the solution of nonsymmetric linear systems", SIAM J. Sci. Stat. Comput. 13, pp631–644 (1992).

(3) Y. Saad and M. H. SCHULTZ, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM J. Sci. Stat. Comput. , vol. 7, pp856-869 (1986).

(4) Duff, I. S. and Meurant, G. A. , "The Effect of Ordering on Preconditioned Conjugate Gradients", CERFACS, TR88/2, Toulouse, France (1988).

(5) Cuthill, E. and McKee, J. , "Reducing the Bandwidth of Sparse Symmetric Matrices", Proc. of the 24th National Conference of the Association of Computing Machinery, Prandon Press, New Jersey, pp. 157–172 (1969).

4.2 SPARSE MATRIX—NONSTATIONARY ITERATIVE METHODS (BASIC ITERATION METHOD FUNCTIONS)

4.2.1 ASL_qxe010, ASL_pxe010 Positive Definite Symmetric Sparse Matrix (ELLPACK Format) (CG method)

(1) **Function**

ASL_qxe010 or ASL_pxe010 uses the scaling preconditioned CG method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having a sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_qxe010 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

Single precision:

ierr = ASL_pxe010 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna×m	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	lna	I	1	Input	Adjustable dimension of array a and ja
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	m denotes the column number of both array a and ja (See Note (d))
5	ja	I*	lna×m	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	b	{D* / R*}	n	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	u	{D* / R*}	n	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	itrmax	I	1	Input	Maximum number of iterations (Default value: n)
9	itr	I*	1	Output	Actual number of iterations
10	epsmax	{D / R}	1	Input	Truncation residual norm Default value : { 10^{-12} (doubleprecision) / 10^{-6} (singleprecision) }
11	eps	{D* / R*}	1	Output	Final residual norm
12	isw	I	1	Input	Processing switch (Default value: 0) (See Note (e)) isw= 0 : Check whether the values ja are all distinct. isw= 1 : Does not check whether the values ja are all distinct.
13	nt	I	1	Input	Number of tasks to be generated (Default value: 1)
14	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c)
- $\mathrm{ja[i-1] = i, ja[(i-1) + lna \times (j-1)] \neq i \quad (i = 1, \cdots, n; j = 2, \cdots, j_i)}$,
  $\mathrm{1 \leq ja[(i-1) + lna \times (j-1)] \leq n \quad (j = 2, \cdots, j_i)}$,
  where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.
- $\mathrm{ja[(i-1) + lna \times j_i] = 0}$, if $j_i < \mathrm{m}$, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{ja[(i-1) + lna \times (j-1)] \quad (j = 1, \cdots, j_i)}$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{a[i-1] \neq 0.0 \quad (i = 1, \cdots, n)}$

(f) $\mathrm{itrmax} \geq 1$

(g) epsmax > underflow decision value

(h) $\mathrm{isw} \in \{0, 1\}$

(i) $\mathrm{nt} \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with itrmax = n.
1200	Restriction (g) was not satisfied.	Processing continues with epsmax $= \left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	Processing continues with isw=0, on the assumption that ja satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with nt = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{u[i-1] \leftarrow 0.0 \quad (i = 1, \cdots, n)}$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	isw was equal to 0.	Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.

ierr value	Meaning	Processing
4000	a has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ is less than the underflow decision value.	
4320	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ is greater than the overflow decision value.	
4410	$\lvert(\boldsymbol{p}_i, A\boldsymbol{p}_i)\rvert$ is less than the underflow decision value.	
4420	$\lvert(\boldsymbol{p}_i, A\boldsymbol{p}_i)\rvert$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

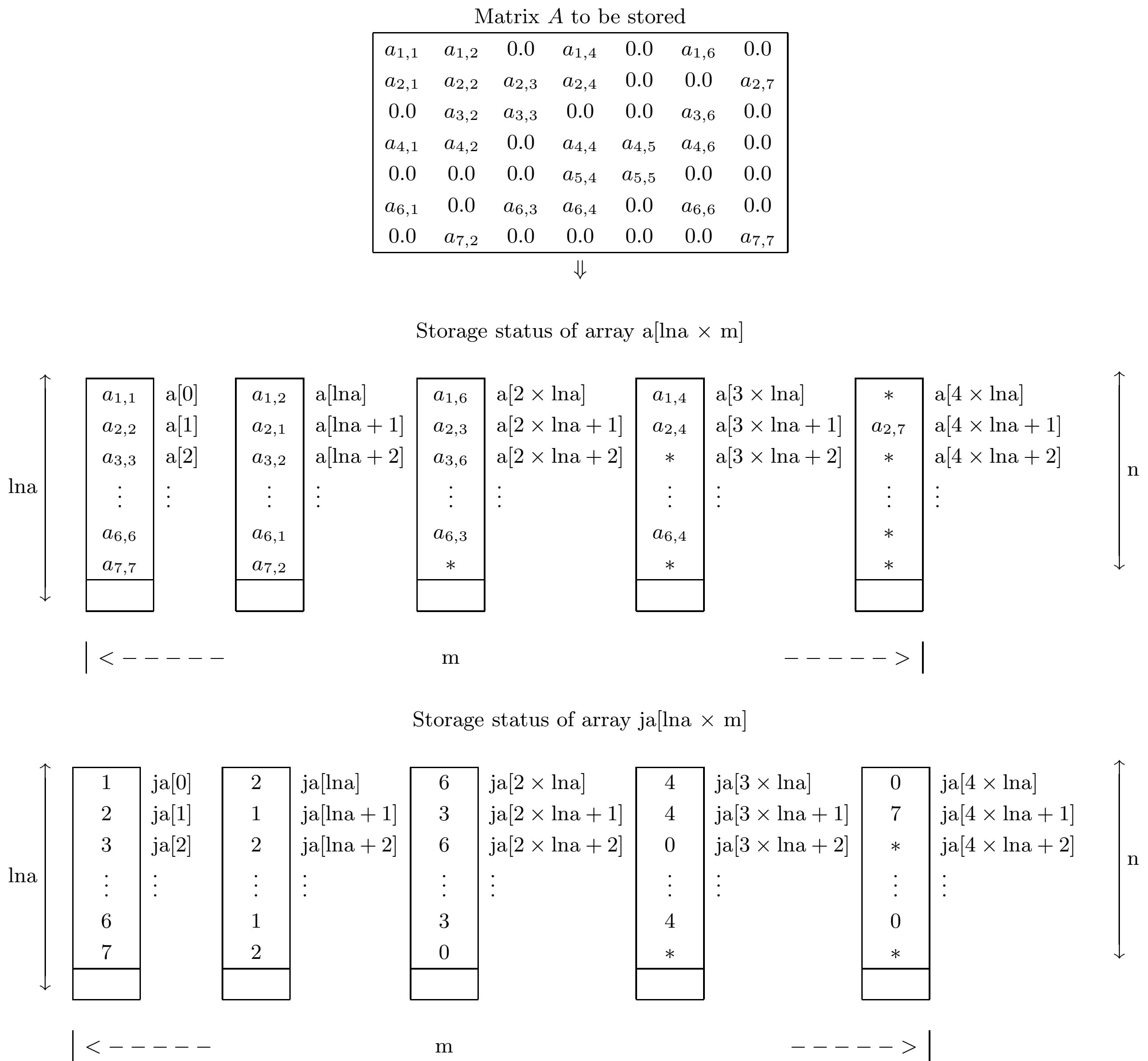
(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL C interface.

Table 4−2 Constants used in ASL C interface

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80 \times 10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40 \times 10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23 \times 10^{-308})$	$2^{-126} \simeq (1.17 \times 10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23 \times 10^{-308})$	$-2^{-126} \simeq (-1.17 \times 10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80 \times 10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40 \times 10^{38})$
Overflow decision value	Maximum value $\times 10^{-3}$	
Underflow decision value	Positive minimum value $\times 10^{3}$	

(b) The storage method used for the arrays a and ja is as follows.

Figure 4−1 Storage format for Input Data



Remarks

a. n is order of Matrix A.

b. lna $\geq$ n must hold.

c. m is the column number of Array a, which contains the nonzero elements of Matrix A.

d. Array a should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though m-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array ja should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array a.
For those rows in which $m - 1$ becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in ja in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array a and array ja are partially changed in order to optimize performance.

(d) m can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting m as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying isw = 0 (Performance will degrade remarkably when isw = 1 is specified). And so, isw = 1 should be specified when it is assured that ja satisfies Restriction (d). When isw = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if isw = 0 was specified but ja doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this function. If the work area could not be successfully allocated, processing will be aborted and ierr returns the value 5000. In this case, the problem cannot be solved by using this function unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & -3 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & -4 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & -5 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & -6 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) Input data

Input Arrays a, ja and b,

lna = 11, n = 10, m = 11, itrmax = 100, epsmax = 10^{-12}, isw = 0, nt = 2

(c) Main program

```
/*      C interface example for ASL_qxe010 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;
```

```
printf( "    *** ASL_qxe010 ***\n" );

lna = 11;
n   = 10;
m   =  5;

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
         a[i+lna*j] = 0.0;
        ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;
a[  lna*2] = -1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;
a[1+lna*3] = -2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;
a[2+lna*3] = -3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;
a[3+lna*4] = -4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;
a[4+lna*4] = -5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;
a[5+lna*4] = -6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
```

```
ja[  lna  ] =  2;
ja[  lna*2] =  4;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;
ja[1+lna*3] =  5;

ja[2      ] =  3;
ja[2+lna  ] =  2;
ja[2+lna*2] =  4;
ja[2+lna*3] =  6;

ja[3      ] =  4;
ja[3+lna  ] =  1;
ja[3+lna*2] =  3;
ja[3+lna*3] =  5;
ja[3+lna*4] =  7;

ja[4      ] =  5;
ja[4+lna  ] =  2;
ja[4+lna*2] =  4;
ja[4+lna*3] =  6;
ja[4+lna*4] =  8;

ja[5      ] =  6;
ja[5+lna  ] =  3;
ja[5+lna*2] =  5;
ja[5+lna*3] =  7;
ja[5+lna*4] =  9;

ja[6      ] =  7;
ja[6+lna  ] =  4;
ja[6+lna*2] =  6;
ja[6+lna*3] =  8;
ja[6+lna*4] = 10;

ja[7      ] =  8;
ja[7+lna  ] =  5;
ja[7+lna*2] =  7;
ja[7+lna*3] =  9;

ja[8      ] =  9;
ja[8+lna  ] =  6;
ja[8+lna*2] =  8;
ja[8+lna*3] = 10;

ja[9      ] = 10;
ja[9+lna  ] =  7;
ja[9+lna*2] =  9;

for( i=0 ; i < n ; i++ )
{
    b[i] = 1.0;
}

for( i=0 ; i < n ; i++ )
{
    u[i] = 0.0;
}

itrmax = 100;
epsmax = 1.0e-12;
isw    = 1;
nt     = 2;

printf( "\n   ***   Input   ***\n\n" );
printf( "\titrmax =  %6d\n", itrmax );
printf( "\tepsmax =%8.3g\n", epsmax );
printf( "\tnt     =  %6d\n", nt );


    ierr = ASL_qxe010
(a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


printf( "\n   ***   Output   ***\n\n" );
printf( "\tierr   =  %6d\n", ierr );
printf( "\titr    =  %6d\n", itr );
printf( "\teps    =%8.3g\n\n", eps );

for( i=0 ; i < n ; i++ )
{
    printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
}

free( a );
```

```
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) Output results

```
     *** ASL_qxe010 ***

    ***   Input   ***

     itrmax =     100
     epsmax =   1e-12
     nt     =       2

    ***   Output   ***

     ierr   =       0
     itr    =      10
     eps    =3.46e-16

     u[ 0 ] =   0.335
     u[ 1 ] =    0.22
     u[ 2 ] = -0.0808
     u[ 3 ] =   0.224
     u[ 4 ] =   0.136
     u[ 5 ] =  -0.151
     u[ 6 ] =   0.247
     u[ 7 ] =  0.0408
     u[ 8 ] = -0.0926
     u[ 9 ] =    0.17
```

4.2.2 ASL_qxe020, ASL_pxe020 Asymmetric Sparse Matrix (ELLPACK Format) (CGS method)

(1) **Function**

ASL_qxe020 or ASL_pxe020 uses the scaling preconditioned CGS method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_qxe020 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

Single precision:

ierr = ASL_pxe020 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×m	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	lna	I	1	Input	Adjustable dimension of array a and ja
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	m denotes the column number of both array a and ja (See Note (d))
5	ja	I*	lna×m	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	b	{D*, R*}	n	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	u	{D*, R*}	n	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	itrmax	I	1	Input	Maximum number of iterations (Default value: n)
9	itr	I*	1	Output	Actual number of iterations
10	epsmax	{D, R}	1	Input	Truncation residual norm Default value: {10^{-12} (doubleprecision), 10^{-6} (singleprecision)}
11	eps	{D*, R*}	1	Output	Final residual norm
12	isw	I	1	Input	Processing switch (Default value: 0) (See Note (e)) isw= 0 : Check whether the values ja are all distinct. isw= 1 : Does not check whether the values ja are all distinct.
13	nt	I	1	Input	Number of tasks to be generated (Default value: 1)
14	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \le \mathrm{n} \le \mathrm{lna}$

(b) $1 \le \mathrm{m} \le \mathrm{n}$

(c)
- $\mathrm{ja[i-1] = i, ja[(i-1) + lna \times (j-1)] \neq i \quad (i = 1, \cdots, n; j = 2, \cdots, j_i)}$,
  $\mathrm{1 \le ja[(i-1) + lna \times (j-1)] \le n \quad (j = 2, \cdots, j_i)}$,
  where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

- $\mathrm{ja[(i-1) + lna \times j_i] = 0}$, if $j_i < \mathrm{m}$, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{ja[(i-1) + lna \times (j-1)] \quad (j = 1, \cdots, j_i)}$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{a[i-1] \neq 0.0 \quad (i = 1, \cdots, n)}$

(f) $\mathrm{itrmax} \ge 1$

(g) $\mathrm{epsmax} > underflowdecisionvalue$

(h) $\mathrm{isw} \in \{0, 1\}$

(i) $\mathrm{nt} \ge 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with itrmax = n.
1200	Restriction (g) was not satisfied.	Processing continues with epsmax = $\left\{ \begin{array}{ll} 10^{-12} & (\text{doubleprecision}) \\ 10^{-6} & (\text{singleprecision}) \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	isw has the input value 0. Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with nt = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{u[i-1] \leftarrow 0.0 \quad (i = 1, \cdots, n)}$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	isw was equal to 0.	Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	

ierr value	Meaning	Processing
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.
4000	a has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\lvert(\boldsymbol{r}_0, \boldsymbol{r}_i)\rvert$ is less than the underflow decision value.	
4320	$\lvert(\boldsymbol{r}_0, \boldsymbol{r}_i)\rvert$ is greater than the overflow decision value.	
4410	$\lvert(\boldsymbol{r}_0, A\boldsymbol{p}_i)\rvert$ is less than the underflow decision value.	
4420	$\lvert(\boldsymbol{r}_0, A\boldsymbol{p}_i)\rvert$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

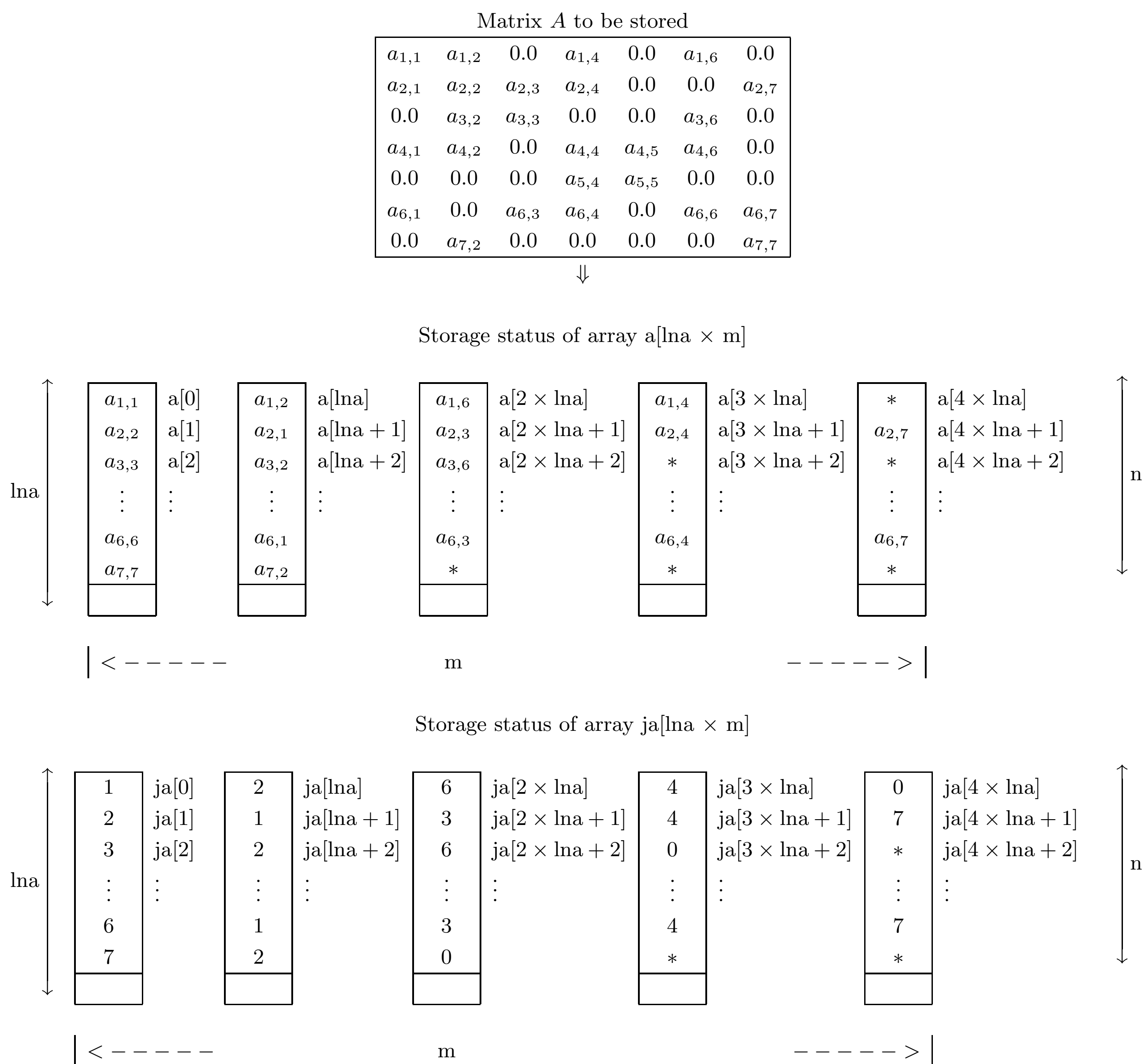
(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL C interface.

Table 4−3 Constants used in ASL C interface

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80 \times 10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40 \times 10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23 \times 10^{-308})$	$2^{-126} \simeq (1.17 \times 10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23 \times 10^{-308})$	$-2^{-126} \simeq (-1.17 \times 10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80 \times 10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40 \times 10^{38})$
Overflow decision value	Maximum value $\times 10^{-3}$	
Underflow decision value	Positive minimum value $\times 10^{3}$	

(b) The storage method used for the arrays a and ja is as follows.

Figure 4−2 Storage format for Input Data



Remarks

a. n is order of Matrix A.

b. lna $\geq$ n must hold.

c. m is the column number of Array a, which contains the nonzero elements of Matrix A.

d. Array a should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though m-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '∗'.

e. Array ja should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array a.
For those rows in which m − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in ja in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '∗'.

(c) The input values of array a and array ja are partially changed in order to optimize performance.

(d) m can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting m as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying isw = 0 (Performance will degrade remarkably when isw = 1 is specified). And so, isw = 1 should be specified when it is assured that ja satisfies Restriction (d). When isw = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if isw = 0 was specified but ja doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this function. If the work area could not be successfully allocated, processing will be aborted and ierr returns the value 5000. In this case, the problem cannot be solved by using this function unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem
Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) Input data
Input Arrays a, ja and b,
$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) Main program

```
/*      C interface example for ASL_qxe020 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;
```

```
printf( "    *** ASL_qxe020 ***\n" );

lna = 11;
n   = 10;
m   =  5;

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;
```

```
    ja[2      ] =  3;
    ja[2+lna  ] =  2;
    ja[2+lna*2] =  4;

    ja[3      ] =  4;
    ja[3+lna  ] =  1;
    ja[3+lna*2] =  3;
    ja[3+lna*3] =  5;

    ja[4      ] =  5;
    ja[4+lna  ] =  2;
    ja[4+lna*2] =  4;
    ja[4+lna*3] =  6;

    ja[5      ] =  6;
    ja[5+lna  ] =  3;
    ja[5+lna*2] =  5;
    ja[5+lna*3] =  7;

    ja[6      ] =  7;
    ja[6+lna  ] =  4;
    ja[6+lna*2] =  6;
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe020
    (a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) Output results

```
 *** ASL_qxe020 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =6.19e-16

 u[ 0 ] =   0.296
 u[ 1 ] =   0.111
 u[ 2 ] =  0.0732
 u[ 3 ] =  0.0882
 u[ 4 ] =  0.0707
 u[ 5 ] =  0.0185
 u[ 6 ] =   0.104
 u[ 7 ] =  0.0335
 u[ 8 ] = 0.00671
 u[ 9 ] =  0.0795
```

4.2.3 ASL_qxe030, ASL_pxe030 Asymmetric Sparse Matrix (ELLPACK Format) (BiCGSTAB method)

(1) **Function**

ASL_qxe030 or ASL_pxe030 uses the scaling preconditioned BiCGSTAB method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_qxe030 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

Single precision:

ierr = ASL_pxe030 (a,lna,n,m,ja,b,u,itrmax, &itr,epsmax, &eps,isw,nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	$\begin{Bmatrix} D* \\ R* \end{Bmatrix}$	lna×m	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	lna	I	1	Input	Adjustable dimension of array a and ja
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	m denotes the column number of both array a and ja (See Note (d))
5	ja	I*	lna×m	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	b	$\begin{Bmatrix} D* \\ R* \end{Bmatrix}$	n	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	u	$\begin{Bmatrix} D* \\ R* \end{Bmatrix}$	n	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	itrmax	I	1	Input	Maximum number of iterations (Default value: n)
9	itr	I*	1	Output	Actual number of iterations
10	epsmax	$\begin{Bmatrix} D \\ R \end{Bmatrix}$	1	Input	Truncation residual norm Default value: $\begin{Bmatrix} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{Bmatrix}$
11	eps	$\begin{Bmatrix} D* \\ R* \end{Bmatrix}$	1	Output	Final residual norm
12	isw	I	1	Input	Processing switch(Default value: 0) (See Note (e)) isw= 0 : Check whether the values ja are all distinct. isw= 1 : Does not check whether the values ja are all distinct.
13	nt	I	1	Input	Number of tasks to be generated (Default value: 1)
14	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c) • $\mathrm{ja[i-1] = i, ja[(i-1) + lna \times (j-1)] \neq i \quad (i = 1, \cdots, n; j = 2, \cdots, j_i)}$,
$\mathrm{1 \leq ja[(i-1) + lna \times (j-1)] \leq n \quad (j = 2, \cdots, j_i)}$,
where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

• $\mathrm{ja[(i-1) + lna \times j_i] = 0}$, if $j_i < \mathrm{m}$, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{ja[(i-1) + lna \times (j-1)] \quad (j = 1, \cdots, j_i)}$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{a[i-1] \neq 0.0 \quad (i = 1, \cdots, n)}$

(f) $\mathrm{itrmax} \geq 1$

(g) $\mathrm{epsmax} > underflowdecisionvalue$

(h) $\mathrm{isw} \in \{0, 1\}$

(i) $\mathrm{nt} \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with itrmax = n.
1200	Restriction (g) was not satisfied.	Processing continues with epsmax = $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (h) was not satisfied.	isw has the input value 0. Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
1400	Restriction (i) was not satisfied.	Processing continues with nt = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	$\mathrm{u[i-1] \leftarrow 0.0 \quad (i = 1, \cdots, n)}$ is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	isw was equal to 0.	Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.

ierr value	Meaning	Processing
4000	a has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The relative norm of the residual $\boldsymbol{r}$ is greater than the overflow decision value.	
4310	$\lvert\boldsymbol{\rho}_i\rvert$ is less than the underflow decision value.	
4320	$\lvert\boldsymbol{\rho}_i\rvert$ is greater than the overflow decision value.	
4410	$\lvert(\boldsymbol{r}_0, \boldsymbol{v}_i)\rvert$ is less than the underflow decision value.	
4420	$\lvert(\boldsymbol{r}_0, \boldsymbol{v}_i)\rvert$ is greater than the overflow decision value.	
4510	$\lvert(\boldsymbol{t}, \boldsymbol{t})\rvert$ is less than the underflow decision value.	
4520	$\lvert(\boldsymbol{t}, \boldsymbol{t})\rvert$ is greater than the overflow decision value.	
4610	$\lvert\boldsymbol{\omega}_i\rvert$ is less than the underflow decision value.	
4620	$\lvert\boldsymbol{\omega}_i\rvert$ is greater than the overflow decision value.	
4700	$\lvert\boldsymbol{\beta}\rvert$ is greater than the overflow decision value.	
4800	$\lvert\boldsymbol{\alpha}\rvert$ is greater than the overflow decision value.	
4900	$\lvert(\boldsymbol{t}, \boldsymbol{s})\rvert$ is greater than the overflow decision value.	
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

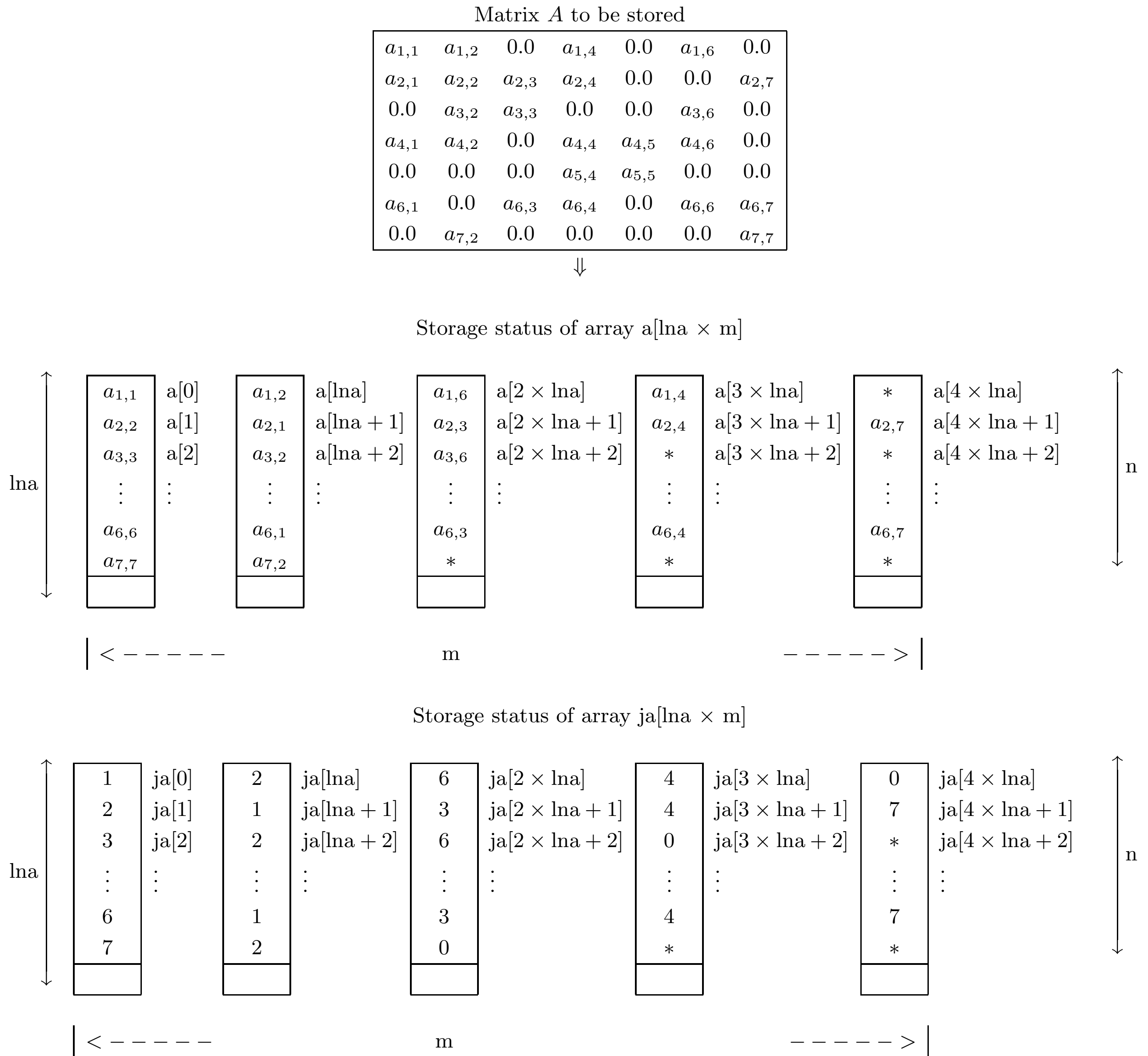
(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL C interface.

Table 4−4 Constants used in ASL C interface

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times 10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times 10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times 10^{-308})$	$2^{-126} \simeq (1.17\times 10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times 10^{-308})$	$-2^{-126} \simeq (-1.17\times 10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times 10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times 10^{38})$
Overflow decision value	Maximum value $\times 10^{-3}$	
Underflow decision value	Positive minimum value $\times 10^{3}$	

(b) The storage method used for the arrays a and ja is as follows.

Figure 4−3 Storage format for Input Data



Remarks

a. n is order of Matrix A.

b. lna $\geq$ n must hold.

c. m is the column number of Array a, which contains the nonzero elements of Matrix A.

d. Array a should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though m-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array ja should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array a.
For those rows in which m − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in ja in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array a and array ja are partially changed in order to optimize performance.

(d) m can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting m as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying isw = 0 (Performance will degrade remarkably when isw = 1 is specified). And so, isw = 1 should be specified when it is assured that ja satisfies Restriction (d). When isw = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if isw = 0 was specified but ja doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this function. If the work area could not be successfully allocated, processing will be aborted and ierr returns the value 5000. In this case, the problem cannot be solved by using this function unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) Input data

Input Arrays a, ja and b,

$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) Main program

```
/*      C interface example for ASL_qxe030 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;
```

```
printf( "    *** ASL_qxe030 ***\n" );

lna = 11;
n   = 10;
m   =  5;

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;
```

```
    ja[2      ] =  3;
    ja[2+lna  ] =  2;
    ja[2+lna*2] =  4;

    ja[3      ] =  4;
    ja[3+lna  ] =  1;
    ja[3+lna*2] =  3;
    ja[3+lna*3] =  5;

    ja[4      ] =  5;
    ja[4+lna  ] =  2;
    ja[4+lna*2] =  4;
    ja[4+lna*3] =  6;

    ja[5      ] =  6;
    ja[5+lna  ] =  3;
    ja[5+lna*2] =  5;
    ja[5+lna*3] =  7;

    ja[6      ] =  7;
    ja[6+lna  ] =  4;
    ja[6+lna*2] =  6;
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe030
    (a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) Output results

```
 *** ASL_qxe030 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =3.88e-16

 u[ 0 ] =   0.296
 u[ 1 ] =   0.111
 u[ 2 ] =  0.0732
 u[ 3 ] =  0.0882
 u[ 4 ] =  0.0707
 u[ 5 ] =  0.0185
 u[ 6 ] =   0.104
 u[ 7 ] =  0.0335
 u[ 8 ] = 0.00671
 u[ 9 ] =  0.0795
```

4.2.4 ASL_qxe040, ASL_pxe040 Asymmetric Sparse Matrix (ELLPACK Format) (GMRES(m) method)

(1) **Function**

ASL_qxe040 or ASL_pxe040 uses the scaling preconditioned GMRES(m) method to solve the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$ having an asymmetric sparse matrix A as the coefficient matrix.

(2) **Usage**

Double precision:

ierr = ASL_qxe040 (a,lna,n,m,ja,b,u,itrmax, &itr,mgmrs,epsmax, &eps,isw,nt);

Single precision:

ierr = ASL_pxe040 (a,lna,n,m,ja,b,u,itrmax, &itr,mgmrs,epsmax, &eps,isw,nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×m	Input	Array of nonzero element values of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
2	lna	I	1	Input	Adjustable dimension of array a and ja
3	n	I	1	Input	Order of matrix A
4	m	I	1	Input	m denotes the column number of both array a and ja (See Note (d))
5	ja	I*	lna×m	Input	Array storing the nonzero structure data of the coefficient matrix (For the storage format(See Note (b)))
				Output	Values updated for optimization (See Note (c))
6	b	{D*, R*}	n	Input	Right-hand side vector $\boldsymbol{b}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
7	u	{D*, R*}	n	Output	Solution vector $\boldsymbol{u}$ of the simultaneous linear equations $A\boldsymbol{u} = \boldsymbol{b}$
8	itrmax	I	1	Input	Maximum number of iterations (Default value: n)
9	itr	I*	1	Output	Actual number of iterations
10	mgmrs	I	1	Input	GMRES(m) parameter m (Default value: 10)
11	epsmax	{D, R}	1	Input	Truncation residual norm Default value: 10^{-12} (doubleprecision), 10^{-6} (singleprecision)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
12	eps	$\begin{Bmatrix} \text{D*} \\ \text{R*} \end{Bmatrix}$	1	Output	Final residual norm
13	isw	I	1	Input	Processing switch(Default value: 0) (See Note (e)) isw= 0 : Check whether the values ja are all distinct. isw= 1 : Does not check whether the values ja are all distinct.
14	nt	I	1	Input	Number of tasks to be generated (Default value: 1)
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \le \mathrm{n} \le \mathrm{lna}$

(b) $1 \le \mathrm{m} \le \mathrm{n}$

(c) • $\mathrm{ja[i-1] = i, ja[(i-1) + lna \times (j-1)] \ne i \quad (i = 1, \cdots, n; j = 2, \cdots, j_i)}$,
$1 \le \mathrm{ja[(i-1) + lna \times (j-1)]} \le \mathrm{n} \quad (\mathrm{j} = 2, \cdots, \mathrm{j_i})$,
where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

• $\mathrm{ja[(i-1) + lna \times j_i] = 0}$, if $j_i < \mathrm{m}$, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(d) $\mathrm{ja[(i-1) + lna \times (j-1)]} \quad (\mathrm{j} = 1, \cdots, \mathrm{j_i})$ are all distinct for a fixed value i, where $j_i \quad (i = 1, \cdots, \mathrm{n})$ is the number of nonzero elements that are contained in the i-th column of Matrix A.

(e) $\mathrm{a[i-1] \ne 0.0 \quad (i = 1, \cdots, n)}$

(f) $\mathrm{itrmax} \ge 1$

(g) $\mathrm{mgmrs} \ge 1$

(h) $\mathrm{epsmax} > underflowdecisionvalue$

(i) $\mathrm{isw} \in \{0, 1\}$

(j) $\mathrm{nt} \ge 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	Restriction (f) was not satisfied.	Processing continues with itrmax = n.
1100	Restriction (g) was not satisfied.	Processing continues with mgmrs = 10.
1200	Restriction (h) was not satisfied.	Processing continues with epsmax = $\left\{ \begin{array}{ll} 10^{-12} & \text{(doubleprecision)} \\ 10^{-6} & \text{(singleprecision)} \end{array} \right\}$.
1300	Restriction (i) was not satisfied.	isw has the input value 0. Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
1400	Restriction (j) was not satisfied.	Processing continues with nt = 1.
2000	The absolute norm of the right-hand side vector $\boldsymbol{b}$ is less than the underflow decision value.	u[i − 1] ← 0.0 (i = 1, · · · , n) is set for the solution.
2100	Restriction (e) was not satisfied.	Processing continues.
2200	isw was equal to 0.	Processing continues on the assumption that ja satisfies Restriction (d) (See Note (e)).
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	
3500	The norm of the Maximum number of iterations has been reached.	The result obtained at that time is returned.
4000	a has a diagonal term whose absolute value is smaller than the underflow decision value.	Processing is aborted.
4100	The norm of the right-hand side vector $\boldsymbol{b}$ is greater than the overflow decision value.	
4210	The norm of the initial residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ is greater than the overflow decision value.	
4220	The norm of the residual $\boldsymbol{r}$ of the final solution is greater than the overflow decision value.	
4310	$\|AM^{-1}v_i\|_2{}^2$ is less than the underflow decision value. In this case, it is very possible that AM^{-1} is degenerate.	The solution obtained by the (i − 1) iteration is returned, and processing is aborted.
4320	$\|AM^{-1}\boldsymbol{v}_i\|_2{}^2$ is greater than the overflow decision value.	Processing is aborted.

ierr value	Meaning	Processing
4410	$\|w\|_2{}^2$ after Gram-Schmidt orthogonalization became less than the underflow decision value, and processing could not be continued. This indicates that the iterative solution has already converged. It is very possible that the convergence decision conditions are too strict.	The solution obtained by that time is returned, and processing is aborted.
5000	Processing to reserve the work area required to execute the basic iterative algorithm failed (See Note (f)).	

(6) **Notes**

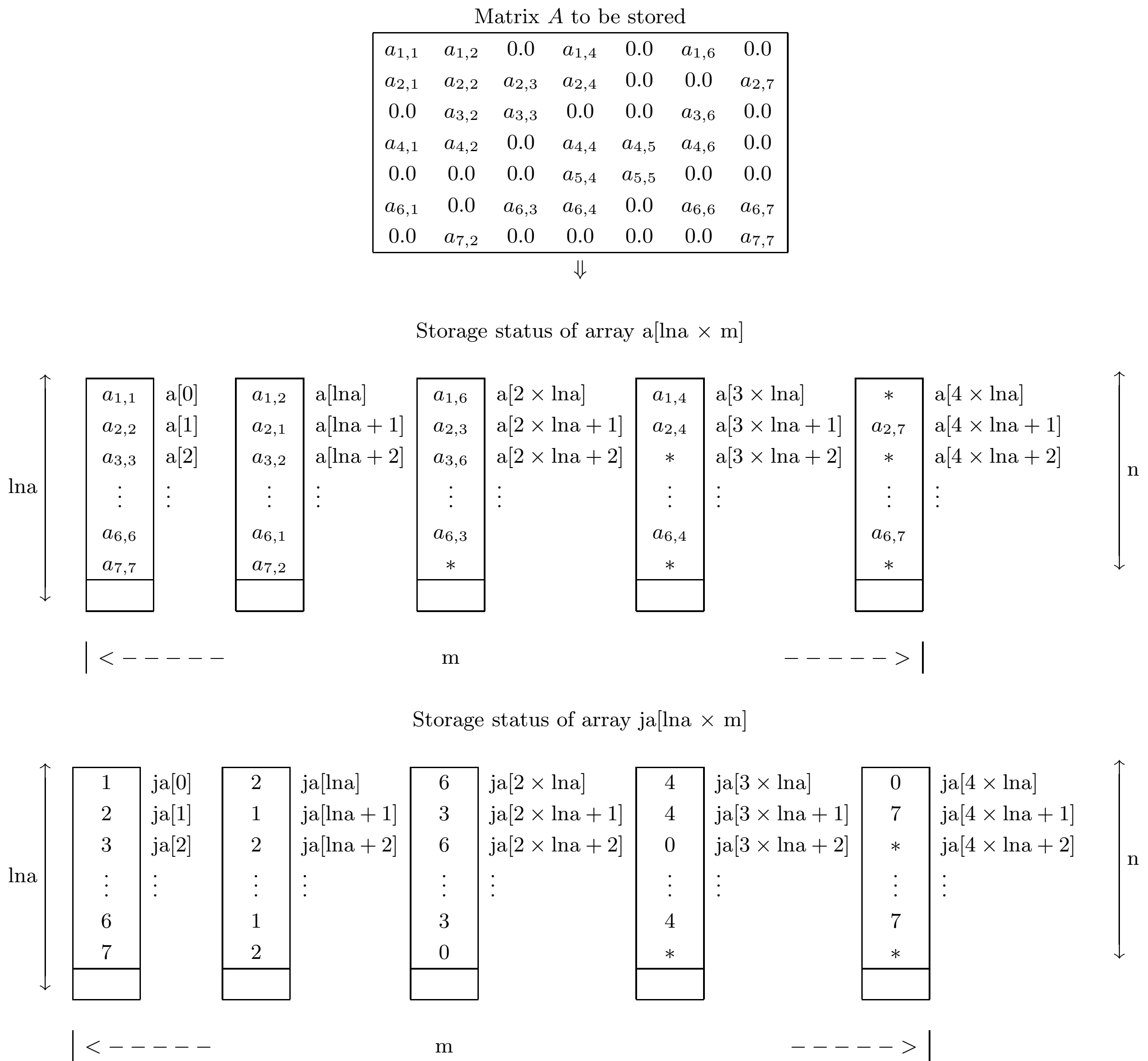
(a) The table below shows maximum and minimum values of floating point data and so on, that are defined within ASL C interface.

Table 4−5 Constants used in ASL C interface

	Double-precision	Single-precision
Maximum value	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
Positive minimum value	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
Negative maximum value	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
Minimum value	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
Overflow decision value	Maximum value $\times10^{-3}$	
Underflow decision value	Positive minimum value $\times10^{3}$	

(b) The storage method used for the arrays a and ja is as follows.

Figure 4−4 Storage format for Input Data



Remarks

a. n is order of Matrix A.

b. $\text{lna} \geq \text{n}$ must hold.

c. m is the column number of Array a, which contains the nonzero elements of Matrix A.

d. Array a should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though m-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array ja should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array a.
For those rows in which $\text{m} - 1$ becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region in ja in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

(c) The input values of array a and array ja are partially changed in order to optimize performance.

(d) m can take an arbitrary value as long as it is equal to or greater than the maximum of the numbers of nonzero elements in a row, but it is recommended from the viewpoint of calculation cost to make the padded area as little as possible by setting m as close as possible to the maximum of the numbers of nonzero elements in a row.

(e) A better performance will be achieved by specifying isw = 0 (Performance will degrade remarkably when isw = 1 is specified). And so, isw = 1 should be specified when it is assured that ja satisfies Restriction (d). When isw = 1 is specified, the user should be very careful in giving the coefficient matrix data as input because the check for Restriction (d) is omitted. The validity of the result will not be guaranteed if isw = 0 was specified but ja doesn't satisfy Restriction (d).

(f) The work area is automatically allocated within this function. If the work area could not be successfully allocated, processing will be aborted and ierr returns the value 5000. In this case, the problem cannot be solved by using this function unless the problem size is reduced or the machine environment is enhanced in memory size.

(7) **Example**

(a) Problem

Solve $A\boldsymbol{u} = \boldsymbol{b}$ for the following matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$:

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

(b) Input data

Input Arrays a, ja and b,

$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{mgmrs} = 5, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) Main program

```
/*      C interface example for ASL_qxe040 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    int mgmrs;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
```

```
int i,j;

printf( "    *** ASL_qxe040 ***\n" );

lna = 11;
n   = 10;
m   =  5;

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
         a[i+lna*j] = 0.0;
        ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
```

```
    ja[1+lna*2] =  3;

    ja[2      ] =  3;
    ja[2+lna  ] =  2;
    ja[2+lna*2] =  4;

    ja[3      ] =  4;
    ja[3+lna  ] =  1;
    ja[3+lna*2] =  3;
    ja[3+lna*3] =  5;

    ja[4      ] =  5;
    ja[4+lna  ] =  2;
    ja[4+lna*2] =  4;
    ja[4+lna*3] =  6;

    ja[5      ] =  6;
    ja[5+lna  ] =  3;
    ja[5+lna*2] =  5;
    ja[5+lna*3] =  7;

    ja[6      ] =  7;
    ja[6+lna  ] =  4;
    ja[6+lna*2] =  6;
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    mgmrs  = 10;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tmgmrs  =  %6d\n", mgmrs );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe040
    (a,lna,n,m,ja,b,u,itrmax,&itr,mgmrs,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) Output results

```
 *** ASL_qxe040 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 mgmrs  =      10
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =3.02e-16

 u[ 0 ] =   0.296
 u[ 1 ] =   0.111
 u[ 2 ] =  0.0732
 u[ 3 ] =  0.0882
 u[ 4 ] =  0.0707
 u[ 5 ] =  0.0185
 u[ 6 ] =   0.104
 u[ 7 ] =  0.0335
 u[ 8 ] = 0.00671
 u[ 9 ] =  0.0795
```

Chapter 5

EIGENVALUES AND EIGENVECTORS

5.1 INTRODUCTION

This chapter describes functions that obtain eigenvalues and eigenvectors of symmetric matrices.
Function described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

In the standard eigenvalue problem, obtain the value λ and corresponding vector $\boldsymbol{x}$ which satisfy the following equation for a given matrix A:

$A\boldsymbol{x} = \lambda\boldsymbol{x}$.

The value λ and the corresponding vector $\boldsymbol{x}$ are called an eigenvalue and the corresponding eigenvector, respectively.

In generalized eigenvalue problem, obtain the value λ and corresponding vector $\boldsymbol{x}$ which satisfy one of the following equations for given matrices A and B:

$A\boldsymbol{x} = \lambda B\boldsymbol{x}$

or

$AB\boldsymbol{x} = \lambda\boldsymbol{x}$ (both A and B are Hermitian, B is positive definite)

or

$BA\boldsymbol{x} = \lambda\boldsymbol{x}$ (both A and B are Hermitian, B is positive definite).

These λ and $\boldsymbol{x}$ are also called an eigenvalue and an eigenvector. If both A and B are Hermitian and B is positive definite, all the eigenvalues are real and the eigenvectors for different eigenvalues are orthogonal for each other.

The functions contained in this chapter provide functions corresponding to the following four categories.

All eigenvalues and all eigenvectors: Obtain all eigenvalues and the corresponding eigenvectors.

All eigenvalues: Obtain all eigenvalues only.

Eigenvalues and eigenvectors: Obtain a number of the largest or smallest eigenvalues and the corresponding eigenvectors.

Eigenvalues: Obtain a number of the largest or smallest eigenvalues.

5.1.1 Notes

(1) In general, functions corresponding to 'All eigenvalues and all eigenvectors' or 'Eigenvalues and eigenvectors' require more processing time and memory than functions corresponding to 'All eigenvalues' or 'Eigenvalues' respectively.

(2) In general, it is more efficient to use functions corresponding to 'Eigenvalues and eigenvectors' or 'Eigenvalues' if you want to obtain no more than 20% of the total number of eigenvalues. To obtain more than 20% of the total number of eigenvalues, functions corresponding to 'All eigenvalues and all eigenvectors' or 'All eigenvalues' require less processing time.

(3) In this library, the functions of the generalized eigenvalue require the condition that B is positive definite. In the following cases, however, the eigenvalues and the eigenvectors can be obtained even if B is not positive definite.

   (a) Matrix B is not positive definite but matrix A is positive definite

   $$Bv = \lambda^{-1} Av$$

   gives non-zero eigenvalues.

   (b) Both of A and B are not positive definite but $A + B$ is positive definite

   $$Av = \frac{\lambda}{1+\lambda}(A+B)v$$

   gives the eigenvalues which are not -1.

(4) If the input matrix is a symmetric matrix or Hermitian matrix, the use of the exclusive functions requires less processing time.

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

5.1.2 Algorithms Used

5.1.2.1 Transforming a real symmetric matrix to a real symmetric tridiagonal matrix

The Householder method is used to transform an $n \times n$ real symmetric matrix A to a real symmetric tridiagonal matrix T. That is, $A_1 = A$ is assumed, and for $k = 1, 2, \cdots, n-2$, a vector u_k is taken so that:

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

and all elements below the subdiagonal component in column k of:

$$A_{k+1} = P_k A_k P_k$$

become 0. A_{n-1} becomes the obtained real symmetric tridiagonal matrix. In addition, the transformation matrix P_k is an orthogonal symmetric matrix.

5.1.2.2 Transforming a Hermitian matrix to a real symmetric tridiagonal matrix

First, the Householder method is used to transform an $n \times n$ Hermitian matrix A to a Hermitian tridiagonal matrix S.

$$S = P_{n-2} \cdots P_2 P_1 A P_1 P_2 \cdots P_{n-2}$$

Then a regular complex diagonal matrix D is used (similarity transformation) to transform matrix S to a real symmetric tridiagonal matrix T.

$$T = D^* S D$$

5.1.2.3 The Householder transformation by block algorithm

As for the Householder transformation, the block algorithm is used. This method simplifies the update of a matrix by applying the lump sum of a rank-one matrix update that transforms the original real symmetric matrix into a real symmetric tridiagonal matrix. Let A_{k+1} be the symmetric matrix that is generated after similarity transformations are performed for k times on the original symmetric matrix A. Then it holds that:

$$A_{k+1} = P_k A_k P_k = A_k - \boldsymbol{u}_k \boldsymbol{v}_k^T - \boldsymbol{v}_k \boldsymbol{u}_k^T$$

where P_k is an orthogonal matrix. Also the following relationship holds:

$$A_1 = A$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$\boldsymbol{y}_k = A_k \boldsymbol{u}_k$$

$$\boldsymbol{v}_k = \frac{(\boldsymbol{y}_k - \frac{(\boldsymbol{u}_k^T \boldsymbol{y}_k)\boldsymbol{u}_k}{2H_k})}{H_k}$$

The Householder transformation updates the symmetric matrix twice for one similarity transformation.
The A_{k+1} can be expressed without using A_k explicitly.

$$A_{k+1} = A_{k-1} - \boldsymbol{u}_{k-1}\boldsymbol{v}_{k-1}{}^T - \boldsymbol{v}_{k-1}\boldsymbol{u}_{k-1}{}^T - \boldsymbol{u}_k\boldsymbol{v}_k{}^T - \boldsymbol{v}_k\boldsymbol{u}_k{}^T$$

Similarly, matrix A_{p+1} after the p times of mirror transformation becomes:

$$A_{p+1} = A_1 - \sum_{i=1}^{p}(\boldsymbol{u}_i\boldsymbol{v}_i^T + \boldsymbol{v}_i\boldsymbol{u}_i^T)$$

Therefore, matrix A_{p+1} can be computed at a higher speed if matrix A_1 is updated in the lump, once per $2p$ matrices. If the original matrix is Hermitian, the transpose notation "^{T}" should be replaced with the Hermite conjugate notation "*". For details, refer to (3) and (4) in the reference bibliography.

5.1.2.4 QR method

For a tridiagonal matrix T, there is a unitary matrix Q and an upper triangular matrix R (for which all diagonal components are positive) such that T is uniquely decomposed into $T = QR$. $T_k = T$ is assumed, T_k is decomposed into $T_k = Q_k R_k$, and these are multiplied in the reverse order to form:

$$T_{k+1} \leftarrow R_k Q_k = Q_k^* T_k Q_k (Q_k^* \text{ is the adjoint matrix of } Q_k)(k = 1, \cdots)$$

If $T_1, T_2, \cdots, T_k, T_{k+1}$ are created, they are all tridiagonal matrices. As $k \to \infty, T_k$ converges to a diagonal matrix having the eigenvalues of T as its diagonal elements.
To accelerate convergence in the actual QR method, the values μ_k are taken as approximations of the eigenvalues, $T_k - \mu_k I$ are created in place of T_k by performing an origin shift, and these are decomposed into:

$$T_k - \mu_k I = Q_k R_k$$

To calculate the approximation of an eigenvalue, consider the case when there is an adjacent eigenvalue (or an eigenvalue having a close absolute value). Let the eigenvalue μ_k of the lower-right corner submatrix obtained by the root-free QR method. If $T_{k+1} = R_k Q_k + \mu_k I$ is created, then T_{k+1} becomes:

$$T_{k+1} = Q_k^* T_k Q_k$$

After this operation is iterated until the sequence of matrices converges, the values adjusted by the cumulative amount the origin was shifted become the eigenvalues.
The eigenvectors of the original matrix before tridiagonalization are obtained by the following procedures. First, sequentially accumulate the transformation matrices used when obtaining the trigonal matrix T according to the Householder transformation method. Next, accumulate the transformation matrices $Q_1, Q_2, \cdots, Q_k$ obtained according to the QR method.

5.1.2.5 root-free QR method

The root-free QR method, which eliminates the square root calculations of the QR method, is faster when only seeking the eigenvalues of a real symmetric tridiagonal matrix. Let $\alpha_1, \cdots, \alpha_n$ be the diagonal elements and $\beta_1, \cdots, \beta_{n-1}$ be the subdiagonal elements. Let one of the components of the transformation matrix during the calculation be $P^{(i)}$ and let S_i and C_i be $\sin\theta$ and $\cos\theta$ within $P^{(i)}$. Although square roots must be computed in the calculations:

$$P_i = \alpha_i C_{i-1} - \beta_{i-1} S_{i-1} C_{i-2}$$

$$S_i = \frac{\beta_i}{\sqrt{P_i^2 + \beta_i^2}}$$

$$C_i = \frac{P_i}{\sqrt{P_i^2 + \beta_i^2}}$$

$$\text{New } \alpha_{i-1} = \alpha_i + P_{i-1}C_{i-2} - P_iC_{i-1}$$

$$\text{New } \beta_{i-2} = S_{i-2}\sqrt{P_{i-1}^2 + \beta_{i-1}^2}$$

If these calculations are performed using the squared formats of each of P_i, β_i, S_i, and C_i, and if the following values are assumed: $C_0 = 1, S_0 = 0, r_1 = \alpha_1, P_1^2 = \alpha_1^2, \alpha_{n+1} = \beta_{n+1} = 0$ then the equations can be expressed as follows:

$$t_i^2 = P_i^2 + \beta_{i+1}^2$$

$$\text{New } \beta_i^2 = S_{i-1}^2 t_i^2$$

$$S_i^2 = \frac{\beta_{i+1}^2}{t_i^2}, C_i^2 = \frac{P_i^2}{t_i^2}$$

$$P_{i+1}^2 = \alpha_{i+1}^2 C_i^2 - 2\alpha_i + S_i^2\gamma_i + \beta_{i+1}^2 S_i^2 C_{i-1}^2$$

$$\gamma_{i+1} = \alpha_{i+1}C_i^2 = S_i^2\gamma_i$$

$$\text{New } \alpha_i = \alpha_{i+1} + \gamma_i - \gamma_{i+1}$$

and the calculations can be performed without computing any square roots.
For details, refer to (9) in the reference bibliography.

5.1.2.6 Bisection method

The bisection method obtains several of the largest or smallest eigenvalues of a real symmetric tridiagonal matrix T. If we let $d_1, d_2, \cdots, d_n$ be the diagonal components of T, let $s_1, s_2, \cdots, s_{n-1}$ be the subdiagonal components, let λ be a variable, and create the sequence of functions:

$$f_0(\lambda) = 1$$

$$f_1(\lambda) = d_1 - \lambda$$

$$f_i(\lambda) = (d_i - \lambda)f_{i-1}(\lambda) - s_{i-1}^2 f_{i-2}(\lambda) \quad (i = 2, \cdots, n)$$

then $f_0(\lambda), f_1(\lambda), \cdots, f_m(\lambda)$ is the Sturm sequence. That is, if we let $L(\lambda)$ be the number of non-matching signs for the successive sequence of functions for a given λ, then this $L(\lambda)$ is equal to the number of eigenvalues less than λ. To prevent overflow or underflow, if $g_i(\lambda)$ is actually defined as:

$$g_i(\lambda) = \frac{f_i(\lambda)}{f_{i-1}(\lambda)} \quad (i = 1, 2, \cdots, n)$$

$L(\lambda)$ becomes the number of $g_i(\lambda)$ that are negative. Furthermore, $g_i(\lambda)$ satisfies the following:

$$g_1(\lambda) = d_1 - \lambda$$

$$g_i = (d_i - \lambda) - \frac{s_{i-1}^2}{g_{i-1}(\lambda)} \quad (i = 2, \cdots, n)$$

If $g_{i-1}(\lambda)$=0, then $g_i(\lambda)$ is assumed to be:

$$g_i(\lambda) = (d_i - \lambda) - \frac{|s_{i-1}|}{\varepsilon} \quad (\varepsilon : \text{Units for determining error})$$

Assume that the eigenvalues of T satisfy:

$$\lambda_1 \le \lambda_2 \le \cdots\cdots \le \lambda_m$$

From the Gerschgorin theorem, the lower limit ($x_{\min}$) and upper limit ($x_{\max}$) of all the eigenvalues are given by:

$$x_{\max} = \max(d_i + (|s_{i-1}| + |s_i|)) \quad (1 \le i \le n)$$

$$x_{\min} = \min(d_i - (|s_{i-1}| + |s_i|)) \quad (1 \le i \le n)$$

where, $s_0 = s_n = 0$ is assumed.
We continue to make the eigenvalue existence range smaller by repeatedly subdividing the interval while counting the number of eigenvalues as described above, based on $x_{\min}$ and $x_{\max}$. In this way, both ends of the infinitesimal interval can be made to converge to a given eigenvalue.
For information about the Sturm sequence of functions and the Gerschgorin theorem, refer to entries (2) and (5) of the bibliography.

5.1.2.7 Accumulation of similarity (unitary) transformation by block algorithm

In seeking the eigenvectors of a real symmetric matrix using the QR method or the inverse iteration method, it is necessary to calculate the accumulation of the similarity (unitary) matrices that had already been computed in the preceding Householder transformation. It is very effective to apply the block algorithm to this accumulating procedure for getting better performance.
Let P_k be a transformation matrix that is obtained in Householder transformation which transform the real symmetric matrix to a tridiagonal matrix.

$$P_k = \boldsymbol{I} - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

The accumulation of the transformation matrix P_k becomes to:

$$P_1 P_2 \cdots P_{n-2} = \boldsymbol{I} - \Sigma_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T$$

where $\boldsymbol{w}_i^T$ is expressed by the following recurrence formula.

$$\boldsymbol{w}_{n-2}^T = \frac{\boldsymbol{u}_{n-2}^T}{H_{n-2}}$$

$$\boldsymbol{w}_i^T = \frac{\boldsymbol{u}_i^T - \displaystyle\sum_{j=i-1}^{n-2} (\boldsymbol{u}_i^T \boldsymbol{u}_j) \boldsymbol{w}_j^T}{H_i}$$

If we let $\boldsymbol{V}$ be the eigenvectors of a real symmetric tridiagonal matrix which are obtained with the QR method or the inverse iteration method. The eigenvectors $\boldsymbol{X}$ of the original matrix is obtained by the following.

$$\begin{aligned} \boldsymbol{X} &= P_1 \cdots P_{n-2} \boldsymbol{V} \\ &= \boldsymbol{V} - \sum_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T \boldsymbol{V} \end{aligned}$$

A product of a similarity (unitary) matrix P_k and the eigenvectors $\boldsymbol{V}$ is a rank-one update. Therefore, the accumulation of the transformation matrices can be obtained at a higher speed if the matrix updates are performed in the lump. If the original matrix is Hermitian, the transpose notation "^{T}" should be replaced with the Hermite conjugate notation "*".

5.1.2.8 Inverse iteration method

The inverse iteration method is used to obtain the eigenvector corresponding to the eigenvalues obtained by the root-free QR method or bisection method.
Assume the approximate value μ_k of a given eigenvalue λ_k of the real symmetric tridiagonal matrix T has been obtained. If a suitable initial vector $\boldsymbol{v_0}$ has been chosen at this time and the linear simultaneous equations:

$$(T - \mu_k I)\boldsymbol{v}_i = \boldsymbol{v}_{i-1} \quad (i = 1, 2, \cdots)$$

are iteratively solved, then if $\boldsymbol{v_i}$ satisfy the convergence conditions, they converge to the eigenvector.
To solve the simultaneous linear equations, partial pivoting is performed while using the Gauss method to perform an LU decomposition. Then forward elimination and back substitution are performed.

5.1.2.9 Generalized eigenvalue problem

A Cholesky decomposition of B is performed:

$$B = LL^*$$

in the generalized eigenvalue problem for a Hermitian matrix:

$$A\boldsymbol{x} = \lambda B\boldsymbol{x} \quad (A: \text{ Hermitian}, B: \text{ Positive Hermitian})$$

yielding:

$$(L^{-1}A(L^*)^{-1})(L^*\boldsymbol{x}) = \lambda(L^*\boldsymbol{x})$$

If we set:

$$P = L^{-1}A(L^*)^{-1}$$

$$L^*\boldsymbol{x} = \boldsymbol{y}$$

then the generalized eigenvalue problem is transformed to a standard eigenvalue problem for the Hermitian matrix P.

$$P\boldsymbol{y} = \lambda\boldsymbol{y}$$

The eigenvector of matrix A is given by:

$$\boldsymbol{x} = (L^*)^{-1}\boldsymbol{y}.$$

Generalized eigenvalue problems for Hermitian matrix other than $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (B : Positive Hermitian) are classified into two cases by the position of positive Hermitian Matrix B as:

$$AB\boldsymbol{x} = \lambda\boldsymbol{x}$$

and

$$BA\boldsymbol{x} = \lambda\boldsymbol{x}$$

The eigenvalues λ and the eigenvectors x of these equations can be obtained by reducing them to standard eigenvalue problems using the following procedure:

① Apply the Cholesky decomposition to the positive matrix B as $B = L^*L$. (Where L is a lower triangle matrix.)

② $AB\boldsymbol{x} = \lambda\boldsymbol{x}$ is reduced to the eigenvalue problem for $C = LAL^*$, and the eigenvectors are obtained by multiplying the inverse of L.

③ $BA\boldsymbol{x} = \lambda\boldsymbol{x}$ is reduced to the eigenvalue problem for $C = LAL^*$, and the eigenvectors are obtained by multiplying L^*.

5.1.3 Reference Bibliography

(1) Wilkinson, J. H. and Reinsch, C. , "Handbook for Automatic Computation, Vol. II, Linear Algebra", Springer-Verlag, (1971).

(2) Wilkinson, J. H. , "The Algebraic Eigenvalue Problem", Clarendon Press, Oxford, (1965).

(3) Dongarra J. J., Sorensen D. C., and Hammarling A. J., "Block reduction of matrices to condensed forms for eigenvalue computations", Journal of Computational and Applied Mathematics, Vol.27, PP.215-227 (1989).

(4) Dongarra J. J. and van de Geijn R. A. , "Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures", LAPACK Working Note 30, PP.1-12(1991).

(5) Francis, J. G. F. , "The QR transformation, I, II", Comput. J. 4, pp. 265-271, pp.332-345(1961, 1962).

(6) Cuppen, J. J. M., "A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem", Numer. Math. 36, pp. 177-195(1981).

(7) Gu, M. and Eisenstat, S. C., "A Stable and Efficient Algorithm for the rank-1 modification of the symmetric eigenproblem", SIAM J. Matrix Anal. Appl. 15, pp. 1266-1276(1994).

(8) Gu, M. and Eisenstat, S. C., "A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem", SIAM J. Matrix Anal. Appl. 16, pp. 172-191(1995).

(9) Y. Beppu and I. Ninomiya, "HQRII—A Fast Diagonalization Subroutine", Computers and Chemistry Vol.6(1982).

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

5.2.1 ASL_qcsmaa, ASL_pcsmaa All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix

(1) **Function**

ASL_qcsmaa or ASL_pcsmaa uses the Householder method and QR method to obtain all eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:
  ierr = ASL_qcsmaa (a, lna, n, e, w1, nt);
Single precision:
  ierr = ASL_pcsmaa (a, lna, n, e, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vectors) corresponding to each eigenvalue
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	e	{D*, R*}	n	Output	Eigenvalues
5	w1	{D*, R*}	n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna

(b) nt≥1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0] and a[0] ← 1.0 are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalues obtained. ($1 \leq i \leq$ n)	Eigenvalues correctly obtained by this time are entered in e$[0], \cdots,$ e$[i-2]$ and eigenvectors corresponding to them are entered in a. (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array a.

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal system.

(d) If eigenvectors are not required, use 5.2.2 {ASL_qcsman, ASL_pcsman}.

(7) **Example**

(a) Problem
Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data
Matrix A, lna=11, n=4 and nt=2.

(c) Main program

```
/*      C interface example for ASL_qcsmaa */

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

int main()
{
    double *a;
    int lda=11;
    int n;
    double *e;
    double *w1;
    int nt = 2;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

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

```
}

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

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

mod = n % 4;

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

e = ( double * )malloc((size_t)( sizeof(double) * n ));
if( e == NULL )
{
    printf( "no enough memory for array e\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( "\tn = %6d\n", n );
printf( "\tnt= %6d\n\n", nt );
printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+lda*j] );
    }
}

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

fclose( fp );


ierr = ASL_qcsmaa(a, lda, n, e, w1, nt);


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

for( k=0 ; k<n-3 ; k = k+4 )
{
    printf( "\n" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvalue " );
    }
    printf( "\n" );
    for( i=k ; i< k+4 ; i++ )
    {
        printf( "\t%8.3g     ", e[i] );
    }
    printf( "\n" );

    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvector " );
    }
    printf( "\n" );
    for( j=0 ; j<n ; j++ )
    {
        for( i=k ; i< k+4 ; i++ )
        {
            printf( "\t%8.3g     ", a[j+lda*i] );
        }
        printf( "\n" );
```

```
        }
    }

    if( mod != 0 )
    {
        printf( "\n" );
        for( i=n-mod ; i<n ; i++ )
        {
            printf( "\tEigenvalue " );
        }
        printf( "\n" );
        for( i=n-mod ; i<n ; i++ )
        {
            printf( "\t%8.3g     ", e[i] );
        }
        printf( "\n" );

        for( i=n-mod ; i<n ; i++ )
        {
            printf( "\tEigenvector " );
        }
        printf( "\n" );
        for( j=1 ; j<n ; j++ )
        {
            for( i=n-mod ; i<n ; i++ )
            {
                printf( "\t%8.3g     ", a[j+lda*i] );
            }
            printf( "\n" );
        }
    }

    free( a );
    free( e );
    free( w1 );

    return 0;
}
```

(d) Output results

```
    *** ASL_qcsmaa ***

     ** Input **

    n =      4
    nt=      2

    Input Matrix a

           6         4         4         1
           4         6         1         4
           4         1         6         4
           1         4         4         6

     ** Output **

    ierr =      0

    Eigenvalue  Eigenvalue  Eigenvalue  Eigenvalue
          -1               5               5              15
    Eigenvector     Eigenvector     Eigenvector     Eigenvector
         0.5           0.707               0             0.5
        -0.5        2.78e-17          -0.707             0.5
        -0.5       -1.67e-16           0.707             0.5
         0.5          -0.707        8.33e-17             0.5
```

5.2.2 ASL_qcsman, ASL_pcsman All Eigenvalues of a Real Symmetric Matrix

(1) **Function**

ASL_qcsman or ASL_pcsman uses the Householder method and root-free QR method to obtain all eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:
ierr = ASL_qcsman (a, lna, n, e, w1, nt);
Single precision:
ierr = ASL_pcsman (a, lna, n, e, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D∗ / R∗}	lna×n	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	e	{D∗ / R∗}	n	Output	Eigenvalues
5	w1	{D∗ / R∗}	n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < n \leq lna$

(b) $nt \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]←a[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalues obtained. ($1 \leq i \leq$ n)	Eigenvalues correctly obtained by this time are entered in $\text{e}[0], \cdots, \text{e}[\text{i}-2]$. (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array a.

(b) Eigenvalues are stored in ascending order.

5.2.3 ASL_qcsmss, ASL_pcsmss Eigenvalues and Eigenvectors of a Real Symmetric Matrix

(1) **Function**

ASL_qcsmss or ASL_pcsmss uses the Householder method, root free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type) and the inverse iteration method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_qcsmss (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, nt);

Single precision:

ierr = ASL_pcsmss (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	eps	{D, R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalues convergence test. (See Note (d))
5	e	{D*, R*}	m	Output	Eigenvalues
6	m	I	1	Input	The number m of eigenvalues to be obtained.
7	ve	{D*, R*}	lnv×m	Output	Eigenvectors (column vector) corresponding to each eigenvalue.
8	lnv	I	1	Input	Adjustable dimension of array ve
9	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
10	iw1	I*	m	Output	Eigenvectors flag (See Note (e))
11	w1	{D*, R*}	8×n	Work	Work area
12	nt	I	1	Input	Number of tasks to be generated
13	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna, lnv

(b) 0<m≤n

(c) nt≥1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0] and ve[0] ← 1.0 are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array a.

(b) If isw≥0, the eigenvalues are stored in descending order. If isw<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (ierr = 2000 is output), the following processing is performed.
If iw1[i − 1] = 0: The i-th eigenvector calculation is normally terminated.
If iw1[i − 1] ≠ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for iw1[i − 1].
If processing is normally terminated (ierr = 0 is output), iw1[i − 1] = 0 is set.

(f) The eigenvectors are an orthonormal system.

(g) If eigenvectors are not required, use 5.2.4 $\left\{\begin{matrix}\text{ASL_qcsmsn}\\ \text{ASL_pcsmsn}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain the three smallest eigenvalues of the matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, lna=11, n=6, eps=−1.0, m=3, lnv=11, isw=−1 and nt=2.

(c) Main program

```
/*      C interface example for ASL_qcsmss */

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

int main()
{
    double *a;
    int lda=11;
    int n;
    double ceps = -1.0;
    double *e;
    int m;
    double *ve;
    int ldv=11;
    int isw = -1;
    int *iw1;
    double *w1;
    int nt = 2;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 4;

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

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

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

    iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( iw1 == NULL )
```

```
{
    printf( "no enough memory for array iw1\n" );
    return -1;
}

w1 = ( double * )malloc((size_t)( sizeof(double) * (8*n) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

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

printf( "\n\tInput Matrix a\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+lda*j] );
    }
}

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

fclose( fp );


ierr = ASL_qcsmss(a, lda, n, ceps, e, m, ve, ldv, isw, iw1, w1, nt);


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

for( k=0 ; k<m-3 ; k = k+4 )
{
    printf( "\n" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvalue  " );
    }
    printf( "\n" );
    for( i=k ; i<k+4 ; i++ )
    {
        printf( "\t%8.3g     ", e[i] );
    }
    printf( "\n" );

    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvector " );
    }
    printf( "\n" );
    for( j=0 ; j<n ; j++ )
    {
        for( i=k ; i<k+4 ; i++ )
        {
            printf( "\t%8.3g     ", ve[j+ldv*i] );
        }
        printf( "\n" );
    }
}

if( mod != 0 )
{
    printf( "\n" );
    for( i= m-mod ; i<m ; i++ )
    {
        printf( "\tEigenvalue  " );
    }
    printf( "\n" );
    for( i= m-mod ; i<m ; i++ )
    {
        printf( "\t%8.3g     ", e[i] );
    }
```

```
            printf( "\n" );

            for( i= m-mod ; i<m ; i++ )
            {
                printf( "\tEigenvector " );
            }
            printf( "\n" );
            for( j=0 ; j<n ; j++ )
            {
                for( i= m-mod ; i<m ; i++ )
                {
                    printf( "\t%8.3g     ", ve[j+ldv*i] );
                }
                printf( "\n" );
            }
        }

        free( a );
        free( e );
        free( ve );
        free( iw1 );
        free( w1 );

        return 0;
    }
```

(d) Output results

```
        *** ASL_qcsmss ***

         ** Input **

        n =      4
        m =      3
        nt=      2


        Input Matrix a

               4         1         0         0
               1         3         1         0
               0         1         3         1
               0         0         1         4

         ** Output **

        ierr =      0

        Eigenvalue      Eigenvalue      Eigenvalue
            1.59               3            4.41
        Eigenvector     Eigenvector     Eigenvector
          -0.271             0.5          -0.653
           0.653            -0.5          -0.271
          -0.653            -0.5           0.271
           0.271             0.5           0.653
```

5.2.4 ASL_qcsmsn, ASL_pcsmsn Eigenvalues of a Real Symmetric Matrix

(1) **Function**

ASL_qcsmsn or ASL_pcsmsn uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the real symmetric matrix A (two-dimensional array type) (upper triangular type).

(2) **Usage**

Double precision:

ierr = ASL_qcsmsn (a, lna, n, eps, e, m, isw, w1, nt);

Single precision:

ierr = ASL_pcsmsn (a, lna, n, eps, e, m, isw, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna×n	Input	Real symmetric matrix A (two-dimensional array type)(upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	eps	{D, R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalues convergence test. (See Note (d))
5	e	{D*, R*}	m	Output	Eigenvalues
6	m	I	1	Input	The number m of eigenvalues to be obtained.
7	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
8	w1	{D*, R*}	5×n	Work	Work area
9	nt	I	1	Input	Number of tasks to be generated
10	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0 < n \leq lna$

(b) $0 < m \leq n$

(c) $nt \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]←a[0] is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portion of array a.

(b) If $isw \geq 0$, the eigenvalues are stored in descending order. If $isw < 0$, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If $eps \leq 0$, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

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

5.3.1 ASL_hchraa, ASL_gchraa All Eigenvalues and All Eigenvectors of a Hermitian Matrix

(1) **Function**

ASL_hchraa or ASL_gchraa uses the Householder method and QR method to obtain all eigenvalues of the Hermitian matrix A=(ar, ai) (two-dimensional array type) (upper triangular type) (real argument type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_hchraa (ar, ai, lna, n, e, vr, vi, lnv, w1, nt);

Single precision:

ierr = ASL_gchraa (ar, ai, lna, n, e, vr, vi, lnv, w1, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{ D* / R* }	lna × n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
2	ai	{ D* / R* }	lna × n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	e	{ D* / R* }	n	Output	Eigenvalues
6	vr	{ D* / R* }	lnv × n	Output	Real part (column vector) of eigenvectors corresponding to each eigenvalue
7	vi	{ D* / R* }	lnv × n	Output	Imaginary part (column vectors) of eigenvectors corresponding to each eigenvalue
8	lnv	I	1	Input	Adjustable dimension of arrays vr and vi
9	w1	{ D* / R* }	3×n	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) 0<n≤lna, lnv

(b) nt≥1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0], vr[0] ← 1.0 and vi[0] ← 0.0 are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \le i \le$ n).	Eigenvalues obtained by this time are entered in e[0], $\cdots$, e[i − 2]. (However, the order is irregular.) Not eigenvector is obtained at this time.

(6) **Notes**

(a) Real and imaginary parts of the Hermitian matrix are stored only in the upper triangular portions of arrays ar and ai respectively. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal set.

(d) If eigenvectors are not required, use 5.3.2 $\left\{\begin{matrix}\text{ASL_hchran}\\ \text{ASL_gchran}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Real part ar and imaginary part ai of matrix A, lna=11, n=4, lnv=10 and nt=2.

(c) Main program

```
/*      C interface example for ASL_hchraa */

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

int main()
{
    double *ar;
    double *ai;
    int lda=11;
    int n;
    double *e;
    double *vr;
    double *vi;
    int ldv=11;
    double *w1;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;
```

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

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

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

mod = n % 2;

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

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

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

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

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

w1 = ( double * )malloc((size_t)( sizeof(double) * (3*n) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\tn = %6d\n", n );
printf( "\tnt= %6d\n\n", nt );

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+lda*j] );
        fscanf( fp, "%lf", &ai[i+lda*j] );
    }
}

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

fclose( fp );

ierr = ASL_hchraa(ar, ai, lda, n, e, vr, vi, ldv, w1, nt);

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

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

        for( j=0 ; j<n-1 ; j = j+2 )
        {
            printf( "\n" );
            for( i=0 ; i<2 ; i++ )
            {
                printf( "\tEigenvalue              " );
            }
            printf( "\n" );
            printf( "\t%8.3g                \t%8.3g\n",
                e[j], e[j+1] );

            for( i=0 ; i<2 ; i++ )
            {
                printf( "\tEigenvector             " );
            }
            printf( "\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
                    vr[i+ldv*j], vi[i+ldv*j], vr[i+ldv*(j+1)],vi[i+ldv*(j+1)] );
            }
        }

        if( mod != 0 )
        {
            printf( "\n" );
            printf( "\tEigenvalue\n" );
            printf( "\t%8.3g\n", e[n-1] );
            printf( "\tEigenvector\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t%8.3g , %8.3g\n",
                        vr[i+ldv*(n-1)], vi[i+ldv*(n-1)] );
            }
        }

        free( ar );
        free( ai );
        free( e );
        free( vr );
        free( vi );
        free( w1 );

        return 0;
    }
```

(d) Output results

```
        *** ASL_hchraa ***

         ** Input **

        n =      4
        nt=      2


        Input Matrix a ( Real , Imaginary )

        (       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
        (       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
        (       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
        (      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

         ** Output **

        ierr =      0

        Eigenvalue                      Eigenvalue
               0                               8
        Eigenvector                     Eigenvector
             0.5 ,        0               -0.707 ,        0
            -0.5 ,        0             6.38e-16 ,        0
        1.67e-16 ,      0.5                0.354 ,    0.354
       -1.11e-16 ,      0.5               -0.354 ,    0.354

        Eigenvalue                      Eigenvalue
               8                              12
        Eigenvector                     Eigenvector
               0 ,        0                  0.5 ,        0
         -0.0999 ,      0.7                  0.5 ,        0
            -0.3 ,     -0.4                  0.5 , -5.55e-16
            -0.4 ,      0.3                 -0.5 , -4.44e-16
```

5.3.2 ASL_hchran, ASL_gchran All Eigenvalues of a Hermitian Matrix

(1) **Function**

ASL_hchran or ASL_gchran uses the Householder method and root-free QR method to obtain all eigenvalues of the Hermitian matrix A=(ar, ai) (two-dimensional array type) (upper triangular type) (real argument type).

(2) **Usage**

Double precision:
ierr = ASL_hchran (ar, ai, lna, n, e, w1, nt);
Single precision:
ierr = ASL_gchran (ar, ai, lna, n, e, w1, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	e	{D*, R*}	n	Output	Eigenvalues
6	w1	{D*, R*}	3×n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $0<n\leq lna$

(b) $nt\geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]←ar[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ n).	Eigenvalues obtained by this time are entered in $\mathrm{e}[0], \cdots, \mathrm{e}[\mathrm{i}-2]$. (However, the order is irregular.)

(6) **Notes**

(a) Real and imaginary parts of the Hermitian matrix are stored only in the upper triangular portions of arrays ar and ai respectively. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

5.3.3 ASL_hchrss, ASL_gchrss Eigenvalues and Eigenvectors of a Hermitian Matrix

(1) **Function**

ASL_hchrss or ASL_gchrss uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A=(ar, ai) (two-dimensional array type) (upper triangular type) (real argument type) and the inverse iteration method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_hchrss (ar, ai, lna, n, eps, e, m, vr, vi, lnv, isw, iw1, w1, nt);

Single precision:

ierr = ASL_gchrss (ar, ai, lna, n, eps, e, m, vr, vi, lnv, isw, iw1, w1, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	eps	{D, R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
6	e	{D*, R*}	m	Output	Eigenvalues
7	m	I	1	Input	The number of m of eigenvalues to be obtained.
8	vr	{D*, R*}	lnv × m	Output	Real part (column vector) of eigenvectors corresponding to each eigenvalue
9	vi	{D*, R*}	lnv × m	Output	Imaginary parts (column vectors) of eigenvectors corresponding to each eigenvalue
10	lnv	I	1	Input	Adjustable dimension of arrays vr and vi
11	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
12	iw1	I*	m	Output	Eigenvector flag (See Note (e))
13	w1	{D*, R*}	10×n	Work	Work area
14	nt	I	1	Input	Number of tasks to be generated
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

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

(b) $0 < m \leq n$

(c) $nt \geq 1$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← ar[0], vr[0] ← 1.0 and vi[0] ← 0.0 are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The real and imaginary parts of the Hermitian matrix should be stored only in the upper triangular portions of arrays ar and ai respectively (See Appendix A).

(b) If $isw \geq 0$, the eigenvalues are stored in descending order. If $isw < 0$, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (ierr = 2000 is output), the following processing is performed.
If iw1[i − 1] = 0: The *i*-th eigenvector calculation is normally terminated.
If iw1[i − 1] ≠ 0: The convergence condition is not satisfied for the *i*-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for iw1[i − 1].
If processing is normally terminated (ierr = 0 is output), iw1[i − 1] = 0 is set.

(f) The eigenvectors are an orthonormal set.

(g) If eigenvectors are not required, use 5.3.4 {ASL_hchrsn, ASL_gchrsn}.

(7) **Example**

(a) Problem

Obtain the three largest eigenvalues of the Hermitian matrix A.

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Real part ar and imaginary part ai of matrix A, lna=11, n=4, eps=−1.0, m=3, lnv=11, isw=1 and nt=2.

(c) Main program

```
/*      C interface example for ASL_hchrss */

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

int main()
{
    double *ar;
    double *ai;
    int lda=11;
    int n;
    double ceps= -1.0;
    double *e;
    int m;
    double *vr;
    double *vi;
    int ldv=11;
    int isw=1;
    int *iw1;
    double *w1;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 2;

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

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

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

    vr = ( double * )malloc((size_t)( sizeof(double) * (ldv*m) ));
```

```
if( vr == NULL )
{
    printf( "no enough memory for array vr\n" );
    return -1;
}

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

iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
if( iw1 == NULL )
{
    printf( "no enough memory for array iw1\n" );
    return -1;
}

w1 = ( double * )malloc((size_t)( sizeof(double) * (10*n) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

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

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+lda*j] );
        fscanf( fp, "%lf", &ai[i+lda*j] );
    }
}

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

fclose( fp );


ierr = ASL_hchrss(ar, ai, lda, n, ceps, e, m, vr, vi, ldv, isw, iw1, w1, nt);


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

for( j=0 ; j<m-1 ; j = j+2 )
{
    printf( "\n" );
    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvalue              " );
    }
    printf( "\n" );
    printf( "\t%8.3g                \t%8.3g\n",
        e[j], e[j+1] );

    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvector             " );
    }
    printf( "\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
            vr[i+ldv*j], vi[i+ldv*j], vr[i+ldv*(j+1)],vi[i+ldv*(j+1)] );
    }
}

if( mod != 0 )
{
    printf( "\n" );
```

```
        printf( "\tEigenvalue\n" );
        printf( "\t%8.3g\n", e[m-1] );
        printf( "\tEigenvector\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g\n",
                    vr[i+ldv*(m-1)], vi[i+ldv*(m-1)] );
        }
    }

    free( ar );
    free( ai );
    free( e );
    free( vr );
    free( vi );
    free( iw1 );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_hchrss ***

 ** Input **

n =      4
m =      3
nt=      2

Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
      12                          8
Eigenvector                 Eigenvector
     0.5 ,        0                0 ,        0
     0.5 , 1.02e-16          -0.0999 ,      0.7
     0.5 , -5.41e-16            -0.3 ,     -0.4
    -0.5 , -4.58e-16            -0.4 ,      0.3

Eigenvalue
       8
Eigenvector
   0.707 ,        0
-6.66e-16 , -5.09e-17
  -0.354 ,   -0.354
   0.354 ,   -0.354
```

5.3.4 ASL_hchrsn, ASL_gchrsn Eigenvalues of a Hermitian Matrix

(1) **Function**

ASL_hchrsn or ASL_gchrsn uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A=(ar, ai) (two-dimensional array type) (upper triangular type) (real argument type).

(2) **Usage**

Double precision:
  ierr = ASL_hchrsn (ar, ai, lna, n, eps, e, m, isw, w1, nt);
Single precision:
  ierr = ASL_gchrsn (ar, ai, lna, n, eps, e, m, isw, w1, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrix A
5	eps	{D, R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
6	e	{D*, R*}	m	Output	Eigenvalues
7	m	I	1	Input	The number of m of eigenvalues to be obtained.
8	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
9	w1	{D*, R*}	5×n	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) 0<n≤lna

(b) 0<m≤n

(c) nt≥1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← ar[0], is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The real and imaginary parts of the Hermitian matrix should be stored only in the upper triangular portions of arrays ar and ai respectively (See Appendix A).

(b) If isw≥0, the eigenvalues are stored in descending order. If isw<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

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

5.4.1 ASL_hcheaa, ASL_gcheaa All Eigenvalues and All Eigenvectors of a Hermitian Matrix

(1) **Function**

ASL_hcheaa or ASL_gcheaa uses the Householder method or QR method to obtain all eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type) and all corresponding eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_hcheaa (a, lna, n, e, w1, w2, nt);

Single precision:

ierr = ASL_gcheaa (a, lna, n, e, w1, w2, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z∗ / C∗}	lna × n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vector) corresponding to each eigenvalue
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	e	{D∗ / R∗}	n	Output	Eigenvalues
5	w1	{D∗ / R∗}	n	Work	Work area
6	w2	{Z∗ / C∗}	n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna

(b) nt≥ 1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← creal(a[0]) and a[0] ←(1.0, 0.0) are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \le i \le$ n).	Eigenvalues obtained by this time are entered in e[0], $\cdots$, e[i−2] (However, the order is irregular). No eigenvector is obtained at this time.

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array a. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

(c) The eigenvectors are an orthonormal set.

(d) If eigenvectors are not required, use 5.4.2 $\left\{\begin{array}{c}\text{ASL_hchean}\\ \text{ASL_gchean}\end{array}\right\}$.

(7) **Example**

(a) Problem

Obtain all eigenvalues of the matrix:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, lna=11 and n=4.

(c) Main program

```
/*      C interface example for ASL_hcheaa */

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

int main()
{
    double _Complex *a;
    int lda=11;
    int n;
    double *e;
    double *w1;
```

```
double _Complex *w2;
int nt = 2;
int ierr;
int i,j;
FILE *fp;

int mod;

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

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

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

mod = n % 2;

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

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

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

w2 = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
if( w2 == NULL )
{
    printf( "no enough memory for array w2\n" );
    return -1;
}

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

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        double tmp_re, tmp_im;
        fscanf( fp, "%lf", &tmp_re );
        fscanf( fp, "%lf", &tmp_im );
        a[i+lda*j] = tmp_re + tmp_im * _Complex_I;
    }
}

for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", creal(a[j+lda*i]), -cimag(a[j+lda*i]) );
    }
    for( j=i ; j<n ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", creal(a[i+lda*j]), cimag(a[i+lda*j]) );
    }
    printf( "\n" );
}

fclose( fp );


ierr = ASL_hcheaa(a, lda, n, e, w1, w2, nt);


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

for( j=0 ; j<n-1 ; j = j+2 )
{
```

```
        printf( "\n" );
        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvalue              " );
        }
        printf( "\n" );
        printf( "\t%8.3g                \t%8.3g\n",
            e[j], e[j+1] );

        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvector             " );
        }
        printf( "\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
                creal(a[i+lda*j]), cimag(a[i+lda*j]), creal(a[i+lda*(j+1)]), cimag(a[i+lda*(j+1)]) );
        }
    }

    if( mod != 0 )
    {
        printf( "\n" );
        printf( "\tEigenvalue\n" );
        printf( "\t%8.3g\n", e[n-1] );
        printf( "\tEigenvector\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g\n",
                    creal(a[i+lda*(n-1)]), cimag(a[i+lda*(n-1)]) );
        }
    }

    free( a );
    free( e );
    free( w1 );
    free( w2 );

    return 0;
}
```

(d) Output results

```
*** ASL_hcheaa ***

 ** Input **

n =      4

Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
       0                           8
Eigenvector                 Eigenvector
     0.5 ,        0            -0.707 ,        0
    -0.5 ,        0          6.11e-16 ,        0
2.22e-16 ,      0.5             0.354 ,    0.354
-1.11e-16 ,      0.5           -0.354 ,    0.354

Eigenvalue                  Eigenvalue
       8                          12
Eigenvector                 Eigenvector
       0 ,        0               0.5 ,        0
 -0.0999 ,      0.7               0.5 ,        0
    -0.3 ,     -0.4               0.5 , -5.55e-16
    -0.4 ,      0.3              -0.5 , -4.44e-16
```

5.4.2 ASL_hchean, ASL_gchean All Eigenvalues of a Hermitian Matrix

(1) **Function**

ASL_hchean or ASL_gchean uses the Householder method or root-free QR method to obtain all eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type).

(2) **Usage**

Double precision:

ierr = ASL_hchean (a, lna, n, e, w1, w2, nt);

Single precision:

ierr = ASL_gchean (a, lna, n, e, w1, w2, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z*, C*}	lna × n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A
4	e	{D*, R*}	n	Output	Eigenvalues
5	w1	{D*, R*}	n	Work	Work area
6	w2	{Z*, C*}	n	Work	Work area
7	nt	I	1	Input	Number of tasks to be generated
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna

(b) nt≥ 1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← creal(a[0]) is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained ($1 \leq i \leq$ n).	Eigenvalues obtained by this time are entered in $\mathrm{e}[0], \cdots, \mathrm{e}[\mathrm{i}-2]$. (However, the order is irregular.)

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array a. (See Appendix A)

(b) Eigenvalues are stored in ascending order.

5.4.3 ASL_hchess, ASL_gchess Eigenvalues and Eigenvectors of a Hermitian Matrix

(1) **Function**

ASL_hchess or ASL_gchess uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type) and the inverse iterative method to obtain the corresponding eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_hchess (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, w2, nt);

Single precision:

ierr = ASL_gchess (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, w2, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z* / C*}	lna × n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a.
3	n	I	1	Input	Order of matrix A
4	eps	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
5	e	{D* / R*}	m	Output	Eigenvalues
6	m	I	1	Input	The number of m of eigenvalues to be obtained.
7	ve	{Z* / C*}	lnv × m	Output	Eigenvectors (column vector) corresponding to each eigenvalue
8	lnv	I	1	Input	Adjustable dimension of array ve
9	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
10	iw1	I*	m	Output	Eigenvector flag (See Note (e))
11	w1	{D* / R*}	8×n	Work	Work area
12	w2	{Z* / C*}	n	Work	Work area
13	nt	I	1	Input	Number of tasks to be generated
14	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) 0<n≤lna, lnv

(b) 0<m≤n

(c) nt≥ 1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← creal(a[0]) and ve[0] ← (1.0, 0.0) are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array A. (See Appendix A)

(b) If isw≥0, the eigenvalues are stored in descending order. If isw < 0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (ierr = 2000 is output), the following processing is performed.

If iw1[i − 1] = 0: The i-th eigenvector calculation is normally terminated.

If iw1[i − 1] ≠ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for iw1[i − 1].

If processing is normally terminated (ierr = 0 is output), iw1[i − 1] = 0 is set.

(f) The eigenvectors are an orthonormal set.

(g) If eigenvectors are not required, use 5.4.4 $\left\{\begin{matrix}\text{ASL_hchesn}\\ \text{ASL_gchesn}\end{matrix}\right\}$.

(7) **Example**

(a) Problem

Obtain the three largest eigenvalues of the following Hermitian matrix A:

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

and their corresponding eigenvectors.

(b) Input data

Matrix A, lna=11, n=4, eps=−1.0, m=3, lnv=11 and isw=1.

(c) Main program

```
/*      C interface example for ASL_hchess */

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

int main()
{
    double _Complex *a;
    int lda=11;
    int n;
    double ceps= -1.0;
    double *e;
    int m;
    double _Complex *ve;
    int ldv=11;
    int isw=1;
    int *iw1;
    double *w1;
    double _Complex *w2;
    int nt=2;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 2;

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

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

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

    iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( iw1 == NULL )
    {
        printf( "no enough memory for array iw1\n" );
        return -1;
    }

    w1 = ( double * )malloc((size_t)( sizeof(double) * (8*n) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

    w2 = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
    if( w2 == NULL )
    {
        printf( "no enough memory for array w2\n" );
        return -1;
    }

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

    printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
    for( i=0 ; i<n ; i++ )
```

```
    {
        for( j=i ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            a[i+lda*j] = tmp_re + tmp_im * _Complex_I;
        }
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[j+lda*i]), -cimag(a[j+lda*i]) );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+lda*j]), cimag(a[i+lda*j]) );
        }
        printf( "\n" );
    }
    fclose( fp );


    ierr = ASL_hchess(a, lda, n, ceps, e, m, ve, ldv, isw, iw1, w1, w2, nt);


    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    for( j=0 ; j<m-1 ; j = j+2 )
    {
        printf( "\n" );
        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvalue              " );
        }
        printf( "\n" );
        printf( "\t%8.3g                 \t%8.3g\n",
            e[j], e[j+1] );
        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvector             " );
        }
        printf( "\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
                creal(ve[i+ldv*j]), cimag(ve[i+ldv*j]), creal(ve[i+ldv*(j+1)]), cimag(ve[i+ldv*(j+1)]) );
        }
    }
    if( mod != 0 )
    {
        printf( "\n" );
        printf( "\tEigenvalue\n" );
        printf( "\t%8.3g\n", e[m-1] );
        printf( "\tEigenvector\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g\n",
                    creal(ve[i+ldv*(m-1)]), cimag(ve[i+ldv*(m-1)]) );
        }
    }
    free( a );
    free( e );
    free( ve );
    free( iw1 );
    free( w1 );
    free( w2 );
    return 0;
}
```

(d) Output results

```
*** ASL_hchess ***

 ** Input **

n =      4
m =      3
```

```
Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
      12                           8
Eigenvector                 Eigenvector
     0.5 ,        0                0 ,        0
     0.5 , 1.02e-16          -0.0999 ,      0.7
     0.5 ,    -5e-16            -0.3 ,     -0.4
    -0.5 , -4.44e-16            -0.4 ,      0.3

Eigenvalue
       8
Eigenvector
   0.707 ,        0
-6.66e-16 ,        0
  -0.354 ,   -0.354
   0.354 ,   -0.354
```

5.4.4 ASL_hchesn, ASL_gchesn Eigenvalues of a Hermitian Matrix

(1) **Function**

ASL_hchesn or ASL_gchesn uses the Householder method, root-free QR method, or Bisection method to obtain the m largest or m smallest eigenvalues of the Hermitian matrix A (two-dimensional array type) (upper triangular type) (complex argument type).

(2) **Usage**

Double precision:

ierr = ASL_hchesn (a, lna, n, eps, e, m, isw, w1, w2, nt);

Single precision:

ierr = ASL_gchesn (a, lna, n, eps, e, m, isw, w1, w2, nt);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{Z∗ / C∗}	lna × n	Input	Hermitian matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a.
3	n	I	1	Input	Order of matrix A
4	eps	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
5	e	{D∗ / R∗}	m	Output	Eigenvalues
6	m	I	1	Input	The number of m of eigenvalues to be obtained.
7	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
8	w1	{D∗ / R∗}	2×n	Work	Work area
9	w2	{Z∗ / C∗}	n	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna

(b) 0<m≤n

(c) nt≥ 1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← creal(a[0]) is performed.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	

(6) **Notes**

(a) Only the upper triangular portion of the Hermitian matrix should be stored in array a. (See Appendix A)

(b) If isw≥0, the eigenvalues are stored in descending order. If isw<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

5.5 GENERALIZED EIGENVALUE PROBLEM FOR A REAL SYMMETRIC MATRIX (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($Ax = \lambda Bx$)

5.5.1 ASL_qcgsaa, ASL_pcgsaa
All Eigenvalues and All Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

ASL_qcgsaa or ASL_pcgsaa uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and QR method to obtain all eigenvalues and corresponding all eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_qcgsaa (a, lna, n, b, lnb, e, w1, nt);

Single precision:

ierr = ASL_pcgsaa (a, lna, n, b, lnb, e, w1, nt);

(3) Arguments and Return Value

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna × n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Eigenvectors (column vector) corresponding to each eigenvalue
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A and B
4	b	{D*, R*}	lnb × n	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D*, R*}	n	Output	Eigenvalues
7	w1	{D*, R*}	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) 0<n≤lna, lnb

(b) nt≥1

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0]/b[0] and a[0] ← $1.0/\sqrt{\text{b[0]}}$ are performed.
2100	B has a diagonal element very close to zero.	Some eigenvectors may be obtained with low precision, and processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained (1≤i≤n).	Eigenvalues obtained by this time are entered in e[0], ⋯, e[i−2]. (However, the order is irregular.) No eigenvector is obtained at this time.

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays a and b.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(d) If eigenvectors are not required, use 5.5.2 $\left\{\begin{matrix}\text{ASL_qcgsan}\\ \text{ASL_pcgsan}\end{matrix}\right\}$.

(7) **Example**

(a) Problem
Obtain all eigenvalues of $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ and their corresponding eigenvectors, where matrices A and B are as follows:

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

$$B = \begin{bmatrix} 153 & 31 & 58 & -58 \\ 31 & 153 & -53 & 58 \\ 58 & -58 & 153 & 31 \\ -58 & 58 & 31 & 153 \end{bmatrix}$$

(b) Input data
Matrix A, lna=11, n=4, matrix B, lnb=11 and nt=2.

(c) Main program

```
/*      C interface example for ASL_qcgsaa */

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

int main()
{
    double *a;
    int na=11;
```

```
int nn;
double *b;
int nb=11;
double *e;
double *wk;
int nt;
int ierr;
int i,j,k;
FILE *fp;

int mod;

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

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

fscanf( fp, "%d", &nn );
fscanf( fp, "%d", &nt );

mod = nn % 4;

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

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

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

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

printf( "\tn = %6d\n", nn );
printf( "\tnt= %6d\n\n", nt );

printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &a[i+na*j] );
    }
}

for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "%8.3g", a[j+na*i] );
    }
    for( j=i ; j<nn ; j++ )
    {
        printf( "%8.3g", a[i+na*j] );
    }
    printf( "\n" );
}

printf( "\n\tInput Matrix b\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &b[i+nb*j] );
    }
}

for( i=0 ; i<nn ; i++ )
```

```
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "%8.3g", b[j+nb*i] );
    }
    for( j=i ; j<nn ; j++ )
    {
        printf( "%8.3g", b[i+nb*j] );
    }
    printf( "\n" );
}

fclose( fp );


ierr = ASL_qcgsaa(a, na, nn, b, nb, e, wk, nt);


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

for( k=0 ; k<nn-3 ; k = k+4 )
{
    printf( "\n\t" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "Eigenvalue   " );
    }
    printf( "\n\t" );
    for( i=k ; i<k+4 ; i++ )
    {
        printf( " %8.3g    ", e[i] );
    }
    printf( "\n" );

    printf( "\t" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "Eigenvector  " );
    }
    printf( "\n" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "\t" );
        for( i=k ; i<k+4 ; i++ )
        {
            printf( " %8.3g    ", a[j+na*i] );
        }
        printf( "\n" );
    }
}

if( mod != 0 )
{
    printf( "\n\t" );
    for( i= nn-mod ; i<nn ; i++ )
    {
        printf( "Eigenvalue  " );
    }
    printf( "\n\t" );
    for( i= nn-mod ; i<nn ; i++ )
    {
        printf( " %8.3g    ", e[i] );
    }
    printf( "\n" );

    printf( "\t" );
    for( i= nn-mod ; i<nn ; i++ )
    {
        printf( "Eigenvector " );
    }
    printf( "\n" );
    for( j=0 ; j<nn ; j++ )
    {
        printf( "\t" );
        for( i= nn-mod ; i<nn ; i++ )
        {
            printf( " %8.3g    ", a[j+na*i] );
        }
        printf( "\n" );
    }

}

free( a );
free( b );
free( e );
free( wk );
```

```
    return 0;
}
```

(d) Output results

```
*** ASL_qcgsaa ***

 ** Input **

n =      4
nt=      2

Input Matrix a

       2       1       1       2
       1       1       1       1
       1       1       2       2
       2       1       2       4

Input Matrix b

     153      31      58     -58
      31     153     -58      58
      58     -58     153      31
     -58      58      31     153

 ** Output **

ierr =      0

Eigenvalue   Eigenvalue   Eigenvalue   Eigenvalue
 0.000648      0.00537       0.0274        0.217
Eigenvector  Eigenvector  Eigenvector  Eigenvector
   0.0294       0.0498      -0.0161        0.205
  -0.0469       0.0377       0.0686       -0.193
   0.0311      -0.0194        0.086       -0.192
  -0.0196      -0.0332      0.00145         0.21
```

5.5.2 ASL_qcgsan, ASL_pcgsan All Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

ASL_qcgsan or ASL_pcgsan uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and QR method to obtain all eigenvalues.

(2) **Usage**

Double precision:
ierr = ASL_qcgsan (a, lna, n, b, lnb, e, w1, nt);
Single precision:
ierr = ASL_pcgsan (a, lna, n, b, lnb, e, w1, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna × n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A and B
4	b	{D* / R*}	lnb × n	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D* / R*}	n	Output	Eigenvalues
7	w1	{D* / R*}	n	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna, lnb

(b) nt≥1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0]/b[0] is performed.
2100	B has a diagonal element very close to zero.	Processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue is obtained. (1≤i≤n)	Eigenvalues obtained by this time are entered in e[0], ···, e[i−2]. (However, the order is irregular.)

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays a and b.

(b) Eigenvalues are stored in ascending order.

5.5.3 ASL_qcgsss, ASL_pcgsss Eigenvalues and Eigenvectors of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

ASL_qcgsss or ASL_pcgsss uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and the root-free QR method or Bisection method to obtain the m largest eigenvalues or m smallest eigenvalues, and uses the reverse iterative method to obtain the eigenvectors.

(2) **Usage**

Double precision:

ierr = ASL_qcgsss (a, lna, n, b, lnb, eps, e, m, ve, lnv, isw, iw1, w1, nt);

Single precision:

ierr = ASL_pcgsss (a, lna, n, b, lnb, eps, e, m, ve, lnv, isw, iw1, w1, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna × n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A and B
4	b	{D* / R*}	lnb × n	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	The strict upper triangular portion is not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	eps	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
7	e	{D* / R*}	m	Output	Eigenvalues
8	m	I	1	Input	The number of m of eigenvalues to be obtained.
9	ve	{D* / R*}	lnv × m	Output	Eigenvectors (column vector) corresponding to each eigenvalue
10	lnv	I	1	Input	Adjustable dimension of array ve
11	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
12	iw1	I*	m	Output	Eigenvector flag (See Note (a))
13	w1	{D* / R*}	9×n	Work	Work area
14	nt	I	1	Input	Number of tasks to be generated
15	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) 0<n≤lna, lnb, lnv

(b) 0<m≤n

(c) nt≥1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0]/b[0] and ve[0] ← $1.0/\sqrt{\text{b[0]}}$ are performed.
2000	The maximum number of iterations was exceeded by the inverse iterations for obtaining eigenvectors.	Some eigenvectors are obtained with low precision, and processing continues. (See Note (e))
2100	B has a diagonal element very close to zero.	Some eigenvectors may be obtained with low precision, and processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
4000	B was not positive definite.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays a and b.

(b) If isw≥0, the eigenvalues are stored in descending order. If isw<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

(e) If the maximum number of iterations is exceeded when using the inverse iteration method (ierr = 2000 is output), the following processing is performed.
If iw1[i − 1] = 0: The i-th eigenvector calculation is normally terminated.
If iw1[i − 1] ≠ 0: The convergence condition is not satisfied for the i-th eigenvector calculation, and the eigenvector precision is low. In this case, the iteration count is set for iw1[i − 1].
If processing is normally terminated (ierr = 0 is output), iw1[i − 1] = 0 is set.

(f) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(g) If eigenvectors are not required, use 5.5.4 $\left\{\begin{matrix}\text{ASL_qcgssn}\\\text{ASL_pcgssn}\end{matrix}\right\}$.

(7) **Example**

(a) ProblemObtain all eigenvalues of $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ and their corresponding eigenvectors, where matrices A and B are as follows:

$$A = \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

$$B = \begin{bmatrix} 170 & 18 & 33 & -21 & -17 & 13 & 25 & -36 \\ 18 & 171 & -21 & 22 & 13 & -17 & -36 & 25 \\ 33 & -21 & 171 & 18 & 25 & -36 & -17 & 13 \\ -21 & 22 & 18 & 171 & -36 & 25 & 13 & -17 \\ -17 & 13 & 25 & -36 & 171 & 18 & 33 & -21 \\ 13 & -17 & -36 & 25 & 18 & 171 & -21 & -3 \\ 25 & -36 & -17 & 13 & 33 & -21 & 171 & 18 \\ -36 & 25 & 13 & -17 & -21 & -3 & 18 & 171 \end{bmatrix}$$

(b) Input data

Matrix A, lna=11, n=8, matrix B, lnb=11, eps=−1.0, m=2, lnv=10 and isw=−1.

(c) Main program

```
/*      C interface example for ASL_qcgsss */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a;
    int na=11;
    int nn;
    double *b;
    int nb=11;
    double ceps= -1.0;
    double *e;
    int mm;
    double *ve;
    int nv=10;
    int ksw= -1;
    int *kw1;
    double *wk;
    int nt;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

    fp = fopen( "qcgsss.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_qcgsss ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &nn );
    fscanf( fp, "%d", &mm );
    fscanf( fp, "%d", &nt );

    mod = mm % 4;
```

```
a = ( double * )malloc((size_t)( sizeof(double) * (na*nn) ));
if( a == NULL )
{
    printf( "no enough memory for array a\n" );
    return -1;
}

b = ( double * )malloc((size_t)( sizeof(double) * (nb*nn) ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}

e = ( double * )malloc((size_t)( sizeof(double) * mm ));
if( e == NULL )
{
    printf( "no enough memory for array e\n" );
    return -1;
}

ve = ( double * )malloc((size_t)( sizeof(double) * (nv*mm) ));
if( ve == NULL )
{
    printf( "no enough memory for array ve\n" );
    return -1;
}

kw1 = ( int * )malloc((size_t)( sizeof(int) * mm ));
if( kw1 == NULL )
{
    printf( "no enough memory for array kw1\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (9*nn) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tn = %6d\n", nn );
printf( "\tm = %6d\n", mm );
printf( "\tnt= %6d\n\n", nt );

printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &a[i+na*j] );
    }
}

for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "%8.3g", a[j+na*i] );
    }
    for( j=i ; j<nn ; j++ )
    {
        printf( "%8.3g", a[i+na*j] );
    }
    printf( "\n" );
}

printf( "\n\tInput Matrix b\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &b[i+nb*j] );
    }
}

for( i=0 ; i<nn ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "%8.3g", b[j+nb*i] );
    }
    for( j=i ; j<nn ; j++ )
    {
        printf( "%8.3g", b[i+nb*j] );
    }
    printf( "\n" );
```

```
    }

    fclose( fp );

    ierr = ASL_qcgsss(a, na, nn, b, nb, ceps, e, mm, ve, nv, ksw, kw1, wk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    for( k=0 ; k<mm-3 ; k = k+4 )
    {
        printf( "\n" );
        printf( "\t" );
        for( i=0 ; i<4 ; i++ )
        {
            printf( "Eigenvalue   " );
        }
        printf( "\n" );
        printf( "\t" );
        for( i=k ; i<k+4 ; i++ )
        {
            printf( " %8.3g   ", e[i] );
        }
        printf( "\n" );

        printf( "\t" );
        for( i=0 ; i<4 ; i++ )
        {
            printf( "Eigenvector  " );
        }
        printf( "\n" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "\t" );
            for( i=k ; i<k+4 ; i++ )
            {
                printf( " %8.3g   ", ve[j+nv*i] );
            }
            printf( "\n" );
        }
    }

    if( mod != 0 )
    {
        printf( "\n" );
        printf( "\t" );
        for( i= mm-mod ; i<mm ; i++ )
        {
            printf( "Eigenvalue   " );
        }
        printf( "\n" );
        printf( "\t" );
        for( i= mm-mod ; i<mm ; i++ )
        {
            printf( " %8.3g   ", e[i] );
        }
        printf( "\n" );

        printf( "\t" );
        for( i= nn-mod ; i<nn ; i++ )
        {
            printf( "Eigenvector  " );
        }
        printf( "\n" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "\t" );
            for( i= mm-mod ; i<mm ; i++ )
            {
                printf( " %8.3g   ", ve[j+nv*i] );
            }
            printf( "\n" );
        }

    }

    free( a );
    free( b );
    free( e );
    free( ve );
    free( kw1 );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_qcgsss ***

 ** Input **

n =      8
m =      2
nt=      2

Input Matrix a

     611      196     -192      407       -8      -52      -49       29
     196      899      113     -192      -71      -43       -8      -44
    -192      113      899      196       61       49        8       52
     407     -192      196      611        8       44       59      -23
      -8      -71       61        8      411     -599      208      208
     -52      -43       49       44     -599      411      208      208
     -49       -8        8       59      208      208       99     -911
      29      -44       52      -23      208      208     -911       99

Input Matrix b

     170       18       33      -21      -17       13       25      -36
      18      171      -21       22       13      -17      -36       25
      33      -21      171       18       25      -36      -17       13
     -21       22       18      171      -36       25       13      -17
     -17       13       25      -36      171       18       33      -21
      13      -17      -36       25       18      171      -21       -3
      25      -36      -17       13       33      -21      171       18
     -36       25       13      -17      -21       -3       18      171

 ** Output **

ierr =      0

Eigenvalue   Eigenvalue
     -5.3    -1.04e-15
Eigenvector  Eigenvector
 0.000789    -0.00329
  0.00146    -0.00658
 0.000624     0.00658
 -0.00168     0.00329
  -0.0247     -0.0461
   -0.019     -0.0461
   0.0479      -0.023
   0.0445      -0.023
```

5.5.4 ASL_qcgssn, ASL_pcgssn Eigenvalues of a Real Symmetric Matrix (Generalized Eigenvalue Problem $Ax = \lambda Bx$, B: Positive)

(1) **Function**

ASL_qcgssn or ASL_pcgssn uses the Cholesky method to transform the real symmetric matrix (two-dimensional array type) (upper triangular type) generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: Real symmetric matrix, B: Positive real symmetric matrix) to a standard eigenvalue problem and uses the Householder method and the root-free QR method or Bisection method to obtain the m largest eigenvalues or m smallest eigenvalues.

(2) **Usage**

Double precision:
ierr = ASL_qcgssn (a, lna, n, b, lnb, eps, e, m, isw, w1, nt);
Single precision:
ierr = ASL_pcgssn (a, lna, n, b, lnb, eps, e, m, isw, w1, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna × n	Input	Real symmetric matrix A (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrix A and B
4	b	{D* / R*}	lnb × n	Input	Positive symmetric matrix B (two-dimensional array type) (upper triangular type)
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	eps	{D / R}	1	Input	Parameter that assigns an upper limit to the absolute error for use in the eigenvalue convergence test. (See Note (d))
7	e	{D* / R*}	m	Output	Eigenvalues
8	m	I	1	Input	The number of m of eigenvalues to be obtained.
9	isw	I	1	Input	Processing switch isw≥0: Obtain m eigenvalues from the largest one. isw<0: Obtain m eigenvalues from the smallest one.
10	w1	{D* / R*}	5×n	Work	Work area
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) 0<n≤lna, lnb

(b) 0<m≤n

(c) nt≥1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] ← a[0]/b[0] is performed.
2100	B has a diagonal element very close to zero.	Processing continues.
3000	Restriction (a) or (b) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
4000	B was not positive definite.	

(6) **Notes**

(a) Data should be stored only in the upper triangular portions of arrays a and b.

(b) If isw≥0, the eigenvalues are stored in descending order. If isw<0, they are stored in ascending order.

(c) Eigenvalue calculations are appropriately divided up between the root-free QR method and Bisection method internally.

(d) If eps ≤ 0, the optimum value is automatically set internally. Normally, a negative value should be set so that this value will be set automatically. eps is used to obtain eigenvalues by using the Bisection method.

5.6 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($ABx = \lambda x$)

5.6.1 ASL_qcgjaa, ASL_pcgjaa
All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{x} = \lambda \boldsymbol{x}$$

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{x}$.

(2) **Usage**

Double precision:
ierr = ASL_qcgjaa (a, lna, n, b, lnb, e, work, nt);
Single precision:
ierr = ASL_pcgjaa (a, lna, n, b, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer; long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D*, R*}	lna × n	Input	Real symmetric matrix A
				Output	Eigenvectors $\boldsymbol{x}$
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrices A and B
4	b	{D*, R*}	lnb × n	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D*, R*}	n	Output	Eigenvalues λ
7	work	{D*, R*}	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0] $\leftarrow$ a[0] $*$ b[0] and a[0] $\leftarrow \dfrac{1.0}{\sqrt{\text{b[0]}}}$ are performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ n)	Eigenvalues obtained by this time are entered in e[0], $\cdots$, e[i $-$ 2]. (However, the order is irregular.) No eigenvector is obtained at this time.

(6) **Notes**

(a) Arrays a and b should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.6.2 {ASL_qcgjan, ASL_pcgjan} should be used if the eigenvectors are not needed.

(e) 5.7.1 {ASL_qcgkaa, ASL_pcgkaa} should be used if matrix A is only positive.

(7) **Example**

(a) Problem
Obtain all eigenvalues and their corresponding eigenvectors non-symmetric matrix AB when A and B are positive symmetric matrices.

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

Note The input data A and B are defined as

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

where

$$P(a^2, \mathrm{n})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt \quad (1 \le i \le \mathrm{n}; 1 \le j \le \mathrm{n}).$$

All eigenvalues of the each matrix exist within a finit interval which is independent of n.

(b) Input data

n = 4, lna = lnb = 4, nt = 2 and symmetric matrices A, B.

(c) Main program

```
/*      C interface example for ASL_qcgjaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *e, *work;
    double one,tre,fiv;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4;nt=4;
    one=1.0;tre=3.0;fiv=5.0;
    printf( "    *** ASL_qcgjaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    a = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n");
        return -1;
    }
    b = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n");
        return -1;
    }
    e = ( double * )malloc((size_t)(sizeof(double)*n));
    if ( e == NULL )
    {
        printf( "no enough memory for array e\n");
        return -1;
    }
    work = ( double * )malloc((size_t)(sizeof(double)*(2*n)));
    if ( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tlna = %6d\n",lna );
    printf( "\tlnb = %6d\n",lnb );
    printf( "\tnt  = %6d\n",nt  );
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            a[i+n*j]=one/(one+tre*(i+j+2)*(i+j+2))
                +one/(one+tre*(i-j)*(i-j));
            b[i+n*j]=one/(one+fiv*(i+j+2)*(i+j+2))
                +one/(one+fiv*(i-j)*(i-j));
        }
    }
    printf( "\n\tInput Matrix a\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
        printf( "\n" );
    }
    printf( "\n\tInput Matrix b\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
```

```
            printf( "%8.3g",b[i+n*j]);
        }
        printf( "\n" );
    }

    ierr = ASL_qcgjaa(a, lna, n, b, lnb, e, work, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t          Eigenvalue   " );
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "%8.3g",e[j]);
    }
    printf( "\n\t          Eigenvector     " );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
    }
    printf( "\n" );
    free(a);
    free(b);
    free(e);
    free(work);
    return 0;
}
```

(d) Output results

```
    *** ASL_qcgjaa ***

     ** Input **

    n   =      4
    lna =      4
    lnb =      4
    nt  =      4

    Input Matrix a

        1.08   0.286  0.0973  0.0489
       0.286    1.02   0.263  0.0861
      0.0973   0.263    1.01   0.257
      0.0489  0.0861   0.257    1.01

    Input Matrix b

        1.05   0.188    0.06  0.0297
       0.188    1.01   0.175  0.0531
        0.06   0.175    1.01   0.171
      0.0297  0.0531   0.171       1

     ** Output **

    ierr =      0

              Eigenvalue
       0.503   0.706    1.15    2.13
              Eigenvector
       -0.36  -0.573     0.6   0.414
       0.702   0.497   0.257   0.494
      -0.718    0.42  -0.389    0.46
       0.406  -0.626  -0.604   0.324
```

5.6.2 ASL_qcgjan, ASL_pcgjan All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $ABx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{x} = \lambda \boldsymbol{x}$$

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:
  ierr = ASL_qcgjan (a, lna, n, b, lnb, e, work, nt);
Single precision:
  ierr = ASL_pcgjan (a, lna, n, b, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna × n	Input	Real symmetric matrix A
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrices A and B
4	b	{D* / R*}	lnb × n	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D* / R*}	n	Output	Eigenvalues λ
7	work	{D* / R*}	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] is performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ n).	Eigenvalues obtained by this time are entered in $\mathrm{e}[0], \cdots, \mathrm{e}[\mathrm{i}-2]$ (However, the order is irregular).

(6) **Notes**

(a) Arrays a and b should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.6.1 {ASL_qcgjaa, ASL_pcgjaa} should be used if the eigenvectors are needed.

(d) 5.7.2 {ASL_qcgkan, ASL_pcgkan} should be used if matrix A is only positive.

5.7 GENERALIZED EIGENVALUE PROBLEM FOR REAL SYMMETRIC MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) ($BAx = \lambda x$)

5.7.1 ASL_qcgkaa, ASL_pcgkaa All Eigenvalues and All Eigenvectors of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$BA\boldsymbol{x} = \lambda \boldsymbol{x}$$

(A: Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{x}$.

(2) **Usage**

Double precision:

ierr = ASL_qcgkaa (a, lna, n, b, lnb, e, work, nt);

Single precision:

ierr = ASL_pcgkaa (a, lna, n, b, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D∗ / R∗}	lna × n	Input	Real symmetric matrix A
				Output	Eigenvectors $\boldsymbol{x}$
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrices A and B
4	b	{D∗ / R∗}	lnb × n	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D∗ / R∗}	n	Output	Eigenvalues λ
7	work	{D∗ / R∗}	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] and a[0]← $\sqrt{\text{b[0]}}$ are performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ n).	Eigenvalues obtained by this time are entered in e[0], $\cdots$, e[i−2] (However, the order is irregular). No eigenvector is obtained at this time.

(6) **Notes**

(a) Arrays a and b should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^T B^{-1} \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.7.2 {ASL_qcgkan, ASL_pcgkan} should be used if the eigenvectors are not needed.

(e) 5.6.1 {ASL_qcgjaa, ASL_pcgjaa} should be used if matrix A is only positive.

(7) **Example**

(a) Problem
Obtain all eigenvalues and their corresponding eigenvectors non-symmetric matrix AB when A and B are positive symmetric matrices.

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

Note The input data A and B are defined as

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

where

$$P(a^2, \mathrm{n})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt(1 \le i \le \mathrm{n}; 1 \le j \le \mathrm{n}).$$

All eigenvalues of the each matrix exist within a finit interval which is independent of n.

(b) Input data

n=4, lna=lnb=4, nt=2 and symmetric matrices A, B.

(c) Main program

```
/*      C interface example for ASL_qcgkaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *e, *work;
    double one,tre,fiv;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4;nt=4;
    one=1.0;tre=3.0;fiv=5.0;
    printf( "    *** ASL_qcgkaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    a = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n");
        return -1;
    }
    b = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n");
        return -1;
    }
    e = ( double * )malloc((size_t)(sizeof(double)*n));
    if ( e == NULL )
    {
        printf( "no enough memory for array e\n");
        return -1;
    }
    work = ( double * )malloc((size_t)(sizeof(double)*(2*n)));
    if ( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tlna = %6d\n",lna );
    printf( "\tlnb = %6d\n",lnb );
    printf( "\tnt  = %6d\n",nt  );
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            a[i+n*j]=one/(one+tre*(i+j+2)*(i+j+2))
                +one/(one+tre*(i-j)*(i-j));
            b[i+n*j]=one/(one+fiv*(i+j+2)*(i+j+2))
                +one/(one+fiv*(i-j)*(i-j));
        }
    }
    printf( "\n\tInput Matrix a\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
        printf( "\n" );
    }
    printf( "\n\tInput Matrix b\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
```

```
                printf( "%8.3g",b[i+n*j]);
            }
            printf( "\n" );
        }

        ierr = ASL_qcgkaa(a, lna, n, b, lnb, e, work, nt);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\t          Eigenvalue   " );
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",e[j]);
        }
        printf( "\n\t          Eigenvector     " );
        for(i=0; i<n; i++)
        {
            printf( "\n\t" );
            for(j=0; j<n; j++)
            {
                printf( "%8.3g",a[i+n*j]);
            }
        }
        printf( "\n" );
        free(a);
        free(b);
        free(e);
        free(work);
        return 0;
    }
```

(d) Output results

```
        *** ASL_qcgkaa ***

         ** Input **

        n   =      4
        lna =      4
        lnb =      4
        nt  =      4

        Input Matrix a

            1.08   0.286  0.0973  0.0489
           0.286    1.02   0.263  0.0861
          0.0973   0.263    1.01   0.257
          0.0489  0.0861   0.257    1.01

        Input Matrix b

            1.05   0.188    0.06  0.0297
           0.188    1.01   0.175  0.0531
            0.06   0.175    1.01   0.171
          0.0297  0.0531   0.171       1

         ** Output **

        ierr =      0

                  Eigenvalue
           0.503   0.706    1.15    2.13
                  Eigenvector
          -0.276    -0.5   0.635   0.564
            0.54   0.436   0.273   0.676
          -0.551   0.368  -0.414   0.629
           0.311  -0.547  -0.641   0.442
```

5.7.2 ASL_qcgkan, ASL_pcgkan All Eigenvalues of Real Symmetric Matrices (Generalized Eigenvalue Problem $BAx = \lambda x$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$BA\boldsymbol{x} = \lambda \boldsymbol{x}$$

(A : Real symmetric, B: Positive real symmetric) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:
  ierr = ASL_qcgkan (a, lna, n, b, lnb, e, work, nt);
Single precision:
  ierr = ASL_pcgkan (a, lna, n, b, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	lna × n	Input	Real symmetric matrix A
				Output	Input-time contents are not retained.
2	lna	I	1	Input	Adjustable dimension of array a
3	n	I	1	Input	Order of matrices A and B
4	b	{D* / R*}	lnb × n	Input	Real symmetric matrix B
				Output	Input-time contents are not retained.
5	lnb	I	1	Input	Adjustable dimension of array b
6	e	{D* / R*}	n	Output	Eigenvalues λ
7	work	{D* / R*}	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] is performed.
2100	B has a diagonal element very close to zero.	Some results may be obtained with low precision
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000+i	The sequence did not converge in the step where the eigenvalue was obtained ($1 \leq i \leq$ n).	Eigenvalues obtained by this time are entered in $\mathrm{e}[0], \cdots, \mathrm{e}[\mathrm{i}-2]$. (However, the order is irregular.)

(6) **Notes**

(a) Arrays a and b should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.7.1 $\left\{\begin{matrix}\text{ASL_qcgkaa} \\ \text{ASL_pcgkaa}\end{matrix}\right\}$ should be used if the eigenvectors are needed.

(d) 5.6.2 $\left\{\begin{matrix}\text{ASL_qcgjan} \\ \text{ASL_pcgjan}\end{matrix}\right\}$ should be used if matrix A is only positive.

5.8 GENERALIZED EIGENVALUE PROBLEM ($Az = \lambda Bz$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.8.1 ASL_hcgraa, ASL_gcgraa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$A\boldsymbol{z} = \lambda B\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

ierr = ASL_hcgraa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

Single precision:

ierr = ASL_gcgraa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* / R*}	lna × n	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	ai	{D* / R*}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D* / R*}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D* / R*}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D* / R*}	n	Output	Eigenvalues λ
9	work	{D* / R*}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $1 \leq$ n $\leq$ lna , lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]$\leftarrow \frac{\text{a[0]}}{\text{b[0]}}$ and a[0]$\leftarrow \frac{1.0}{\sqrt{\text{b[0]}}}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.8.2 $\left\{\begin{array}{l}\text{ASL_hcgran}\\ \text{ASL_gcgran}\end{array}\right\}$ should be used if the eigenvectors are not needed.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem.

(b) Input data

n=4, lna=4, matrix A, lnb=4, nt=2 and matrix B.

(c) Main program

```
/*       C interface example for ASL_hcgraa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4,nt=4;
```

```
printf( "    *** ASL_hcgraa ***\n" );
printf( "\n     ** Input **\n\n" );
ar = ( double * )malloc(sizeof(double)*n*n);
if( ar == NULL )
{
    printf( "no enough memory for array ar\n");
    return -1;
}
br = ( double * )malloc(sizeof(double)*n*n);
if( br == NULL )
{
    printf( "no enough memory for array br\n");
    return -1;
}
ai = ( double * )malloc(sizeof(double)*n*n);
if( ai == NULL )
{
    printf( "no enough memory for array ai\n");
    return -1;
}
bi = ( double * )malloc(sizeof(double)*n*n);
if( bi == NULL )
{
    printf( "no enough memory for array bi\n");
    return -1;
}
e = ( double * )malloc(sizeof(double)*n);
if ( e == NULL )
{
    printf( "no enough memory for array e\n");
    return -1;
}
work = ( double * )malloc(sizeof(double)*4*n);
if ( work == NULL )
{
    printf( "no enough memory for array work\n");
    return -1;
}
printf( "\tn   = %6d\n" , n );
printf( "\tlna = %6d\n",lna );
printf( "\tlnb = %6d\n",lnb );
printf( "\tnt  = %6d\n",nt  );
br[0+n*0]=8.0;bi[0+n*0]=0.0;
br[1+n*1]=9.0;bi[1+n*1]=0.0;
br[2+n*2]=10.0;bi[2+n*2]=0.0;
br[3+n*3]=11.0;bi[3+n*3]=0.0;
br[0+n*1]=3.0;bi[0+n*1]=0.0;
br[0+n*2]=1.0;bi[0+n*2]=2.0;
br[0+n*3]=-1.0;bi[0+n*3]=2.0;
br[1+n*2]=1.0;bi[1+n*2]=-2.0;
br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
br[2+n*3]=-3.0;bi[2+n*3]=0.0;
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        br[i+n*j]= br[j+n*i];
        bi[i+n*j]=-bi[j+n*i];
    }
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        ar[i+n*j]= br[i+n*j];
        ai[i+n*j]=-bi[i+n*j];
    }
}
printf( "\n\tInput Matrix a\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)  ",ai[i+n*j]);
    }
    printf( "\n" );
}
printf( "\n\tInput Matrix b\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",br[i+n*j]);
        printf( "%8.3g)  ",bi[i+n*j]);
    }
    printf( "\n" );
}
```

```
        ierr = ASL_hcgraa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\t          Eigenvalue   " );
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "       %8.3g       ",e[j]);
        }
        printf( "\n\t          Eigenvector     " );
        for(i=0; i<n; i++)
        {
            printf( "\n\t" );
            for(j=0; j<n; j++)
            {
                printf( "(%8.3g,",ar[i+n*j]);
                printf( "%8.3g)   ",ai[i+n*j]);
            }
        }
        printf( "\n" );
        free(ar);
        free(br);
        free(ai);
        free(bi);
        free(e);
        free(work);
        return 0;
}
```

(d) Output results

```
*** ASL_hcgraa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
          0.231                    1                      1                  4.33
          Eigenvector
(   0.175,       0)   (       0,       0)   (   0.211,       0)   (   0.364,       0)
(   -0.16, 0.00182)   (-6.89e-19,-2.03e-16)   (   0.211,-1.71e-18)   (  -0.333,-0.00378)
(-0.00199,  -0.149)   (  -0.192, 0.00302)   ( -0.0313,-4.35e-17)   (-0.00414,    0.31)
( 0.00029,  -0.138)   (   0.192,-0.00302)   (  0.0313,-1.66e-16)   (0.000603,   0.287)
```

5.8.2 ASL_hcgran, ASL_gcgran All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $Az = \lambda Bz$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$A\boldsymbol{z} = \lambda B\boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:
  ierr = ASL_hcgran (ar, ai, lna, n, br, bi, lnb, e, work, nt);
Single precision:
  ierr = ASL_gcgran (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer, long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D*, R*}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D*, R*}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D*, R*}	n	Output	Eigenvalues λ
9	work	{D*, R*}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$, lnb

(b) $\mathrm{nt} \geq 1$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	$e[0] \leftarrow \frac{a[0]}{b[0]}$ is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.8.1 $\left\{\begin{array}{l}\text{ASL_hcgraa}\\\text{ASL_gcgraa}\end{array}\right\}$ should be used if the eigenvectors are needed.

5.9 GENERALIZED EIGENVALUE PROBLEM ($ABz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.9.1 ASL_hcgjaa, ASL_gcgjaa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{z} = \lambda \boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

ierr = ASL_hcgjaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

Single precision:

ierr = ASL_gcgjaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer, long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D*, R*}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D*, R*}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D*, R*}	n	Output	Eigenvalues λ
9	work	{D*, R*}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) $1 \leq$ n $\leq$ lna , lnb

(b) nt ≥ 1

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] and a[0]← $\frac{1.0}{\sqrt{b[0]}}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.9.2 $\left\{\begin{matrix}\text{ASL_hcgjan}\\ \text{ASL_gcgjan}\end{matrix}\right\}$ should be used if the eigenvectors are not needed.

(e) 5.10.1 $\left\{\begin{matrix}\text{ASL_hcgkaa}\\ \text{ASL_gcgkaa}\end{matrix}\right\}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem $ABx = \lambda x$.

(b) Input data

n=4, lna=4, matrix A, lnb=4, nt=2, matrix B.

(c) Main program

```
/*      C interface example for ASL_hcgjaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4,nt=4;
    printf( "    *** ASL_hcgjaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    ar = ( double * )malloc(sizeof(double)*n*n);
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n");
        return -1;
    }
    br = ( double * )malloc(sizeof(double)*n*n);
    if( br == NULL )
    {
        printf( "no enough memory for array br\n");
        return -1;
    }
    ai = ( double * )malloc(sizeof(double)*n*n);
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n");
        return -1;
    }
    bi = ( double * )malloc(sizeof(double)*n*n);
    if( bi == NULL )
```

```
{
    printf( "no enough memory for array bi\n");
    return -1;
}
e = ( double * )malloc(sizeof(double)*n);
if ( e == NULL )
{
    printf( "no enough memory for array e\n");
    return -1;
}
work = ( double * )malloc(sizeof(double)*4*n);
if ( work == NULL )
{
    printf( "no enough memory for array work\n");
    return -1;
}
printf( "\tn   = %6d\n" , n );
printf( "\tlna = %6d\n",lna );
printf( "\tlnb = %6d\n",lnb );
printf( "\tnt  = %6d\n",nt  );
br[0+n*0]=8.0;bi[0+n*0]=0.0;
br[1+n*1]=9.0;bi[1+n*1]=0.0;
br[2+n*2]=10.0;bi[2+n*2]=0.0;
br[3+n*3]=11.0,bi[3+n*3]=0.0;
br[0+n*1]=3.0;bi[0+n*1]=0.0;
br[0+n*2]=1.0;bi[0+n*2]=2.0;
br[0+n*3]=-1.0;bi[0+n*3]=2.0;
br[1+n*2]=1.0;bi[1+n*2]=-2.0;
br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
br[2+n*3]=-3.0;bi[2+n*3]=0.0;
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        br[i+n*j]= br[j+n*i];
        bi[i+n*j]=-bi[j+n*i];
    }
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        ar[i+n*j]= br[i+n*j];
        ai[i+n*j]=-bi[i+n*j];
    }
}
printf( "\n\tInput Matrix a\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)  ",ai[i+n*j]);
    }
    printf( "\n" );
}
printf( "\n\tInput Matrix b\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",br[i+n*j]);
        printf( "%8.3g)  ",bi[i+n*j]);
    }
    printf( "\n" );
}

ierr = ASL_hcgjaa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\t          Eigenvalue   " );
printf( "\n\t" );
for(j=0; j<n; j++)
{
    printf( "      %8.3g       ",e[j]);
}
printf( "\n\t          Eigenvector     " );
for(i=0; i<n; i++)
{
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)   ",ai[i+n*j]);
    }
```

```
    }
    printf( "\n" );
    free(ar);
    free(br);
    free(ai);
    free(bi);
    free(e);
    free(work);
    return 0;
}
```

(d) Output results

```
*** ASL_hcgjaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
           16.7                     37                    106                   218
          Eigenvector
(  -0.399,        0)   (  -0.109,        0)   (    0.17,        0)   (  0.0917,        0)
(   0.345,0.000936)   (   0.102,    0.016)   (   0.203, 0.00834)   (   0.101,-0.00404)
( 0.00726,   -0.132)   (  0.0187,   -0.327)   ( -0.0997,-0.00601)   (   0.147,-0.00401)
(-0.00421,   -0.125)   (  0.0156,   -0.281)   (    0.13,-0.000644)   (  -0.166, 0.00269)
```

5.9.2 ASL_hcgjan, ASL_gcgjan All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $ABz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$AB\boldsymbol{z} = \lambda \boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

ierr = ASL_hcgjan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

Single precision:

ierr = ASL_gcgjan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D∗ R∗}	lna × n	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	ai	{D∗ R∗}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D∗ R∗}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D∗ R∗}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D∗ R∗}	n	Output	Eigenvalues λ
9	work	{D∗ R∗}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq \text{n} \leq \text{lna}$, lnb

(b) $\text{nt} \geq 1$

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.9.1 {ASL_hcgjaa, ASL_gcgjaa} should be used if the eigenvectors are needed.

(d) 5.10.2 {ASL_hcgkan, ASL_gcgkan} should be used if matrix A is only positive.

5.10 GENERALIZED EIGENVALUE PROBLEM ($BAz = \lambda z$) FOR HERMITIAN MATRICES (TWO-DIMENSIONAL ARRAY TYPE) (UPPER TRIANGULAR TYPE) (REAL ARGUMENT TYPE)

5.10.1 ASL_hcgkaa, ASL_gcgkaa
All Eigenvalues and All Eigenvectors of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$BA\boldsymbol{z} = \lambda \boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ and corresponding all eigenvectors $\boldsymbol{z}$.

(2) **Usage**

Double precision:

ierr = ASL_hcgkaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

Single precision:

ierr = ASL_gcgkaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D* / R*}	lna × n	Input	Real part of Hermitian matrix A
				Output	Real part of eigenvector x
2	ai	{D* / R*}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Imaginary part of eigenvector x
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D* / R*}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D* / R*}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D* / R*}	n	Output	Eigenvalues λ
9	work	{D* / R*}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna , lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] and a[0]← $\sqrt{\text{b[0]}}$ are performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) Eigenvectors $\boldsymbol{v}_i$ are an orthonormal set so that $\boldsymbol{v}_j^* B^{-1} \boldsymbol{v}_k = \delta_{j,k}$

(d) 5.10.2 $\left\{\begin{array}{l}\text{ASL_hcgkan}\\ \text{ASL_gcgkan}\end{array}\right\}$ should be used if the eigenvectors are not needed.

(e) 5.9.1 $\left\{\begin{array}{l}\text{ASL_hcgjaa}\\ \text{ASL_gcgjaa}\end{array}\right\}$ should be used if matrix A is only positive.

(7) **Example**

(a) Problem

For Hermitian matrix of the degree 4

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

and its conjugate Hermitian matrix

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

obtain eigenvector of generalized eigenvalue problem $BAx = \lambda x$.

(b) Input data

n=4, lna=4, matrix A, lnb=4, nt=2 and matrix B.

(c) Main program

```
/*      C interface example for ASL_hcgkaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
```

```
n=4;
lna=4;lnb=4,nt=4;
printf( "    *** ASL_hcgkaa ***\n" );
printf( "\n     ** Input **\n\n" );
ar = ( double * )malloc(sizeof(double)*n*n);
if( ar == NULL )
{
    printf( "no enough memory for array ar\n");
    return -1;
}
br = ( double * )malloc(sizeof(double)*n*n);
if( br == NULL )
{
    printf( "no enough memory for array br\n");
    return -1;
}
ai = ( double * )malloc(sizeof(double)*n*n);
if( ai == NULL )
{
    printf( "no enough memory for array ai\n");
    return -1;
}
bi = ( double * )malloc(sizeof(double)*n*n);
if( bi == NULL )
{
    printf( "no enough memory for array bi\n");
    return -1;
}
e = ( double * )malloc(sizeof(double)*n);
if ( e == NULL )
{
    printf( "no enough memory for array e\n");
    return -1;
}
work = ( double * )malloc(sizeof(double)*4*n);
if ( work == NULL )
{
    printf( "no enough memory for array work\n");
    return -1;
}
printf( "\tn   = %6d\n" , n );
printf( "\tlna = %6d\n",lna );
printf( "\tlnb = %6d\n",lnb );
printf( "\tnt  = %6d\n",nt  );
br[0+n*0]=8.0;bi[0+n*0]=0.0;
br[1+n*1]=9.0;bi[1+n*1]=0.0;
br[2+n*2]=10.0;bi[2+n*2]=0.0;
br[3+n*3]=11.0,bi[3+n*3]=0.0;
br[0+n*1]=3.0;bi[0+n*1]=0.0;
br[0+n*2]=1.0;bi[0+n*2]=2.0;
br[0+n*3]=-1.0;bi[0+n*3]=2.0;
br[1+n*2]=1.0;bi[1+n*2]=-2.0;
br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
br[2+n*3]=-3.0;bi[2+n*3]=0.0;
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        br[i+n*j]= br[j+n*i];
        bi[i+n*j]=-bi[j+n*i];
    }
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        ar[i+n*j]= br[i+n*j];
        ai[i+n*j]=-bi[i+n*j];
    }
}
printf( "\n\tInput Matrix a\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)  ",ai[i+n*j]);
    }
    printf( "\n" );
}
printf( "\n\tInput Matrix b\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",br[i+n*j]);
        printf( "%8.3g)  ",bi[i+n*j]);
    }
```

```
        printf( "\n" );
    }

    ierr = ASL_hcgkaa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t          Eigenvalue   " );
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "       %8.3g       ",e[j]);
    }
    printf( "\n\t          Eigenvector     " );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "(%8.3g,",ar[i+n*j]);
            printf( "%8.3g)   ",ai[i+n*j]);
        }
    }
    printf( "\n" );
    free(ar);
    free(br);
    free(ai);
    free(bi);
    free(e);
    free(work);
    return 0;
}
```

(d) Output results

```
*** ASL_hcgkaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
           16.7                  37                  106                   218
          Eigenvector
(   -1.63, 0.00149)   (   0.656,  0.0713)   (    1.76,    0.08)   (    1.35, -0.0577)
(    1.41,-0.00511)   (  -0.625,  0.0299)   (    2.09, 0.00933)   (     1.5,-0.00416)
(  0.0301,    0.54)   (   0.102,   -1.99)   (   -1.03,  0.0151)   (    2.16, -0.0331)
( -0.0167,    0.51)   (  0.0905,   -1.71)   (    1.34,  0.0676)   (   -2.45,  0.0648)
```

5.10.2 ASL_hcgkan, ASL_gcgkan All Eigenvalues of Hermitian Matrices (Generalized Eigenvalue Problem $BAz = \lambda z$, B: Positive)

(1) **Function**

Generalized eigenvalue problem

$$B A \boldsymbol{z} = \lambda \boldsymbol{z}$$

(A: Hermitian, B: Positive Hermitian) is solved by using the Cholesky method, the Householder method and QR method to obtain all eigenvalues λ .

(2) **Usage**

Double precision:

ierr = ASL_hcgkan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

Single precision:

ierr = ASL_gcgkan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	ar	{D*, R*}	lna × n	Input	Real part of Hermitian matrix A
				Output	Input-time contents are not retained.
2	ai	{D*, R*}	lna × n	Input	Imaginary part of Hermitian matrix A
				Output	Input-time contents are not retained.
3	lna	I	1	Input	Adjustable dimension of arrays ar and ai
4	n	I	1	Input	Order of matrices A and B
5	br	{D*, R*}	lnb × n	Input	Real part of Hermitian matrix B
				Output	Input-time contents are not retained.
6	bi	{D*, R*}	lnb × n	Input	Imaginary part of Hermitian matrix B
				Output	Input-time contents are not retained.
7	lnb	I	1	Input	Adjustable dimension of arrays br and bi
8	e	{D*, R*}	n	Output	Eigenvalues λ
9	work	{D*, R*}	4×n	Work	Work area
10	nt	I	1	Input	Number of tasks
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $1 \leq$ n $\leq$ lna , lnb

(b) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	e[0]← a[0]*b[0] is performed.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
4000	B was not positive definite.	
5000	The sequence did not converge in the step where the eigenvalue was obtained.	

(6) **Notes**

(a) Arrays ar, ai, br and bi should be stored only in the upper triangular portions.

(b) Eigenvalues are stored in ascending order.

(c) 5.10.1 {ASL_hcgkaa, ASL_gcgkaa} should be used if the eigenvectors are needed.

(d) 5.9.2 {ASL_hcgjan, ASL_gcgjan} should be used if matrix A is only positive.

Chapter 6

FOURIER TRANSFORMS AND THEIR APPLICATIONS

6.1 INTRODUCTION

This chapter describes functions that perform fast Fourier transforms, convolutions, correlations, power spectrum analysis.

Function described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.

The following functions are provided for computing the discrete Fourier transform according to the data characteristics. You can perform processing efficiently if your data satisfies any of the characteristics described below.

(1) Multiple one-dimensional complex Fourier transform
Data values are complex numbers and are multiple one-dimensional data.

(2) Multiple one-dimensional real Fourier transform
Data values are real numbers and are multiple one-dimensional data.

(3) Two-dimensional complex Fourier transform
Data values are complex numbers and are two-dimensional data.

(4) Two-dimensional real Fourier transform
Data values are real numbers and are two-dimensional data.

(5) Three-dimensional complex Fourier transform
Data values are complex numbers and are three-dimensional data.

(6) Three-dimensional real Fourier transform
Data values are real numbers and are three-dimensional data.

In addition, although the number of equal divisions n of the input data can be transformed by the fast Fourier transforms handled in this chapter no matter what prime number is used as radix, calculation efficiency decreases for a sequence formed from a large prime number. Therefore, the number of equal divisions n should be able to be factored into small radices (2, 3, 5, 7).

The functions, which handles the following data, are provided for performing the convolutions, correlations, power spectrum analysis.

(1) Two-dimensional data

(2) Three-dimensional data

6.1.1 Notes

(1) For a two-dimensional complex Fourier transform or two-dimensional real Fourier transform, the number of sample points nx, ny (nx, ny, nz for a three-dimensional complex Fourier transform or three-dimensional real Fourier transform) corresponds to a sample on a single period $(0, 2\pi)$.

(2) You first must use a transform function that includes initialization. This function performs such tasks as creating a trigonometric function table. If you plan to continue computing Fourier transforms for the same number of data N, processing will be more efficient if you use the post-initialization transform function thereafter since the contents of work areas are retained.

6.1.2 Algorithms Used

6.1.2.1 Two-dimensional complex Fourier transform

The complex Fourier forward transform C_{j_x,j_y} for one period c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ of complex multiperiodic data $\hat{c}_{k_x,k_y}$ satisfying $\hat{c}_{k_x,k_y} = \hat{c}_{k_x+n_x,k_y+n_y}$ for arbitrary integers k_x and k_y is defined by the following equation.

$$
\begin{aligned}
C_{j_x,j_y} &= \frac{1}{\alpha}\sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\
&(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)
\end{aligned}
$$

α is an arbitrary constant for which 1 or $n_x n_y$ normally is selected. At this time, the complex data after the transform C_{j_x,j_y} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ also corresponds to one period of complex multiperiodic data $\hat{C}_{j_x,j_y}$ satisfying $\hat{C}_{j_x,j_y} = \hat{C}_{j_x+n_x,j_y+n_y}$ for an arbitrary integers j_x and j_y. The corresponding backward transform is as follows:

$$
\begin{aligned}
c_{k_x,k_y} &= \frac{1}{n_x n_y}\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1} (\alpha C_{j_x,j_y}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\
&(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)
\end{aligned}
$$

The functions in this manual obtain $\alpha C_{j_x,j_y}$ and $n_x n_y c_{k_x,k_y}$ except for a constant factor, from the normal Fourier transform definition.

6.1.2.2 Two-dimensional real Fourier transform

When the data for which the two-dimensional Fourier forward transform is to be calculated is real, the following relationship is satisfied.

$$
\begin{aligned}
\alpha C^*_{n_x-j_x,n_y-j_y} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c^*_{k_x,k_y} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}\}} \\
&= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}\}} \\
&= \alpha C_{j_x,j_y}
\end{aligned}
$$

Here, z^* represents the conjugate complex number of the complex number z. In particular, $C_{0,0}$ is real and when n_x and n_y are even, then $C_{\frac{n_x}{2},\frac{n_y}{2}}$ is also real.
Similarly, the following relationship is satisfied.

$$
\begin{aligned}
\alpha C^*_{n_x-j_x,j_y} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c^*_{k_x,k_y} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{j_y k_y}{n_y}\}} \\
&= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}\}} \\
&= \alpha C_{j_x,n_y-j_y}
\end{aligned}
$$

Therefore, the calculations for a Fourier transform can be executed using half the data for the case of general complex data (for c_{k_x,k_y}, only the real part, and for C_{j_x,j_y}, half of its period for either j_x or j_y).

6.1.2.3 Three-dimensional complex Fourier transform

The complex Fourier forward transform C_{j_x,j_y,j_z} for one period c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ of complex multiperiodic data $\hat{c}_{k_x,k_y,k_z}$ satisfying $\hat{c}_{k_x,k_y,k_z} = \hat{c}_{k_x+n_x,k_y+n_y,k_z+n_z}$ for arbitrary integers k_x, k_y and k_z is defined by the following equation.

$$
\begin{aligned}
C_{j_x,j_y,j_z} &= \frac{1}{\alpha} \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})} \\
&\quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)
\end{aligned}
$$

α is an arbitrary constant for which 1 or $n_x n_y n_z$ normally is selected. At this time, the complex data after the transform C_{j_x,j_y,j_z} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ also corresponds to one period of complex multiperiodic data $\hat{C}_{j_x,j_y,j_z}$ satisfying $\hat{C}_{j_x,j_y,j_z} = \hat{C}_{j_x+n_x,j_y+n_y,j_z+n_z}$ for an arbitrary integers j_x, j_y and j_z. The corresponding backward transform is as follows:

$$
\begin{aligned}
c_{k_x,k_y,k_z} &= \frac{1}{n_x n_y n_z} \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \sum_{j_z=0}^{n_z-1} (\alpha C_{j_x,j_y,j_z}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})} \\
&\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)
\end{aligned}
$$

The functions in this manual obtain $\alpha C_{j_x,j_y,j_z}$ and $n_x n_y n_z c_{k_x,k_y,k_z}$ except for a constant factor, from the normal Fourier transform definition.

6.1.2.4 Three-dimensional real Fourier transform

When the data for which the three-dimensional Fourier forward transform is to be calculated is real, the following relationship is satisfied.

$$
\begin{aligned}
\alpha C^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c^*_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x} + \frac{(n_y-j_y)k_y}{n_y} + \frac{(n_z-j_z)k_z}{n_z}\}} \\
&= \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}\{\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z}\}} \\
&= \alpha C_{j_x,j_y,j_z}
\end{aligned}
$$

Here, z^* represents the conjugate complex number of the complex number z. In particular, $C_{0,0,0}$ is real and when n_x, n_y and n_z are all even, then $C_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}$ is also real.
Similarly, the following kinds of relationships are satisfied.

$$
C^*_{n_x-j_x,j_y,j_z} = C_{j_x,n_y-j_y,n_z-j_z}
$$

That is, the complex number after substitutions are made in the subscripts using the following correspondence relationships for all of j_x, j_y and j_z is mutually related as a complex conjugate to the complex number before the substitutions are made.

$$
\begin{aligned}
j_x &\leftrightarrow n_x - j_x \\
j_y &\leftrightarrow n_y - j_y \\
j_z &\leftrightarrow n_z - j_z
\end{aligned}
$$

Therefore, the calculations for a Fourier transform can be executed using half the data for the case of general complex data (for c_{k_x,k_y,k_z}, only the real part, and for C_{j_x,j_y,j_z}, half of its period for either j_x, j_y, or j_z).

6.1.3 Reference Bibliography

(1) Pease, M. C, “An Adaption of the Fast Fourier Transform for Parallel Processing”, J. Assn. Comput. Mach., 15, 252(1968).

(2) Stockham, T. G. , “High Speed Convolution and Correlation”, AFIPS Conf. Proc. , 28, 229(1966).

(3) Swarztrauber, P. N. , “Vectorizing the FFTs”, Parallel Computations, 51(1982).

(4) Singleton, R. C. , “An Algorithm for Computing the Mixed Radix Fast Fourier Transform”, IEEE Trans. Audio and Electroacoust. , AU-17, 93(1969).

(5) Singleton, R. C. , “ALGOL Procedures for the Fast Fourier Transform”, Commun. ACM, 11, 773(1968).

(6) Petersen, W. P. , “Vector Fortran for Numerical Problems on CRAY-1”, Commun. ACM, 26, 1008(1983).

(7) Brigham, E. O. , “The Fast Fourier Transform”, Prentice-Hall Inc. , 1974.

(8) Temperton, C. , “Implementation of a Self-Sorting In-Place Prime-Factor FFT Algorithm”, J. Comp. Phys., 58, 283(1985).

(9) Temperton, C. , “Self-Sorting Mixed-Radix Fast Fourier Transform”, J. Comp. Phys., 52, 1(1983).

(10) Temperton, C. , “Fast Mixed-Radix Real Fourier Transforms”, J. Comp. Phys., 52, 340(1983)

(11) Willemstein, T. , “Two-dimensional Fourier Transforms on the Cray-1S”, Supercomputer, 10(1985)

6.2 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.2.1 [DEPRECATED]ASL_qfcmfb, ASL_pfcmfb Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

ASL_qfcmfb or ASL_pfcmfb computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

ASL_qfcmfb or ASL_pfcmfb computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

ierr = ASL_qfcmfb (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfcmfb (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	cr	{D* / R*}	See Contents	Input	Real part of input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Real part of output data $d_{j,l}$ (See Notes (b) and (c))
4	ci	{D* / R*}	See Contents	Input	Imaginary part of input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Imaginary part of output data $d_{j,l}$ (See Notes (b) and (c))
5	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
6	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
7	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
8	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
9	trigs	{D* / R*}	2 × n	Output	Trigonometric function table (See Note (d))
10	wk	{D* / R*}	2 × m × n	Work	Work area
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n $\geq$ 1, m $\geq$ 1

(b) incn $\geq$ 1, incm $\geq$ 1

(c) incn $\geq$ m $\times$ gcm (incn, incm) or
incm $\geq$ n $\times$ gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw $\in \{0, 1, -1\}$

(e) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of transformed data n can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting n = 289(=17^2), it is usually more efficient to set n = 300(=$2^2 \times 3 \times 5^2$), n = 320(=$2^6 \times 5$), n = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be $\Re\{c_{k,l}\}$ and $\Im\{c_{k,l}\}$, respectively, the $c_{k,l}$ and elements of arrays cr and ci are associated as follows.

$$\Re\{c_{k,l}\} \leftrightarrow \mathrm{cr[incn * k + incm * (l-1)]}$$
$$\Im\{c_{k,l}\} \leftrightarrow \mathrm{ci[incn * k + incm * (l-1)]}$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \mathrm{cr[k + n * (l-1)]}, \quad \Im\{c_{k,l}\} \leftrightarrow \mathrm{ci[k + n * (l-1)]}$$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \mathrm{cr[(l-1) + m * k]}, \quad \Im\{c_{k,l}\} \leftrightarrow \mathrm{ci[(l-1) + m * k]}$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of arrays cr and ci is not stored do not change when this function is called.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform

immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data n, you should call this function once, and then use the after-initialization transform 6.2.2 $\left\{\begin{matrix}\text{ASL_qfcmbf}\\ \text{ASL_pfcmbf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.2.2 $\left\{\begin{matrix}\text{ASL_qfcmbf}\\ \text{ASL_pfcmbf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for arrays cr and ci.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.2.2 (7).

6.2.2 [DEPRECATED]ASL_qfcmbf, ASL_pfcmbf Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_qfcmbf or ASL_pfcmbf computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

ASL_qfcmbf or ASL_pfcmbf computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

ierr = ASL_qfcmbf (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfcmbf (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer ; long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	cr	{D* / R*}	See Contents	Input	Real part of input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Real part of output data $d_{j,l}$ (See Notes (b) and (c))
4	ci	{D* / R*}	See Contents	Input	Imaginary part of input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Imaginary part of output data $d_{j,l}$ (See Notes (b) and (c))
5	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
6	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
7	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
8	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
9	trigs	{D* / R*}	2 × n	Input	Trigonometric function table (See Note (a))
10	wk	{D* / R*}	2×m×n	Work	Work area
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) or
incm ≥ n × gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw ∈ {1, −1}

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of transformed data n after the including-initialization function 6.2.1 $\left\{\begin{array}{l}\text{ASL_qfcmfb}\\ \text{ASL_pfcmfb}\end{array}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) If we let the real and imaginary parts of the complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be $\Re\{c_{k,l}\}$ and $\Im\{c_{k,l}\}$, respectively, the $c_{k,l}$ and elements of arrays cr and ci are associated as follows.

$$\begin{array}{lcl}\Re\{c_{k,l}\} & \leftrightarrow & \mathrm{cr[incn * k + incm * (l-1)]}\\ \Im\{c_{k,l}\} & \leftrightarrow & \mathrm{ci[incn * k + incm * (l-1)]}\end{array}$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \mathrm{cr[k + n * (l-1)]},\quad \Im\{c_{k,l}\} \leftrightarrow \mathrm{ci[k + n * (l-1)]}$$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$$\Re\{c_{k,l}\} \leftrightarrow \mathrm{cr[(l-1) + m * k]},\quad \Im\{c_{k,l}\} \leftrightarrow \mathrm{ci[(l-1) + m * k]}$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of arrays cr and ci is not stored do not change when this function is called.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform

immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional complex Fourier transform using the following sequence of numbers as input data.

cr[0]= 1.000 ci[0]=4.000
cr[1]= 2.000 ci[1]=3.000
cr[2]= 3.000 ci[2]=2.000
cr[3]= 4.000 ci[3]=1.000
cr[4]= 4.000 ci[4]=1.000
cr[5]= 3.000 ci[5]=2.000
cr[6]= 2.000 ci[6]=3.000
cr[7]= 1.000 ci[7]=4.000
cr[9]= 1.000 ci[9]=2.000
cr[10]= 1.000 ci[10]=2.000
cr[11]= 2.000 ci[11]=1.000
cr[12]= 2.000 ci[12]=1.000
cr[13]= 2.000 ci[13]=1.000
cr[14]= 2.000 ci[14]=1.000
cr[15]= 1.000 ci[15]=2.000
cr[16]= 1.000 ci[16]=2.000
cr[18]= 1.000 ci[18]=2.000
cr[19]= 1.000 ci[19]=2.000
cr[20]= 1.000 ci[20]=2.000
cr[21]= 1.000 ci[21]=2.000
cr[22]= 2.000 ci[22]=1.000
cr[23]= 2.000 ci[23]=1.000

cr[24]= 2.000 ci[24]=1.000
cr[25]= 2.000 ci[25]=1.000
cr[27]= 1.000 ci[27]=1.000
cr[28]= 1.000 ci[28]=1.000
cr[29]= 1.000 ci[29]=1.000
cr[30]= 1.000 ci[30]=1.000
cr[31]= 1.000 ci[31]=1.000
cr[32]= 1.000 ci[32]=1.000
cr[33]= 1.000 ci[33]=1.000
cr[34]= 1.000 ci[34]=1.000

(b) Input data

Array cr and ci, n=8, m=4, incn=1, incm=9, isw=1 (forward transform), isw=−1 (backward transform) and nt=2.

(c) Main program

```
/*      C interface example for ASL_qfcmfb , ASL_qfcmbf */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int ld=35;
    int n;      int m;
    double *cr; double *ci;
    int incn;   int incm;
    int isw;
    int ifax[20];   double *trigs;
    double *wk;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "qfcmbf.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_qfcmfb , ASL_qfcmbf ***\n" );
    printf( "\n     ** Input **\n\n" );

    cr = ( double * )malloc((size_t)( sizeof(double) * ld ));
    if( cr == NULL )
    {
        printf( "no enough memory for array cr\n" );
        return -1;
    }

    ci = ( double * )malloc((size_t)( sizeof(double) * ld ));
    if( ci == NULL )
    {
        printf( "no enough memory for array ci\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*ld) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (2*ld) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    fscanf( fp, "%d,%d,%d,%d,%d", &n, &m, &incn, &incm, &nt );

    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
```

```
        {
            fscanf( fp, "%lf,%lf", &cr[i*incn+j*incm], &ci[i*incn+j*incm] );
        }
    }

    printf("\t   n    = %d \n", n );
    printf("\t   m    = %d \n", m );
    printf("\t   incn = %d \n", incn );
    printf("\t   incm = %d \n", incm );
    printf("\t   nt   = %d \n\n", nt );

    printf( "\t  Real Part                  Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n",
                i*incn+j*incm, cr[i*incn+j*incm],
                i*incn+j*incm, ci[i*incn+j*incm] );
        }
    }

    fclose( fp );


    printf( "\n     ** Output **\n" );

    isw = 1;
    ierr = ASL_qfcmfb(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            cr[i*incn+j*incm] /= n ;
            ci[i*incn+j*incm] /= n ;
        }
    }

    printf( "\n\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n",
                i*incn+j*incm, cr[i*incn+j*incm],
                i*incn+j*incm, ci[i*incn+j*incm] );
        }
    }

    isw = -1;
    ierr = ASL_qfcmbf(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

    printf( "\n\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n",
                i*incn+j*incm, cr[i*incn+j*incm],
                i*incn+j*incm, ci[i*incn+j*incm] );
        }
    }


    free( cr );
    free( ci );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
    *** ASL_qfcmfb , ASL_qfcmbf ***

     ** Input **
```

```
  n    = 8
  m    = 4
  incn = 1
  incm = 9
  nt   = 2

 Real Part                     Imaginary Part
 cr[  0] =        1            ci[  0] =        4
 cr[  1] =        2            ci[  1] =        3
 cr[  2] =        3            ci[  2] =        2
 cr[  3] =        4            ci[  3] =        1
 cr[  4] =        4            ci[  4] =        1
 cr[  5] =        3            ci[  5] =        2
 cr[  6] =        2            ci[  6] =        3
 cr[  7] =        1            ci[  7] =        4
 cr[  9] =        1            ci[  9] =        2
 cr[ 10] =        1            ci[ 10] =        2
 cr[ 11] =        2            ci[ 11] =        1
 cr[ 12] =        2            ci[ 12] =        1
 cr[ 13] =        2            ci[ 13] =        1
 cr[ 14] =        2            ci[ 14] =        1
 cr[ 15] =        1            ci[ 15] =        2
 cr[ 16] =        1            ci[ 16] =        2
 cr[ 18] =        1            ci[ 18] =        2
 cr[ 19] =        1            ci[ 19] =        2
 cr[ 20] =        1            ci[ 20] =        2
 cr[ 21] =        1            ci[ 21] =        2
 cr[ 22] =        2            ci[ 22] =        1
 cr[ 23] =        2            ci[ 23] =        1
 cr[ 24] =        2            ci[ 24] =        1
 cr[ 25] =        2            ci[ 25] =        1
 cr[ 27] =        1            ci[ 27] =        1
 cr[ 28] =        1            ci[ 28] =        1
 cr[ 29] =        1            ci[ 29] =        1
 cr[ 30] =        1            ci[ 30] =        1
 cr[ 31] =        1            ci[ 31] =        1
 cr[ 32] =        1            ci[ 32] =        1
 cr[ 33] =        1            ci[ 33] =        1
 cr[ 34] =        1            ci[ 34] =        1

 ** Output **

< Forward Transform >
ierr =      0

Solution

 Real Part                     Imaginary Part
 cr[  0] =      2.5            ci[  0] =      2.5
 cr[  1] =    -1.03            ci[  1] =    0.427
 cr[  2] =        0            ci[  2] =        0
 cr[  3] =  -0.0732            ci[  3] =  -0.0303
 cr[  4] =        0            ci[  4] =        0
 cr[  5] =   0.0303            ci[  5] =   0.0732
 cr[  6] =        0            ci[  6] =        0
 cr[  7] =   -0.427            ci[  7] =     1.03
 cr[  9] =      1.5            ci[  9] =      1.5
 cr[ 10] =   -0.427            ci[ 10] =    0.177
 cr[ 11] =        0            ci[ 11] =        0
 cr[ 12] =    0.177            ci[ 12] =   0.0732
 cr[ 13] =        0            ci[ 13] =        0
 cr[ 14] =  -0.0732            ci[ 14] =   -0.177
 cr[ 15] =        0            ci[ 15] =        0
 cr[ 16] =   -0.177            ci[ 16] =    0.427
 cr[ 18] =      1.5            ci[ 18] =      1.5
 cr[ 19] =    0.177            ci[ 19] =    0.427
 cr[ 20] =        0            ci[ 20] =        0
 cr[ 21] =  -0.0732            ci[ 21] =    0.177
 cr[ 22] =        0            ci[ 22] =        0
 cr[ 23] =   -0.177            ci[ 23] =   0.0732
 cr[ 24] =        0            ci[ 24] =        0
 cr[ 25] =   -0.427            ci[ 25] =   -0.177
 cr[ 27] =        1            ci[ 27] =        1
 cr[ 28] =        0            ci[ 28] =        0
 cr[ 29] =        0            ci[ 29] =        0
 cr[ 30] =        0            ci[ 30] =        0
 cr[ 31] =        0            ci[ 31] =        0
 cr[ 32] =        0            ci[ 32] =        0
 cr[ 33] =        0            ci[ 33] =        0
 cr[ 34] =        0            ci[ 34] =        0

< Backward Transform >
ierr =      0

Solution

 Real Part                     Imaginary Part
 cr[  0] =        1            ci[  0] =        4
 cr[  1] =        2            ci[  1] =        3
 cr[  2] =        3            ci[  2] =        2
 cr[  3] =        4            ci[  3] =        1
 cr[  4] =        4            ci[  4] =        1
 cr[  5] =        3            ci[  5] =        2
 cr[  6] =        2            ci[  6] =        3
```

```
cr[  7] =        1          ci[  7] =        4
cr[  9] =        1          ci[  9] =        2
cr[ 10] =        1          ci[ 10] =        2
cr[ 11] =        2          ci[ 11] =        1
cr[ 12] =        2          ci[ 12] =        1
cr[ 13] =        2          ci[ 13] =        1
cr[ 14] =        2          ci[ 14] =        1
cr[ 15] =        1          ci[ 15] =        2
cr[ 16] =        1          ci[ 16] =        2
cr[ 18] =        1          ci[ 18] =        2
cr[ 19] =        1          ci[ 19] =        2
cr[ 20] =        1          ci[ 20] =        2
cr[ 21] =        1          ci[ 21] =        2
cr[ 22] =        2          ci[ 22] =        1
cr[ 23] =        2          ci[ 23] =        1
cr[ 24] =        2          ci[ 24] =        1
cr[ 25] =        2          ci[ 25] =        1
cr[ 27] =        1          ci[ 27] =        1
cr[ 28] =        1          ci[ 28] =        1
cr[ 29] =        1          ci[ 29] =        1
cr[ 30] =        1          ci[ 30] =        1
cr[ 31] =        1          ci[ 31] =        1
cr[ 32] =        1          ci[ 32] =        1
cr[ 33] =        1          ci[ 33] =        1
cr[ 34] =        1          ci[ 34] =        1
```

6.3 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.3.1 [DEPRECATED]ASL_hfcmfb, ASL_gfcmfb Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

ASL_hfcmfb or ASL_gfcmfb computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

ASL_hfcmfb or ASL_gfcmfb computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

ierr = ASL_hfcmfb (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfcmfb (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer; long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	c	{Z∗ C∗}	See Contents	Input	Input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Output data $d_{j,l}$ (See Notes (b) and (c))
4	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
8	trigs	{D∗ R∗}	2 × n	Output	Trigonometric function table (See Note (d))
9	wk	{Z∗ C∗}	m×n	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) or
incm ≥ n × gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw $\in \{0, 1, -1\}$

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of transformed data n can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting n = 289(=17^2), it is usually more efficient to set n = 300(=$2^2 \times 3 \times 5^2$), n = 320(=$2^6 \times 5$), n = 384(=$2^7 \times 3$) or the like.

(b) The complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array c are associated as follows.

$$c_{k,l} \leftrightarrow \mathrm{c[incn * k + incm * (l - 1)]}$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[k + n * (l - 1)]}$$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[(l - 1) + m * k]}$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of array c is not stored do not change when this function is called.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data n, you should call this function once, and then use the after-initialization transform 6.3.2 $\left\{\begin{array}{l}\text{ASL_hfcmbf}\\ \text{ASL_gfcmbf}\end{array}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and

trigs so they can be used as input to the function 6.3.2 $\left\{\begin{matrix}\text{ASL_hfcmbf}\\ \text{ASL_gfcmbf}\end{matrix}\right\}$.

To perform initialization only by setting isw=0, you need not set input data for array c.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.3.2 (7).

6.3.2 [DEPRECATED]ASL_hfcmbf, ASL_gfcmbf Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_hfcmbf or ASL_gfcmbf computes the m-fold one-dimensional complex Fourier forward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Backward transform

ASL_hfcmbf or ASL_gfcmbf computes the m-fold one-dimensional complex Fourier backward transform (arbitrary radix) for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **Usage**

Double precision:

ierr = ASL_hfcmbf (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfcmbf (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	c	{Z* / C*}	See Contents	Input	Input data $c_{k,l}$ (See Note (b)) **Size:** incn × (n − 1) + incm × (m − 1) + 1
				Output	Output data $d_{j,l}$ (See Notes (b) and (c))
4	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
8	trigs	{D* / R*}	2 × n	Input	Trigonometric function table (See Note (a))
9	wk	{Z* / C*}	m×n	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) or
incm ≥ n × gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw ∈ {1, −1}

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n=1.	Input data is output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of transformed data n after the including-initialization function 6.3.1 $\left\{\begin{array}{l}\text{ASL_hfcmfb}\\ \text{ASL_gfcmfb}\end{array}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The complex data $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array c are associated as follows.

$$c_{k,l} \leftrightarrow \mathrm{c[incn * k + incm * (l - 1)]}$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[k + n * (l - 1)]}$$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[(l - 1) + m * k]}$$

and the data is stored so that it is packed consecutively for subscript l. Similarly, for the complex data $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$. Values in areas where the data of array c is not stored do not change when this function is called.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that

is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional complex Fourier transform using the following sequence of numbers as input data.

c[0]= (1.000, 4.000)
c[1]= (2.000, 3.000)
c[2]= (3.000, 2.000)
c[3]= (4.000, 1.000)
c[4]= (4.000, 1.000)
c[5]= (3.000, 2.000)
c[6]= (2.000, 3.000)
c[7]= (1.000, 4.000)
c[9]= (1.000, 2.000)
c[10]= (1.000, 2.000)
c[11]= (2.000, 1.000)
c[12]= (2.000, 1.000)
c[13]= (2.000, 1.000)
c[14]= (2.000, 1.000)
c[15]= (1.000, 2.000)
c[16]= (1.000, 2.000)
c[18]= (1.000, 2.000)
c[19]= (1.000, 2.000)
c[20]= (1.000, 2.000)
c[21]= (1.000, 2.000)
c[22]= (2.000, 1.000)
c[23]= (2.000, 1.000)
c[24]= (2.000, 1.000)
c[25]= (2.000, 1.000)
c[27]= (1.000, 1.000)
c[28]= (1.000, 1.000)
c[29]= (1.000, 1.000)
c[30]= (1.000, 1.000)
c[31]= (1.000, 1.000)
c[32]= (1.000, 1.000)
c[33]= (1.000, 1.000)
c[34]= (1.000, 1.000)

(b) Input data

Array c, n=8, m=4, incn=1, incm=9, isw=1 (forward transform), isw=−1 (backward transform) and

nt=2.

(c) Main program

```
/*      C interface example for ASL_hfcmfb , ASL_hfcmbf */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>

int main()
{
    int ld=35;
    int n;      int m;
    double _Complex *c;
    int incn;   int incm;
    int isw;
    int ifax[20];   double *trigs;
    double _Complex *wk;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "hfcmbf.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_hfcmfb , ASL_hfcmbf ***\n" );
    printf( "\n     ** Input **\n\n" );

    c = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * ld ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*ld) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * ld ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    fscanf( fp, "%d,%d,%d,%d,%d", &n, &m, &incn, &incm, &nt );

    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 );
            c[i*incn+j*incm] = tmp_re + tmp_im * _Complex_I;
        }
    }

    printf("\t   n    = %d \n", n );
    printf("\t   m    = %d \n", m );
    printf("\t   incn = %d \n", incn );
    printf("\t   incm = %d \n", incm );
    printf("\t   nt   = %d \n\n", nt );

    printf( "\t  Real Part                   Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  creal(c[%3d]) = %8.3g         cimag(c[%3d]) = %8.3g\n",
                i*incn+j*incm, creal(c[i*incn+j*incm]),
                i*incn+j*incm, cimag(c[i*incn+j*incm]) );
        }
    }

    fclose( fp );


    printf( "\n     ** Output **\n" );
```

```
    isw = 1;
    ierr = ASL_hfcmfb(n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            c[i*incn+j*incm] /= n ;
        }
    }

    printf( "\n\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  creal(c[%3d]) = %8.3g        cimag(c[%3d]) = %8.3g\n",
                i*incn+j*incm, creal(c[i*incn+j*incm]),
                i*incn+j*incm, cimag(c[i*incn+j*incm]) );
        }
    }

    isw = -1;
    ierr = ASL_hfcmbf(n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

    printf( "\n\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t  creal(c[%3d]) = %8.3g        cimag(c[%3d]) = %8.3g\n",
                i*incn+j*incm, creal(c[i*incn+j*incm]),
                i*incn+j*incm, cimag(c[i*incn+j*incm]) );
        }
    }


    free( c );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_hfcmfb , ASL_hfcmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 9
   nt   = 2

  Real Part                  Imaginary Part
  creal(c[  0]) =        1        cimag(c[  0]) =        4
  creal(c[  1]) =        2        cimag(c[  1]) =        3
  creal(c[  2]) =        3        cimag(c[  2]) =        2
  creal(c[  3]) =        4        cimag(c[  3]) =        1
  creal(c[  4]) =        4        cimag(c[  4]) =        1
  creal(c[  5]) =        3        cimag(c[  5]) =        2
  creal(c[  6]) =        2        cimag(c[  6]) =        3
  creal(c[  7]) =        1        cimag(c[  7]) =        4
  creal(c[  9]) =        1        cimag(c[  9]) =        2
  creal(c[ 10]) =        1        cimag(c[ 10]) =        2
  creal(c[ 11]) =        2        cimag(c[ 11]) =        1
  creal(c[ 12]) =        2        cimag(c[ 12]) =        1
  creal(c[ 13]) =        2        cimag(c[ 13]) =        1
  creal(c[ 14]) =        2        cimag(c[ 14]) =        1
  creal(c[ 15]) =        1        cimag(c[ 15]) =        2
  creal(c[ 16]) =        1        cimag(c[ 16]) =        2
  creal(c[ 18]) =        1        cimag(c[ 18]) =        2
  creal(c[ 19]) =        1        cimag(c[ 19]) =        2
  creal(c[ 20]) =        1        cimag(c[ 20]) =        2
  creal(c[ 21]) =        1        cimag(c[ 21]) =        2
```

```
  creal(c[ 22]) =        2        cimag(c[ 22]) =        1
  creal(c[ 23]) =        2        cimag(c[ 23]) =        1
  creal(c[ 24]) =        2        cimag(c[ 24]) =        1
  creal(c[ 25]) =        2        cimag(c[ 25]) =        1
  creal(c[ 27]) =        1        cimag(c[ 27]) =        1
  creal(c[ 28]) =        1        cimag(c[ 28]) =        1
  creal(c[ 29]) =        1        cimag(c[ 29]) =        1
  creal(c[ 30]) =        1        cimag(c[ 30]) =        1
  creal(c[ 31]) =        1        cimag(c[ 31]) =        1
  creal(c[ 32]) =        1        cimag(c[ 32]) =        1
  creal(c[ 33]) =        1        cimag(c[ 33]) =        1
  creal(c[ 34]) =        1        cimag(c[ 34]) =        1

 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                    Imaginary Part
  creal(c[  0]) =      2.5        cimag(c[  0]) =      2.5
  creal(c[  1]) =    -1.03        cimag(c[  1]) =    0.427
  creal(c[  2]) =        0        cimag(c[  2]) =        0
  creal(c[  3]) =  -0.0732        cimag(c[  3]) =  -0.0303
  creal(c[  4]) =        0        cimag(c[  4]) =        0
  creal(c[  5]) =   0.0303        cimag(c[  5]) =   0.0732
  creal(c[  6]) =        0        cimag(c[  6]) =        0
  creal(c[  7]) =   -0.427        cimag(c[  7]) =     1.03
  creal(c[  9]) =      1.5        cimag(c[  9]) =      1.5
  creal(c[ 10]) =   -0.427        cimag(c[ 10]) =    0.177
  creal(c[ 11]) =        0        cimag(c[ 11]) =        0
  creal(c[ 12]) =    0.177        cimag(c[ 12]) =   0.0732
  creal(c[ 13]) =        0        cimag(c[ 13]) =        0
  creal(c[ 14]) =  -0.0732        cimag(c[ 14]) =   -0.177
  creal(c[ 15]) =        0        cimag(c[ 15]) =        0
  creal(c[ 16]) =   -0.177        cimag(c[ 16]) =    0.427
  creal(c[ 18]) =      1.5        cimag(c[ 18]) =      1.5
  creal(c[ 19]) =    0.177        cimag(c[ 19]) =    0.427
  creal(c[ 20]) =        0        cimag(c[ 20]) =        0
  creal(c[ 21]) =  -0.0732        cimag(c[ 21]) =    0.177
  creal(c[ 22]) =        0        cimag(c[ 22]) =        0
  creal(c[ 23]) =   -0.177        cimag(c[ 23]) =   0.0732
  creal(c[ 24]) =        0        cimag(c[ 24]) =        0
  creal(c[ 25]) =   -0.427        cimag(c[ 25]) =   -0.177
  creal(c[ 27]) =        1        cimag(c[ 27]) =        1
  creal(c[ 28]) =        0        cimag(c[ 28]) =        0
  creal(c[ 29]) =        0        cimag(c[ 29]) =        0
  creal(c[ 30]) =        0        cimag(c[ 30]) =        0
  creal(c[ 31]) =        0        cimag(c[ 31]) =        0
  creal(c[ 32]) =        0        cimag(c[ 32]) =        0
  creal(c[ 33]) =        0        cimag(c[ 33]) =        0
  creal(c[ 34]) =        0        cimag(c[ 34]) =        0

< Backward Transform >
ierr =      0

Solution

  Real Part                    Imaginary Part
  creal(c[  0]) =        1        cimag(c[  0]) =        4
  creal(c[  1]) =        2        cimag(c[  1]) =        3
  creal(c[  2]) =        3        cimag(c[  2]) =        2
  creal(c[  3]) =        4        cimag(c[  3]) =        1
  creal(c[  4]) =        4        cimag(c[  4]) =        1
  creal(c[  5]) =        3        cimag(c[  5]) =        2
  creal(c[  6]) =        2        cimag(c[  6]) =        3
  creal(c[  7]) =        1        cimag(c[  7]) =        4
  creal(c[  9]) =        1        cimag(c[  9]) =        2
  creal(c[ 10]) =        1        cimag(c[ 10]) =        2
  creal(c[ 11]) =        2        cimag(c[ 11]) =        1
  creal(c[ 12]) =        2        cimag(c[ 12]) =        1
  creal(c[ 13]) =        2        cimag(c[ 13]) =        1
  creal(c[ 14]) =        2        cimag(c[ 14]) =        1
  creal(c[ 15]) =        1        cimag(c[ 15]) =        2
  creal(c[ 16]) =        1        cimag(c[ 16]) =        2
  creal(c[ 18]) =        1        cimag(c[ 18]) =        2
  creal(c[ 19]) =        1        cimag(c[ 19]) =        2
  creal(c[ 20]) =        1        cimag(c[ 20]) =        2
  creal(c[ 21]) =        1        cimag(c[ 21]) =        2
  creal(c[ 22]) =        2        cimag(c[ 22]) =        1
```

```
creal(c[ 23]) =        2        cimag(c[ 23]) =        1
creal(c[ 24]) =        2        cimag(c[ 24]) =        1
creal(c[ 25]) =        2        cimag(c[ 25]) =        1
creal(c[ 27]) =        1        cimag(c[ 27]) =        1
creal(c[ 28]) =        1        cimag(c[ 28]) =        1
creal(c[ 29]) =        1        cimag(c[ 29]) =        1
creal(c[ 30]) =        1        cimag(c[ 30]) =        1
creal(c[ 31]) =        1        cimag(c[ 31]) =        1
creal(c[ 32]) =        1        cimag(c[ 32]) =        1
creal(c[ 33]) =        1        cimag(c[ 33]) =        1
creal(c[ 34]) =        1        cimag(c[ 34]) =        1
```

6.4 MULTIPLE ONE-DIMENSIONAL REAL FOURIER TRANSFORM

6.4.1 [DEPRECATED]ASL_qfrmfb, ASL_pfrmfb Multiple One-Dimensional Real Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

ASL_qfrmfb or ASL_pfrmfb obtains a half period of the m-fold one-dimensional Fourier forward transform (arbitrary radix) for the real data $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$.

$$c_{j,l} = \sum_{k=0}^{n-1} r_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j,l} = c_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j,l}(j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$ for n complex data groups $c_{j,l}(j=0,\cdots,n-1;\ l=1,\cdots,m)$ satisfying $c^*_{n-j,l}=c_{j,l}$, ASL_qfrmfb or ASL_pfrmfb obtains the m-fold one-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k,l} &= \sum_{j=0}^{n-1} c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2\sum_{j=1}^{\lceil\frac{n}{2}\rceil-1} \Re\{c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2\sum_{j=1}^{\lceil\frac{n}{2}\rceil-1} \left[\Re\{c_{j,l}\}\cos(2\pi\frac{jk}{n}) - \Im\{c_{j,l}\}\sin(2\pi\frac{jk}{n})\right] \\ &\quad (k=0,\cdots,n-1;\ l=1,\cdots,m) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ and $\Im\{z\}$ represent the real and imaginary parts of the complex number z, respectively. Also, when n is odd, $\hat{c}_{\frac{n}{2},l}=0$, and when n is even, $\hat{c}_{\frac{n}{2},l}=c_{\frac{n}{2},l}$.

(2) **Usage**

Double precision:

ierr = ASL_qfrmfb (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfrmfb (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	r	{D* / R*}	See Contents	Input	Input data $r_{k,l}$ (Forward transform) or $c_{j,l}$ (Backward transform) (See Note (b)). **Size:** incn $\times$ (n) + incm $\times$ (m − 1) + 1, where n is an odd, or incn $\times$ (n + 1) + incm $\times$ (m − 1) + 1, where n is an even.
				Output	Output results $c_{j,l}$ (Forward transform), or $r_{k,l}$ (Backward transform) (See Notes (b) and (c))
4	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
8	trigs	{D* / R*}	n	Output	Trigonometric function table (See Note (d))
9	wk	{D* / R*}	See Contents	Work	Work area **Size:** (n+1)×m, where n is an odd, or (n+2)×m, where n is an even.
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n≥1, m≥1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) or:
In the case where n is an odd:
incm ≥ (n+1) × gcm (incn, incm)
or if n is an even:
incm ≥ (n+2) × gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw $\in \{0, 1, -1\}$

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Input-time contents are output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of data n can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting n = 289(=17^2), it is usually more efficient to set n = 300(=$2^2 \times 3 \times 5^2$), n = 320(=$2^6 \times 5$), n = 384(=$2^7 \times 3$) or the like.

(b) The real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array r are associated as follows.

$$r_{k,l} \leftrightarrow \mathrm{r[incn * k + incm * (l - 1)]}$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{r[k + n * (l - 1)]}$$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{r[(l - 1) + m * k]}$$

and the data is stored so that it is packed consecutively for subscript l. When computing the backward transform, if n(=n) is odd, then r[incn ∗ n + incm ∗ (l − 1)] = 0, and when n is even, then r[incn ∗ n + incm ∗ (l − 1)] = r[incn ∗ (n + 1) + incm ∗ (l − 1)] = 0. If we let the real and imaginary parts of the complex data $c_{j,l}$ $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ be $\Re\{c_{j,l}\}$ and $\Im\{c_{j,l}\}$, respectively, the $c_{j,l}$ and

elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j,l}\} \quad \leftrightarrow \quad \mathrm{r[incn*(2j)+incm*(l-1)]}$$
$$\Im\{c_{j,l}\} \quad \leftrightarrow \quad \mathrm{r[incn*(2j+1)+incm*(l-1)]}$$

From the properties of a real Fourier transform, when n is odd, $\Im\{c_{0,l}\} = 0$, and when n is even, $\Im\{c_{0,l}\} = \Im\{c_{\frac{n}{2},l}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements $c_{j,l} \quad (j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ can be obtained according to the following relationship from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c_{n-j,l} = c^*_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{r}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{r}_{k,l} = nr_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data n, you should call this function once, and then use the after-initialization transform 6.4.2 $\left\{\begin{matrix}\text{ASL_qfrmbf} \\ \text{ASL_pfrmbf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.4.2 $\left\{\begin{matrix}\text{ASL_qfrmbf} \\ \text{ASL_pfrmbf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for array r.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.4.2 (7).

6.4.2 [DEPRECATED]ASL_qfrmbf, ASL_pfrmbf Multiple One-Dimensional Real Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_qfrmbf or ASL_pfrmbf obtains a half period of the m-fold one-dimensional Fourier forward transform (arbitrary radix) for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$.

$$c_{j,l} = \sum_{k=0}^{n-1} r_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j,l} = c_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j,l}(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ for n complex data groups $c_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ satisfying $c^*_{n-j,l} = c_{j,l}$, ASL_qfrmbf or ASL_pfrmbf obtains the m-fold one-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k,l} &= \sum_{j=0}^{n-1} c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{c_{j,l} e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_{0,l} + (-1)^k \hat{c}_{\frac{n}{2},l} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[\Re\{c_{j,l}\} \cos(2\pi \frac{jk}{n}) - \Im\{c_{j,l}\} \sin(2\pi \frac{jk}{n}) \right] \\ & (k = 0, \cdots, n-1;\ l = 1, \cdots, m) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ and $\Im\{z\}$ represent the real and imaginary parts of the complex number z, respectively. Also, when n is odd, $\hat{c}_{\frac{n}{2},l} = 0$, and when n is even, $\hat{c}_{\frac{n}{2},l} = c_{\frac{n}{2},l}$.

(2) **Usage**

Double precision:

ierr = ASL_qfrmbf (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfrmbf (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	n	I	1	Input	Number of transformed data values n (See Note (a))
2	m	I	1	Input	Multiplicity m
3	r	{D* / R*}	See Contents	Input	Input data $r_{k,l}$ (Forward transform) or $c_{j,l}$ (Backward transform) (See Note (b)). **Size:** incn × (n) + incm × (m − 1) + 1, where n is an odd, or incn × (n + 1) + incm × (m − 1) + 1, where n is an even.
				Output	Output results $c_{j,l}$ (Forward transform), or $r_{k,l}$ (Backward transform) (See Notes (b) and (c))
4	incn	I	1	Input	The stride between each transformed datum in storage (See Note (b))
5	incm	I	1	Input	The stride between the first elements of each transformed data in storage (See Note (b))
6	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
8	trigs	{D* / R*}	n	Input	Trigonometric function table (See Note (a))
9	wk	{D* / R*}	See Contents	Work	Work area **Size:** (n+1)×m, where n is an odd, or (n+2)×m, where n is an even.
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n≥1, m≥1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) or
In the case where n is an odd:
incm ≥ (n+1) × gcm (incn, incm)
or if n is an even:
incm ≥ (n+2) × gcm (incn, incm)
where, gcm(i, j) is the greatest common measure between i and j.

(d) isw $\in \{1, -1\}$

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 1.	Input-time contents are output unchanged.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of transformed data n after the including-initialization function 6.4.1 $\left\{\begin{array}{c}\text{ASL_qfrmfb}\\ \text{ASL_pfrmfb}\end{array}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array r are associated as follows.

$r_{k,l} \leftrightarrow \mathrm{r[incn * k + incm * (l-1)]}$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$r_{k,l} \leftrightarrow \mathrm{r[k + n * (l-1)]}$

and the data is stored so that it is packed consecutively for subscript k. If we let incn=m and incm=1, then the associations are as follows:

$r_{k,l} \leftrightarrow \mathrm{r[(l-1) + m * k]}$

and the data is stored so that it is packed consecutively for subscript l. When computing the backward transform, if n(=n) is odd, then $\mathrm{r[incn * n + incm * (l-1)]} = 0$, and when n is even, then $\mathrm{r[incn * n + incm * (l-1)]} = \mathrm{r[incn * (n+1) + incm * (l-1)]} = 0$. If we let the real and imaginary parts of the complex data $c_{j,l}$ $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ be $\Re\{c_{j,l}\}$ and $\Im\{c_{j,l}\}$, respectively, the $c_{j,l}$ and

elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j,l}\} \quad \leftrightarrow \quad \mathrm{r[incn*(2j)+incm*(l-1)]}$$
$$\Im\{c_{j,l}\} \quad \leftrightarrow \quad \mathrm{r[incn*(2j+1)+incm*(l-1)]}$$

From the properties of a real Fourier transform, when n is odd, $\Im\{c_{0,l}\} = 0$, and when n is even, $\Im\{c_{0,l}\} = \Im\{c_{\frac{n}{2},l}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements $c_{j,l} \quad (j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ can be obtained according to the following relationship from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c_{n-j,l} = c^*_{j,l}$$

Here, z^* represents the conjugate complex number of the complex number z.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of transformed data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ be represented by $\hat{r}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$, then the following relationship holds.

$$\hat{r}_{k,l} = n r_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the multiple one-dimensional real Fourier forward and backward transforms using the following sequence of numbers as input data.

r[0]= 1.000
r[1]= 2.000
r[2]= 3.000
r[3]= 4.000
r[4]= 5.000
r[5]= 6.000
r[6]= 7.000
r[7]= 8.000
r[12]= 1.000
r[13]= 1.000
r[14]= 2.000
r[15]= 2.000
r[16]= 3.000
r[17]= 3.000
r[18]= 4.000
r[19]= 4.000
r[24]= 1.000
r[25]= 1.000
r[26]= 1.000
r[27]= 1.000
r[28]= 2.000
r[29]= 2.000
r[30]= 2.000
r[31]= 2.000
r[36]= 1.000
r[37]= 1.000
r[38]= 1.000
r[39]= 1.000
r[40]= 1.000
r[41]= 1.000
r[42]= 1.000
r[43]= 1.000

(b) Input data

Array r, n=8, m=4, incn=1, incm=12, isw=1(Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C interface example for ASL_qfrmfb , ASL_qfrmbf */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int ld=46;
```

```
int n;      int m;
double *r;
int incn;   int incm;
int isw;
int ifax[20];   double *trigs;
double *wk;
int nt;
int ierr;
int i,j;
FILE *fp;

fp = fopen( "qfrmbf.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

printf( "    *** ASL_qfrmfb , ASL_qfrmbf ***\n" );
printf( "\n     ** Input **\n\n" );

r = ( double * )malloc((size_t)( sizeof(double) * ld ));
if( r == NULL )
{
    printf( "no enough memory for array r\n" );
    return -1;
}

trigs = ( double * )malloc((size_t)( sizeof(double) * ld ));
if( trigs == NULL )
{
    printf( "no enough memory for array trigs\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * ld ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

fscanf( fp, "%d,%d,%d,%d,%d", &n, &m, &incn, &incm, &nt );

for( j=0 ; j<m ; j++ )
{
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &r[i*incn+j*incm] );
    }
}

printf("\t   n    = %d \n", n );
printf("\t   m    = %d \n", m );
printf("\t   incn = %d \n", incn );
printf("\t   incm = %d \n", incm );
printf("\t   nt   = %d \n\n", nt );

printf( "\t  Real Part\n" );
for( j=0 ; j<m ; j++ )
{
    printf( "\t" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "    r[%3d] =%4.1f",
            i*incn+j*incm, r[i*incn+j*incm] );
        if((i+1)%4==0) printf( "\n\t" );
    }
    printf( "\n" );
}

fclose( fp );


printf( "\n     ** Output **\n" );

isw = 1;
ierr = ASL_qfrmfb(n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

for( j=0 ; j<m ; j++ )
{
    for( i=0 ; i<n+2 ; i++ )
    {
        r[i*incn+j*incm] /= n ;
    }
}

printf( "\n\t< Forward Transform >\n" );
printf( "\tierr = %6d\n", ierr );
```

```
        printf( "\n\tSolution\n\n" );
        printf( "\t  Real Part                  Imaginary Part\n" );
        for( j=0 ; j<m ; j++ )
        {
            for( i=0 ; i<n+2 ; i=i+2 )
            {
                printf( "\t  r[%3d] = %8.3g           r[%3d] = %8.3g\n",
                    i*incn+j*incm, r[i*incn+j*incm],
                    (i+1)*incn+j*incm, r[(i+1)*incn+j*incm] );
            }
            printf( "\n" );
        }

        isw = -1;
        ierr = ASL_qfrmbf(n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

        printf( "\n\t< Backward Transform >\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\n\tSolution\n\n" );
        printf( "\t  Real Part\n" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "\t" );
            for( i=0 ; i<n+2 ; i++ )
            {
                printf( "    r[%3d] =%4.1f",
                    i*incn+j*incm, r[i*incn+j*incm] );
                if((i+1)%4==0) printf( "\n\t" );
            }
            printf( "\n" );
        }


        free( r );
        free( trigs );
        free( wk );

        return 0;
    }
```

(d) Output results

```
        *** ASL_qfrmfb , ASL_qfrmbf ***

         ** Input **

           n    = 8
           m    = 4
           incn = 1
           incm = 12
           nt   = 2

          Real Part
            r[  0] = 1.0    r[  1] = 2.0    r[  2] = 3.0    r[  3] = 4.0
            r[  4] = 5.0    r[  5] = 6.0    r[  6] = 7.0    r[  7] = 8.0

            r[ 12] = 1.0    r[ 13] = 1.0    r[ 14] = 2.0    r[ 15] = 2.0
            r[ 16] = 3.0    r[ 17] = 3.0    r[ 18] = 4.0    r[ 19] = 4.0

            r[ 24] = 1.0    r[ 25] = 1.0    r[ 26] = 1.0    r[ 27] = 1.0
            r[ 28] = 2.0    r[ 29] = 2.0    r[ 30] = 2.0    r[ 31] = 2.0

            r[ 36] = 1.0    r[ 37] = 1.0    r[ 38] = 1.0    r[ 39] = 1.0
            r[ 40] = 1.0    r[ 41] = 1.0    r[ 42] = 1.0    r[ 43] = 1.0


         ** Output **

        < Forward Transform >
        ierr =      0

        Solution

          Real Part                  Imaginary Part
          r[  0] =      4.5           r[  1] =        0
          r[  2] =     -0.5           r[  3] =     1.21
          r[  4] =     -0.5           r[  5] =      0.5
          r[  6] =     -0.5           r[  7] =    0.207
          r[  8] =     -0.5           r[  9] =        0

          r[ 12] =      2.5           r[ 13] =        0
          r[ 14] =    -0.25           r[ 15] =    0.604
          r[ 16] =    -0.25           r[ 17] =     0.25
          r[ 18] =    -0.25           r[ 19] =    0.104
          r[ 20] =        0           r[ 21] =        0

          r[ 24] =      1.5           r[ 25] =        0
          r[ 26] =   -0.125           r[ 27] =    0.302
```

```
  r[ 28] =        0           r[ 29] =        0
  r[ 30] =   -0.125           r[ 31] =   0.0518
  r[ 32] =        0           r[ 33] =        0

  r[ 36] =        1           r[ 37] =        0
  r[ 38] =        0           r[ 39] =        0
  r[ 40] =        0           r[ 41] =        0
  r[ 42] =        0           r[ 43] =        0
  r[ 44] =        0           r[ 45] =        0


< Backward Transform >
ierr =      0

Solution

  Real Part
    r[  0] = 1.0    r[  1] = 2.0    r[  2] = 3.0    r[  3] = 4.0
    r[  4] = 5.0    r[  5] = 6.0    r[  6] = 7.0    r[  7] = 8.0
    r[  8] = 0.0    r[  9] = 0.0
    r[ 12] = 1.0    r[ 13] = 1.0    r[ 14] = 2.0    r[ 15] = 2.0
    r[ 16] = 3.0    r[ 17] = 3.0    r[ 18] = 4.0    r[ 19] = 4.0
    r[ 20] = 0.0    r[ 21] = 0.0
    r[ 24] = 1.0    r[ 25] = 1.0    r[ 26] = 1.0    r[ 27] = 1.0
    r[ 28] = 2.0    r[ 29] = 2.0    r[ 30] = 2.0    r[ 31] = 2.0
    r[ 32] = 0.0    r[ 33] = 0.0
    r[ 36] = 1.0    r[ 37] = 1.0    r[ 38] = 1.0    r[ 39] = 1.0
    r[ 40] = 1.0    r[ 41] = 1.0    r[ 42] = 1.0    r[ 43] = 1.0
    r[ 44] = 0.0    r[ 45] = 0.0
```

6.5 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.5.1 [DEPRECATED]ASL_qfc2fb, ASL_pfc2fb Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_qfc2fb or ASL_pfc2fb computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$$

Backward transform

ASL_qfc2fb or ASL_pfc2fb computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$$

(2) **Usage**

Double precision:

ierr = ASL_qfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real   Z:Double precision complex   I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	cr	{D* / R*}	lx×ly	Input	Real part of input data c_{k_x,k_y} (See Note (b))
				Output	Real part of output data d_{j_x,j_y} (See Notes (b) and (c))
4	ci	{D* / R*}	lx×ly	Input	Imaginary part of input data c_{k_x,k_y} (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y} (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array cr and ci (See Note (b))
6	ly	I	1	Input	Second dimension of array cr and ci (See Note (b))
7	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
8	ifax	I*	40	Output	Factorization results and number of factors (See Note (d))
9	trigs	{D* / R*}	2 × (nx + ny)	Output	Trigonometric function table (See Note (d))
10	wk	{D* / R*}	2 × lx × ly	Work	Work area
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw $\in \{0, 1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data nx or ny can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300 (=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be $\Re\{c_{k_x,k_y}\}$ and $\Im\{c_{k_x,k_y}\}$, respectively, the c_{k_x,k_y} and elements of arrays cr and ci are associated as follows.

$$\Re\{c_{k_x,k_y}\} \leftrightarrow \mathrm{cr}[\mathrm{k_x} + \mathrm{lx} * \mathrm{k_y}]$$
$$\Im\{c_{k_x,k_y}\} \leftrightarrow \mathrm{ci}[\mathrm{k_x} + \mathrm{lx} * \mathrm{k_y}]$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions lx **and** ly **of arrays** cr **and** ci **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is even,** lx=nx**+1 is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny), you should call this function once, and then use the after-initialization transform 6.5.2 {ASL_qfc2bf, ASL_pfc2bf}, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.5.2 {ASL_qfc2bf, ASL_pfc2bf}.
To perform initialization only by setting isw=0, you need not set input data for arrays cr and ci.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period,

the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.5.2 (7).

6.5.2 [DEPRECATED]ASL_qfc2bf, ASL_pfc2bf Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_qfc2bf or ASL_pfc2bf computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

ASL_qfc2bf or ASL_pfc2bf computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:
  ierr = ASL_qfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);
Single precision:
  ierr = ASL_pfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	cr	{D* R*}	lx×ly	Input	Real part of input data c_{k_x,k_y} (See Note (b))
				Output	Real part of output data d_{j_x,j_y} (See Notes (b) and (c))
4	ci	{D* R*}	lx×ly	Input	Imaginary part of input data c_{k_x,k_y}
				Output	Imaginary part of output results d_{j_x,j_y} (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array cr and ci (See Note (b))
6	ly	I	1	Input	Second dimension of array cr and ci (See Note (b))
7	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
8	ifax	I*	40	Input	Factorization results and number of factors (See Note (a))
9	trigs	{D* R*}	2 × (nx + ny)	Input	Trigonometric function table (See Note (a))
10	wk	{D* R*}	2 × lx × ly	Work	Work area
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw $\in \{1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny) after the including-initialization function 6.5.1 {ASL_qfc2fb, ASL_pfc2fb} has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be $\Re\{c_{k_x,k_y}\}$ and $\Im\{c_{k_x,k_y}\}$, respectively, the c_{k_x,k_y} and elements of arrays cr and ci are associated as follows.

$$\Re\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{cr[k_x + lx * k_y]}$$
$$\Im\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{ci[k_x + lx * k_y]}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions lx **and** ly **of arrays** cr **and** ci **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the two-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y} = (k_x+1)+(k_y+1)+\sqrt{-1}\frac{(k_x+1)(k_y+1)}{n_x n_y}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$$

as input data.

(b) Input data

Array cr and ci, nx=5, ny=4, lx=5, ly=5, isw=1 (forward transform), isw=−1 (backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_qfc2fb , ASL_qfc2bf */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int nx = 5; int ny = 4;
    int lx = 5; int ly = 5;
    double *cr; double *ci;
    int isw;
    int ifax[40];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j;

    printf( "    *** ASL_qfc2fb , ASL_qfc2bf ***\n" );
    printf( "\n     ** Input **\n" );

    cr = ( double * )malloc((size_t)( sizeof(double) * (lx*ly) ));
    if( cr == NULL )
    {
        printf( "no enough memory for array cr\n" );
        return -1;
    }

    ci = ( double * )malloc((size_t)( sizeof(double) * (lx*ly) ));
    if( ci == NULL )
    {
        printf( "no enough memory for array ci\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*(nx+ny)) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (2*lx*ly) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnt = %6d\n", nt );

    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            cr[(i-1)+lx*(j-1)]=(double)(i+j) ;
            ci[(i-1)+lx*(j-1)]=(double)(i*j)/(double)(nx*ny) ;
        }
    }

    printf( "\tcr[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
```

```
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
            }
            printf( "\n" );
        }

        isw = 1;
        ierr = ASL_qfc2fb(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

        for( i=0 ; i<lx*ly ; i++)
        {
            cr[i] /= (double)(nx*ny);
            ci[i] /= (double)(nx*ny);
        }

        printf( "\n     ** Output **\n" );

        printf( "\t< Forward Transform >\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\tcr[ix][iy]\n" );
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
            }
            printf( "\n" );
        }

        isw = -1;
        ierr = ASL_qfc2bf(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

        printf( "\t< Backward Transform >\n" );
        printf( "\tierr = %6d\n", ierr );

        printf( "\tcr[ix][iy]\n" );
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
            }
            printf( "\n" );
        }


        free( cr );
        free( ci );
        free( trigs );
        free( wk );

        return 0;
    }
```

(d) Output results

```
*** ASL_qfc2fb , ASL_qfc2bf ***

 ** Input **
nx =      5
ny =      4
nt =      2
cr[ix][iy]
(       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
(       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
(       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
(       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
(       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)

 ** Output **
< Forward Transform >
ierr =      0
cr[ix][iy]
(     5.5,   0.375) (  -0.575,   0.425) (    -0.5,  -0.075) (  -0.425,  -0.575)
(  -0.586,   0.626) (  0.0297, -0.0047) (  0.0172,  0.0125) (  0.0047,  0.0297)
(   -0.52,     0.1) (  0.0166, 0.00844) ( 0.00406,  0.0125) (-0.00844,  0.0166)
(   -0.48,  -0.225) ( 0.00844,  0.0166) (-0.00406,  0.0125) ( -0.0166, 0.00844)
(  -0.414,  -0.751) ( -0.0047,  0.0297) ( -0.0172,  0.0125) ( -0.0297, -0.0047)
< Backward Transform >
ierr =      0
cr[ix][iy]
(       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
(       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
(       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
(       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
(       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)
```

6.6 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.6.1 [DEPRECATED]ASL_hfc2fb, ASL_gfc2fb Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_hfc2fb or ASL_gfc2fb computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$$

Backward transform

ASL_hfc2fb or ASL_gfc2fb computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \quad (j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$$

(2) **Usage**

Double precision:

ierr = ASL_hfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	c	{Z* / C*}	lx×ly	Input	Input data c_{k_x,k_y} (See Note (b))
				Output	Output results d_{j_x,j_y} (See Notes (b) and (c))
4	lx	I	1	Input	Adjustable dimension of array c (See Note (b))
5	ly	I	1	Input	Second dimension of array c (See Note (b))
6	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	40	Output	Factorization results and number of factors (See Note (d))
8	trigs	{D* / R*}	2 × (nx + ny)	Output	Trigonometric function table (See Note (d))
9	wk	{Z* / C*}	lx × ly	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw $\in \{0, 1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data nx or ny can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300 (=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) The complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ and elements of array c are associated as follows.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{c[k_x + lx * k_y]}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x-1;\ j_y = 0, \cdots, n_y-1)$.
The adjustable dimensions lx **and** ly **of array** c **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny), you should call this function once, and then use the after-initialization transform 6.6.2 $\left\{\begin{array}{c}\text{ASL_hfc2bf}\\ \text{ASL_gfc2bf}\end{array}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.6.2 $\left\{\begin{array}{c}\text{ASL_hfc2bf}\\ \text{ASL_gfc2bf}\end{array}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for array c.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time

function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.6.2 (7).

6.6.2 [DEPRECATED]ASL_hfc2bf, ASL_gfc2bf Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_hfc2bf or ASL_gfc2bf computes the two-dimensional complex Fourier forward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

Backward transform

ASL_hfc2bf or ASL_gfc2bf computes the two-dimensional complex Fourier backward transform (arbitrary radix) for the two-dimensional complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$d_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

(2) **Usage**

Double precision:

ierr = ASL_hfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer ; long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	c	{Z* ; C*}	lx×ly	Input	Input data c_{k_x,k_y} (See Note (b))
				Output	Output results d_{j_x,j_y} (See Notes (b) and (c))
4	lx	I	1	Input	Adjustable dimension of array c (See Note (b))
5	ly	I	1	Input	Second dimension of array c (See Note (b))
6	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	40	Input	Factorization results and number of factors (See Note (a))
8	trigs	{D* ; R*}	2 × (nx + ny)	Input	Trigonometric function table (See Note (a))
9	wk	{Z* ; C*}	lx × ly	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw $\in \{1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny) after the including-initialization function 6.6.1 $\left\{\begin{matrix}\text{ASL_hfc2fb}\\\text{ASL_gfc2fb}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The complex data c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array c are associated as follows.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{c[k_x + lx * k_y]}$$

Similarly, for the complex data $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$.
The adjustable dimensions lx **and** ly **of array** c **should be set to odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1)$ be represented by $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the two-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y} = (k_x+1)+(k_y+1)+\sqrt{-1}\frac{(k_x+1)(k_y+1)}{n_x n_y}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$$

as input data.

(b) Input data

Array c, nx=5, ny=4, lx=5, ly=5, isw=1(Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_hfc2fb , ASL_hfc2bf */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <asl.h>

int main()
{
    int nx = 5; int ny = 4;
    int lx = 5; int ly = 5;
    double _Complex *c;
    int isw;
    int ifax[40];
    double *trigs;
    double _Complex *wk;
    int nt = 2;
    int ierr;
    int i,j;

    printf( "    *** ASL_hfc2fb , ASL_hfc2bf ***\n" );
    printf( "\n     ** Input **\n" );

    c = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lx*ly) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*(nx+ny)) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lx*ly) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnt = %6d\n", nt );

    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            c[(i-1)+lx*(j-1)]=(double)(i+j) + (double)(i*j)/(double)(nx*ny) * _Complex_I;
        }
    }

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }
```

```
    isw = 1;
    ierr = ASL_hfc2fb(nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly ; i++)
    {
        c[i] /= (double)(nx*ny);
    }

    printf( "\n     ** Output **\n" );

    printf( "\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_hfc2bf(nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }


    free( c );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
    *** ASL_hfc2fb , ASL_hfc2bf ***

     ** Input **
    nx =      5
    ny =      4
    nt =      2
    c[ix][iy]
    (       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
    (       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
    (       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
    (       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
    (       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)

     ** Output **
    < Forward Transform >
    ierr =      0
    c[ix][iy]
    (     5.5,   0.375) (  -0.575,   0.425) (    -0.5,  -0.075) (  -0.425,  -0.575)
    (  -0.586,   0.626) (  0.0297, -0.0047) (  0.0172,  0.0125) (  0.0047,  0.0297)
    (   -0.52,     0.1) (  0.0166, 0.00844) ( 0.00406,  0.0125) (-0.00844,  0.0166)
    (   -0.48,  -0.225) ( 0.00844,  0.0166) (-0.00406,  0.0125) ( -0.0166, 0.00844)
    (  -0.414,  -0.751) ( -0.0047,  0.0297) ( -0.0172,  0.0125) ( -0.0297, -0.0047)
    < Backward Transform >
    ierr =      0
    c[ix][iy]
    (       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
    (       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
    (       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
    (       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
    (       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)
```

6.7 TWO-DIMENSIONAL REAL FOURIER TRANSFORM

6.7.1 [DEPRECATED]ASL_qfr2fb, ASL_pfr2fb Two-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_qfr2fb or ASL_pfr2fb obtains a half period of the two-dimensional Fourier forward transform (arbitrary radix) for the two-dimensional real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$c_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} r_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ for $n_x n_y$ complex data $c_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ satisfying $c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$ and $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$, ASL_qfr2fb or ASL_pfr2fb obtains the two-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1} \{c_{0,j_y} + (-1)^{k_x} \hat{c}_{\frac{n_x}{2},j_y}\} e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2 \sum_{j_y=0}^{n_y-1} \sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil - 1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})}\} \\ &\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$.

(2) **Usage**

Double precision:

ierr = ASL_qfr2fb (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfr2fb (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	r	{D* / R*}	lx×ly	Input	Input data r_{k_x,k_y} (Forward transform), or c_{j_x,j_y} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y} (Forward transform), or r_{k_x,k_y} (Backward transform) (See Notes (b) and (c))
4	lx	I	1	Input	Adjustable dimension of array r (See Note (b))
5	ly	I	1	Input	Second dimension of array r (See Note (b))
6	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
7	ifax	I*	40	Output	Factorization results and number of factors (See Note (d))
8	trigs	{D* / R*}	nx+2×ny	Output	Trigonometric function table (See Note (d))
9	wk	{D* / R*}	lx × ly	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) In the case where nx is an odd, nx+1≤lx, ny≤ly
or if nx is an even, nx+2≤lx, ny≤ly

(c) isw $\in \{0, 1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data nx or ny can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300 (=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) The real data r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array r are associated as follows.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{r[k_x + lx * k_y]}$$

When computing the backward transform, if nx(=n_x) is odd, then $\mathrm{r[nx + lx * k_y] = 0}$, and when nx is even, then $\mathrm{r[nx + lx * k_y] = r[nx + 1 + lx * k_y] = 0}$. Also, when entering the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ into array r, the corresponding zeros mentioned above need not be specifically stored.

If we let the real and imaginary parts of the complex data $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ be $\Re\{c_{j_x,j_y}\}$ and $\Im\{c_{j_x,j_y}\}$, respectively, the c_{j_x,j_y} and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j_x,j_y}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + lx * j_y]}$$
$$\Im\{c_{j_x,j_y}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * j_y]}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0}\} = 0$, and when nx and ny are both even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c^*_{n_x - j_x, n_y - j_y} = c_{j_x,j_y}$$
$$c^*_{n_x - j_x, j_y} = c_{j_x, n_y - j_y}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** r **should be set so that** lx/2 **and** ly **are odd numbers to avoid bank conflict of main memory. Usually, when** nx, **for example, is (a multiple of 4)+2,** lx=nx+4 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny), you should call this function once, and then use the after-initialization transform 6.7.2 $\left\{\begin{matrix}\text{ASL_qfr2bf}\\ \text{ASL_pfr2bf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.7.2 $\left\{\begin{matrix}\text{ASL_qfr2bf}\\ \text{ASL_pfr2bf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for array r.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.7.2 (7).

6.7.2 [DEPRECATED]ASL_qfr2bf, ASL_pfr2bf Two-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_qfr2bf or ASL_pfr2bf obtains a half period of the two-dimensional Fourier forward transform (arbitrary radix) for the two-dimensional real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$.

$$c_{j_x,j_y} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} r_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \quad (j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ for $n_x n_y$ complex data $c_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ satisfying $c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$ and $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$, ASL_qfr2bf or ASL_pfr2bf obtains the two-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1} \{c_{0,j_y} + (-1)^{k_x} \hat{c}_{\frac{n_x}{2},j_y}\} e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2 \sum_{j_y=0}^{n_y-1} \sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil - 1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})}\} \\ & (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$.

(2) **Usage**

Double precision:

```
ierr = ASL_qfr2bf (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);
```

Single precision:

```
ierr = ASL_pfr2bf (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);
```

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	r	{D* / R*}	lx×ly	Input	Input data r_{k_x,k_y} (Forward transform), or c_{j_x,j_y} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y} (Forward transform), or r_{k_x,k_y} (Backward transform) (See Notes (b) and (c))
4	lx	I	1	Input	Adjustable dimension of array r (See Note (b))
5	ly	I	1	Input	Second dimension of array r (See Note (b))
6	isw	I	1	Input	Processing switch isw= 1: Forward transform isw=−1: Backward transform
7	ifax	I*	40	Input	Factorization results and number of factors (See Note (a))
8	trigs	{D* / R*}	nx + 2 × ny	Input	Trigonometric function table (See Note (a))
9	wk	{D* / R*}	lx × ly	Work	Work area
10	nt	I	1	Input	Number of tasks to be generated
11	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2

(b) In the case where nx is an odd, nx+1≤lx, ny≤ly
or if nx is an even, nx+2≤lx, ny≤ly

(c) isw $\in \{1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny) after the including-initialization function 6.7.1 $\left\{\begin{matrix}\text{ASL_qfr2fb}\\ \text{ASL_pfr2fb}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The real data r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ and elements of array r are associated as follows.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{r[k_x + lx * k_y]}$$

When computing the backward transform, if nx($=n_x$) is odd, then $\mathrm{r[nx + lx * k_y]} = 0$, and when nx is even, then $\mathrm{r[nx + lx * k_y]} = \mathrm{r[nx + 1 + lx * k_y]} = 0$. Also, when entering the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ into array r, the corresponding zeros mentioned above need not be specifically stored.

If we let the real and imaginary parts of the complex data $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ be $\Re\{c_{j_x,j_y}\}$ and $\Im\{c_{j_x,j_y}\}$, respectively, the c_{j_x,j_y} and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j_x,j_y}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + lx * j_y]}$$
$$\Im\{c_{j_x,j_y}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * j_y]}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0}\} = 0$, and when nx and ny are both even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c^*_{n_x - j_x, n_y - j_y} = c_{j_x,j_y}$$
$$c^*_{n_x - j_x, j_y} = c_{j_x, n_y - j_y}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** r **should be set so that** lx/2 **and** ly **are odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is (a multiple of 4)+2,** lx=nx+4 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ be represented by $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the two-dimensional real Fourier forward and backward transforms using

$$r_{k_x,k_y} = \frac{n_x + n_y}{(k_x + 1) + (k_y + 1)}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

as input data.

(b) Input data

Array r, nx=6, ny=4, lx=10, ly=5, isw=1(Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_qfr2fb , ASL_qfr2bf */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx =  6;    int ny =  4;
    int lx = 10;    int ly =  5;
    double *r;
    int isw;
    int ifax[40];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j;

    printf( "    *** ASL_qfr2fb , ASL_qfr2bf ***\n" );
    printf( "\n     ** Input **\n" );

    r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly) ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (nx+2*ny) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }
```

```
    wk = ( double * )malloc((size_t)( sizeof(double) * (lx*ly) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnt = %6d\n", nt );

    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            r[i-1+lx*(j-1)]=(double)(nx+ny)/(double)(i+j) ;
        }
    }

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }


    isw = 1;
    ierr = ASL_qfr2fb(nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly ; i++)
    {
        r[i] /= (double)(nx*ny);
    }

    printf( "\n     ** Output **\n" );

    printf( "\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx+2 ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_qfr2bf(nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }


    free( r );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_qfr2fb , ASL_qfr2bf ***

 ** Input **
nx =      6
ny =      4
nt =      2
r[ix][iy]
       5        3.33         2.5           2
    3.33         2.5           2        1.67
     2.5           2        1.67        1.43
```

```
        2         1.67         1.43         1.25
     1.67         1.43         1.25         1.11
     1.43         1.25         1.11            1

 ** Output **
< Forward Transform >
ierr =      0
r[ix][iy]
     1.94        0.249        0.219        0.249
        0       -0.155            0        0.155
    0.296       0.0585       0.0761        0.119
   -0.247      -0.0939      -0.0447     -0.00945
    0.229       0.0557        0.058       0.0794
  -0.0928      -0.0535      -0.0186       0.0102
    0.219       0.0637       0.0547       0.0637
        0      -0.0301            0       0.0301
< Backward Transform >
ierr =      0
r[ix][iy]
        5         3.33          2.5            2
     3.33          2.5            2         1.67
      2.5            2         1.67         1.43
        2         1.67         1.43         1.25
     1.67         1.43         1.25         1.11
     1.43         1.25         1.11            1
```

6.8 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

6.8.1 [DEPRECATED]ASL_qfc3fb, ASL_pfc3fb Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

ASL_qfc3fb or ASL_pfc3fb computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z}\right)}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Backward transform

ASL_qfc3fb or ASL_pfc3fb computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}\left(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z}\right)}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

(2) **Usage**

Double precision:

ierr = ASL_qfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	cr	{D* / R*}	lx×ly×lz	Input	Real part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Real part of output data d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	ci	{D* / R*}	lx×ly×lz	Input	Imaginary part of input data $c_{k,s,u}$ (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
6	lx	I	1	Input	Adjustable dimension of array cr and ci (See Note (b))
7	ly	I	1	Input	Second dimension of array cr and ci (See Note (b))
8	lz	I	1	Input	Third dimension of array cr and ci (See Note (b))
9	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
10	ifax	I*	60	Output	Factorization results and number of factors (See Note (d))
11	trigs	{D* / R*}	2 × (nx + ny + nz)	Output	Trigonometric function table (See Note (d))
12	wk	{D* / R*}	2×lx×ly×lz	Work	Work area
13	nt	I	1	Input	Number of tasks to be generated
14	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx≥2, ny≥2 or nz≥2

(b) nx≤lx, ny≤ly, or nz≤lz

(c) isw $\in \{0, 1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data nx, ny or nz can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300(=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y,k_z} ($k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1$) be $\Re\{c_{k_x,k_y,k_z}\}$ and $\Im\{c_{k_x,k_y,k_z}\}$, respectively, the c_{k_x,k_y,k_z} and elements of arrays cr and ci are associated as follows.

$$\Re\{c_{k_x,k_y,k_z}\} \leftrightarrow \mathrm{cr}[\mathrm{k_x} + \mathrm{lx} * (\mathrm{k_y} + \mathrm{ly} * \mathrm{k_z})]$$
$$\Im\{c_{k_x,k_y,k_z}\} \leftrightarrow \mathrm{ci}[\mathrm{k_x} + \mathrm{lx} * (\mathrm{k_y} + \mathrm{ly} * \mathrm{k_z})]$$

Similarly, for the complex data d_{j_x,j_y,j_z} ($j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1$).
The adjustable dimensions lx**,** ly**, and** lz **of arrays** cr **and** ci **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** cr **and** ci**. Usually, when** nx**, for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data c_{k_x,k_y,k_z} ($k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1$) be represented by $\hat{c}_{k_x,k_y,k_z}$ ($k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1$), then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny, nz), you should call this function once, and then use the after-initialization transform 6.8.2 $\left\{\begin{matrix}\text{ASL_qfc3bf}\\ \text{ASL_pfc3bf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.8.2 $\left\{\begin{matrix}\text{ASL_qfc3bf}\\ \text{ASL_pfc3bf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for arrays cr and ci.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**
See the example in Section 6.8.2 (7).

6.8.2 [DEPRECATED]ASL_qfc3bf, ASL_pfc3bf Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_qfc3bf or ASL_pfc3bf computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}\left(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z}\right)}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

Backward transform

ASL_qfc3bf or ASL_pfc3bf computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}\left(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z}\right)}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

(2) **Usage**

Double precision:

ierr = ASL_qfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	cr	{D* / R*}	lx×ly×lz	Input	Real part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Real part of output data d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	ci	{D* / R*}	lx×ly×lz	Input	Imaginary part of input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Imaginary part of output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
6	lx	I	1	Input	Adjustable dimension of array cr and ci (See Note (b))
7	ly	I	1	Input	Second dimension of array cr and ci (See Note (b))
8	lz	I	1	Input	Third dimension of array cr and ci (See Note (b))
9	isw	I	1	Input	Processing switch isw= 1 :Forward transform isw=−1 :Backward transform
10	ifax	I*	60	Input	Factorization results and number of factors (See Note (d))
11	trigs	{D* / R*}	2 × (nx + ny + nz)	Input	Trigonometric function table (See Note (d))
12	wk	{D* / R*}	2×lx×ly× lz	Work	Work area
13	nt	I	1	Input	Number of tasks to be generated
14	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw $\in \{1, -1\}$

(d) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny, nz) after the including-initialization function 6.8.1 $\left\{\begin{array}{l}\text{ASL_qfc3fb}\\ \text{ASL_pfc3fb}\end{array}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) If we let the real and imaginary parts of the complex data c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1)$ be $\Re\{c_{k_x,k_y,k_z}\}$ and $\Im\{c_{k_x,k_y,k_z}\}$, respectively, the c_{k_x,k_y,k_z} and elements of arrays cr and ci are associated as follows.

$$\Re\{c_{k_x,k_y,k_z}\} \quad \leftrightarrow \quad \mathrm{cr}[\mathrm{k_x} + \mathrm{lx} * (\mathrm{k_y} + \mathrm{ly} * \mathrm{k_z})]$$
$$\Im\{c_{k_x,k_y,k_z}\} \quad \leftrightarrow \quad \mathrm{ci}[\mathrm{k_x} + \mathrm{lx} * (\mathrm{k_y} + \mathrm{ly} * \mathrm{k_z})]$$

Similarly, for the complex data $d_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$.
The adjustable dimensions lx, ly, **and** lz **of arrays** cr **and** ci **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** cr **and** ci**. Usually, when** nx**, for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem
Compute the three-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y,k_z} = \frac{n_x + n_y + n_z}{(k_x+1)+(k_y+1)+(k_z+1)} + \sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

as input data.

(b) Input data
Array cr and ci, nx=5, ny=4, nz=3, lx=5, ly=5, lz=3, isw=1 (Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_qfc3fb , ASL_qfc3bf */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int nx = 5; int ny = 4; int nz = 3;
    int lx = 5; int ly = 5; int lz = 3;
    double *cr; double *ci;
    int isw;
    int ifax[60];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j,k;

    printf( "    *** ASL_qfc3fb , ASL_qfc3bf ***\n" );
    printf( "\n     ** Input **\n" );

    cr = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz) ));
    if( cr == NULL )
    {
        printf( "no enough memory for array cr\n" );
        return -1;
    }

    ci = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz) ));
    if( ci == NULL )
    {
        printf( "no enough memory for array ci\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*(nx+ny+nz)) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (2*lx*ly*lz) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnz = %6d\n", nz );
    printf( "\tnt = %6d\n", nt );

    for( k=1 ; k<=nz ; k++ )
    {
        for( j=1 ; j<=ny ; j++ )
        {
            for( i=1 ; i<=nx ; i++ )
            {
```

```
                cr[i-1+lx*(j-1)+lx*ly*(k-1)]=(double)(nx+ny+nz)/(double)(i+j+k) ;
                ci[i-1+lx*(j-1)+lx*ly*(k-1)]=(double)(i*j*k)/(double)(nx*ny*nz) ;
            }
        }
    }
    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
        }
        printf( "\n" );
    }
    printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }
    printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }

    isw = 1;
    ierr = ASL_qfc3fb(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);
    for( i=0 ; i<lx*ly*lz ; i++)
    {
        cr[i] /= (double)(nx*ny*nz);
        ci[i] /= (double)(nx*ny*nz);
    }
    printf( "\n     ** Output **\n" );
    printf( "\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
        }
        printf( "\n" );
    }
    printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }
    printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }
    isw = -1;
    ierr = ASL_qfc3bf(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);
    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
```

```
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
            }
            printf( "\n" );
        }

        printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
            }
            printf( "\n" );
        }

        printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
            }
            printf( "\n" );
        }


        free( cr );
        free( ci );
        free( trigs );
        free( wk );

        return 0;
    }
```

(d) Output results

```
*** ASL_qfc3fb , ASL_qfc3bf ***

 ** Input **
nx =      5
ny =      4
nz =      3
nt =      2
cr[ix][iy][1] ci[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
cr[ix][iy][2] ci[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
cr[ix][iy][3] ci[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)

 ** Output **
< Forward Transform >
ierr =       0
cr[ix][iy][1] ci[ix][iy][1]
(    1.74,    0.25) (   0.102,   -0.16) (   0.137,   -0.05) (   0.202,    0.06)
(   0.108,  -0.189) (  0.0379, -0.0469) (  0.0406, -0.0125) (  0.0525,   0.016)
(   0.125, -0.0784) (   0.034, -0.0168) (  0.0261, 0.00288) (  0.0254,  0.0209)
(   0.152,-0.00492) (  0.0366, 0.00116) (  0.0207,  0.0138) (   0.012,   0.028)
(   0.223,   0.106) (  0.0462,  0.0236) (  0.0177,  0.0292) (-0.00166,  0.0406)
cr[ix][iy][2] ci[ix][iy][2]
(   0.106,  -0.127) (  0.0407, -0.0223) (  0.0315, 0.00295) (  0.0297,  0.0255)
(  0.0419, -0.0317) (-0.00167,-0.00877) (  0.0025,-0.00799) ( 0.00901,-0.00976)
(  0.0317,-0.00698) ( 0.00134,-0.00743) ( 0.00423,-0.00524) ( 0.00924,-0.00424)
(  0.0297,  0.0084) ( 0.00473,-0.00711) ( 0.00655,-0.00336) (  0.0108,  0.0001)
(  0.0318,  0.0285) (  0.0112,-0.00921) (  0.0118,-0.00179) (   0.016, 0.00627)
cr[ix][iy][3] ci[ix][iy][3]
(   0.178, 0.00231) (  0.0403,   0.014) (   0.017,   0.022) ( 0.00125,  0.0329)
(  0.0484, 0.00885) ( 0.00516, -0.0104) ( 0.00849,-0.00569) (  0.0153, -0.0016)
(  0.0244,  0.0163) ( 0.00692, -0.0061) (  0.0076,-0.00159) (  0.0107, 0.00322)
(  0.0129,  0.0239) ( 0.00961, -0.0029) ( 0.00799, 0.00185) ( 0.00849,  0.0078)
( 0.00117,   0.036) (  0.0163,0.000768) (  0.0106, 0.00714) ( 0.00733,  0.0161)
< Backward Transform >
ierr =       0
cr[ix][iy][1] ci[ix][iy][1]
```

```
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
cr[ix][iy][2] ci[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
cr[ix][iy][3] ci[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)
```

6.9 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

6.9.1 [DEPRECATED]ASL_hfc3fb, ASL_gfc3fb Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_hfc3fb or ASL_gfc3fb computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

Backward transform

ASL_hfc3fb or ASL_gfc3fb computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

(2) **Usage**

Double precision:

ierr = ASL_hfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	c	{Z* / C*}	lx×ly×lz	Input	Input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array c (See Note (b))
6	ly	I	1	Input	Second dimension of array c (See Note (b))
7	lz	I	1	Input	Third dimension of array c (See Note (b))
8	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
9	ifax	I*	60	Output	Factorization results and number of factors (See Note (d))
10	trigs	{D* / R*}	2 × (nx + ny + nz)	Output	Trigonometric function table (See Note (d))
11	wk	{Z* / C*}	lx × ly × lz	Work	Work area
12	nt	I	1	Input	Number of tasks to be generated
13	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw $\in \{0, 1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) When the number of data nx, ny or nz can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300(=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) The complex data c_{k_x,k_y,k_z} $(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ and elements of array c are associated as follows.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{c[k_x + lx * (k_y + ly * k_z)]}$$

Similarly, for the complex data $d_{j_x,j_y,j_z}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$.
The adjustable dimensions lx, ly, and lz of array c should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array c. Usually, when nx, for example, is even, lx=nx+1 is set.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1,\ k_z = 0,\cdots,n_z-1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny, nz), you should call this function once, and then use the after-initialization transform 6.9.2 $\left\{\begin{matrix}\text{ASL_hfc3bf}\\ \text{ASL_gfc3bf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.9.2 $\left\{\begin{matrix}\text{ASL_hfc3bf}\\ \text{ASL_gfc3bf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for array c.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period,

the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.9.2 (7).

6.9.2 [DEPRECATED]ASL_hfc3bf, ASL_gfc3bf Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_hfc3bf or ASL_gfc3bf computes the three-dimensional complex Fourier forward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

Backward transform

ASL_hfc3bf or ASL_gfc3bf computes the three-dimensional complex Fourier backward transform (arbitrary radix) for the three-dimensional complex data $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$d_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}$$
$$(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$$

(2) **Usage**

Double precision:

ierr = ASL_hfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_gfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	c	{Z∗ / C∗}	lx×ly×lz	Input	Input data c_{k_x,k_y,k_z} (See Note (b))
				Output	Output results d_{j_x,j_y,j_z} (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array c (See Note (b))
6	ly	I	1	Input	Second dimension of array c (See Note (b))
7	lz	I	1	Input	Third dimension of array c (See Note (b))
8	isw	I	1	Input	Processing switch isw= 1 :Forward transform isw=−1 :Backward transform
9	ifax	I*	60	Input	Factorization results and number of factors (See Note (a))
10	trigs	{D∗ / R∗}	2 × (nx + ny + nz)	Input	Trigonometric function table (See Note (a))
11	wk	{Z∗ / C∗}	lx × ly × lz	Work	Work area
12	nt	I	1	Input	Number of tasks to be generated
13	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) nx≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw $\in \{1, -1\}$

(d) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny, nz) after the including-initialization function 6.9.1 $\left\{\begin{matrix}\text{ASL_hfc3fb}\\ \text{ASL_gfc3fb}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The complex data c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array c are associated as follows.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{c[k_x + lx * (k_y + ly * k_z)]}$$

The adjustable dimensions lx, ly, **and** lz **of array** c **should be set to odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** c**. Usually, when** nx, **for example, is even,** lx=nx+1 **is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the complex data $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1,\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the three-dimensional complex Fourier forward and backward transforms using

$$c_{k_x,k_y,k_z} = \frac{n_x + n_y + n_z}{(k_x+1)+(k_y+1)+(k_z+1)} + \sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

as input data.

(b) Input data

Array cr and ci, nx=5, ny=4, nz=3, lx=5, ly=5, lz=3, isw=1 (Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_hfc3fb , ASL_hfc3bf */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <asl.h>

int main()
{
    int nx = 5; int ny = 4; int nz = 3;
    int lx = 5; int ly = 5; int lz = 3;
    double _Complex *c;
    int isw;
    int ifax[60];
    double *trigs;
    double _Complex *wk;
    int nt = 2;
    int ierr;
    int i,j,k;

    printf( "    *** ASL_hfc3fb , ASL_hfc3bf ***\n" );
    printf( "\n     ** Input **\n" );

    c = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lx*ly*lz) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }

    trigs = ( double * )malloc((size_t)( sizeof(double) * (2*(nx+ny+nz)) ));
    if( trigs == NULL )
    {
        printf( "no enough memory for array trigs\n" );
        return -1;
    }

    wk = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lx*ly*lz) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnz = %6d\n", nz );
    printf( "\tnt = %6d\n", nt );

    for( k=1 ; k<=nz ; k++ )
    {
        for( j=1 ; j<=ny ; j++ )
        {
            for( i=1 ; i<=nx ; i++ )
            {
                c[(i-1)+lx*(j-1)+lx*ly*(k-1)]=(double)(nx+ny+nz)/(double)(i+j+k)
                                             +(double)(i*j*k)/(double)(nx*ny*nz) * _Complex_I;
            }
        }
    }

    printf( "\tc[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
```

```
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
        }
        printf( "\n" );
    }


    isw = 1;
    ierr = ASL_hfc3fb(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly*lz ; i++)
    {
        c[i] /= (double)(nx*ny*nz);
    }

    printf( "\n     ** Output **\n" );

    printf( "\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tc[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_hfc3bf(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tc[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
```

```
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
        }
        printf( "\n" );
    }
    printf( "\tc[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
        }
        printf( "\n" );
    }


    free( c );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_hfc3fb , ASL_hfc3bf ***

 ** Input **
nx =       5
ny =       4
nz =       3
nt =       2
c[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
c[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
c[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)

 ** Output **
< Forward Transform >
ierr =       0
c[ix][iy][1]
(    1.74,    0.25) (   0.102,   -0.16) (   0.137,   -0.05) (   0.202,    0.06)
(   0.108,  -0.189) (  0.0379, -0.0469) (  0.0406, -0.0125) (  0.0525,   0.016)
(   0.125, -0.0784) (   0.034, -0.0168) (  0.0261, 0.00288) (  0.0254,  0.0209)
(   0.152,-0.00492) (  0.0366, 0.00116) (  0.0207,  0.0138) (   0.012,   0.028)
(   0.223,   0.106) (  0.0462,  0.0236) (  0.0177,  0.0292) (-0.00166,  0.0406)
c[ix][iy][2]
(   0.106,  -0.127) (  0.0407, -0.0223) (  0.0315, 0.00295) (  0.0297,  0.0255)
(  0.0419, -0.0317) (-0.00167,-0.00877) (  0.0025,-0.00799) ( 0.00901,-0.00976)
(  0.0317,-0.00698) ( 0.00134,-0.00743) ( 0.00423,-0.00524) ( 0.00924,-0.00424)
(  0.0297,  0.0084) ( 0.00473,-0.00711) ( 0.00655,-0.00336) (  0.0108,  0.0001)
(  0.0318,  0.0285) (  0.0112,-0.00921) (  0.0118,-0.00179) (   0.016, 0.00627)
c[ix][iy][3]
(   0.178, 0.00231) (  0.0403,   0.014) (   0.017,   0.022) ( 0.00125,  0.0329)
(  0.0484, 0.00885) ( 0.00516, -0.0104) ( 0.00849,-0.00569) (  0.0153, -0.0016)
(  0.0244,  0.0163) ( 0.00692, -0.0061) (  0.0076,-0.00159) (  0.0107, 0.00322)
(  0.0129,  0.0239) ( 0.00961, -0.0029) ( 0.00799, 0.00185) ( 0.00849,  0.0078)
( 0.00117,   0.036) (  0.0163,0.000768) (  0.0106, 0.00714) ( 0.00733,  0.0161)
< Backward Transform >
ierr =       0
c[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
c[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
c[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
```

```
(       2,      0.1) (    1.71,      0.2) (     1.5,      0.3) (    1.33,      0.4)
(    1.71,     0.15) (     1.5,      0.3) (    1.33,     0.45) (     1.2,      0.6)
(     1.5,      0.2) (    1.33,      0.4) (     1.2,      0.6) (    1.09,      0.8)
(    1.33,     0.25) (     1.2,      0.5) (    1.09,     0.75) (       1,        1)
```

6.10 THREE-DIMENSIONAL REAL FOURIER TRANSFORM

6.10.1 [DEPRECATED]ASL_qfr3fb, ASL_pfr3fb Three-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_qfr3fb or ASL_pfr3fb obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1)$.

$$c_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$$
$$c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$$
$$c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ for $n_x n_y n_z$ complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ satisfying $c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$, and $c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$, ASL_qfr3fb or ASL_pfr3fb obtains the three-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned}
r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})} \\
&= \sum_{j_z=0}^{n_z-1} \sum_{j_y=0}^{n_y-1} \{c_{0,j_y,j_z} + (-1)^{k_x} \hat{c}_{\frac{n_x}{2},j_y,j_z}\} e^{2\pi\sqrt{-1}(\frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})} \\
&+ 2 \sum_{j_z=0}^{n_z-1} \sum_{j_y=0}^{n_y-1} \sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil - 1} \Re\{c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y} + \frac{j_z k_z}{n_z})}\} \\
&(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)
\end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = c_{\frac{n_x}{2},j_y,j_z}$.

(2) **Usage**

Double precision:

ierr = ASL_qfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	r	{D* / R*}	lx×ly×lz	Input	Input data r_{k_x,k_y,k_z} (Forward transform), or c_{j_x,j_y,j_z} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y,j_z} (Forward transform), or r_{k_x,k_y,k_z} (Backward transform) (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array r (See Note (b))
6	ly	I	1	Input	Second dimension of array r (See Note (b))
7	lz	I	1	Input	Third dimension of array r (See Note (b))
8	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
9	ifax	I*	60	Output	Factorization results and number of factors (See Note (d))
10	trigs	{D* / R*}	nx + 2 × (ny + nz)	Output	Trigonometric function table (See Note (d))
11	wk	{D* / R*}	lx × ly × lz	Work	Work area
12	nt	I	1	Input	Number of tasks to be generated
13	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx≥2, ny≥2, nz≥2

(b) In the case where nx is an odd, nx+1≤lx, ny≤ly, nz≤lz
or if nx is an even, nx+2≤lx, ny≤ly, nz≤lz

(c) lx should be even number.

(d) isw $\in \{0, 1, -1\}$

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) When the number of data nx, ny or nz can be adjusted, the calculations can be performed more efficiently by setting a number for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc.). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300(=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like.

(b) The real data r_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array r are associated as follows.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{r[k_x + lx * (k_y + ly * k_z)]}$$

When computing the backward transform, if nx(=n_x) is odd, then $\mathrm{r[nx + lx * (k_y + ly * k_z)]} = 0$, and when nx is even, then $\mathrm{r[nx + lx * (k_y + ly * k_z)]} = \mathrm{r[nx + 1 + lx * (k_y + ly * k_z)]} = 0$. Also, when entering the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ into array r, the corresponding zeros mentioned above need not be specifically stored.
If we let the real and imaginary parts of the complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ be $\Re\{c_{j_x,j_y,j_z}\}$ and $\Im\{c_{j_x,j_y,j_z}\}$, respectively, the c_{j_x,j_y,j_z} and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j_x,j_y,j_z}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + lx * (j_y + ly * j_z)]}$$
$$\Im\{c_{j_x,j_y,j_z}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * (j_y + ly * j_z)]}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0,0}\} = 0$, and when nx, ny and nz are all even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y,j_z} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$$
$$c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$$
$$c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$$

Here, z^* represents the conjugate complex number of the complex number z. **The adjustable dimensions of array** r **should be set so that** lx/**2,** ly**, and** lz **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** r**. Usually, when** nx**, for example, is (a multiple of 4)+2,** lx=nx+**4 is set.**

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{r}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) To repeatedly compute the transform for the same number of data (nx, ny, nz), you should call this function once, and then use the after-initialization transform 6.10.2 $\left\{\begin{matrix}\text{ASL_qfr3bf}\\ \text{ASL_pfr3bf}\end{matrix}\right\}$, thereafter. This enables processing to be performed more efficiently since initialization (factorization or the creation of trigonometric tables) is performed only once. However, in this case, you must retain the contents of arrays ifax and trigs so they can be used as input to the function 6.10.2 $\left\{\begin{matrix}\text{ASL_qfr3bf}\\ \text{ASL_pfr3bf}\end{matrix}\right\}$.
To perform initialization only by setting isw=0, you need not set input data for array r.

(e) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c (t - iT)}{\pi (t - iT)}$$

(f) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

See the example in Section 6.10.2 (7).

6.10.2 [DEPRECATED]ASL_qfr3bf, ASL_pfr3bf Three-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_qfr3bf or ASL_pfr3bf obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1)$.

$$c_{j_x,j_y,j_z} = \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$$
$$c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$$
$$c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ for $n_x n_y n_z$ complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ satisfying $c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$, and $c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$, ASL_qfr3bf or ASL_pfr3bf obtains the three-dimensional Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &= \sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1} \{c_{0,j_y,j_z} + (-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y,j_z}\} e^{2\pi\sqrt{-1}(\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad + 2\sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil \frac{n_x}{2} \rceil -1} \Re\{c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})}\} \\ &\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1) \end{aligned}$$

Here, $\lceil x \rceil$ represents the minimum integer greater than or equal to x, and $\Re\{z\}$ represents the real part of the complex number z. Also, when n_x is odd, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = 0$, and when n_x is even, $\hat{c}_{\frac{n_x}{2},j_y,j_z} = c_{\frac{n_x}{2},j_y,j_z}$.

(2) **Usage**

Double precision:

ierr = ASL_qfr3bf (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

Single precision:

ierr = ASL_pfr3bf (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Number of data values in the first dimension, n_x (See Note (a))
2	ny	I	1	Input	Number of data values in the second dimension, n_y (See Note (a))
3	nz	I	1	Input	Number of data values in the third dimension, n_z (See Note (a))
4	r	{D∗ / R∗}	lx×ly×lz	Input	Input data r_{k_x,k_y,k_z} (Forward transform), or c_{j_x,j_y,j_z} (Backward transform) (See Note (b))
				Output	Output results c_{j_x,j_y,j_z} (Forward transform), or r_{k_x,k_y,k_z} (Backward transform) (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array r (See Note (b))
6	ly	I	1	Input	Second dimension of array r (See Note (b))
7	lz	I	1	Input	Third dimension of array r (See Note (b))
8	isw	I	1	Input	Processing switch isw= 1: Forward transform isw=−1: Backward transform
9	ifax	I*	60	Input	Factorization results and number of factors (See Note (a))
10	trigs	{D∗ / R∗}	nx + 2 × (ny + nz)	Input	Trigonometric function table (See Note (a))
11	wk	{D∗ / R∗}	lx × ly × lz	Work	Work area
12	nt	I	1	Input	Number of tasks to be generated
13	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx≥2, ny≥2, nz≥2

(b) In the case where nx is an odd, nx+1≤lx, ny≤ly, nz≤lz
or if nx is an even, nx+2≤lx, ny≤ly, nz≤lz

(c) lx should be even number.

(d) isw $\in \{1, -1\}$

(e) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data (nx, ny, nz) after the including-initialization function 6.10.1 $\left\{\begin{matrix}\text{ASL_qfr3fb}\\ \text{ASL_pfr3fb}\end{matrix}\right\}$ has been used. In this case, you must retain the contents of arrays ifax and trigs so they can be used as input in this function.

(b) The real data r_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ and elements of array r are associated as follows.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{r[k_x + lx * (k_y + ly * k_z)]}$$

When computing the backward transform, if nx($=n_x$) is odd, then $\mathrm{r[nx + lx * (k_y + ly * k_z)] = 0}$, and when nx is even, then $\mathrm{r[nx + lx * (k_y + ly * k_z)] = r[nx + 1 + lx * (k_y + ly * k_z)] = 0}$. Also, when entering the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ into array r, the corresponding zeros mentioned above need not be specifically stored.
If we let the real and imaginary parts of the complex data $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ be $\Re\{c_{j_x,j_y,j_z}\}$ and $\Im\{c_{j_x,j_y,j_z}\}$, respectively, the c_{j_x,j_y,j_z} and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\Re\{c_{j_x,j_y,j_z}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + lx * (j_y + ly * j_z)]}$$
$$\Im\{c_{j_x,j_y,j_z}\} \quad \leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * (j_y + ly * j_z)]}$$

From the properties of a real Fourier transform, $\Im\{c_{0,0,0}\} = 0$, and when nx, ny and nz are all even, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\} = 0$. Therefore, even if nonzero values are set for the corresponding elements of array r, they are considered to be zero when processing is performed. Since the elements c_{j_x,j_y,j_z} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ can be obtained according to the following relationships from the symmetry of the real Fourier transform, they need not be assigned as input when computing the backward transform. Also, they are not output when computing the forward transform.

$$c^*_{n_x-j_x,n_y-j_y,n_z-j_z} = c_{j_x,j_y,j_z}$$
$$c^*_{n_x-j_x,j_y,j_z} = c_{j_x,n_y-j_y,n_z-j_z}$$
$$c^*_{n_x-j_x,n_y-j_y,j_z} = c_{j_x,j_y,n_z-j_z}$$

The adjustable dimensions of array r should be set so that lx/2, ly, and lz are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array r. Usually, when nx, for example, is (a multiple of 4)+2, lx=nx+4 is set.

(c) When this function is used to compute the backward transform immediately following the forward transform, the values of the data obtained will be the original data multiplied by the number of data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ be represented by $\hat{r}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$, then the following relationship holds.

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

Therefore, normalization must be performed for the result of either the forward transform or the backward transform. Note that in some of the entries in the Reference Bibliography, the definitions of the forward and backward transforms are reversed from those in this book, and in some of the entries a normalized result is defined.

(d) Since a discrete Fourier transform is assumed to be a periodic function for which the data sequences before and after the transform are assumed to have the number of data (n_x or n_y or n_z) as the period, the number of samples or sampling interval must be set with this taken into account when sampling to approximate the continuous Fourier transform. According to **the sampling theorem**, for a time function $h(t)$ that is bandwidth limited by the frequency f_c, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $h(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{h(iT)\}$ as follows.

$$h(t) = T \sum_{i=-\infty}^{\infty} h(iT) \frac{\sin 2\pi f_c(t - iT)}{\pi(t - iT)}$$

(e) **DEPRECATED** This function will be removed in the future. Use **ASL Unified Interface**, the higher performance alternative implementation instead.

(7) **Example**

(a) Problem

Compute the three-dimensional real Fourier forward and backward transforms using

$$r_{k_x,k_y,k_z} = \frac{(k_x + 1)(k_y + 1)(k_z + 1)}{n_x n_y n_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

as input data.

(b) Input data

Array r, nx=6, ny=4, nz=3, lx=10, ly=5, lz=3, isw=1 (Forward transform), isw=−1 (Backward transform) and nt=2.

(c) Main program

```
/*      C Interface example for ASL_qfr3fb , ASL_qfr3bf */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx =  6;    int ny =  4;    int nz =  3;
    int lx = 10;    int ly =  5;    int lz =  3;
```

```
double *r;
int isw;
int ifax[60];
double *trigs;
double *wk;
int nt = 2;
int ierr;
int i,j,k;

printf( "    *** ASL_qfr3fb , ASL_qfr3bf ***\n" );
printf( "\n     ** Input **\n" );

r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz) ));
if( r == NULL )
{
    printf( "no enough memory for array r\n" );
    return -1;
}

trigs = ( double * )malloc((size_t)( sizeof(double) * (nx+2*(ny+nz)) ));
if( trigs == NULL )
{
    printf( "no enough memory for array trigs\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tnx = %6d\n", nx );
printf( "\tny = %6d\n", ny );
printf( "\tnz = %6d\n", nz );
printf( "\tnt = %6d\n", nt );

for( k=1 ; k<=nz ; k++ )
{
    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            r[(i-1)+lx*(j-1)+lx*ly*(k-1)]=(double)(i*j*k)/(double)(nx*ny*nz) ;
        }
    }
}

printf( "\tr[ix][iy][1]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j        ] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][2]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][3]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
    }
    printf( "\n" );
}


isw = 1;
ierr = ASL_qfr3fb(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

for( i=0 ; i<lx*ly*lz ; i++)
{
    r[i] /= (double)(nx*ny*nz);
}

printf( "\n     ** Output **\n" );
```

```
    printf( "\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tr[ix][iy][1]\n" );
    for( i=0 ; i<nx+2 ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j        ] );
        }
        printf( "\n" );
    }

    printf( "\tr[ix][iy][2]\n" );
    for( i=0 ; i<nx+2 ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tr[ix][iy][3]\n" );
    for( i=0 ; i<nx+2 ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_qfr3bf(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tr[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j        ] );
        }
        printf( "\n" );
    }

    printf( "\tr[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tr[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }


    free( r );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_qfr3fb , ASL_qfr3bf ***

 ** Input **
nx =      6
ny =      4
nz =      3
nt =      2
r[ix][iy][1]
  0.0139      0.0278      0.0417      0.0556
```

```
   0.0278      0.0556      0.0833       0.111
   0.0417      0.0833       0.125       0.167
   0.0556       0.111       0.167       0.222
   0.0694       0.139       0.208       0.278
   0.0833       0.167        0.25       0.333
r[ix][iy][2]
   0.0278      0.0556      0.0833       0.111
   0.0556       0.111       0.167       0.222
   0.0833       0.167        0.25       0.333
    0.111       0.222       0.333       0.444
    0.139       0.278       0.417       0.556
    0.167       0.333         0.5       0.667
r[ix][iy][3]
   0.0417      0.0833       0.125       0.167
   0.0833       0.167        0.25       0.333
    0.125        0.25       0.375         0.5
    0.167       0.333         0.5       0.667
    0.208       0.417       0.625       0.833
     0.25         0.5        0.75           1

 ** Output **
< Forward Transform >
ierr =       0
r[ix][iy][1]
    0.243     -0.0486     -0.0486     -0.0486
        0      0.0486           0     -0.0486
  -0.0347    -0.00508     0.00694       0.019
   0.0601      -0.019      -0.012    -0.00508
  -0.0347     0.00294     0.00694       0.011
     0.02      -0.011    -0.00401     0.00294
  -0.0347     0.00694     0.00694     0.00694
        0    -0.00694           0     0.00694
r[ix][iy][2]
  -0.0608     0.00514      0.0122      0.0192
   0.0351     -0.0192    -0.00702     0.00514
4.63e-18      0.00401   -1.54e-18    -0.00401
    -0.02     0.00401     0.00401     0.00401
  0.00579    0.000847    -0.00116    -0.00316
    -0.01     0.00316       0.002    0.000847
  0.00868   -0.000734    -0.00174    -0.00274
 -0.00501     0.00274       0.001   -0.000734
r[ix][iy][3]
  -0.0608      0.0192      0.0122     0.00514
  -0.0351    -0.00514     0.00702      0.0192
   0.0174    -0.00147    -0.00347    -0.00548
    -0.01     0.00548       0.002    -0.00147
   0.0116    -0.00231    -0.00231    -0.00231
        0     0.00231           0    -0.00231
  0.00868    -0.00274    -0.00174   -0.000734
  0.00501    0.000734      -0.001    -0.00274
< Backward Transform >
ierr =       0
r[ix][iy][1]
   0.0139      0.0278      0.0417      0.0556
   0.0278      0.0556      0.0833       0.111
   0.0417      0.0833       0.125       0.167
   0.0556       0.111       0.167       0.222
   0.0694       0.139       0.208       0.278
   0.0833       0.167        0.25       0.333
r[ix][iy][2]
   0.0278      0.0556      0.0833       0.111
   0.0556       0.111       0.167       0.222
   0.0833       0.167        0.25       0.333
    0.111       0.222       0.333       0.444
    0.139       0.278       0.417       0.556
    0.167       0.333         0.5       0.667
r[ix][iy][3]
   0.0417      0.0833       0.125       0.167
   0.0833       0.167        0.25       0.333
    0.125        0.25       0.375         0.5
    0.167       0.333         0.5       0.667
    0.208       0.417       0.625       0.833
     0.25         0.5        0.75           1
```

6.11 CONVOLUTIONS

6.11.1 ASL_qfcn2d, ASL_pfcn2d Two-Dimensional Convolutions

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ of period (m_x, m_y) satisfying:

$$
\begin{aligned}
f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\
g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\
&\quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1)
\end{aligned}
$$

for arbitrary integers L_x and L_y take nonzero values within their basic periods only for $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$ and $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$. Here, $[0, a] \times [0, b]$ is the direct product region (region contained in the square for which the point (0, 0) and the point (a, b) are diagonal points) on the plane in which the plane coordinates (i, j) lie. At this time, ASL_qfcn2d or ASL_pfcn2d calculates the discrete convolution $p(k_x, k_y)$ defined as follows:

$$
\begin{aligned}
p(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x - i_x, k_y - i_y) \\
&= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} g(j_x, j_y) f(k_x - j_x, k_y - j_y) \\
&\quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1)
\end{aligned}
$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$ and $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ and M_x and M_y are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$ and $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, respectively. The two-dimensional real Fourier transform of $p(k_x, k_y)$ can also be obtained.

(2) **Usage**

Double precision:

ierr = ASL_qfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y)$
2	ny1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y)$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
3	nx2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y)$
4	ny2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y)$
5	r1	$\left\{\begin{matrix} D* \\ R* \end{matrix}\right\}$	lx1×ly1	Input	Values of discrete function $f(i_x, i_y)$ (See Note (a))
				Output	When isw $\geq$ 1, result of two-dimensional real Fourier transform of discrete function $f(i_x, i_y)$ (period (M_x, M_y))
6	lx1	I	1	Input	Adjustable dimension of array r1
7	ly1	I	1	Input	Second dimension of array r1
8	r2	$\left\{\begin{matrix} D* \\ R* \end{matrix}\right\}$	lx2×ly2	Input	Values of discrete function $g(j_x, j_y)$ (See Note (a))
				Output	Value of discrete function $p(k_x, k_y)$ or its two-dimensional real Fourier transform (See Note (b))
9	lx2	I	1	Input	Adjustable dimension of array r2
10	ly2	I	1	Input	Second dimension of array r2
11	mx	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $p(k_x, k_y)$ (See Note (c))
12	my	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $p(k_x, k_y)$ (See Note (c))
13	isw	I	1	Input	Processing switch (See Note (d)) isw= 0: Calculate the convolution according to the definition. isw= 1: Calculate the convolution according to the FFT method. isw= 2: Calculate the real Fourier transform of the convolution.
14	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 40 (When isw $\geq$ 1)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
15	wk	{D* R*}	See Contents	Work	Work area **Size**: nx2 × ny2 ((When isw= 0 and nx2 is odd) (nx2 + 1) × ny2 (When isw= 0 and nx2 is even) $mx+2\times my+\max(lx1\times ly1, lx2\times ly2)$ (When isw ≥ 1)
16	nt	I	1	Input	Number of tasks to be generated
17	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 > 1 and ny1 > 1

(c) nx2 > 1 and ny2 > 1

(d) mx $\geq$ MAX(nx1, nx2) and my $\geq$ MAX(ny1, ny2)

(e) lx1 $\geq$ nx1 and ly1 $\geq$ ny1 (When isw=0)
lx1 $\geq$ mx + 1 and ly1 $\geq$ my (When isw $\geq$ 1 and mx is odd)
lx1 $\geq$ mx + 2 and ly1 $\geq$ my (When isw $\geq$ 1 and mx is even)

(f) lx2 $\geq$ mx and ly2 $\geq$ my (When isw=0)
lx2 $\geq$ mx + 1 and ly2 $\geq$ my (When isw $\geq$ 1 and mx is odd)
lx2 $\geq$ mx + 2 and ly2 $\geq$ my (When isw $\geq$ 1 and mx is even)

(g) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1+nx2−1 or my < ny1+ny2−1	Overlapping occurred during the convolution calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ and the elements of arrays r1 and r2 are associated as follows.

$$f(i_x, i_y) \leftrightarrow \mathrm{r1[i_x + lx1 * i_y]}$$
$$g(j_x, j_y) \leftrightarrow \mathrm{r2[j_x + lx2 * j_y]}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1;\ i_y = 0, \cdots, n_y^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1;\ j_y = 0, \cdots, n_y^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** r1 **and** r2 **should be set so that** lx1**/2,** ly1**,** lx2**/2, and** ly2 **are odd numbers to avoid bank conflict of main memory. Usually, when** mx**, for example, is a multiple of 4,** lx1=mx**+3 is set.**

(b) The values of the discrete convolution $p(k_x, k_y)$ and the elements of array r2 are associated as follows.

$$p(k_x, k_y) \leftrightarrow \mathrm{r2[k_x + lx2 * k_y]}$$

Here, $k_x = 0, \cdots, M_x - 1;\ k_y = 0, \cdots, M_y - 1$. When isw=2 is set to obtain the two-dimensional real Fourier transform $P(j_x, j_y)$ of the discrete convolution $p(k_x, k_y)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$P(j_x, j_y) = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} p(k_x, k_y) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y}\right)}$$
$$\left(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor\right)$$

the following associations are made:

$$\Re\{P(j_x, j_y)\} \leftrightarrow \mathrm{r2[2 * j_x + lx2 * j_y]}$$
$$\Im\{P(j_x, j_y)\} \leftrightarrow \mathrm{r2[2 * j_x + 1 + lx2 * j_y]}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$P(M_x - j_x, M_y - j_y)^* = P(j_x, j_y)$$
$$P(M_x - j_x, j_y)^* = P(j_x, M_y - j_y)$$

(Here, z^* represents the conjugate complex number of the complex number z.)

(c) If mx $\geq$ nx1+nx2$-$1 and my $\geq$ ny1+ny2$-$1 are set, the convolution can be calculated without causing an overlap with the convolution of the next period. When mx $>$ nx1 + nx2 $-$ 1 or my $>$ ny1 + ny2 $-$ 1, the following correspondences are made:

$$p(k_x, k_y) \leftrightarrow \mathrm{r2[k_x + lx2 * k_y]}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{nx1 + nx2} - 1, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1$ or $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = \mathrm{ny1 + ny2} - 1, \cdots, \mathrm{my} - 1$. When isw=0, mx = nx1 + nx2 $-$ 1 and my = ny1 + ny2 $-$ 1 should be set. When isw $\geq$ 1, the calculations can be performed more efficiently by setting a value for mx or my for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if nx1=nx2=145, then when isw=0, mx = 289(=17^2) should be set. However, when isw $\geq$ 1, it is usually more efficient to set mx = 300(=$2^2 \times 3 \times 5^2$), mx = 320(=$2^6 \times 5$), mx = 384(=$2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** isw=**1 to calculate the FFT convolution.** However, to conserve work area or if there is a restriction on the method of selecting the parameter mx or my, the calculations should be performed by setting isw=0.

(e) To calculate the convolution of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$ be defined as follows:

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

and apply this function to $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$. Let $\hat{p}(k_x, k_y)$ represent the result that was obtained, and the convolution $p(k_x, k_y)$ of the original functions $f(i_x, i_y)$ and $g(j_x, j_y)$ is given as follows:

$$p(k_x, k_y) = \hat{p}(k_x + (i_0 + j_0), k_y)$$

That is, the desired results are obtained if you shift $f(i_x, i_y)$ and $g(j_x, j_y)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete convolution, and then shift the calculated value of the convolution after applying this function by $i_0 + j_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y.

(f) The sampling interval squared multiplied by the discrete convolution calculated by this function is the square approximation (or approximation by using the trapezoidal formula) of the continuous convolution integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous convolution, it is easiest to let $p(n_x^{(f)} + n_x^{(g)} - 1, k_y) = 0$ and $p(k_x, n_y^{(f)} + n_y^{(g)} - 1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ data of $p(k_x, k_y)$ ($k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1$; $k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1$). In this case, the coordinate (0, 0) element usually is associated with $p(0, 0)$. However,
When isw=0,
then $\mathrm{lx1} = \mathrm{nx1}, \mathrm{ly1} = \mathrm{ny1}, \mathrm{lx2} = \mathrm{mx}, \mathrm{ly2} = \mathrm{my}$, and
$\mathrm{nwk} = \mathrm{nx2} \times \mathrm{ny2}$(when nx2 is odd) or
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times \mathrm{ny2}$(when nx2 is even)
When isw ≥ 1,
then lx1=lx2=mx+1 (when mx is odd) or
lx1=lx2=mx+2 (when mx is even),
ly1=ly2=my, and $\mathrm{nwk} = \mathrm{mx} + (\mathrm{lx1} + 2) \times \mathrm{my}$.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete convolution.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x, y) = \begin{cases} x_g - x & ((x, y) \in [0, x_g] \times [0, y_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data
Sampling data
$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * \mathrm{i_y}] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta)$ ($\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1$; $\mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1$),
$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * \mathrm{j_y}] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ ($\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1$; $\mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1$).

Here, $\Delta = 0.5$.

nx1, ny1, nx2, ny2, mx, my and isw.

(c) Main program

```
/*      C interface example for ASL_qfcn2d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nx2;
    int ny2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    double *r2;
    int lx2;
    int ly2;
    int mx;
    int my;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
    int nwk;
    int ierr;
    int i,j;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0;
    double xg=2.0,yg=2.0;


    printf( "    *** ASL_qfcn2d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
    ny1=(int) yf/dt;
    nx2=(int) xg/dt;
    ny2=(int) yg/dt;
    mx=my=m0;
    lx1=lx2=m0+2;
    ly1=ly2=m0;
    nwk=mx+2*my+lx2*my;

    r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1) ));
    if( r1 == NULL )
    {
        printf( "no enough memory for array r1\n" );
        return -1;
    }
    r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2) ));
    if( r2 == NULL )
    {
        printf( "no enough memory for array r2\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\t isw = %6d\n\t (nx1, ny1) = (%3d,%3d)\n",
        isw, nx1, ny1);
    printf( "\t (nx2, ny2) = (%3d,%3d)\n",
        nx2, ny2);
    printf( "\t (mx , my ) = (%3d,%3d)\n\n",
        mx, my);

    for( j=0 ; j<ny1 ; j++ )
        for( i=0 ; i<nx1 ; i++ )
        {
            t=i*dt;
            r1[i+lx1*j]=t;
        }
```

```
    for( j=0 ; j<ny2 ; j++ )
        for( i=0 ; i<nx2 ; i++ )
        {
            t=i*dt;
            r2[i+lx2*j]=xg-t;
        }
    printf( "\tData r1[i+%3d*j]\n", lx1 );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx1 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny1 ; j++ )
            printf( "%8.3g", r1[i+lx1*j] );
        printf( "\n");
    }
    printf( "\n");
    printf( "\tData r2[i+%3d*j]\n", lx2 );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx2 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny2 ; j++ )
            printf( "%8.3g", r2[i+lx2*j] );
        printf( "\n");
    }

    ierr = ASL_qfcn2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
         r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tConvolution r2[i+%3d*j]\n", lx2 );
    printf( "\ti/j   0      1      2      3      4" );
    printf( "      5      6      7\n" );
    printf( "\t------------------------------------");
    printf( "--------------------\n" );
    for( i=0 ; i<mx ; i++ )
    {
        printf( "\t%2d", i );
        for( j=0 ; j<my ; j++ )
            printf( "%7.2lf", r2[i+lx2*j] );
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) Output results

```
*** ASL_qfcn2d ***

 ** Input **

 isw =       1
 (nx1, ny1) = (  4,  4)
 (nx2, ny2) = (  4,  4)
 (mx , my ) = (  8,  8)

Data r1[i+ 10*j]
i/j       0       1       2       3
------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*j]
i/j       0       1       2       3
------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

 ** Output **

ierr =       0
Convolution r2[i+ 10*j]
```

i/j	0	1	2	3	4	5	6	7
0	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00
1	1.00	2.00	3.00	4.00	3.00	2.00	1.00	-0.00
2	2.75	5.50	8.25	11.00	8.25	5.50	2.75	-0.00
3	5.00	10.00	15.00	20.00	15.00	10.00	5.00	-0.00
4	3.50	7.00	10.50	14.00	10.50	7.00	3.50	-0.00
5	2.00	4.00	6.00	8.00	6.00	4.00	2.00	-0.00
6	0.75	1.50	2.25	3.00	2.25	1.50	0.75	-0.00
7	-0.00	0.00	0.00	0.00	-0.00	-0.00	-0.00	-0.00

6.11.2 ASL_qfcn3d, ASL_pfcn3d Three-Dimensional Convolutions

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ of period (m_x, m_y, m_z) satisfying:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ &\quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

for arbitrary integers L_x, L_y, and L_z take nonzero values within their basic periods only for $(i_x, i_y, i_z) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1] \times [0, n_z^{(f)} - 1]$ and $(j_x, j_y, j_z) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1] \times [0, n_z^{(g)} - 1]$. Here, $[0, a] \times [0, b] \times [0, c]$ is the direct product region (region contained in the cube for which the point (0, 0, 0) and the point (a, b, c) are diagonal points) in the space in which the space coordinates (i, j, k) lie. At this time, ASL_qfcn3d or ASL_pfcn3d calculates the discrete convolution $p(k_x, k_y, k_z)$ defined as follows:

$$\begin{aligned} p(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x - i_x, k_y - i_y, k_z - i_z) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} \sum_{j_z=0}^{m_z-1} g(j_x, j_y, j_z) f(k_x - j_x, k_y - j_y, k_z - j_z) \\ &\quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$, and $m_z = \min(n_z^{(f)} + n_z^{(g)} - 1, M_z)$ and M_x, M_y, and M_z are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, and $M_z \geq \max(n_z^{(f)}, n_z^{(g)})$, respectively. The three-dimensional real Fourier transform of $p(k_x, k_y, k_z)$ can also be obtained.

(2) **Usage**

Double precision:

ierr = ASL_qfcn3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfcn3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real   Z:Double precision complex   I: { int as for 32bit Integer ; long as for 64bit Integer }
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
2	ny1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
3	nz1	I	1	Input	Number of effective data in i_z direction $n_z^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
4	nx2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
5	ny2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
6	nz2	I	1	Input	Number of effective data in j_z direction $n_z^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
7	r1	{ D* ; R* }	lx1×ly1× lz1	Input	Values of discrete function $f(i_x, i_y, i_z)$ (See Note (a))
				Output	When isw $\geq$ 1, result of three-dimensional real Fourier transform of discrete function $f(i_x, i_y, i_z)$ (period (M_x, M_y, M_z))
8	lx1	I	1	Input	Adjustable dimension of array r1
9	ly1	I	1	Input	Second dimension of array r1
10	lz1	I	1	Input	Third dimension of array r1
11	r2	{ D* ; R* }	lx2×ly2× lz2	Input	Values of discrete function $g(j_x, j_y, j_z)$ (See Note (a))
				Output	Value of discrete function $p(k_x, k_y, k_z)$ or its three-dimensional real Fourier transform (See Note (b))
12	lx2	I	1	Input	Adjustable dimension of array r2
13	ly2	I	1	Input	Second dimension of array r2
14	lz2	I	1	Input	Third dimension of array r2

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
15	mx	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))
16	my	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))
17	mz	I	1	Input	Parameter M_z corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $p(k_x, k_y, k_z)$ (See Note (c))
18	isw	I	1	Input	Processing switch (See Note (d)) isw= 0: Calculate the convolution according to the definition. isw= 1: Calculate the convolution according to the FFT method. isw= 2: Calculate the real Fourier transform of the convolution.
19	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 60 (When isw $\geq$ 1)
20	wk	$\begin{Bmatrix} \text{D*} \\ \text{R*} \end{Bmatrix}$	See Contents	Work	Work area **Size**: (nx2 + 1) × (ny2 + 1) × nz2 (When isw= 0, nx2 is even and ny2 is even) nx2 × (ny2 + 1) × nz2 (When isw= 0, nx2 is odd and ny2 is even) (nx2 + 1) × ny2 × nz2 (When isw= 0, nx2 is even and ny2 is odd) nx2 × ny2 × nz2 (When isw= 0, nx2 is odd and ny2 is odd) mx+2×(my+mz)+max(lx1×ly1×lz1, lx2×ly2×lz2) (When isw $\geq$ 1)
21	nt	I	1	Input	Number of tasks to be generated
22	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 > 1 and ny1 > 1 and nz1 > 1

(c) nx2 > 1 and ny2 > 1 and nz2 > 1

(d) mx $\geq$ MAX(nx1, nx2) and my $\geq$ MAX(ny1, ny2) and mz $\geq$ MAX(nz1, nz2)

(e) lx1 $\geq$ nx1 and ly1 $\geq$ ny1 and lz1 $\geq$ nz1 (when isw=0)
lx1 $\geq$ mx + 1 and lx1 is even and ly1 $\geq$ my and lz1 $\geq$ mz (when isw ≥ 1 and mx is odd)
lx1 $\geq$ mx + 2 and lx1 is even and ly1 $\geq$ my and lz1 $\geq$ mz (when isw ≥ 1 and mx is even)

(f) lx2 $\geq$ mx and ly2 $\geq$ ny2 and lz2 $\geq$ nz2 (when isw=0)
lx2 $\geq$ mx + 1 and lx2 is even and ly2 $\geq$ my and lz2 $\geq$ mz (when isw ≥ 1 and mx is odd)
lx2 $\geq$ mx + 2 and lx2 is even and ly2 $\geq$ my and lz2 $\geq$ mz (when isw ≥ 1 and mx is even)

(g) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1+nx2−1 or my < ny1+ny2−1 or mz < nz1 + nz2 − 1	Overlapping occurred during the convolution calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ and the elements of arrays r1 and r2 are associated as follows.

$$f(i_x, i_y, i_z) \quad \leftrightarrow \quad \mathrm{r1[i_x + lx1 * (i_y + ly1 * i_z)]}$$
$$g(j_x, j_y, j_z) \quad \leftrightarrow \quad \mathrm{r2[j_x + lx2 * (j_y + ly2 * j_z)]}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1;\ i_y = 0, \cdots, n_y^{(f)} - 1;\ i_z = 0, \cdots, n_z^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1;\ j_y = 0, \cdots, n_y^{(g)} - 1;\ j_z = 0, \cdots, n_z^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** r1 **and** r2 **should be set so that** lx1**/2,** ly1**,** lz1**,** lx2**/2,** ly2**, and** lz2 **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays** r1 **and** r2**. Usually, when** mx**, for example, is (a multiple of 4)+2,** lx1=mx**+4 is set.**

(b) The values of the discrete convolution $p(k_x, k_y, k_z)$ and the elements of array r2 are associated as follows.

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

Here, $k_x = 0, \cdots, M_x - 1;\ k_y = 0, \cdots, M_y - 1;\ k_z = 0, \cdots, M_z - 1$. When isw=2 is set to obtain the three-dimensional real Fourier transform $P(j_x, j_y, j_z)$ of the discrete convolution $p(k_x, k_y, k_z)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$P(j_x, j_y, j_z) = \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} p(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor)$$

the following associations are made:

$$\Re\{P(j_x, j_y, j_z)\} \leftrightarrow \mathrm{r2}[2 * \mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})]$$
$$\Im\{P(j_x, j_y, j_z)\} \leftrightarrow \mathrm{r2}[2 * \mathrm{j_x} + 1 + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})]$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$P(M_x - j_x, M_y - j_y, M_z - j_z)^* = P(j_x, j_y, j_z)$$
$$P(M_x - j_x, j_y, j_z)^* = P(j_x, M_y - j_y, M_z - j_z)$$
$$P(M_x - j_x, M_y - j_y, j_z)^* = P(j_x, j_y, M_z - j_z)$$

(Here, z^* represents the conjugate complex number of the complex number z.)

(c) If $\mathrm{mx} \geq \mathrm{nx1} + \mathrm{nx2} - 1$ and $\mathrm{my} \geq \mathrm{ny1} + \mathrm{ny2} - 1$ and $\mathrm{mz} \geq \mathrm{nz1} + \mathrm{nz2} - 1$ are set, the convolution can be calculated without causing an overlap with the convolution of the next period. When $\mathrm{mx} > \mathrm{nx1}+\mathrm{nx2}-1$ or $\mathrm{my} > \mathrm{ny1} + \mathrm{ny2} - 1$ or $\mathrm{mz} > \mathrm{nz1} + \mathrm{nz2} - 1$, the following correspondences are made:

$$p(k_x, k_y, k_z) \leftrightarrow \mathrm{r2}[\mathrm{k_x} + \mathrm{lx2} * (\mathrm{k_y} + \mathrm{ly2} * \mathrm{k_z})]$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ or $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ or $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = \mathrm{nz1} + \mathrm{nz2} - 1, \cdots, \mathrm{mz} - 1$. When isw=0, $\mathrm{mx} = \mathrm{nx1}+\mathrm{nx2}-1$, $\mathrm{my} = \mathrm{ny1}+\mathrm{ny2}-1$, and $\mathrm{mz} = \mathrm{nz1}+\mathrm{nz2}-1$ should be set. When isw ≥ 1, the calculations can be performed more efficiently by setting a value for mx, my or mz for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if nx1=nx2=145, then when isw=0, $\mathrm{mx} = 289 (=17^2)$ should be set. However, when isw ≥ 1, it is usually more efficient to set $\mathrm{mx} = 300(=2^2 \times 3 \times 5^2)$, $\mathrm{mx} = 320(=2^6 \times 5)$, $\mathrm{mx} = 384(=2^7 \times 3)$ or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** isw**=1 to calculate the FFT convolution.** However, to conserve work area or if there is a restriction on the method of selecting the parameter mx, my or mz, the calculations should be performed by setting isw=0.

(e) To calculate the convolution of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$ be defined as follows:

$$\hat{f}(i_x, i_y, i_z) = f(i_x - i_0, i_y, i_z), \quad \hat{g}(j_x, j_y, j_z) = g(j_x - j_0, j_y, j_z)$$

and apply this function to $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$. Let $\hat{p}(k_x, k_y, k_z)$ represent the result that was obtained, and the convolution $p(k_x, k_y, k_z)$ of the original functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ is given as follows:

$$p(k_x, k_y, k_z) = \hat{p}(k_x + (i_0 + j_0), k_y, k_z)$$

That is, the desired results are obtained if you shift $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete convolution, and then shift the calculated value of the convolution after applying this function by $i_0 + j_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y and i_z, j_z, and k_z.

(f) The sampling interval cubed multiplied by the discrete convolution calculated by this function is the square approximation (or approximation by using the trapezoidal formula) of the continuous convolution integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous convolution, it is easiest to let $p(n_x^{(f)}+n_x^{(g)}-1, k_y, k_z) = 0$, $p(k_x, n_y^{(f)}+n_y^{(g)}-1, k_z) = 0$, and $p(k_x, k_y, n_z^{(f)}+n_z^{(g)}-1) = 0$ and consider $(n_x^{(f)}+n_x^{(g)})(n_y^{(f)}+n_y^{(g)})(n_z^{(f)}+n_z^{(g)})$ data of $p(k_x, k_y, k_z)$ ($k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1;\ k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1;\ k_z = 0, 1, \cdots, n_z^{(f)} + n_z^{(g)} - 1$). In this case, the coordinate (0, 0, 0) element usually is associated with $p(0, 0, 0)$. However,
When isw=0,
then $\mathrm{lx1} = \mathrm{nx1}, \mathrm{ly1} = \mathrm{ny1}, \mathrm{lz1} = \mathrm{nz1}, \mathrm{lx2} = \mathrm{mx}, \mathrm{ly2} = \mathrm{my}, \mathrm{lz2} = \mathrm{mz}$, and
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times (\mathrm{ny2} + 1) \times \mathrm{nz2}$ (when nx2 is even and ny2 is even) or
$\mathrm{nwk} = \mathrm{nx2} \times (\mathrm{ny2} + 1) \times \mathrm{nz2}$ (when nx2 is odd and ny2 is even) or
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times \mathrm{ny2} \times \mathrm{nz2}$ (when nx2 is even and ny2 is odd) or
$\mathrm{nwk} = \mathrm{nx2} \times \mathrm{ny2} \times \mathrm{nz2}$ (when nx2 is odd and ny2 is odd)
When isw ≥ 1
lx1=lx2=mx+1 (when mx is odd) or
lx1=lx2=mx+2 (when mx is even),
ly1=ly2=my, lz1=lz2=mz, and $\mathrm{nwk} = \mathrm{mx} + 2 \times (\mathrm{my} + \mathrm{mz}) + \mathrm{lx1} \times \mathrm{my} \times \mathrm{mz}$.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete convolution.

$$f(x, y, z) = \begin{cases} x & ((x, y, z) \in [0, x_f] \times [0, y_f] \times [0, z_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x, y, z) = \begin{cases} x_g - x & ((x, y, z) \in [0, x_g] \times [0, y_g] \times [0, z_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data
Sampling data
$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * (\mathrm{i_y} + \mathrm{ly1} * \mathrm{i_z})] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta, \mathrm{i_z}\Delta)$ ($\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1;\ \mathrm{i_z} = 0, 1, \cdots, \mathrm{nz1} - 1$),
$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ ($\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1;\ \mathrm{j_z} = 0, 1, \cdots, \mathrm{nz2} - 1$).
Here, $\Delta = 0.5$.
nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz and isw.

(c) Main program

```
/*      C interface example for ASL_qfcn3d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
```

```
{
    int nx1;
    int ny1;
    int nz1;
    int nx2;
    int ny2;
    int nz2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    int lz1;
    double *r2;
    int lx2;
    int ly2;
    int lz2;
    int mx;
    int my;
    int mz;
    int isw;
    int *iwk;
    int niwk=60;
    double *wk;
    int nwk;
    int ierr;
    int i,j,k;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0,zf=2.0;
    double xg=2.0,yg=2.0,zg=2.0;


    printf( "    *** ASL_qfcn3d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
    ny1=(int) yf/dt;
    nz1=(int) zf/dt;
    nx2=(int) xg/dt;
    ny2=(int) yg/dt;
    nz2=(int) zg/dt;
    mx=my=mz=m0;
    lx1=lx2=(m0+2)/2*2;
    ly1=ly2=my;
    lz1=lz2=mz;
    nwk=mx+2*(my+mz)+lx2*my*mz;

    r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1*lz1) ));
    if( r1 == NULL )
    {
        printf( "no enough memory for array r1\n" );
        return -1;
    }
    r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2*lz2) ));
    if( r2 == NULL )
    {
        printf( "no enough memory for array r2\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\t isw = %6d\n\t (nx1, ny1, nz1) = (%3d,%3d,%3d)\n",
        isw, nx1, ny1, nz1);
    printf( "\t (nx2, ny2, nz2) = (%3d,%3d,%3d)\n",
        nx2, ny2, nz2);
    printf( "\t (mx , my , mz ) = (%3d,%3d,%3d)\n\n",
        mx, my, mz);

    for( k=0 ; k<nz1 ; k++ )
        for( j=0 ; j<ny1 ; j++ )
            for( i=0 ; i<nx1 ; i++ )
            {
                t=i*dt;
                r1[i+lx1*(j+ly1*k)]=t;
            }
    for( k=0 ; k<nz2 ; k++ )
        for( j=0 ; j<ny2 ; j++ )
            for( i=0 ; i<nx2 ; i++ )
            {
```

```
                t=i*dt;
                r2[i+lx2*(j+ly2*k)]=xg-t;
            }
        for( k=0 ; k<nz1 ; k++ )
        {
            printf( "\tData r1[i+%3d*(j+%3d*%3d)]\n", lx1, ly1, k );
            printf( "\ti/j       0       1       2       3\n" );
            printf( "\t------------------------------------\n" );
            for( i=0 ; i<nx1 ; i++ )
            {
                printf( "\t%3d", i );
                for( j=0 ; j<ny1 ; j++ )
                    printf( "%8.3g", r1[i+lx1*(j+ly1*k)] );
                printf( "\n");
            }
            printf( "\n");
        }
        for( k=0 ; k<nz2 ; k++ )
        {
            printf( "\tData r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
            printf( "\ti/j       0       1       2       3\n" );
            printf( "\t------------------------------------\n" );
            for( i=0 ; i<nx2 ; i++ )
            {
                printf( "\t%3d", i );
                for( j=0 ; j<ny2 ; j++ )
                    printf( "%8.3g", r2[i+lx2*(j+ly2*k)] );
                printf( "\n");
            }
            printf( "\n");
        }

        ierr = ASL_qfcn3d(nx1, ny1, nz1, nx2, ny2, nz2,
            r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
            mx, my, mz, isw, iwk, wk, nt);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );

        for( k=0 ; k<mz ; k++ )
        {
            printf( "\tConvolution r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
            printf( "\ti/j   0      1      2      3      4" );
            printf( "      5      6      7\n" );
            printf( "\t-------------------------------------");
            printf( "-------------------\n" );
            for( i=0 ; i<mx ; i++ )
            {
                printf( "\t%2d", i );
                for( j=0 ; j<my ; j++ )
                    printf( "%7.2lf", r2[i+lx2*(j+ly2*k)] );
                printf( "\n");
            }
            printf( "\n");
        }
        free( iwk );
        free( wk );
        free( r2 );
        free( r1 );

        return 0;
    }
```

(d) Output results

```
        *** ASL_qfcn3d ***

         ** Input **

         isw =      1
         (nx1, ny1, nz1) = (  4,  4,  4)
         (nx2, ny2, nz2) = (  4,  4,  4)
         (mx , my , mz ) = (  8,  8,  8)

        Data r1[i+ 10*(j+  8*  0)]
        i/j       0       1       2       3
        ------------------------------------
          0       0       0       0       0
          1     0.5     0.5     0.5     0.5
          2       1       1       1       1
          3     1.5     1.5     1.5     1.5

        Data r1[i+ 10*(j+  8*  1)]
        i/j       0       1       2       3
        ------------------------------------
```

```
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5


 ** Output **

ierr =       0
Convolution r2[i+ 10*(j+  8*  0)]
i/j   0      1      2      3      4      5      6      7
---------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00   0.00
 1   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
 2   2.75   5.50   8.25  11.00   8.25   5.50   2.75   0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00   0.00
 4   3.50   7.00  10.50  14.00  10.50   7.00   3.50   0.00
 5   2.00   4.00   6.00   8.00   6.00   4.00   2.00   0.00
 6   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 7   0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00   0.00

Convolution r2[i+ 10*(j+  8*  1)]
i/j   0      1      2      3      4      5      6      7
---------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
 1   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 2   5.50  11.00  16.50  22.00  16.50  11.00   5.50  -0.00
 3  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
 4   7.00  14.00  21.00  28.00  21.00  14.00   7.00  -0.00
 5   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 6   1.50   3.00   4.50   6.00   4.50   3.00   1.50  -0.00
 7   0.00  -0.00  -0.00   0.00  -0.00  -0.00  -0.00  -0.00

Convolution r2[i+ 10*(j+  8*  2)]
i/j   0      1      2      3      4      5      6      7
---------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00   0.00  -0.00
 1   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 2   8.25  16.50  24.75  33.00  24.75  16.50   8.25  -0.00
 3  15.00  30.00  45.00  60.00  45.00  30.00  15.00  -0.00
 4  10.50  21.00  31.50  42.00  31.50  21.00  10.50  -0.00
 5   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 6   2.25   4.50   6.75   9.00   6.75   4.50   2.25  -0.00
 7   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00  -0.00
```

```
Convolution r2[i+ 10*(j+  8*  3)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0   0.00    0.00    0.00    0.00    0.00    0.00   -0.00    0.00
 1   4.00    8.00   12.00   16.00   12.00    8.00    4.00    0.00
 2  11.00   22.00   33.00   44.00   33.00   22.00   11.00   -0.00
 3  20.00   40.00   60.00   80.00   60.00   40.00   20.00   -0.00
 4  14.00   28.00   42.00   56.00   42.00   28.00   14.00   -0.00
 5   8.00   16.00   24.00   32.00   24.00   16.00    8.00   -0.00
 6   3.00    6.00    9.00   12.00    9.00    6.00    3.00   -0.00
 7  -0.00   -0.00    0.00   -0.00   -0.00    0.00   -0.00    0.00

Convolution r2[i+ 10*(j+  8*  4)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0   0.00   -0.00    0.00    0.00    0.00    0.00   -0.00   -0.00
 1   3.00    6.00    9.00   12.00    9.00    6.00    3.00   -0.00
 2   8.25   16.50   24.75   33.00   24.75   16.50    8.25   -0.00
 3  15.00   30.00   45.00   60.00   45.00   30.00   15.00   -0.00
 4  10.50   21.00   31.50   42.00   31.50   21.00   10.50    0.00
 5   6.00   12.00   18.00   24.00   18.00   12.00    6.00    0.00
 6   2.25    4.50    6.75    9.00    6.75    4.50    2.25   -0.00
 7  -0.00   -0.00    0.00    0.00    0.00   -0.00   -0.00   -0.00

Convolution r2[i+ 10*(j+  8*  5)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0   0.00    0.00    0.00    0.00    0.00    0.00   -0.00    0.00
 1   2.00    4.00    6.00    8.00    6.00    4.00    2.00    0.00
 2   5.50   11.00   16.50   22.00   16.50   11.00    5.50   -0.00
 3  10.00   20.00   30.00   40.00   30.00   20.00   10.00   -0.00
 4   7.00   14.00   21.00   28.00   21.00   14.00    7.00   -0.00
 5   4.00    8.00   12.00   16.00   12.00    8.00    4.00   -0.00
 6   1.50    3.00    4.50    6.00    4.50    3.00    1.50   -0.00
 7   0.00   -0.00    0.00   -0.00   -0.00   -0.00   -0.00    0.00

Convolution r2[i+ 10*(j+  8*  6)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0  -0.00    0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 1   1.00    2.00    3.00    4.00    3.00    2.00    1.00   -0.00
 2   2.75    5.50    8.25   11.00    8.25    5.50    2.75   -0.00
 3   5.00   10.00   15.00   20.00   15.00   10.00    5.00   -0.00
 4   3.50    7.00   10.50   14.00   10.50    7.00    3.50   -0.00
 5   2.00    4.00    6.00    8.00    6.00    4.00    2.00   -0.00
 6   0.75    1.50    2.25    3.00    2.25    1.50    0.75   -0.00
 7  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00

Convolution r2[i+ 10*(j+  8*  7)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 1  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 2  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 3  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 4  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 5  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 6  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 7  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
```

6.12 CORRELATIONS

6.12.1 ASL_qfcr2d, ASL_pfcr2d Two-Dimensional Correlations

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ of period (m_x, m_y) satisfying:

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ &\quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

for arbitrary integers L_x and L_y take nonzero values within their basic periods only for $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$ and $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$. Here, $[0, a] \times [0, b]$ is the direct product region (region contained in the square for which the point (0, 0) and the point (a, b) are diagonal points) on the plane in which the plane coordinates (i, j) lie. At this time, ASL_qfcr2d or ASL_pfcr2d calculates the quantity $\tilde{q}(k_x, k_y)$ obtained by shifting the discrete correlation $q(k_x, k_y)$, which is defined as follows:

$$\begin{aligned} q(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x + i_x, k_y + i_y) \\ &\quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

by $(n_x^{(f)} - 1, n_y^{(f)} - 1)$ in the positive direction for (k_x, k_y), respectively. $\tilde{q}(k_x, k_y)$ is defined as follows:

$$\begin{aligned} \tilde{q}(k_x, k_y) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1)) \\ &\quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$ and $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ and M_x and M_y are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$ and $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, respectively. The two-dimensional real Fourier transform of $q(k_x, k_y)$ can also be obtained.

(2) **Usage**

Double precision:

ierr = ASL_qfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y)$
2	ny1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y)$
3	nx2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y)$
4	ny2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y)$
5	r1	{D* / R*}	lx1×ly1	Input	Values of discrete function $f(i_x, i_y)$ (See Note (a))
				Output	When isw $\geq$ 1, result of two-dimensional real Fourier transform of discrete function $f(i_x, i_y)$ (period (M_x, M_y))
6	lx1	I	1	Input	Adjustable dimension of array r1
7	ly1	I	1	Input	Second dimension of array r1
8	r2	{D* / R*}	lx2×ly2	Input	Values of discrete function $g(j_x, j_y)$ (See Note (a))
				Output	Value of discrete function $\tilde{q}(k_x, k_y)$ or the two-dimensional real Fourier transform of $q(k_x, k_y)$ (See Note (b))
9	lx2	I	1	Input	Adjustable dimension of array r2
10	ly2	I	1	Input	Second dimension of array r2
11	mx	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $\tilde{q}(k_x, k_y)$ (See Note (c))
12	my	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y) of discrete functions $f(i_x, i_y)$, $g(j_x, j_y)$, and $\tilde{q}(k_x, k_y)$ (See Note (c))

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
13	isw	I	1	Input	Processing switch (See Note (d)) isw= 0: Calculate the correlation according to the definition. isw= 1: Calculate the correlation according to the FFT method. isw= 2: Calculate the real Fourier transform of the correlation.
14	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 40 (When isw $\geq$ 1)
15	wk	$\left\{\begin{matrix} D* \\ R* \end{matrix}\right\}$	See Contents	Work	Work area **Size**: nx2 $\times$ ny2 ((When isw= 0 and nx2 is odd) (nx2 + 1) $\times$ ny2 (When isw= 0 and nx2 is even) mx$+2\times$my$+$max(lx1$\times$ly1, lx2$\times$ly2) (When isw $\geq$ 1)
16	nt	I	1	Input	Number of tasks to be generated
17	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 $>$ 1 and ny1 $>$ 1

(c) nx2 $>$ 1 and ny2 $>$ 1

(d) mx $\geq$ MAX(nx1, nx2) and my $\geq$ MAX(ny1, ny2)

(e) lx1 $\geq$ nx1 and ly1 $\geq$ ny1 (When isw=0)
lx1 $\geq$ mx + 1 and ly1 $\geq$ my (When isw $\geq$ 1 and mx is odd)
lx1 $\geq$ mx + 2 and ly1 $\geq$ my (When isw $\geq$ 1 and mx is even)

(f) lx2 $\geq$ mx and ly2 $\geq$ my (When isw=0)
lx2 $\geq$ mx + 1 and ly2 $\geq$ my (When isw $\geq$ 1 and mx is odd)
lx2 $\geq$ mx + 2 and ly2 $\geq$ my (When isw $\geq$ 1 and mx is even)

(g) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1+nx2−1 or my < ny1+ny2−1	Overlapping occurred during the correlation calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ and the elements of arrays r1 and r2 are associated as follows.

$f(i_x, i_y) \quad \leftrightarrow \quad$ r1[ix + lx1 ∗ iy]
$g(j_x, j_y) \quad \leftrightarrow \quad$ r2[jx + lx2 ∗ jy]

Here, $i_x = 0, \cdots, n_x^{(f)} - 1;\ i_y = 0, \cdots, n_y^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1;\ j_y = 0, \cdots, n_y^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays** r1 **and** r2 **should be set so that** lx1**/2,** ly1**,** lx2**/2, and** ly2 **are odd numbers to avoid bank conflict of main memory. Usually, when** mx**, for example, is a multiple of 4,** lx1=mx**+3 is set.**

(b) The values of the discrete correlation $\tilde{q}(k_x, k_y)$ and the elements of array r2 are associated as follows.

$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad$ r2[kx + lx2 ∗ ky]

Here, $k_x = 0, \cdots, M_x - 1;\ k_y = 0, \cdots, M_y - 1$. When isw=2 is set to obtain the two-dimensional real Fourier transform $Q(j_x, j_y)$ of the discrete correlation $q(k_x, k_y)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$Q(j_x, j_y) = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} q(k_x, k_y) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y}\right)}$$
$$\left(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor\right)$$

the following associations are made:

$\Re\{Q(j_x, j_y)\} \quad \leftrightarrow \quad$ r2[2 ∗ jx + lx2 ∗ jy]
$\Im\{Q(j_x, j_y)\} \quad \leftrightarrow \quad$ r2[2 ∗ jx + 1 + lx2 ∗ jy]

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$Q(M_x - j_x, M_y - j_y)^* = Q(j_x, j_y)$$
$$Q(M_x - j_x, j_y)^* = Q(j_x, M_y - j_y)$$

(Here, z^* represents the conjugate complex number of the complex number z.) Now, $Q(j_x, j_y)$ can be thought of as an estimate of the cross spectrum of the original two functions for which the correlation is to be calculated. In this case, M_x and M_y should be thought of as $M_x = n_x^{(f)} + n_x^{(g)}$ and $M_y = n_y^{(f)} + n_y^{(g)}$. In particular, if the original two functions for which the correlation is to be calculated are the same

function, $Q(j_x, j_y)$ corresponds to the raw Fourier periodogram (estimate of the power spectrum), and $Q(j_x, j_y)$ is a real number.

(c) If mx $\geq$ nx1+nx2−1 and my $\geq$ ny1+ny2−1 are set, the correlation can be calculated without causing an overlap with the correlation of the next period. When mx > nx1 + nx2 − 1 or my > ny1 + ny2 − 1, the following correspondences are made:

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * k_y]}$$

and values that match 0.0 within the error range are stored in elements corresponding to k_x = nx1 + nx2 − 1, $\cdots$, mx − 1; $k_y = 0, \cdots,$ my − 1 or $k_x = 0, \cdots,$ mx − 1; k_y = ny1 + ny2 − 1, $\cdots$, my − 1. When isw=0, mx = nx1 + nx2 − 1 and my = ny1 + ny2 − 1 should be set. When isw $\geq$ 1, the calculations can be performed more efficiently by setting a value for mx or my for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if nx1=nx2=145, then when isw=0, mx = 289(=17^2) should be set. However, when isw $\geq$ 1, it is usually more efficient to set mx = 300(=$2^2 \times 3 \times 5^2$), mx = 320(=$2^6 \times 5$), mx = 384(=$2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** isw**=1 to calculate the FFT correlation.** However, to conserve work area or if there is a restriction on the method of selecting the parameter mx or my, the calculations should be performed by setting isw=0.

(e) To calculate the correlation of discrete functions for which the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y)$ and $g(j_x, j_y)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$ be defined as follows:

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

and apply this function to $\hat{f}(i_x, i_y)$ and $\hat{g}(j_x, j_y)$. Let $\tilde{q}(k_x, k_y)$ represent the result that was obtained, and the correlation $q(k_x, k_y)$ of the original functions $f(i_x, i_y)$ and $g(j_x, j_y)$ is given as follows:

$$q(k_x, k_y) = \tilde{q}(k_x - (j_0 - i_0) + (n_x^{(f)} - 1), k_y)$$

Therefore, even when $i_0 = j_0 = 0$, to consider the correlation $q(k_x, k_y)$ that conforms to the normal definition, you must consider shifting the result by $n_x^{(f)} - 1$ in the negative direction of k_x after applying this function or if you shift $f(i_x, i_y)$ and $g(j_x, j_y)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete correlation, you must then shift the calculation result again by $j_0 - i_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y.

(f) The sampling interval squared multiplied by the discrete correlation calculated by this function is the square approximation (or approximation by using the trapezoidal formula) of the continuous correlation integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous correlation, it is easiest to let $q(-n_x^{(f)}, k_y) = \tilde{q}(-1, k_y) = 0$ and $q(k_x, -n_y^{(f)}) = \tilde{q}(k_x, -1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ data of $q(k_x, k_y)$ ($k_x = -n_x^{(f)}, \cdots, -1, 0, 1, \cdots, n_x^{(g)} - 1$; $k_y = -n_y^{(f)}, \cdots, -1, 0, 1, \cdots, n_y^{(g)} - 1$). Of course, this is the same as letting $q(n_x^{(f)} + n_x^{(g)}, k_y) = \tilde{q}(n_x^{(g)}, k_y) = 0$ and $q(k_x, n_y^{(f)} + n_y^{(g)}) = \tilde{q}(k_x, n_y^{(g)}) = 0$ and considering $q(k_x, k_y)$ ($k_x = -(n_x^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_x^{(g)}$; $k_y = -(n_y^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_y^{(g)}$). In this case, the coordinate (0, 0) element usually is associated with $q(0, 0)$. However,

When isw=0,
then $\text{lx1} = \text{nx1}, \text{ly1} = \text{ny1}, \text{lx2} = \text{mx}, \text{ly2} = \text{my}$, and
$\text{nwk} = \text{nx2} \times \text{ny2}$(when nx2 is odd) or
$\text{nwk} = (\text{nx2} + 1) \times \text{ny2}$(when nx2 is even)
When isw ≥ 1,
then lx1=lx2=mx+1 (when mx is odd) or
lx1=lx2=mx+2 (when mx is even),
ly1=ly2=my, and $\text{nwk} = \text{mx} + (\text{lx1} + 2) \times \text{my}$.

(7) **Example**

(a) Problem

Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete correlation.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x, y) = \begin{cases} x_g - x & ((x, y) \in [0, x_g] \times [0, y_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data

Sampling data
$\text{r1}[\text{i}_\text{x} + \text{lx1} * \text{i}_\text{y}] = \text{f}(\text{i}_\text{x}\Delta, \text{i}_\text{y}\Delta)$ $(\text{i}_\text{x} = 0, 1, \cdots, \text{nx1} - 1;\ \text{i}_\text{y} = 0, 1, \cdots, \text{ny1} - 1)$,
$\text{r2}[\text{j}_\text{x} + \text{lx2} * \text{j}_\text{y}] = \text{g}(\text{j}_\text{x}\Delta, \text{j}_\text{y}\Delta)$ $(\text{j}_\text{x} = 0, 1, \cdots, \text{nx2} - 1;\ \text{j}_\text{y} = 0, 1, \cdots, \text{ny2} - 1)$.
Here, $\Delta = 0.5$.
nx1, ny1, nx2, ny2, mx, my and isw.

(c) Main program

```
/*      C interface example for ASL_qfcr2d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nx2;
    int ny2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    double *r2;
    int lx2;
    int ly2;
    int mx;
    int my;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
    int nwk;
    int ierr;
    int i,j;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0;
    double xg=2.0,yg=2.0;


    printf( "    *** ASL_qfcr2d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
    ny1=(int) yf/dt;
```

```
nx2=(int) xg/dt;
ny2=(int) yg/dt;
mx=my=m0;
lx1=lx2=m0+2;
ly1=ly2=m0;
nwk=mx+2*my+lx2*my;

r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1) ));
if( r1 == NULL )
{
    printf( "no enough memory for array r1\n" );
    return -1;
}
r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2) ));
if( r2 == NULL )
{
    printf( "no enough memory for array r2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw = %6d\n\t (nx1, ny1) = (%3d,%3d)\n",
    isw, nx1, ny1);
printf( "\t (nx2, ny2) = (%3d,%3d)\n",
    nx2, ny2);
printf( "\t (mx , my ) = (%3d,%3d)\n\n",
    mx, my);

for( j=0 ; j<ny1 ; j++ )
    for( i=0 ; i<nx1 ; i++ )
    {
        t=i*dt;
        r1[i+lx1*j]=t;
    }
for( j=0 ; j<ny2 ; j++ )
    for( i=0 ; i<nx2 ; i++ )
    {
        t=i*dt;
        r2[i+lx2*j]=xg-t;
    }
printf( "\tData r1[i+%3d*j]\n", lx1 );
printf( "\ti/j       0       1       2       3\n" );
printf( "\t------------------------------------\n" );
for( i=0 ; i<nx1 ; i++ )
{
    printf( "\t%3d", i );
    for( j=0 ; j<ny1 ; j++ )
        printf( "%8.3g", r1[i+lx1*j] );
    printf( "\n");
}
printf( "\n");
printf( "\tData r2[i+%3d*j]\n", lx2 );
printf( "\ti/j       0       1       2       3\n" );
printf( "\t------------------------------------\n" );
for( i=0 ; i<nx2 ; i++ )
{
    printf( "\t%3d", i );
    for( j=0 ; j<ny2 ; j++ )
        printf( "%8.3g", r2[i+lx2*j] );
    printf( "\n");
}


ierr = ASL_qfcr2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
     r2, lx2, ly2, mx, my, isw, iwk, wk, nt);


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

printf( "\tCorrelation r2[i+%3d*j]\n", lx2 );
printf( "\ti/j   0      1      2      3      4" );
printf( "      5      6      7\n" );
printf( "\t-------------------------------------");
printf( "-------------------\n" );
for( i=0 ; i<mx ; i++ )
{
```

```
        printf( "\t%2d", i );
        for( j=0 ; j<my ; j++ )
            printf( "%7.2lf", r2[i+lx2*j] );
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) Output results

```
    *** ASL_qfcr2d ***

     ** Input **

     isw =       1
     (nx1, ny1) = (  4,  4)
     (nx2, ny2) = (  4,  4)
     (mx , my ) = (  8,  8)

    Data r1[i+ 10*j]
    i/j       0       1       2       3
    -------------------------------------
      0       0       0       0       0
      1     0.5     0.5     0.5     0.5
      2       1       1       1       1
      3     1.5     1.5     1.5     1.5

    Data r2[i+ 10*j]
    i/j       0       1       2       3
    -------------------------------------
      0       2       2       2       2
      1     1.5     1.5     1.5     1.5
      2       1       1       1       1
      3     0.5     0.5     0.5     0.5

     ** Output **

    ierr =       0
    Correlation r2[i+ 10*j]
    i/j   0      1      2      3      4      5      6      7
    ---------------------------------------------------------
     0   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
     1   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
     2   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
     3   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
     4   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
     5   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
     6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
     7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
```

6.12.2 ASL_qfcr3d, ASL_pfcr3d Three-Dimensional Correlations

(1) **Function**

Assume that the two multiperiodic discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ of period (m_x, m_y, m_z) satisfying:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ &(i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

for arbitrary integers L_x, L_y, and L_z take nonzero values within their basic periods only for $(i_x, i_y, i_z) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1] \times [0, n_z^{(f)} - 1]$ and $(j_x, j_y, j_z) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1] \times [0, n_z^{(g)} - 1]$. Here, $[0, a] \times [0, b] \times [0, c]$ is the direct product region (region contained in the cube for which the point (0, 0, 0) and the point (a, b, c) are diagonal points) in the space in which the space coordinates (i, j, k) lie. At this time, ASL_qfcr3d or ASL_pfcr3d calculates the quantity $\tilde{q}(k_x, k_y, k_z)$ obtained by shifting the discrete correlation $q(k_x, k_y, k_z)$, which is defined as follows:

$$\begin{aligned} q(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x + i_x, k_y + i_y, k_z + i_z) \\ &(k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

by $(n_x^{(f)} - 1, n_y^{(f)} - 1, n_z^{(f)} - 1)$ in the positive direction for (k_x, k_y, k_z), respectively. $\tilde{q}(k_x, k_y, k_z)$ is defined as follows:

$$\begin{aligned} \tilde{q}(k_x, k_y, k_z) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1), k_z - (n_z^{(f)} - 1)) \\ &(k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

Here, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$, and $m_z = \min(n_z^{(f)} + n_z^{(g)} - 1, M_z)$ and M_x, M_y, and M_z are arbitrary integers satisfying $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, and $M_z \geq \max(n_z^{(f)}, n_z^{(g)})$, respectively. The three-dimensional real Fourier transform of $q(k_x, k_y, k_z)$ can also be obtained.

(2) **Usage**

Double precision:

ierr = ASL_qfcr3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfcr3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real　Z:Double precision complex　I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real　C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx1	I	1	Input	Number of effective data in i_x direction $n_x^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
2	ny1	I	1	Input	Number of effective data in i_y direction $n_y^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
3	nz1	I	1	Input	Number of effective data in i_z direction $n_z^{(f)}$ for discrete function $f(i_x, i_y, i_z)$
4	nx2	I	1	Input	Number of effective data in j_x direction $n_x^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
5	ny2	I	1	Input	Number of effective data in j_y direction $n_y^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
6	nz2	I	1	Input	Number of effective data in j_z direction $n_z^{(g)}$ for discrete function $g(j_x, j_y, j_z)$
7	r1	{D* / R*}	lx1×ly1× lz1	Input	Values of discrete function $f(i_x, i_y, i_z)$ (See Note (a))
				Output	When isw $\geq$ 1, result of three-dimensional real Fourier transform of discrete function $f(i_x, i_y, i_z)$ (period (M_x, M_y, M_z))
8	lx1	I	1	Input	Adjustable dimension of array r1
9	ly1	I	1	Input	Second dimension of array r1
10	lz1	I	1	Input	Third dimension of array r1
11	r2	{D* / R*}	lx2×ly2× lz2	Input	Values of discrete function $g(j_x, j_y, j_z)$ (See Note (a))
				Output	Value of discrete function $\tilde{q}(k_x, k_y, k_z)$ or the three-dimensional real Fourier transform of $q(k_x, k_y, k_z)$ (See Note (b))
12	lx2	I	1	Input	Adjustable dimension of array r2
13	ly2	I	1	Input	Second dimension of array r2
14	lz2	I	1	Input	Third dimension of array r2
15	mx	I	1	Input	Parameter M_x corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
16	my	I	1	Input	Parameter M_y corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))
17	mz	I	1	Input	Parameter M_z corresponding to the period (m_x, m_y, m_z) of discrete functions $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, and $\tilde{q}(k_x, k_y, k_z)$ (See Note (c))
18	isw	I	1	Input	Processing switch (See Note (d)) isw= 0: Calculate the correlation according to the definition. isw= 1: Calculate the correlation according to the FFT method. isw= 2: Calculate the real Fourier transform of the correlation.
19	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 60 (When isw $\geq$ 1)
20	wk	$\begin{Bmatrix} \text{D}* \\ \text{R}* \end{Bmatrix}$	See Contents	Work	Work area **Size**: (nx2 + 1) × (ny2 + 1) × nz2 (When isw= 0, nx2 is even and ny2 is even) nx2 × (ny2 + 1) × nz2 (When isw= 0, nx2 is odd and ny2 is even) (nx2 + 1) × ny2 × nz2 (When isw= 0, nx2 is even and ny2 is odd) nx2 × ny2 × nz2 (When isw= 0, nx2 is odd and ny2 is odd) mx+2×(my+mz)+max(lx1×ly1×lz1, lx2×ly2 × lz2) (When isw $\geq$ 1)
21	nt	I	1	Input	Number of tasks to be generated
22	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 > 1 and ny1 > 1 and nz1 > 1

(c) nx2 > 1 and ny2 > 1 and nz2 > 1

(d) mx $\geq$ MAX(nx1, nx2) and my $\geq$ MAX(ny1, ny2) and mz $\geq$ MAX(nz1, nz2)

(e) lx1 $\geq$ nx1 and ly1 $\geq$ ny1 and lz1 $\geq$ nz1 (when isw=0)
lx1 $\geq$ mx + 1 and lx1 is even and ly1 $\geq$ my and lz1 $\geq$ mz (when isw $\geq$ 1 and mx is odd)
lx1 $\geq$ mx + 2 and lx1 is even and ly1 $\geq$ my and lz1 $\geq$ mz (when isw $\geq$ 1 and mx is even)

(f) lx2 $\geq$ mx and ly2 $\geq$ ny2 and lz2 $\geq$ nz2 (when isw=0)
lx2 $\geq$ mx + 1 and lx2 is even and ly2 $\geq$ my and lz2 $\geq$ mz (when isw $\geq$ 1 and mx is odd)
lx2 $\geq$ mx + 2 and lx2 is even and ly2 $\geq$ my and lz2 $\geq$ mz (when isw $\geq$ 1 and mx is even)

(g) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1+nx2−1 or my < ny1+ny2−1 or mz < nz1 + nz2 − 1	Overlapping occurred during the correlation calculation.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3020	Restriction (c) was not satisfied.	
3030	Restriction (d) was not satisfied.	
3040	Restriction (e) was not satisfied.	
3050	Restriction (f) was not satisfied.	
3060	Restriction (g) was not satisfied.	

(6) **Notes**

(a) The values of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ and the elements of arrays r1 and r2 are associated as follows.

$$f(i_x, i_y, i_z) \leftrightarrow \mathrm{r1[i_x + lx1 * (i_y + ly1 * i_z)]}$$
$$g(j_x, j_y, j_z) \leftrightarrow \mathrm{r2[j_x + lx2 * (j_y + ly2 * j_z)]}$$

Here, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$; $i_z = 0, \cdots, n_z^{(f)} - 1$ and $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$; $j_z = 0, \cdots, n_z^{(g)} - 1$, and no values need be entered in other elements. **The adjustable dimensions of arrays r1 and r2 should be set so that lx1/2, ly1, lz1, lx2/2, ly2, and lz2 are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within arrays r1 and r2. Usually, when mx, for example, is (a multiple of 4)+2, lx1=mx+4 is set.**

(b) The values of the discrete convolution $\tilde{q}(k_x, k_y, k_z)$ and the elements of array r2 are associated as follows.

$$\tilde{q}(k_x, k_y, k_z) \leftrightarrow \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

Here, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$; $k_z = 0, \cdots, M_z - 1$. When isw=2 is set to obtain the three-dimensional real Fourier transform $Q(j_x, j_y, j_z)$ of the discrete correlation $q(k_x, k_y, k_z)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$Q(j_x, j_y, j_z) = \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} q(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z})}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor)$$

the following associations are made:

$$\Re\{Q(j_x, j_y, j_z)\} \leftrightarrow \mathrm{r2[2 * j_x + lx2 * (j_y + ly2 * j_z)]}$$
$$\Im\{Q(j_x, j_y, j_z)\} \leftrightarrow \mathrm{r2[2 * j_x + 1 + lx2 * (j_y + ly2 * j_z)]}$$

In this case, note that the Fourier transform that is obtained is normalized. The remaining half period of the Fourier transform can be obtained from the symmetry of the real Fourier transform as follows:

$$Q(M_x - j_x, M_y - j_y, M_z - j_z)^* = Q(j_x, j_y, j_z)$$
$$Q(M_x - j_x, j_y, j_z)^* = Q(j_x, M_y - j_y, M_z - j_z)$$
$$Q(M_x - j_x, M_y - j_y, j_z)^* = Q(j_x, j_y, M_z - j_z)$$

(Here, z^* represents the conjugate complex number of the complex number z.) Now, $Q(j_x, j_y, j_z)$ can be thought of as an estimate of the cross spectrum of the original two functions for which the correlation is to be calculated. In this case, M_x, M_y, and M_z should be thought of as $M_x = n_x^{(f)} + n_x^{(g)}$, $M_y = n_y^{(f)} + n_y^{(g)}$, and $M_z = n_z^{(f)} + n_z^{(g)}$. In particular, if the original two functions for which the correlation is to be calculated are the same function, $Q(j_x, j_y, j_z)$ corresponds to the raw Fourier periodogram (estimate of the power spectrum), and $Q(j_x, j_y, j_z)$ is a real number.

(c) If $\mathrm{mx} \geq \mathrm{nx1} + \mathrm{nx2} - 1$ and $\mathrm{my} \geq \mathrm{ny1} + \mathrm{ny2} - 1$ and $\mathrm{mz} \geq \mathrm{nz1} + \mathrm{nz2} - 1$ are set, the correlation can be calculated without causing an overlap with the correlation of the next period. When $\mathrm{mx} > \mathrm{nx1}+\mathrm{nx2}-1$ or $\mathrm{my} > \mathrm{ny1} + \mathrm{ny2} - 1$ or $\mathrm{mz} > \mathrm{nz1} + \mathrm{nz2} - 1$, the following correspondences are made:

$$\tilde{q}(k_x, k_y) \leftrightarrow \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1$; $k_y = 0, \cdots, \mathrm{my} - 1$; $k_z = 0, \cdots, \mathrm{mz} - 1$ or $k_x = 0, \cdots, \mathrm{mx} - 1$; $k_y =$

ny1 + ny2 − 1, ⋯ , my − 1; k_z = 0, ⋯ , mz − 1 or k_x = 0, ⋯ , mx − 1; k_y = 0, ⋯ , my − 1; k_z = nz1 + nz2−1, ⋯ , mz−1. When isw=0, mx = nx1+nx2−1, my = ny1+ny2−1, and mz = nz1+nz2−1 should be set. When isw ≥ 1, the calculations can be performed more efficiently by setting a value for mx, my or mz for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, if nx1=nx2=145, then when isw=0, mx = 289(=17^2) should be set. However, when isw ≥ 1, it is usually more efficient to set mx = 300(=$2^2 \times 3 \times 5^2$), mx = 320(=$2^6 \times 5$), mx = 384(=$2^7 \times 3$) or the like.

(d) **Usually, the calculations can be performed more efficiently by setting** isw**=1 to calculate the FFT correlation.** However, to conserve work area or if there is a restriction on the method of selecting the parameter mx, my or mz, the calculations should be performed by setting isw=0.

(e) To calculate the correlation of discrete functions the starting position of the nonzero portions are separated from the origin, first perform the calculations by shifting the functions so that the starting positions are at the origin, and then shift the calculation results again to obtain the final results more efficiently. For example, when the nonzero portions of the discrete functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ are the intervals $[i_0, i_0 + n_x^{(f)} - 1]$ and $[j_0, j_0 + n_x^{(g)} - 1]$ for i_x and j_x, respectively, let $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$ be defined as follows:

$$\hat{f}(i_x, i_y, i_z) = f(i_x - i_0, i_y, i_z), \quad \hat{g}(j_x, j_y, j_z) = g(j_x - j_0, j_y, j_z)$$

and apply this function to $\hat{f}(i_x, i_y, i_z)$ and $\hat{g}(j_x, j_y, j_z)$. Let $\tilde{q}(k_x, k_y, k_z)$ represent the result that was obtained, and the correlation $q(k_x, k_y, k_z)$ of the original functions $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ is given as follows:

$$q(k_x, k_y, k_z) = \tilde{q}(k_x - (j_0 - i_0) + (n_x^{(f)} - 1), k_y, k_z)$$

Therefore, even when $i_0 = j_0 = 0$, to consider the correlation $q(k_x, k_y, k_z)$ that conforms to the normal definition, you must consider shifting the result by $n_x^{(f)} - 1$ in the negative direction of k_x after applying this function or if you shift $f(i_x, i_y, i_z)$ and $g(j_x, j_y, j_z)$ in the negative directions of i_x and j_x by i_0 and j_0, respectively, before calculating the discrete correlation, you must then shift the calculation result again by $j_0 - i_0$ in the positive direction of k_x. Similarly, for i_y, j_y, and k_y and i_z, j_z, and k_z.

(f) The sampling interval cubed multiplied by the discrete correlation calculated by this function is the square approximation (or approximation by using the trapezoidal formula) of the continuous correlation integral of a bandwidth-limited function. Therefore, to raise the approximation precision, you must take a smaller sampling interval and a larger number of sample data. To associate these results with a continuous correlation it is easiest to let $q(-n_x^{(f)}, k_y, k_z) = \tilde{q}(-1, k_y, k_z) = 0$, $q(k_x, -n_y^{(f)}, k_z) = \tilde{q}(k_x, -1, k_z) = 0$, and $q(k_x, k_y, -n_z^{(f)}) = \tilde{q}(k_x, k_y, -1) = 0$ and consider $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})(n_z^{(f)} + n_z^{(g)})$ data of $q(k_x, k_y, k_z)$ ($k_x = -n_x^{(f)}, \cdots, -1, 0, 1, \cdots, n_x^{(g)} - 1$; $k_y = -n_y^{(f)}, \cdots, -1, 0, 1, \cdots, n_y^{(g)} - 1$; $k_z = -n_z^{(f)}, \cdots, -1, 0, 1, \cdots, n_z^{(g)} - 1$). Of course, this is the same as letting $q(n_x^{(f)} + n_x^{(g)}, k_y, k_z) = \tilde{q}(n_x^{(g)}, k_y, k_z) = 0$, $q(k_x, n_y^{(f)} + n_y^{(g)}, k_z) = \tilde{q}(k_x, n_y^{(g)}, k_z) = 0$, and $q(k_x, k_y, n_z^{(f)} + n_z^{(g)}) = \tilde{q}(k_x, k_y, n_y^{(g)}) = 0$ and considering $q(k_x, k_y, k_z)$ ($k_x = -(n_x^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_x^{(g)}$; $k_y = -(n_y^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_y^{(g)}$; $k_z = -(n_z^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_z^{(g)}$). In this case, the coordinate (0, 0, 0) element usually is associated with $q(0, 0, 0)$. However,
When isw=0,
then lx1 = nx1, ly1 = ny1, lz1 = nz1, lx2 = mx, ly2 = my, lz2 = mz, and
nwk = (nx2 + 1) × (ny2 + 1) × nz2 (when nx2 is even and ny2 is even) or
nwk = nx2 × (ny2 + 1) × nz2 (when nx2 is odd and ny2 is even) or
nwk = (nx2 + 1) × ny2 × nz2 (when nx2 is even and ny2 is odd) or
nwk = nx2 × ny2 × nz2 (when nx2 is odd and ny2 is odd)

When isw ≥ 1
lx1=lx2=mx+1 (when mx is odd) or
lx1=lx2=mx+2 (when mx is even),
ly1=ly2=my, lz1=lz2=mz, and nwk = mx + 2 × (my + mz) + lx1 × my × mz.

(7) **Example**

(a) Problem

Use the sampling interval Δ to discretize the two finite waveforms defined by the following equations and calculate the discrete correlation.

$$f(x,y,z) = \begin{cases} x & ((x,y,z) \in [0,x_f] \times [0,y_f] \times [0,z_f]) \\ 0 & \text{(Otherwise)} \end{cases}$$

$$g(x,y,z) = \begin{cases} x_g - x & ((x,y,z) \in [0,x_g] \times [0,y_g] \times [0,z_g]) \\ 0 & \text{(Otherwise)} \end{cases}$$

(b) Input data

Sampling data

$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * (\mathrm{i_y} + \mathrm{ly1} * \mathrm{i_z})] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta, \mathrm{i_z}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1;\ \mathrm{i_z} = 0, 1, \cdots, \mathrm{nz1} - 1)$,

$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1;\ \mathrm{j_z} = 0, 1, \cdots, \mathrm{nz2} - 1)$.

Here, $\Delta = 0.5$.

nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz and isw.

(c) Main program

```
/*      C interface example for ASL_qfcr3d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nz1;
    int nx2;
    int ny2;
    int nz2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    int lz1;
    double *r2;
    int lx2;
    int ly2;
    int lz2;
    int mx;
    int my;
    int mz;
    int isw;
    int *iwk;
    int niwk=60;
    double *wk;
    int nwk;
    int ierr;
    int i,j,k;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0,zf=2.0;
    double xg=2.0,yg=2.0,zg=2.0;


    printf( "    *** ASL_qfcr3d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
```

```
ny1=(int) yf/dt;
nz1=(int) zf/dt;
nx2=(int) xg/dt;
ny2=(int) yg/dt;
nz2=(int) zg/dt;
mx=my=mz=m0;
lx1=lx2=(m0+2)/2*2;
ly1=ly2=my;
lz1=lz2=mz;
nwk=mx+2*(my+mz)+lx2*my*mz;

r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1*lz1) ));
if( r1 == NULL )
{
    printf( "no enough memory for array r1\n" );
    return -1;
}
r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2*lz2) ));
if( r2 == NULL )
{
    printf( "no enough memory for array r2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw = %6d\n\t (nx1, ny1, nz1) = (%3d,%3d,%3d)\n",
    isw, nx1, ny1, nz1);
printf( "\t (nx2, ny2, nz2) = (%3d,%3d,%3d)\n",
    nx2, ny2, nz2);
printf( "\t (mx , my , mz ) = (%3d,%3d,%3d)\n\n",
    mx, my, mz);

for( k=0 ; k<nz1 ; k++ )
    for( j=0 ; j<ny1 ; j++ )
        for( i=0 ; i<nx1 ; i++ )
        {
            t=i*dt;
            r1[i+lx1*(j+ly1*k)]=t;
        }
for( k=0 ; k<nz2 ; k++ )
    for( j=0 ; j<ny2 ; j++ )
        for( i=0 ; i<nx2 ; i++ )
        {
            t=i*dt;
            r2[i+lx2*(j+ly2*k)]=xg-t;
        }
for( k=0 ; k<nz1 ; k++ )
{
    printf( "\tData r1[i+%3d*(j+%3d*%3d)]\n", lx1, ly1, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx1 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny1 ; j++ )
            printf( "%8.3g", r1[i+lx1*(j+ly1*k)] );
        printf( "\n");
    }
    printf( "\n");
}
for( k=0 ; k<nz2 ; k++ )
{
    printf( "\tData r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx2 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny2 ; j++ )
            printf( "%8.3g", r2[i+lx2*(j+ly2*k)] );
        printf( "\n");
    }
    printf( "\n");
}


ierr = ASL_qfcr3d(nx1, ny1, nz1, nx2, ny2, nz2,
```

```
        r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
        mx, my, mz, isw, iwk, wk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    for( k=0 ; k<mz ; k++ )
    {
        printf( "\tCorrelation r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
        printf( "\ti/j   0      1      2      3      4" );
        printf( "      5      6      7\n" );
        printf( "\t-------------------------------------");
        printf( "--------------------\n" );
        for( i=0 ; i<mx ; i++ )
        {
            printf( "\t%2d", i );
            for( j=0 ; j<my ; j++ )
                printf( "%7.2lf", r2[i+lx2*(j+ly2*k)] );
            printf( "\n");
        }
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) Output results

```
*** ASL_qfcr3d ***

 ** Input **

 isw =      1
 (nx1, ny1, nz1) = (  4,  4,  4)
 (nx2, ny2, nz2) = (  4,  4,  4)
 (mx , my , mz ) = (  8,  8,  8)

Data r1[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5
```

```
Data r2[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5


 ** Output **

ierr =      0
Correlation r2[i+ 10*(j+  8*  0)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 1   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
 2   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 3   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
 4   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
 5   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
 6   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00  -0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  1)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 1   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
 2   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 5   0.50   1.00   1.50   2.00   1.50   1.00   0.50  -0.00
 6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  2)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
 1  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
 2  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 3   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
 4   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 5   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 6   0.00   0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 7   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00   0.00

Correlation r2[i+ 10*(j+  8*  3)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 1  17.00  34.00  51.00  68.00  51.00  34.00  17.00  -0.00
 2  16.00  32.00  48.00  64.00  48.00  32.00  16.00  -0.00
 3  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
 4   4.00   8.00  12.00  16.00  12.00   8.00   4.00   0.00
 5   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
 6  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00   0.00
 7  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00

Correlation r2[i+ 10*(j+  8*  4)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
 1  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
 2  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 3   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
 4   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 5   0.75   1.50   2.25   3.00   2.25   1.50   0.75   0.00
 6   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00   0.00  -0.00   0.00  -0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  5)]
i/j   0      1      2      3      4      5      6      7
--------------------------------------------------------
 0   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 1   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
 2   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 5   0.50   1.00   1.50   2.00   1.50   1.00   0.50   0.00
 6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  6)]
```

```
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0   3.00    6.00    9.00   12.00    9.00    6.00    3.00    0.00
 1   4.25    8.50   12.75   17.00   12.75    8.50    4.25   -0.00
 2   4.00    8.00   12.00   16.00   12.00    8.00    4.00   -0.00
 3   2.50    5.00    7.50   10.00    7.50    5.00    2.50   -0.00
 4   1.00    2.00    3.00    4.00    3.00    2.00    1.00    0.00
 5   0.25    0.50    0.75    1.00    0.75    0.50    0.25    0.00
 6   0.00   -0.00   -0.00    0.00    0.00    0.00    0.00    0.00
 7   0.00   -0.00   -0.00   -0.00    0.00    0.00    0.00    0.00

Correlation r2[i+ 10*(j+  8*  7)]
i/j   0       1       2       3       4       5       6       7
---------------------------------------------------------------
 0  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00    0.00    0.00
 1  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 2  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 3  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 4  -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00   -0.00
 5  -0.00   -0.00    0.00    0.00    0.00    0.00   -0.00   -0.00
 6  -0.00   -0.00    0.00    0.00    0.00    0.00    0.00    0.00
 7  -0.00    0.00   -0.00    0.00    0.00    0.00    0.00    0.00
```

6.13 POWER SPECTRUM ANALYSIS

6.13.1 ASL_qfps2d, ASL_pfps2d
Two-Dimensional Fourier Periodograms

(1) **Function**

ASL_qfps2d or ASL_pfps2d obtains the (modified) Fourier periodogram of the series u_{j_x,j_y} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$. The Fourier periodogram p_{k_x,k_y} is defined by the following equation.

$$p_{k_x,k_y} = \frac{\left| \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} w_{j_x}^{(x)} w_{j_y}^{(y)} u_{j_x,j_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \right|^2}{n_x n_y \beta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. $w_{j_x}^{(x)}$ and $w_{j_y}^{(y)}$ are the truncation functions (data windows). For a raw Fourier periodogram, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = 1$ $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ and $\beta = n_x n_y$ are set, and for a modified periodogram β is set as follows:

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1} (w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1} (w_{j_y}^{(y)})^2) & \text{(when a power modification expression according to a data window is used)} \\ n_x n_y & \text{(Otherwise)} \end{cases}$$

The periodogram p_{k_x,k_y} corresponds to a half period (period (n_x, n_y)) and the remainder is obtained from the relationship as follows.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y} &= p_{k_x,k_y} \\ p_{n_x-k_x,k_y} &= p_{k_x,n_y-k_y} \end{aligned}$$

Also, the total power of the corresponding series is as follows.

$$\frac{\sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \{u_{j_x,j_y}\}^2}{n_x n_y}$$

(2) **Usage**

Double precision:

ierr = ASL_qfps2d (nx, ny, r, lx, ly, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfps2d (nx, ny, r, lx, ly, isw, iwk, wk, nt);

(3) Arguments and Return Value

D:Double precision real   Z:Double precision complex   I: { int as for 32bit Integer; long as for 64bit Integer }
R:Single precision real   C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Length n_x in the j_x direction of series u_{j_x,j_y} (See Note (d))
2	ny	I	1	Input	Length n_y in the j_y direction of series u_{j_x,j_y} (See Note (d))
3	r	{D* R*}	lx×ly	Input	Values of series u_{j_x,j_y} (See Note (a))
				Output	Values of Fourier periodogram p_{k_x,k_y} of series u_{j_x,j_y} (See Notes (b) and (c))
4	lx	I	1	Input	Adjustable dimension of array r
5	ly	I	1	Input	Second dimension of array r
6	isw	I	1	Input	Processing switch (See Note (e)) isw= 0: Calculate the raw Fourier periodogram isw=±1: Calculate the periodogram using a user-defined data window isw=±2: Calculate the periodogram using the Hanning window isw=±3: Calculate the periodogram using the Bartlett window isw=±4: Calculate the periodogram using the Welch window isw=±5: Calculate the periodogram using the Parzen window To use a power modification expression according to a data window, set isw > 0; otherwise, set isw < 0.
7	iwk	I*	40	Work	Work area
8	wk	{D* R*}	See Contents	Work	Work area When isw=±1, enter the values of the user-defined data window. (See Note (e)) **Size**: $\mathrm{nx} + 2 \times \mathrm{ny} + \mathrm{lx} \times \mathrm{ly}$
9	nt	I	1	Input	Number of tasks to be generated
10	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx $>$ 1 and ny $>$ 1

(c) lx $\geq$ nx + 1 and ly $\geq$ ny (when nx is odd)
lx $\geq$ nx + 2 and ly $\geq$ ny (when nx is even)

(d) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3030	Restriction (c) was not satisfied.	
3040	Restriction (d) was not satisfied.	
4000	When isw=1, the user-defined data window was $w^{(x)}_{j_x} = 0$ $(j_x = 0, \cdots, n_x - 1)$.	
4010	When isw=1, the user-defined data window was $w^{(y)}_{j_y} = 0$ $(j_y = 0, \cdots, n_y - 1)$.	

(6) **Notes**

(a) The elements of array r and the values of the series u_{j_x,j_y} are associated as follows.

$$u_{j_x,j_y} \quad \leftrightarrow \quad \mathrm{r[j_x + lx * j_y]}$$

Here, $j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$, and no values need be entered in other elements. **The adjustable dimensions of array** r **should be set so that** lx/**2 and** ly **are odd numbers to avoid bank conflict of main memory. Usually, when** nx**, for example, is (a multiple of 4)+2,** lx=nx+**4 is set.**

(b) The values of the Fourier periodogram p_{k_x,k_y} are associated as follows with the elements of array r.

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r[k_x + lx * k_y]} \quad (\mathrm{k_x} = 0, \cdots, \lfloor \tfrac{\mathrm{n_x}}{2} \rfloor;\ \mathrm{k_y} = 0, \cdots, \mathrm{n_y} - 1)$$

$\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

(c) The frequencies (ξ_{k_x}, η_{k_y}) corresponding to obtained Fourier periodogram p_{k_x,k_y} $(k_x = 0, 1, \cdots, n_x - 1;\ k_y = 0, 1, \cdots, n_y - 1)$ are given as follows.

$$\xi_{k_x} = \frac{k_x}{n_x \Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y \Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y \Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

Where Δ is the sampling interval.

(d) The calculations can be performed more efficiently by setting the length nx and ny of the series u_{j_x,j_y} to a value for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300(=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the

like. When the number of data cannot be increased, adjust nx by supplying the required number of zeros at the end of the data to perform the calculations.

(e) The truncation function (data window) can be changed as follows according to the value of the processing switch isw.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{isw}=\pm 2 \text{ (Hanning window)} \\ 1-|2v_j-1| & \text{isw}=\pm 3 \text{ (Bartlett window)} \\ 1-(2v_j-1)^2 & \text{isw}=\pm 4 \text{ (Welch window)} \\ \left\{\begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1-6v_j(v_j-1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1-6v_j(v_{n-j+1}-1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array}\right\} & \text{isw}=\pm 5 \text{ (Parzen window)} \end{cases}$$

Here, $v_j = \frac{j}{n}$, and $j = j_x$ and $n = n_x$ are set for $w^{(x)}_{j_x}$ and $j = j_y$ and $n = n_y$ are set for $w^{(y)}_{j_y}$. Therefore, when the data windows shown above are used, the elements u_{0,j_y} and $u_{j_x,0}$ of the series u_{j_x,j_y} do not affect the calculation of the modified periodogram. To avoid this situation, you should specify values for nx and ny that are larger by 1 than the lengths of the series for which you actually want to calculate the periodogram and set the effective data in elements corresponding to 1 and after for j_x and j_y. The data windows are represented as follows as time (or space) domain functions that are nonzero only for $|x| \le 1$.

$$w(x) = \begin{cases} \dfrac{1+\cos \pi x}{2} = \cos^2 \dfrac{\pi x}{2} & \text{Hanning window} \\ 1-|x| & \text{Bartlett window} \\ 1-x^2 & \text{Welch window} \\ \left\{\begin{array}{ll} 1-6x^2+6|x|^3 & |x| \le \frac{1}{2} \\ 2(1-|x|)^3 & \frac{1}{2} \le |x| \le 1 \end{array}\right\} & \text{Parzen window} \end{cases}$$

Also, to use user-defined data window values $w^{(x)}_{j_x}$ and $w^{(y)}_{j_y}$, set isw $= \pm 1$, set the values in work array wk as follows:

$$\mathrm{wk[j_x] = w^{(x)}_{j_x} \quad (j_x = 0, \cdots, n_x - 1), wk[n_x + j_y] = w^{(y)}_{j_y} \quad (j_y = 0, \cdots, n_y - 1)}$$

and then call this function.

(f) From its definition, the raw periodogram should be regarded as an approximation of a discrete Fourier transform of the autocorrelation function. Since the effective data length of the autocorrelation function of a discrete function having effective number of data n is $2n-1$, approximating the power spectrum of a general function by a raw periodogram corresponds to truncating the function by using a square truncation function $w(k)$ for which one period is given as follows.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n-1 \\ 0 & \text{Otherwise} \end{cases}$$

When the frequency is f for the Fourier transform of the square function, a $\frac{\sin f}{f}$ type function form is assumed having a sidelobe that is not small around the central frequency. Therefore, when a periodic function is sampled, for example, by simply truncating it using a width that is not an integer multiple of one period, since the raw periodogram will be the convolution of the Fourier transform of the periodic function for which the power spectrum is to be obtained and the $\frac{\sin f}{f}$ type function in the frequency domain, an excess frequency component called **leakage** occurs. To suppress this kind of leakage, simple truncation is not performed, and a truncation function having a small sidelobe in the frequency domain, such as the Hanning window, is used. However, in general, the more the leakage is suppressed, the

more the result of the discrete Fourier transform widens and blurs. Therefore, when estimating the power spectrum, you must select a suitable truncation function according to your objectives, that is, according to whether the spectral width or the central frequency is to be the problem, for example.

(g) To raise the resolution (sampling interval in the frequency domain) $\frac{1}{nT}$ of the discrete Fourier transform, you should increase the number of sample data n or increase the sampling interval T. However, to raise the precision of the power spectrum estimate while holding the sampling interval and resolution fixed, a technique is often used of taking m groups of samples for which the number of samples is n, obtaining the modified periodogram for each of those m groups, and then taking the average of those values. In this case, a technique is also proposed in which the m groups of sample data are taken from the series so that they overlap. For details, refer to the Reference Bibliography.

(h) When obtaining the power spectrum, the property related to the frequency transition of the Fourier transform, that is, the multiplication by $e^{2\pi\sqrt{-1}f_0 t}$ in the time (or space) domain, is associated with the shifting of the frequency by f_0 in the frequency domain, and a technique is often used of reducing the number of data points required for the calculation in which the central frequency of the power spectrum is shifted in advance, using the property that the function shape does not change. This kind of operation is known as **modulation**. However, lx=nx+1 (when nx is odd) or
lx=nx+2 (when nx is even)
and ly=ny.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the waveform defined by the following equation and estimate the power spectrum by calculating the Fourier periodogram.

$$f(x, y) = \cos 2\pi f_1 x + \cos 2\pi f_2 y$$

(b) Input data
Sampling data
$\mathrm{r[j_x + lx * j_y] = f(j_x\Delta, j_y\Delta)}$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny} - 1)$.
Here, $\Delta = 0.5$.
nx, ny and isw.

(c) Main program

```
/*      C interface example for ASL_qfps2d */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int n0=8,isw0=4;
    int nx;
    int ny;
    double *r;
    int lx;
    int ly;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
    int nwk;
    int ierr;
    int i,j,m,nd2,is;
    int nt = 2;
    double *p,t,tx,ty,dt,dfx,dfy,f0,f1,f2;

    printf( "    *** ASL_qfps2d ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
nx=n0;
ny=n0;
lx=n0+2;
ly=ny;
nwk=nx+2*ny+lx*ly;

r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*(isw0+2)) ));
if( r == NULL )
{
    printf( "no enough memory for array r\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
p = ( double * )malloc((size_t)( sizeof(double) * (isw0+2) ));
if( p == NULL )
{
    printf( "no enough memory for array p\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw=0, 2 to %6d\n", isw0+1 );
printf( "\t nx=%6d\n\t ny=%6d\n\n", nx,ny );
dt=0.5;
f0=1.0/(2.0*dt);
f1=0.62*f0;
f2=0.14*f0;
nd2=(int) (nx+1)/2;
dfx=1.0/(dt*nx);
dfy=1.0/(dt*ny);
p[isw0+1]=0.0;
for( j=0 ; j<ny ; j++ )
{
    ty=(double) j*dt;
    for( i=0 ; i<nx ; i++ )
    {
        tx=(double) i*dt;
        t=cos(2.0*M_PI*f1*tx)+cos(2.0*M_PI*f2*ty);
        r[i+lx*(j+ly*(isw0+1))]=t;
        p[isw0+1] += (t*t);
    }
}
p[isw0+1] /= (double) (nx*ny);
printf( "\tTime series data r[i+%3d*j]\n", lx);
printf( "  i/j");
for( j=0 ; j<ny ; j++ )
    printf( "%9d", j);
printf( "\n" );
printf( "  -----------------------------------");
printf( "---------------------------------------\n");
for( i=0 ; i<nx ; i++ )
{
    printf( "%5d", i );
    for( j=0 ; j<ny ; j++ )
        printf( "%9.4lf", r[i+lx*(j+ly*(isw0+1))] );
    printf( "\n" );
}
printf( "\n");
printf( "\tTime domain power =%9.4lf\n", p[isw0+1]);
printf( "\tSignal frequency =(%9.4lf, %9.4lf)\n", f1, f2);
is=0;

for( isw=0 ; isw<=isw0 ; isw++ )
{
    for( j=0 ; j<ny ; j++ )
        for( i=0 ; i<nx ; i++ )
            r[i+lx*(j+ly*isw)]=r[i+lx*(j+ly*(isw0+1))];
    if (isw != 0)
        is=isw+1;


    ierr = ASL_qfps2d(nx, ny, &r[lx*ly*isw], lx, ly, is, iwk, wk, nt);
    p[isw]=0.0;
    if (nx%2==0)
    {
        m=nd2-1;
        for( j=0 ; j<ny ; j++ )
            for( i=1 ; i<m ; i++ )
```

```
                p[isw]+=2.0*r[i+lx*(j+ly*isw)];
        for( j=0 ; j<ny ; j++ )
            p[isw]+=r[lx*(j+ly*isw)]+r[m+lx*(j+ly*isw)];

    }
    else
    {
        m=nd2;
        for( j=0 ; j<ny ; j++ )
            for( i=1 ; i<m ; i++ )
                p[isw]+=2.0*r[i+lx*(j+ly*isw)];
        for( j=0 ; j<ny ; j++ )
            p[isw]+=r[lx*(j+ly*isw)];
    }
}

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

isw=0;
printf( "\t(Modified) periodogram (Raw)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=1;
printf( "\t(Modified) periodogram (Hanning)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=2;
printf( "\t(Modified) periodogram (Bartlett)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
```

```
    }
    printf( "\n");

    isw=3;
    printf( "\t(Modified) periodogram (Welch)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  -----------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
        printf( "\n" );
    }
    printf( "\n");

    isw=4;
    printf( "\t(Modified) periodogram (Parzen)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  -----------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
        printf( "\n" );
    }

    free( iwk );
    free( p );
    free( wk );
    free( r );

    return 0;
}
```

(d) Output results

```
    *** ASL_qfps2d ***

     ** Input **

     isw=0, 2 to      5
     nx=     8
     ny=     8

    Time series data r[i+ 10*j]
  i/j       0        1        2        3        4        5        6        7
  ---------------------------------------------------------------------------
    0   2.0000   1.9048   1.6374   1.2487   0.8126   0.4122   0.1237   0.0020
    1   0.6319   0.5367   0.2693  -0.1194  -0.5555  -0.9559  -1.2444  -1.3662
    2   0.2710   0.1759  -0.0915  -0.4803  -0.9163  -1.3168  -1.6053  -1.7270
    3   1.9048   1.8097   1.5423   1.1535   0.7174   0.3170   0.0285  -0.0932
    4   1.0628   0.9676   0.7002   0.3115  -0.1246  -0.5250  -0.8135  -0.9352
    5   0.0489  -0.0462  -0.3136  -0.7024  -1.1384  -1.5388  -1.8274  -1.9491
    6   1.6374   1.5423   1.2748   0.8861   0.4500   0.0496  -0.2389  -0.3606
    7   1.4818   1.3866   1.1192   0.7304   0.2944  -0.1060  -0.3946  -0.5163

    Time domain power =   1.0626
    Signal frequency =(   0.6200,    0.1400)

     ** Output **

    ierr =      0
    (Modified) periodogram (Raw)
    Frequency domain power=   0.9717
  x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
  ----------------------------------------------------------------------
      0.00  0.0158  0.0188  0.0350  0.2150  0.0218  0.2150  0.0350  0.0188
```

```
     0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
     0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
     0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

  (Modified) periodogram (Hanning)
  Frequency domain power=   0.5980
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
     0.00  0.0000  0.0001  0.0054  0.0632  0.0105  0.0632  0.0054  0.0001
     0.25  0.0000  0.0000  0.0013  0.0095  0.0056  0.0236  0.0013  0.0000
     0.50  0.0000  0.0000  0.0000  0.0204  0.0814  0.0204  0.0000  0.0000
     0.75  0.0000  0.0000  0.0000  0.0205  0.0820  0.0205  0.0000  0.0000

  (Modified) periodogram (Bartlett)
  Frequency domain power=   0.5835
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
     0.00  0.0000  0.0000  0.0009  0.0820  0.0109  0.0820  0.0009  0.0000
     0.25  0.0000  0.0000  0.0002  0.0122  0.0025  0.0178  0.0002  0.0000
     0.50  0.0000  0.0005  0.0000  0.0156  0.0855  0.0156  0.0000  0.0005
     0.75  0.0000  0.0004  0.0000  0.0095  0.0762  0.0191  0.0000  0.0004

  (Modified) periodogram (Welch)
  Frequency domain power=   0.7072
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
     0.00  0.0000  0.0000  0.0001  0.1263  0.0124  0.1263  0.0001  0.0000
     0.25  0.0000  0.0000  0.0000  0.0140  0.0014  0.0127  0.0000  0.0000
     0.50  0.0002  0.0003  0.0010  0.0195  0.1065  0.0054  0.0009  0.0003
     0.75  0.0002  0.0003  0.0008  0.0064  0.0941  0.0142  0.0009  0.0003

  (Modified) periodogram (Parzen)
  Frequency domain power=   0.4909
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
     0.00  0.0000  0.0002  0.0093  0.0253  0.0070  0.0253  0.0093  0.0002
     0.25  0.0000  0.0001  0.0022  0.0022  0.0127  0.0279  0.0064  0.0001
     0.50  0.0000  0.0000  0.0006  0.0171  0.0558  0.0323  0.0030  0.0000
     0.75  0.0000  0.0000  0.0013  0.0206  0.0485  0.0214  0.0014  0.0000
```

6.13.2 ASL_qfps3d, ASL_pfps3d Three-Dimensional Fourier Periodograms

(1) **Function**

ASL_qfps3d or ASL_pfps3d obtains the (modified) Fourier periodogram of the series u_{j_x,j_y,j_z} $(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$. The Fourier periodogram p_{k_x,k_y,k_z} is defined by the following equation.

$$p_{k_x,k_y,k_z} = \frac{\left|\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} w_{j_x}^{(x)} w_{j_y}^{(y)} w_{j_z}^{(z)} u_{j_x,j_y,j_z} e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z}\right)}\right|^2}{n_x n_y n_z \beta}$$

$$(k_x = 0,1,\cdots,\lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0,1,\cdots,n_y-1; k_z = 0,1,\cdots,n_z-1)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ and $w_{j_z}^{(z)}$ are the truncation functions (data windows). For a raw Fourier periodogram, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = w_{j_z}^{(z)} = 1$ $(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ and $\beta = n_x n_y n_z$ are set, and for a modified periodogram β is set as follows:

$$\beta = \begin{cases} \left(\sum_{j_x=0}^{n_x-1}(w_{j_x}^{(x)})^2\right)\left(\sum_{j_y=0}^{n_y-1}(w_{j_y}^{(y)})^2\right)\left(\sum_{j_z=0}^{n_z-1}(w_{j_z}^{(z)})^2\right) & \text{(when a power modification expression according to a data window is used)} \\ n_x n_y n_z & \text{(Otherwise)} \end{cases}$$

The periodogram p_{k_x,k_y,k_z} corresponds to a half period (period (n_x, n_y, n_z)) and the remainder is obtained from the relationship as follows.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y,n_z-k_z} &= p_{k_x,k_y,k_z} \\ p_{n_x-k_x,k_y,k_z} &= p_{k_x,n_y-k_y,n_z-k_z} \\ p_{n_x-k_x,n_y-k_y,k_z} &= p_{k_x,k_y,n_z-k_z} \end{aligned}$$

Also, the total power of the corresponding series is as follows.

$$\frac{\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1}\{u_{j_x,j_y,j_z}\}^2}{n_x n_y n_z}$$

(2) **Usage**

Double precision:

ierr = ASL_qfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk, nt);

Single precision:

ierr = ASL_pfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk, nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	nx	I	1	Input	Length n_x in the j_x direction of series u_{j_x,j_y,j_z} (See Note (d))
2	ny	I	1	Input	Length n_y in the j_y direction of series u_{j_x,j_y,j_z} (See Note (d))
3	nz	I	1	Input	Length n_z in the j_z direction of series u_{j_x,j_y,j_z} (See Note (d))
4	r	{D* / R*}	lx×ly× lz	Input	Values of series u_{j_x,j_y,j_z} (See Note (a))
				Output	Values of Fourier periodogram p_{k_x,k_y,k_z} of series u_{j_x,j_y,j_z} (See Notes (b) and (c))
5	lx	I	1	Input	Adjustable dimension of array r
6	ly	I	1	Input	Second dimension of array r
7	lz	I	1	Input	Third dimension of array r
8	isw	I	1	Input	Processing switch (See Note (e)) isw= 0: Calculate the raw Fourier periodogram isw=±1: Calculate the periodogram using a user-defined data window isw=±2: Calculate the periodogram using the Hanning window isw=±3: Calculate the periodogram using the Bartlett window isw=±4: Calculate the periodogram using the Welch window isw=±5: Calculate the periodogram using the Parzen window To use a power modification expression according to a data window, set isw > 0; otherwise, set isw < 0.
9	iwk	I*	60	Work	Work area
10	wk	{D* / R*}	See Contents	Work	Work area When isw=±1, enter the values of the user-defined data window. (See Note (e)) **Size**: nx + 2 × (ny + nz) + lx × ly × lz
11	nt	I	1	Input	Number of tasks to be generated
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx $>$ 1 and ny $>$ 1 and nz $>$ 1

(c) lx $\geq$ nx + 1 and lx is even and ly $\geq$ ny and lz $\geq$ nz (when nx is odd)
lx $\geq$ nx + 2 and lx is even and ly $\geq$ ny and lz $\geq$ nz (when nx is even)

(d) nt $\geq$ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) was not satisfied.	
3030	Restriction (c) was not satisfied.	
3040	Restriction (d) was not satisfied.	
4000	When isw=1, the user-defined data window was $w_{j_x}^{(x)} = 0$ ($j_x = 0, \cdots, n_x - 1$).	
4010	When isw=1, the user-defined data window was $w_{j_y}^{(y)} = 0$ ($j_y = 0, \cdots, n_y - 1$).	
4020	When isw=1, the user-defined data window was $w_{j_z}^{(z)} = 0$ ($j_z = 0, \cdots, n_z - 1$).	

(6) **Notes**

(a) The elements of array r and the values of the series u_{j_x,j_y,j_z} are associated as follows.

$$u_{j_x,j_y,j_z} \quad \leftrightarrow \quad \mathrm{r[j_x + lx * (j_y + ly * j_z)]}$$

Here, $j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ \ j_z = 0, \cdots, n_z - 1$ and no values need be entered in other elements. **The adjustable dimensions of array** r **should be set so that** lx/2**,** ly**, and** lz **are odd numbers to avoid bank conflict of main memory. Also, to increase speed, calculations are executed even for elements outside areas where data is set within array** r**. Usually, when** nx**, for example, is (a multiple of 4)+2,** lx=nx+4 **is set.**

(b) The values of the Fourier periodogram p_{k_x,k_y,k_z} are associated as follows with the elements of array r.

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r[k_x + lx * (k_y + ly * k_z)]}$$
$$(k_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

$\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

(c) The frequencies $(\xi_{k_x}, \eta_{k_y}, \zeta_{k_z})$ corresponding to obtained Fourier periodogram p_{k_x,k_y,k_z} $(k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1\ k_z = 0, \cdots, n_z - 1)$ are given as follows.

$$\xi_{k_x} = \frac{k_x}{n_x\Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y\Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y\Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

$$\zeta_{k_z} = \begin{cases} \dfrac{k_z}{n_z\Delta} & (k_z = 0, 1, \cdots, \lfloor \dfrac{n_z}{2} \rfloor) \\ \dfrac{k_z - n_z}{n_z\Delta} & (k_z = \lfloor \dfrac{n_z}{2} \rfloor + 1, \cdots, n_z - 1) \end{cases}$$

Where Δ is the sampling interval.

(d) The calculations can be performed more efficiently by setting the length nx, ny and nz of the series u_{j_x,j_y,j_z} to a value for which the mixed radix FFT algorithm operates effectively (multiples of 2, 3, 5, etc., which are the mixed radix values of FFT). For example, rather than setting nx = 289(=17^2), it is usually more efficient to set nx = 300(=$2^2 \times 3 \times 5^2$), nx = 320(=$2^6 \times 5$), nx = 384(=$2^7 \times 3$) or the like. When the number of data cannot be increased, adjust nx by supplying the required number of zeros at the end of the data to perform the calculations.

(e) The truncation function (data window) can be changed as follows according to the value of the processing switch isw.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{isw} = \pm 2 \text{ (Hanning window)} \\ 1 - |2v_j - 1| & \text{isw} = \pm 3 \text{ (Bartlett window)} \\ 1 - (2v_j - 1)^2 & \text{isw} = \pm 4 \text{ (Welch window)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & \text{isw} = \pm 5 \text{ (Parzen window)} \end{cases}$$

Here, $v_j = \frac{j}{n}$, and $j = j_x$ and $n = n_x$ are set for $w^{(x)}_{j_x}$, $j = j_y$ and $n = n_y$ are set for $w^{(y)}_{j_y}$, and $j = j_z$ and $n = n_z$ are set for $w^{(z)}_{j_z}$. Therefore, when the data windows shown above are used, the elements u_{0,j_y,j_z}, $u_{j_x,0,j_z}$ and $u_{j_x,j_y,0}$ of the series u_{j_x,j_y,j_z} do not affect the calculation of the modified periodogram. To avoid this situation, you should specify values for nx, ny and nz that are larger by 1 than the lengths of the series for which you actually want to calculate the periodogram and set the effective data in elements corresponding to 1 and after for j_x, j_y and j_z. The data windows are represented as follows as time (or space) domain functions that are nonzero only for $|x| \le 1$.

$$w(x) = \begin{cases} \dfrac{1 + \cos \pi x}{2} = \cos^2 \dfrac{\pi x}{2} & \text{Hanning window} \\ 1 - |x| & \text{Bartlett window} \\ 1 - x^2 & \text{Welch window} \\ \left\{ \begin{array}{ll} 1 - 6x^2 + 6|x|^3 & |x| \le \frac{1}{2} \\ 2(1 - |x|)^3 & \frac{1}{2} \le |x| \le 1 \end{array} \right\} & \text{Parzen window} \end{cases}$$

Also, to use user-defined data window values $w^{(x)}_{j_x}$, $w^{(y)}_{j_y}$ and $w^{(z)}_{j_z}$, set isw = ± 1, set the values in work array wk as follows:

$$\begin{array}{l} \mathrm{wk[j_x] = w^{(x)}_{j_x} \quad (j_x = 0, \cdots, n_x - 1), wk[n_x + j_y] = w^{(y)}_{j_y} \quad (j_y = 0, \cdots, n_y - 1),} \\ \mathrm{wk[n_x + n_y + j_z] = w^{(z)}_{j_z} \quad (j_z = 0, \cdots, n_z - 1)} \end{array}$$

and then call this function.

(f) From its definition, the raw periodogram should be regarded as an approximation of a discrete Fourier transform of the autocorrelation function. Since the effective data length of the autocorrelation function of a discrete function having effective number of data n is $2n - 1$, approximating the power spectrum of a general function by a raw periodogram corresponds to truncating the function by using a square truncation function $w(k)$ for which one period is given as follows.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n - 1 \\ 0 & \text{Otherwise} \end{cases}$$

When the frequency is f for the Fourier transform of the square function, a $\frac{\sin f}{f}$ type function form is assumed having a sidelobe that is not small around the central frequency. Therefore, when a periodic

function is sampled, for example, by simply truncating it using a width that is not an integer multiple of one period, since the raw periodogram will be the convolution of the Fourier transform of the periodic function for which the power spectrum is to be obtained and the $\frac{\sin f}{f}$ type function in the frequency domain, an excess frequency component called **leakage** occurs. To suppress this kind of leakage, simple truncation is not performed, and a truncation function having a small sidelobe in the frequency domain, such as the Hanning window, is used. However, in general, the more the leakage is suppressed, the more the result of the discrete Fourier transform widens and blurs. Therefore, when estimating the power spectrum, you must select a suitable truncation function according to your objectives, that is, according to whether the spectral width or the central frequency is to be the problem, for example.

(g) To raise the resolution (sampling interval in the frequency domain) $\frac{1}{nT}$ of the discrete Fourier transform, you should increase the number of sample data n or increase the sampling interval T. However, to raise the precision of the power spectrum estimate while holding the sampling interval and resolution fixed, a technique is often used of taking m groups of samples for which the number of samples is n, obtaining the modified periodogram for each of those m groups, and then taking the average of those values. In this case, a technique is also proposed in which the m groups of sample data are taken from the series so that they overlap. For details, refer to the Reference Bibliography.

(h) When obtaining the power spectrum, the property related to the frequency transition of the Fourier transform, that is, the multiplication by $e^{2\pi\sqrt{-1}f_0 t}$ in the time (or space) domain, is associated with the shifting of the frequency by f_0 in the frequency domain, and a technique is often used of reducing the number of data points required for the calculation in which the central frequency of the power spectrum is shifted in advance, using the property that the function shape does not change. This kind of operation is known as **modulation**. However, lx=nx+1 (when nx is odd) or
lx=nx+2 (when nx is even)
ly=ny and lz=nz.

(7) **Example**

(a) Problem
Use the sampling interval Δ to discretize the waveform defined by the following equation and estimate the power spectrum by calculating the Fourier periodogram.

$$f(x,y,z) = \cos 2\pi f_1 x + \cos 2\pi f_2 y + \cos 2\pi f_3 z$$

(b) Input data
Sampling data
$\mathrm{r}[\mathrm{j_x} + \mathrm{lx} * (\mathrm{j_y} + \mathrm{ly} * \mathrm{j_z})] = \mathrm{f}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ ($\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx}-1$; $\mathrm{j_y} = 0, 1, \cdots, \mathrm{ny}-1$; $\mathrm{j_z} = 0, 1, \cdots, \mathrm{nz}-1$).
Here, $\Delta = 0.5$.
nx, ny, nz and isw.

(c) Main program

```
/*      C interface example for ASL_qfps3d */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int n0=8,isw0=4;
    int nx;
    int ny;
    int nz;
    double *r;
    int lx;
    int ly;
```

```
int lz;
int isw;
int *iwk;
int niwk=60;
double *wk;
int nwk;
int ierr;
int i,j,k,m,nd2,is;
int nt = 2;
double *p,t,tx,ty,tz,dt,dfx,dfy,dfz,f0,f1,f2,f3;

printf( "    *** ASL_qfps3d ***\n" );
printf( "\n     ** Input **\n\n" );

nx=n0;
ny=n0;
nz=n0;
lx=(n0+2)/2*2;
ly=ny;
lz=nz;
nwk=nx+2*(ny+nz)+lx*ly*lz;

r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz*(isw0+2)) ));
if( r == NULL )
{
    printf( "no enough memory for array r\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
p = ( double * )malloc((size_t)( sizeof(double) * (isw0+2) ));
if( p == NULL )
{
    printf( "no enough memory for array p\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw=0, 2 to %6d\n", isw0+1 );
printf( "\t nx=%6d\n\t ny=%6d\n\t nz=%6d\n\n", nx,ny,nz );
dt=0.5;
f0=1.0/(2.0*dt);
f1=0.62*f0;
f2=0.14*f0;
f3=0.55*f0;
nd2=(int) (nx+1)/2;
dfx=1.0/(dt*nx);
dfy=1.0/(dt*ny);
dfz=1.0/(dt*nz);
p[isw0+1]=0.0;
for( k=0 ; k<nz ; k++ )
{
    tz=(double) k*dt;
    for( j=0 ; j<ny ; j++ )
    {
        ty=(double) j*dt;
        for( i=0 ; i<nx ; i++ )
        {
            tx=(double) i*dt;
            t=cos(2.0*M_PI*f1*tx)+cos(2.0*M_PI*f2*ty)
                +cos(2.0*M_PI*f3*tz);
            r[i+lx*(j+ly*(k+lz*(isw0+1)))]=t;
            p[isw0+1] += (t*t);
        }
    }
}
p[isw0+1] /= (double) (nx*ny*nz);
printf( "\tTime series data\n");
for( k=0 ; k<nz ; k++ )
{
    printf( "\t r[i+%3d*(j+%3d*%3d)]\n", lx,ly,k);
    printf( "  i/j");
    for( j=0 ; j<ny ; j++ )
        printf( "%9d", j);
    printf( "\n" );
    printf( "  -----------------------------------");
    printf( "--------------------------------------\n");
    for( i=0 ; i<nx ; i++ )
    {
        printf( "%5d", i );
        for( j=0 ; j<ny ; j++ )
```

```
            printf( "%9.4lf", r[i+lx*(j+ly*(k+lz*(isw0+1)))] );
        printf( "\n" );
    }
    printf( "\n" );
}
printf( "\n");
printf( "\tTime domain power =%9.4lf\n", p[isw0+1]);
printf( "\tSignal frequency =(%9.4lf, %9.4lf, %9.4lf)\n", f1, f2, f3);
is=0;

for( isw=0 ; isw<=isw0 ; isw++ )
{
    for( k=0 ; k<nz ; k++ )
        for( j=0 ; j<ny ; j++ )
            for( i=0 ; i<nx ; i++ )
                r[i+lx*(j+ly*(k+lz*isw))]=
                    r[i+lx*(j+ly*(k+lz*(isw0+1)))];
    if (isw != 0)
        is=isw+1;

    ierr = ASL_qfps3d(nx, ny, nz, &r[lx*ly*lz*isw],
         lx, ly, lz, is, iwk, wk, nt);
    p[isw]=0.0;
    if (nx%2==0)
    {
        m=nd2-1;
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                for( i=1 ; i<m ; i++ )
                    p[isw]+=2.0
                        *r[i+lx*(j+ly*(k+lz*isw))];
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                p[isw]+=r[lx*(j+ly*(k+lz*isw))]
                    +r[m+lx*(j+ly*(k+lz*isw))];
    }
    else
    {
        m=nd2;
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                for( i=1 ; i<m ; i++ )
                    p[isw]+=2.0
                        *r[i+lx*(j+ly*(k+lz*isw))];
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                p[isw]+=r[lx*(j+ly*(k+lz*isw))];
    }
}


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

isw=0;
printf( "\t(Modified) periodogram (Raw)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
```

```
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  -----------------------------------");
        printf( "-----------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    isw=1;
    printf( "\t(Modified) periodogram (Hanning)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    for( k=(nz+1)/2 ; k<nz ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  -----------------------------------");
        printf( "-----------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    for( k=0 ; k<(nz+1)/2 ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", k*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  -----------------------------------");
        printf( "-----------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    isw=2;
    printf( "\t(Modified) periodogram (Bartlett)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    for( k=(nz+1)/2 ; k<nz ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  -----------------------------------");
        printf( "-----------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
```

```
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    for( k=0 ; k<(nz+1)/2 ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", k*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  ------------------------------------");
        printf( "------------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    isw=3;
    printf( "\t(Modified) periodogram (Welch)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    for( k=(nz+1)/2 ; k<nz ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  ------------------------------------");
        printf( "------------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    for( k=0 ; k<(nz+1)/2 ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", k*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  ------------------------------------");
        printf( "------------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    isw=4;
```

```
    printf( "\t(Modified) periodogram (Parzen)\n");
    printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
    for( k=(nz+1)/2 ; k<nz ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  ------------------------------------");
        printf( "------------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    for( k=0 ; k<(nz+1)/2 ; k++ )
    {
        printf( "\tz-frq=%8.2lf\n", k*dfz );
        printf( "  x/y-freq");
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.2lf", (j-ny)*dfy );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.2lf", j*dfy );
        printf( "\n");
        printf( "  ------------------------------------");
        printf( "------------------------------------\n");
        for( i=0 ; i<nd2 ; i++ )
        {
            printf( "  %8.2lf", i*dfx );
            for( j=(ny+1)/2 ; j<ny ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    free( iwk );
    free( p );
    free( wk );
    free( r );

    return 0;
}
```

(d) Output results

```
    *** ASL_qfps3d ***

     ** Input **

     isw=0, 2 to      5
     nx=     8
     ny=     8
     nz=     8

    Time series data
     r[i+ 10*(j+  8*  0)]
  i/j       0        1        2        3        4        5        6        7
-----------------------------------------------------------------------------
   0   3.0000   2.9048   2.6374   2.2487   1.8126   1.4122   1.1237   1.0020
   1   1.6319   1.5367   1.2693   0.8806   0.4445   0.0441  -0.2444  -0.3662
   2   1.2710   1.1759   0.9085   0.5197   0.0837  -0.3168  -0.6053  -0.7270
   3   2.9048   2.8097   2.5423   2.1535   1.7174   1.3170   1.0285   0.9068
   4   2.0628   1.9676   1.7002   1.3115   0.8754   0.4750   0.1865   0.0648
   5   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
   6   2.6374   2.5423   2.2748   1.8861   1.4500   1.0496   0.7611   0.6394
   7   2.4818   2.3866   2.1192   1.7304   1.2944   0.8940   0.6054   0.4837

     r[i+ 10*(j+  8*  1)]
  i/j       0        1        2        3        4        5        6        7
-----------------------------------------------------------------------------
   0   1.8436   1.7484   1.4810   1.0923   0.6562   0.2558  -0.0327  -0.1545
   1   0.4754   0.3803   0.1129  -0.2759  -0.7119  -1.1123  -1.4009  -1.5226
   2   0.1146   0.0194  -0.2480  -0.6367  -1.0728  -1.4732  -1.7617  -1.8834
   3   1.7484   1.6532   1.3858   0.9971   0.5610   0.1606  -0.1279  -0.2496
```

```
   4   0.9064   0.8112   0.5438   0.1550  -0.2810  -0.6814  -0.9700  -1.0917
   5  -0.1075  -0.2027  -0.4701  -0.8588  -1.2949  -1.6953  -1.9838  -2.1055
   6   1.4810   1.3858   1.1184   0.7297   0.2936  -0.1068  -0.3953  -0.5170
   7   1.3253   1.2301   0.9627   0.5740   0.1379  -0.2625  -0.5510  -0.6727

    r[i+ 10*(j+  8*  2)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
   1  -0.3192  -0.4144  -0.6818  -1.0705  -1.5066  -1.9070  -2.1955  -2.3172
   2  -0.6800  -0.7752  -1.0426  -1.4313  -1.8674  -2.2678  -2.5563  -2.6781
   3   0.9538   0.8586   0.5912   0.2025  -0.2336  -0.6340  -0.9225  -1.0443
   4   0.1117   0.0166  -0.2508  -0.6396  -1.0756  -1.4761  -1.7646  -1.8863
   5  -0.9021  -0.9973  -1.2647  -1.6534  -2.0895  -2.4899  -2.7784  -2.9001
   6   0.6864   0.5912   0.3238  -0.0649  -0.5010  -0.9014  -1.1899  -1.3117
   7   0.5307   0.4355   0.1681  -0.2206  -0.6567  -1.0571  -1.3456  -1.4673

    r[i+ 10*(j+  8*  3)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   2.4540   2.3588   2.0914   1.7027   1.2666   0.8662   0.5777   0.4560
   1   1.0859   0.9907   0.7233   0.3346  -0.1015  -0.5019  -0.7904  -0.9122
   2   0.7250   0.6298   0.3624  -0.0263  -0.4624  -0.8628  -1.1513  -1.2730
   3   2.3588   2.2636   1.9962   1.6075   1.1714   0.7710   0.4825   0.3608
   4   1.5168   1.4216   1.1542   0.7655   0.3294  -0.0710  -0.3595  -0.4812
   5   0.5029   0.4078   0.1404  -0.2484  -0.6844  -1.0849  -1.3734  -1.4951
   6   2.0914   1.9962   1.7288   1.3401   0.9040   0.5036   0.2151   0.0934
   7   1.9357   1.8406   1.5732   1.1844   0.7484   0.3480   0.0594  -0.0623

    r[i+ 10*(j+  8*  4)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   2.8090   2.7138   2.4464   2.0577   1.6216   1.2212   0.9327   0.8110
   1   1.4409   1.3457   1.0783   0.6896   0.2535  -0.1469  -0.4354  -0.5571
   2   1.0800   0.9849   0.7175   0.3287  -0.1073  -0.5077  -0.7963  -0.9180
   3   2.7138   2.6187   2.3513   1.9625   1.5265   1.1261   0.8375   0.7158
   4   1.8718   1.7766   1.5092   1.1205   0.6844   0.2840  -0.0045  -0.1262
   5   0.8580   0.7628   0.4954   0.1067  -0.3294  -0.7298  -1.0183  -1.1401
   6   2.4464   2.3513   2.0839   1.6951   1.2591   0.8587   0.5701   0.4484
   7   2.2908   2.1956   1.9282   1.5395   1.1034   0.7030   0.4145   0.2927

    r[i+ 10*(j+  8*  5)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   1.2929   1.1977   0.9303   0.5416   0.1055  -0.2949  -0.5834  -0.7051
   1  -0.0752  -0.1704  -0.4378  -0.8265  -1.2626  -1.6630  -1.9515  -2.0733
   2  -0.4361  -0.5312  -0.7987  -1.1874  -1.6235  -2.0239  -2.3124  -2.4341
   3   1.1977   1.1025   0.8351   0.4464   0.0103  -0.3901  -0.6786  -0.8003
   4   0.3557   0.2605  -0.0069  -0.3956  -0.8317  -1.2321  -1.5206  -1.6423
   5  -0.6582  -0.7533  -1.0207  -1.4095  -1.8455  -2.2459  -2.5345  -2.6562
   6   0.9303   0.8351   0.5677   0.1790  -0.2571  -0.6575  -0.9460  -1.0677
   7   0.7746   0.6795   0.4121   0.0233  -0.4127  -0.8131  -1.1017  -1.2234

    r[i+ 10*(j+  8*  6)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   1.4122   1.3170   1.0496   0.6609   0.2248  -0.1756  -0.4641  -0.5858
   1   0.0441  -0.0511  -0.3185  -0.7072  -1.1433  -1.5437  -1.8322  -1.9539
   2  -0.3168  -0.4119  -0.6793  -1.0681  -1.5041  -1.9045  -2.1931  -2.3148
   3   1.3170   1.2219   0.9545   0.5657   0.1297  -0.2707  -0.5593  -0.6810
   4   0.4750   0.3798   0.1124  -0.2763  -0.7124  -1.1128  -1.4013  -1.5230
   5  -0.5388  -0.6340  -0.9014  -1.2902  -1.7262  -2.1266  -2.4151  -2.5369
   6   1.0496   0.9545   0.6871   0.2983  -0.1377  -0.5381  -0.8267  -0.9484
   7   0.8940   0.7988   0.5314   0.1427  -0.2934  -0.6938  -0.9823  -1.1041

    r[i+ 10*(j+  8*  7)]
 i/j        0        1        2        3        4        5        6        7
 ---------------------------------------------------------------------------
   0   2.8910   2.7958   2.5284   2.1397   1.7036   1.3032   1.0147   0.8930
   1   1.5229   1.4277   1.1603   0.7716   0.3355  -0.0649  -0.3534  -0.4751
   2   1.1620   1.0669   0.7995   0.4107  -0.0253  -0.4257  -0.7143  -0.8360
   3   2.7958   2.7007   2.4333   2.0445   1.6085   1.2080   0.9195   0.7978
   4   1.9538   1.8586   1.5912   1.2025   0.7664   0.3660   0.0775  -0.0442
   5   0.9400   0.8448   0.5774   0.1886  -0.2474  -0.6478  -0.9364  -1.0581
   6   2.5284   2.4333   2.1659   1.7771   1.3410   0.9406   0.6521   0.5304
   7   2.3728   2.2776   2.0102   1.6215   1.1854   0.7850   0.4965   0.3747

   Time domain power =    1.6439
   Signal frequency =(    0.6200,     0.1400,     0.5500)

    ** Output **

   ierr =       0
   (Modified) periodogram (Raw)
   Frequency domain power=    1.5531
   z-frq=    -1.00
 x/y-freq    -1.00    -0.75    -0.50    -0.25     0.00     0.25     0.50     0.75
 ------------------------------------------------------------------------------
     0.00  0.0000   0.0000   0.0000   0.0000   0.0007   0.0000   0.0000   0.0000
     0.25  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
     0.50  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
     0.75  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

```
  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0158  0.0188  0.0350  0.2150  0.0583  0.2150  0.0350  0.0188
    0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000


  (Modified) periodogram (Hanning)
  Frequency domain power=    1.0699
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0001  0.0004  0.0001  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0164  0.0045  0.0053  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0027  0.0004  0.0023  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000
```

```
  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0036  0.0427  0.0087  0.0427  0.0036  0.0001
    0.25  0.0000  0.0000  0.0009  0.0064  0.0041  0.0158  0.0009  0.0000
    0.50  0.0000  0.0000  0.0000  0.0136  0.0543  0.0136  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0137  0.0547  0.0137  0.0000  0.0000

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0053  0.0045  0.0164  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0006  0.0031  0.0058  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000


  (Modified) periodogram (Bartlett)
  Frequency domain power=   1.0593
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0000  0.0079  0.0346  0.0050  0.0000  0.0002
    0.25  0.0000  0.0000  0.0000  0.0014  0.0062  0.0009  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0003  0.0001  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
    0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0141  0.0037  0.0074  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0022  0.0005  0.0017  0.0000  0.0000
    0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
    0.75  0.0000  0.0001  0.0000  0.0012  0.0096  0.0024  0.0000  0.0001

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0007  0.0594  0.0070  0.0594  0.0007  0.0000
    0.25  0.0000  0.0000  0.0001  0.0089  0.0017  0.0129  0.0001  0.0000
    0.50  0.0000  0.0003  0.0000  0.0113  0.0622  0.0113  0.0000  0.0003
    0.75  0.0000  0.0003  0.0000  0.0069  0.0554  0.0139  0.0000  0.0003

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0074  0.0037  0.0141  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0011  0.0011  0.0030  0.0000  0.0000
    0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
    0.75  0.0000  0.0001  0.0000  0.0014  0.0108  0.0027  0.0000  0.0001

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
    0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

  z-frq=    0.75
```

```
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0000  0.0050  0.0346  0.0079  0.0000  0.0002
    0.25  0.0000  0.0000  0.0000  0.0009  0.0065  0.0015  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0008  0.0002  0.0000  0.0000


  (Modified) periodogram (Welch)
  Frequency domain power=   1.2154
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0005  0.0030  0.0005  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0001  0.0001  0.0004  0.0051  0.0357  0.0028  0.0003  0.0001
    0.25  0.0000  0.0000  0.0000  0.0005  0.0038  0.0003  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0003  0.0004  0.0011  0.0103  0.1247  0.0186  0.0011  0.0004
    0.25  0.0000  0.0000  0.0001  0.0011  0.0131  0.0020  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0017  0.0002  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0122  0.0012  0.0090  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0013  0.0002  0.0009  0.0000  0.0000
    0.50  0.0000  0.0000  0.0001  0.0017  0.0095  0.0005  0.0001  0.0000
    0.75  0.0000  0.0000  0.0001  0.0005  0.0078  0.0012  0.0001  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.1034  0.0186  0.1034  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0115  0.0021  0.0104  0.0000  0.0000
    0.50  0.0002  0.0003  0.0008  0.0158  0.0862  0.0044  0.0007  0.0003
    0.75  0.0002  0.0002  0.0007  0.0052  0.0760  0.0115  0.0007  0.0002

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0090  0.0012  0.0122  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0010  0.0001  0.0012  0.0000  0.0000
    0.50  0.0000  0.0000  0.0001  0.0016  0.0086  0.0004  0.0001  0.0000
    0.75  0.0000  0.0000  0.0001  0.0006  0.0083  0.0012  0.0001  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0003  0.0004  0.0011  0.0186  0.1247  0.0103  0.0011  0.0004
    0.25  0.0000  0.0000  0.0001  0.0020  0.0133  0.0011  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0004  0.0031  0.0003  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0004  0.0001  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0001  0.0001  0.0003  0.0028  0.0357  0.0051  0.0004  0.0001
    0.25  0.0000  0.0000  0.0000  0.0003  0.0037  0.0005  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0003  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0005  0.0001  0.0000  0.0000


  (Modified) periodogram (Parzen)
  Frequency domain power=   0.9132
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0023  0.0053  0.0023  0.0001  0.0000
    0.25  0.0000  0.0000  0.0001  0.0010  0.0023  0.0010  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0001  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0006  0.0092  0.0209  0.0089  0.0006  0.0000
    0.25  0.0000  0.0000  0.0003  0.0039  0.0090  0.0038  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0002  0.0005  0.0002  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

```
   z-frq=   -0.50
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0000  0.0014  0.0162  0.0313  0.0112  0.0005  0.0000
      0.25 0.0000  0.0000  0.0005  0.0062  0.0120  0.0044  0.0002  0.0000
      0.50 0.0000  0.0000  0.0000  0.0003  0.0007  0.0004  0.0000  0.0000
      0.75 0.0000  0.0000  0.0000  0.0003  0.0007  0.0003  0.0000  0.0000

   z-frq=   -0.25
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0001  0.0030  0.0125  0.0054  0.0016  0.0012  0.0000
      0.25 0.0000  0.0000  0.0008  0.0019  0.0010  0.0030  0.0010  0.0000
      0.50 0.0000  0.0000  0.0001  0.0032  0.0106  0.0063  0.0006  0.0000
      0.75 0.0000  0.0000  0.0003  0.0046  0.0108  0.0048  0.0003  0.0000

   z-frq=    0.00
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0001  0.0047  0.0117  0.0007  0.0117  0.0047  0.0001
      0.25 0.0000  0.0000  0.0011  0.0006  0.0055  0.0140  0.0033  0.0001
      0.50 0.0000  0.0000  0.0003  0.0089  0.0291  0.0168  0.0016  0.0000
      0.75 0.0000  0.0000  0.0007  0.0107  0.0253  0.0111  0.0007  0.0000

   z-frq=    0.25
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0000  0.0012  0.0016  0.0054  0.0125  0.0030  0.0001
      0.25 0.0000  0.0000  0.0002  0.0005  0.0086  0.0110  0.0019  0.0000
      0.50 0.0000  0.0000  0.0002  0.0048  0.0151  0.0085  0.0008  0.0000
      0.75 0.0000  0.0000  0.0003  0.0047  0.0110  0.0049  0.0003  0.0000

   z-frq=    0.50
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0000  0.0005  0.0112  0.0313  0.0162  0.0014  0.0000
      0.25 0.0000  0.0000  0.0003  0.0055  0.0155  0.0081  0.0007  0.0000
      0.50 0.0000  0.0000  0.0001  0.0010  0.0028  0.0014  0.0001  0.0000
      0.75 0.0000  0.0000  0.0000  0.0003  0.0008  0.0003  0.0000  0.0000

   z-frq=    0.75
 x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
 -------------------------------------------------------------------------
      0.00 0.0000  0.0000  0.0006  0.0089  0.0209  0.0092  0.0006  0.0000
      0.25 0.0000  0.0000  0.0002  0.0039  0.0091  0.0040  0.0003  0.0000
      0.50 0.0000  0.0000  0.0000  0.0003  0.0006  0.0003  0.0000  0.0000
      0.75 0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

Chapter 7

SORTING

7.1 INTRODUCTION

This chapter describes the functions for sorting data.
Function described in this chapter divides up and allocates internal processing among threads and executes allocated processing in parallel.
This library provides functions having the following operations.

(1) Sorting a list of data

(2) Sorting a list of pairwise data

7.1.1 Notes

Since the parallel processing overhead significantly affects the computation cost if the number of data to be sorted is small, performance may be worse than when a non-parallel function is used.

7.1.2 Algorithms Used

The algorithms for sorting in ascending order are shown below. The algorithms for sorting in descending order, which differ only in terms of the relative magnitudes, are similar.

(1) Shell sort

(1) Set the spacing h.

(2) Take all subsequences of spacing h from the data sequence.

(3) Compare adjacent pairs within each subsequence to arrange them in ascending order. If they are in the reverse order, exchange their positions and confirm again the relative order with the preceding data. If they are again in the reverse order, exchange the positions and work back toward the beginning.

(4) Decrease the spacing h and repeat steps (2) and (3). When the processing for h =1 ends, sorting is completed.

(2) Heap sort

(1) Organize the assigned data into a heap tree (well-ordered binary tree in which parents have value greater than or equal to those of children).

(2) Exchange the root and the data at the very end of the tree.

(3) Let the position excluding the very last data be A.

(4) Consider A to be a new tree, and organize this into a heap tree again.

(5) Repeat steps (2) to (4). When the remaining data is only the root, sorting is completed.

(3) Quick sort

(1) Count the number of data values within the sort interval.

(2) Do the following depending on the number of data values.

- If the number of data values is less than or equal to one:
  Do nothing.
- If the number of data values is 2:
  If they are in the reverse order, exchange their positions.
- If the number of data values is greater than or equal to three:
  ① Select one pivot value from within the interval.
  ② Divide the data within the interval into two intervals consisting of values greater than the pivot value and values less than the pivot value.

(3) Repeat steps (1) and (2). When the number of data values in all data intervals is less than or equal to two, sorting is completed.

(4) Merge sort

(1) Count the number of data values within the sort interval.

(2) Do the following depending on the number of data values.

- If the number of data value is one:
  Do nothing.
- If the number of data values is two:
  If they are in the reverse order, exchange the positions.
- If the number of data values is greater than or equal to three:
  ① Divide the data within the interval in half into the top half and bottom half.

② Recursively merge sort the top half. Recursively merge sort the bottom half.

③ Merge the sorted top half and bottom half.

7.1.3 Reference Bibliography

(1) Niklaus Wirth, “ALGORITHMS + DATA STRUCTURES = PROGRAMS”, Prentice–Hall Inc. (1976).

(2) Hiroto Namihira,“Sorting and Searching”, CQ Publishing Co. Ltd.

(3) Yoshiyuki Kondo,“Algorithms and Data Structures”, Softbank Publishing Inc.

7.2 SORTING

7.2.1 ASL_qssta1, ASL_pssta1 Sorting a List of Data

(1) **Function**

Given n data values $a_{i_k}(k = 1, 2, \cdots, n)$, the ASL_qssta1 or ASL_pssta1 obtains the data sequence $a_{j_k}(k = 1, 2, \cdots, n)$ consisting of the original data values a_i rearranged in ascending or descending order. Here,a_j satisfies ths following relationship.

For ascending order : $a_{j_1} \le a_{j_2} \le \cdots \le a_{j_n}$

For descending order : $a_{j_1} \ge a_{j_2} \ge \cdots \ge a_{j_n}$

(2) **Usage**

Double precision:
ierr = ASL_qssta1 (a,n,isw,wk,iwk,nt);
Single precision:
ierr = ASL_pssta1 (a,n,isw,wk,iwk,nt);

(3) **Arguments and Return Value**

D:Double precision real Z:Double precision complex
R:Single precision real C:Single precision complex
I: { int as for 32bit Integer / long as for 64bit Integer }

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	n	Input	Data to be sorted a_i
				Output	Sorted data a_j
2	n	I	1	Input	Size of array a
3	isw	I	1	Input	Sort method selection switch (See Note (a))
4	wk	{D* / R*}	n	Work	Work area
5	iwk	I*	2×n	Work	Work area
6	nt	I	1	Input	Number of tasks to be generated
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n≥1

(b) isw $\in \{\pm1, \pm2, \pm3, \pm4\}$

(c) nt ≥ 1

(5) **Error indicator (Return Value)**

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	

(6) **Notes**

(a) The sort methods to be selected by isw are as follows.

isw	Sort Method	isw	Sort Method
1	Shell sort (ascending order)	−4	Shell sort (descending order)
2	Heap sort (ascending order)	−2	Heap sort (descending order)
3	Quick sort (ascending order)	−3	Quick sort (descending order)
4	Merge sort (ascending order)	−4	Merge sort (descending order)

The user should select an appropriate sort method according to the properties of the input data. The features of each sort method are shown below.

· Shell sort

The average of the amount of calculation is on the order of $0(n^{1.5})$. A fast, stable sort is performed for any kind of data. Sorting is faster if part of the data sequence has been sorted.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

No work area is necessary.

· Heap sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

No work area is necessary.

· Quick sort

Although the average of the amount of calculation is on the order of $O(n \log n)$, this is an extremely inefficient sort for data having certain types of regularities such as data that has been partially sorted to begin with. This is the fastest sort method for random data.

Retention of the ordinal relationships among data having the same value is not guaranteed between before and after sorting.

· Merge sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.

The ordinal relationships before sorting among data having the same value is retained after sorting.

(7) **Example**

(a) Problem

Sort the following data in ascending order by using a shell sort.

a[0] = 5.0

a[1] = 4.0

a[2] = 9.0
a[3] = 6.0
a[4] = 2.0
a[5] = 5.0

(b) Input data

Array a, n=6 and isw=1.

(c) Main program

```
/*      C interface example for ASL_qssta1 */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*wk;
    int n,isw,*iwk,ierr;
    int nt;
    int i;
    FILE *fp;

    fp = fopen( "qssta1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw=1;
    n=6;
    nt=4;

    printf( "    *** ASL_qssta1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

    a = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*n) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\tArray a" );

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a[i] );
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    fclose( fp );

    ierr = ASL_qssta1(a, n, isw, wk, iwk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );

    printf( "\tArray a" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    free( a );
```

```
    free( wk );
    free( iwk );

    return 0;
}
```

(d) Output results

```
    *** ASL_qssta1 ***

     ** Input **

    n=  6

    Array a       5       4       9       6       2       5

     ** Output **

    ierr =      0

    Array a       2       4       5       5       6       9
```

7.2.2 ASL_qssta2, ASL_pssta2 Sorting a List of Pairwise Data

(1) **Function**

Given two sets of n data values $a_{i_k}(k = 1, 2, \cdots, n), b_{i_k}(k = 1, 2, \cdots, n)$, the ASL_qssta2 or ASL_pssta2 obtains the data sequence $a_{j_k}(k = 1, 2, \cdots, n)$ consisting of the original a_i data values rearranged in ascending or ascending order and the data sequence $b_{j_k}(k = 1, 2, \cdots, n)$ corresponding to it.
Here, a_j satisfies the following relationship.

For ascending order : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

For descending order : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

If a second order sort is specified, the function determines $j = j_1, j_2, \cdots, j_n$ so that the following relationship is satisfied for any k for which $a_{j_k} = a_{j_{k+1}}$ is satisfied.

For ascending order : $b_{j_k} \leq b_{j_{k+1}}$

For descending order : $b_{j_k} \geq b_{j_{k+1}}$

(2) **Usage**

Double precision:
ierr = ASL_qssta2 (a,n,b,isw1,isw2,wk,iwk,nt);
Single precision:
ierr = ASL_pssta2 (a,n,b,isw1,isw2,wk,iwk,nt);

(3) Arguments and Return Value

D:Double precision real Z:Double precision complex I: { int as for 32bit Integer / long as for 64bit Integer }
R:Single precision real C:Single precision complex

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	a	{D* / R*}	n	Input	Data to be sorted a_i
				Output	Sorted data a_j
2	n	I	1	Input	Size of array a
3	b	{D* / R*}	n	Input	Data b_i corresponding to a_i
				Output	Data b_j corresponding to sorted a_j
4	isw1	I	1	Input	Sort method selection switch (See Note (a))
5	isw2	I	1	Input	Second order sort switch isw2=0 : Do not perform second order sort isw2=1 : Perform second order sort
6	wk	{D* / R*}	2×n	Work	Work area
7	iwk	I*	2×n	Work	Work area
8	nt	I	1	Input	Number of tasks to be generated
9	ierr	I	1	Output	Error indicator (Return Value)

(4) Restrictions

(a) n≥1

(b) isw1 $\in \{\pm1, \pm2, \pm3, \pm4\}$

(c) isw2 $\in \{0, 1\}$

(d) nt ≥ 1

(5) Error indicator (Return Value)

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3100	Restriction (b) was not satisfied.	
3200	Restriction (c) was not satisfied.	
3300	Restriction (d) was not satisfied.	

(6) **Notes**

(a) The sort methods to be selected by isw1 are as follows.

isw1	Sort Method	isw1	Sort Method
1	Shell sort (ascending order)	−1	Shell sort (descending order)
2	Heap sort (ascending order)	−2	Heap sort (descending order)
3	Quick sort (ascending order)	−3	Quick sort (descending order)
4	Merge sort (ascending order)	−4	Merge sort (descending order)

The user should select an appropriate sort method according to the properties of the input data. The features of each sort method are shown below.

· Shell sort

The average of the amount of calculation is on the order of $O(n^{1.5})$. A fast, stable sort is performed for any kind of data. Sorting is faster if part of the data sequence has been sorted.
It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.
No work area is necessary.

· Heap sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.
It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.
No work area is necessary.

· Quick sort

Although the average of the amount of calculation is on the order of $O(n \log n)$, this is an extremely inefficient sort for data having certain types of regularities such as data that has been partially sorted to begin with. This is the fastest sort method for random data.
It is not guaranteed that ordinal relations among plural values of the second set having the same value of the first set keep unchanged between before and after sorting.

· Merge sort

Although the amount of calculation is on the order of $O(n \log n)$, the constant term portion is large. The sort time does not change much according to the properties of the input data.
The ordinal relationships before sorting among data having the same value is retained after sorting.

(7) **Example**

(a) Problem

Sort the following data for a in ascending order by using a shell sort and rearrange the corresponding data for b according to the sorted data for a. Also perform a second order sort.

a[0] = 5.0, b[0] = 3.0
a[1] = 4.0, b[1] = 4.0
a[2] = 9.0, b[2] = 2.0
a[3] = 6.0, b[3] = 3.0
a[4] = 2.0, b[4] = 8.0
a[5] = 5.0, b[5] = 1.0

(b) Input data

Arrays a and b, n=6, isw1=1 and isw2=1.

(c) Main program

```
/*      C interface example for ASL_qssta2 */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*b,*wk;
    int n,isw1,isw2,*iwk,ierr;
    int nt;
    int i;
    FILE *fp;


    fp = fopen( "qssta2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw1=1;
    isw2=1;
    n=6;
    nt=4;

    printf( "    *** ASL_qssta2 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

    a = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (2*n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*n) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf", &a[i], &b[i] );
    }
    printf( "\t   <Array a>    <Array b>\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g     %8.3g\n", a[i], b[i] );
    }

    fclose( fp );

    ierr = ASL_qssta2(a, n, b, isw1, isw2, wk, iwk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );

    printf( "\t   <Array a>    <Array b>\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g     %8.3g\n", a[i], b[i] );
    }

    free( a );
    free( b );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) Output results

```
*** ASL_qssta2 ***

 ** Input **

n=  6

   <Array a>    <Array b>
       5            3
       4            4
       9            2
       6            3
       2            8
       5            1

 ** Output **

ierr =      0

   <Array a>    <Array b>
       2            8
       4            4
       5            1
       5            3
       6            3
       9            2
```

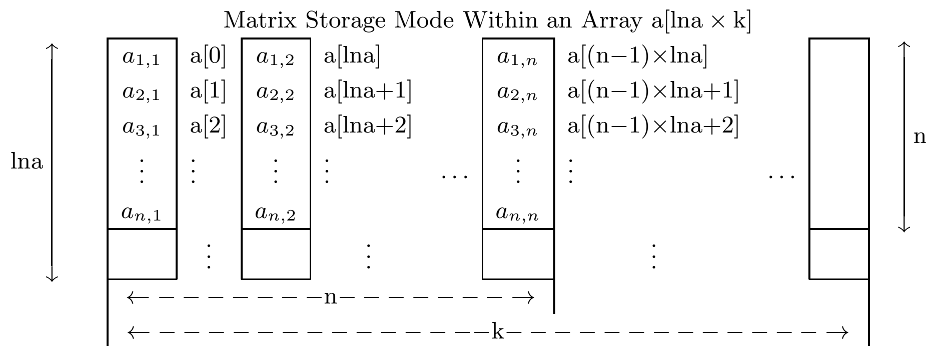
Appendix A

METHODS OF HANDLING ARRAY DATA

A.1 Methods of handling array data corresponding to matrix

Since the ASL C interface function library uses array data corresponding to matrix, this section describes various methods of handling arrays.

To call a function that uses array data, you must declare that array in advance in the calling program. If the declared array is a[lna × k], then $n \times n$ matrix $A = (a_{i,j})$ $(i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$ is stored in array a as shown in the figure below.



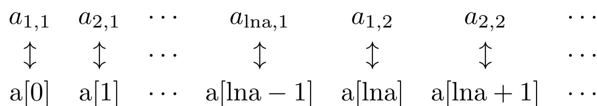
Remarks

a. lna $\geq n$ and k $\geq n$ must hold.

b. Matrix element $a_{i,j}$ corresponds to the array element a[$(i-1)$ + lna $\times$ $(j-1)$].

Figure A−1 Matrix Storage Mode Within an Array a[lna × k]

lna is called an adjustable dimension. If a two-dimensional array is used as an argument, the adjustable must be passed to the function as an argument in addition to the array name and order of the array. Because the matrix storage mode in ASL C interface corresponds to that of FORTRAN and the matrix elements $a_{i,j}$ $(i = 1, 2, \cdots, \text{lna}; j = 1, 2, \cdots, \text{k})$ must correspond to the array element a[ℓ] $(\ell = 0, 1, 2, \cdots, \text{lna} \times \text{k} - 1)$, as follows on the main memory.



Example ASL_dam1ad (Real matrix addition)

Add 3×2 matrices A and B placing the sum in matrix C. If you declare arrays of size $[5 \times 4]$, the declaration is as follows.

```
/*      C interface example for ASL_dam1ad */
#include <stdio.h>
#include <stdlib.h>

#include <asl.h>

int main()
{
    double *a, *b, *c;
    int lma, lmb, lmc;
    int m, n, ierr;
```

```
    int k;
    lma = lmb = lmc = 5;
    k = 4;
    m = 3;
    n = 2;
    a = (double *)malloc((size_t) sizeof(double) * lma*k);
    if(a == NULL)
    {
        printf("no enough memory for array a\n");
        return -1;
    }
    b = (double *)malloc((size_t) sizeof(double) * lmb*k);
    if(b == NULL)
    {
        printf("no enough memory for array b\n");
        return -1;
    }
    c = (double *)malloc((size_t) sizeof(double) * lmc*k);
    if(c == NULL)
    {
        printf("no enough memory for array c\n");
        return -1;
    }

           :
    ierr = ASL _dam1ad(a, lma , m, n, b, lmb, c, lmc);
           :

    free(a);
    free(b);
    free(c);
    return 0;
}
```

Data is stored in a as follows. Data are stored in b and c in the same way.

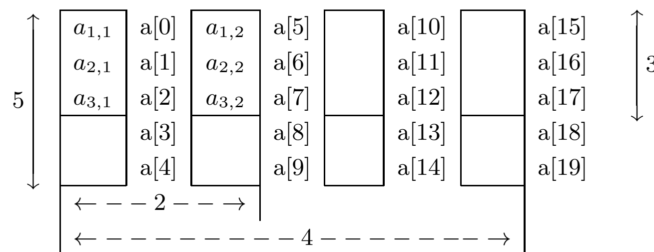


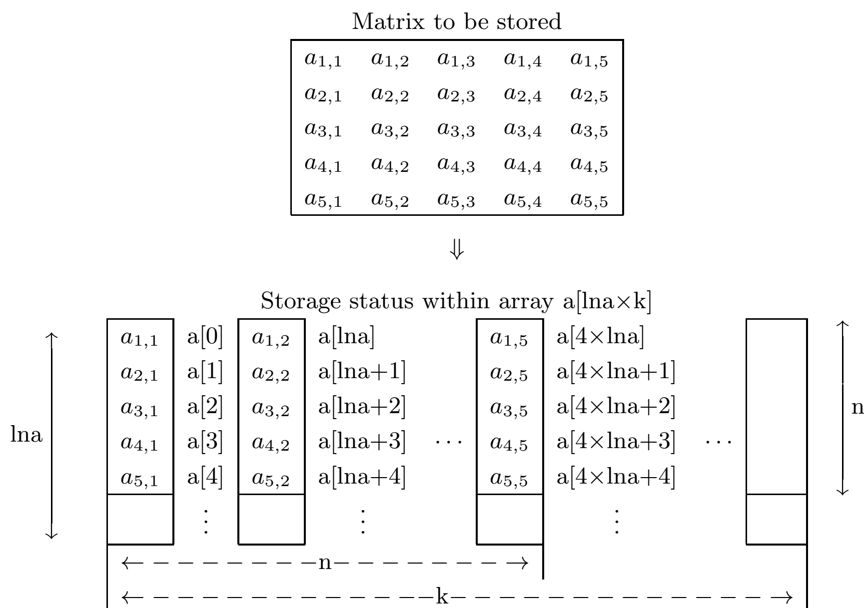
Figure A−2 Matrix Storage Mode Within an Array a[lna × k]

If you will be manipulating several arrays having different orders as data, you can prepare one array having lna equal to the largest order and use that array successively for each array. However, you must always assign the lna value as an adjustable dimension.

A.2 Data storage modes

Matrix data storage modes differ according to the matrix type. Storage modes for each type of matrix are shown below.

A.2.1 Real matrix (two-dimensional array type)



Remarks

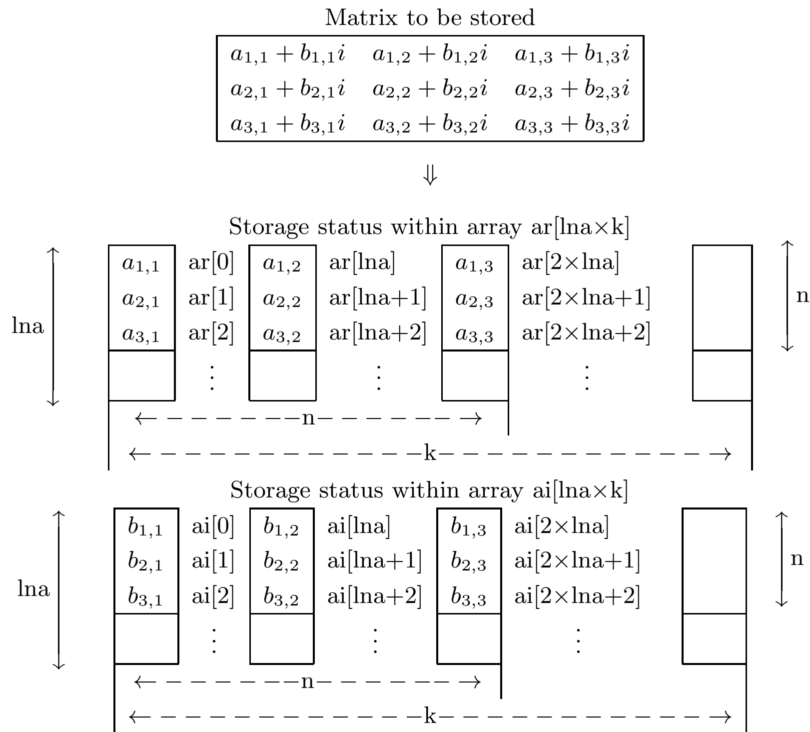
a. lna ≥ n and k ≥ n must hold.

Figure A−3 Real Matrix (Two-Dimensional Array Type) Storage Mode

A.2.2 Complex matrix

(1) **Two-dimensional array type, real argument type**

Real and imaginary parts are stored in separate arrays.

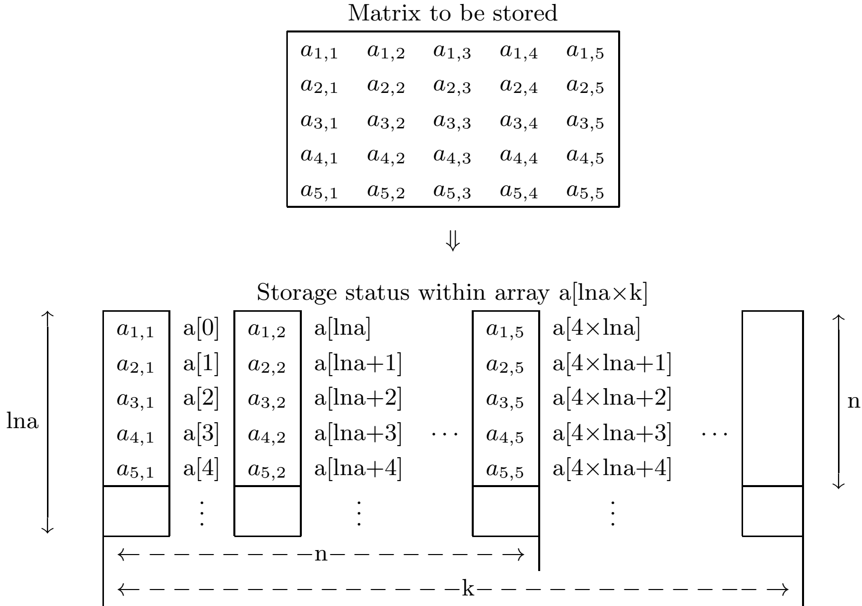


Remarks

a. lna ≥ n and k ≥ n must hold.

Figure A−4 Complex Matrix (Two-dimensional Array Type) (Real Argument Type) Storage Mode

(2) **Two-dimensional array type, complex argument type**



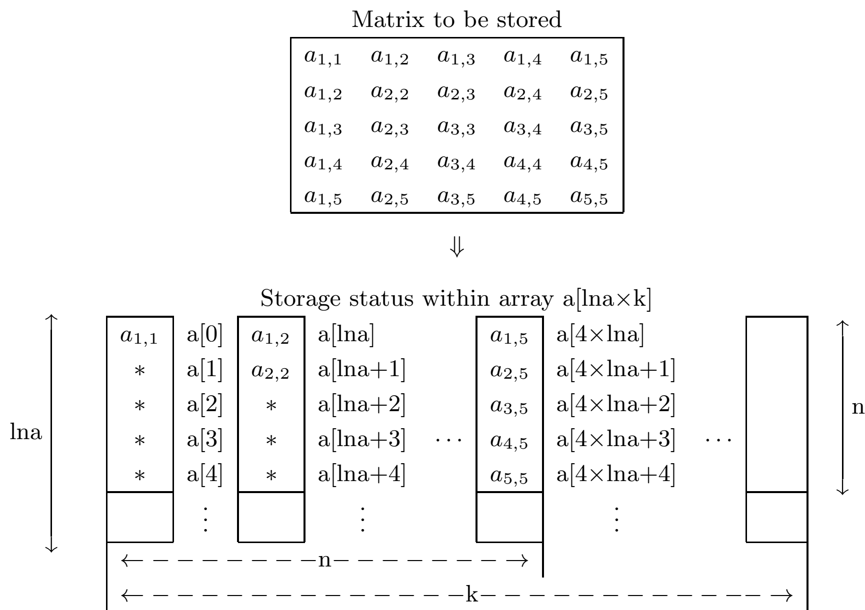
Remarks

a. lna ≥ n and k ≥ n must hold.

Figure A−5 Complex Matrix (Two-dimensional Array Type)(Complex Argument Type) Storage Mode

A.2.3 Real symmetric matrix and positive symmetric matrix

(1) Two-dimensional array type, upper triangular type



Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. lna ≥ n and k ≥ n must hold.

Figure A−6 Real Symmetric Matrix (Two-dimensional Array Type) (Upper Triangular Type) Storage mode

(2) **Two-dimensional array type, lower triangular type**

Matrix to be stored

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{1,2}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{1,3}$	$a_{2,3}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
$a_{1,4}$	$a_{2,4}$	$a_{3,4}$	$a_{4,4}$	$a_{4,5}$
$a_{1,5}$	$a_{2,5}$	$a_{3,5}$	$a_{4,5}$	$a_{5,5}$

⇓

Storage status within array a[lna×k]

$a_{1,1}$	a[0]	$*$	a[lna]		$*$	a[4×lna]		
$a_{1,2}$	a[1]	$a_{2,2}$	a[lna+1]		$*$	a[4×lna+1]		
$a_{1,3}$	a[2]	$a_{2,3}$	a[lna+2]		$*$	a[4×lna+2]		
$a_{1,4}$	a[3]	$a_{2,4}$	a[lna+3]	⋯	$*$	a[4×lna+3]	⋯	
$a_{1,5}$	a[4]	$a_{2,5}$	a[lna+4]		$a_{5,5}$	a[4×lna+4]		
	⋮		⋮			⋮		

lna (column height); n (rows used); ←— n —→ (columns used); ←— k —→ (total columns)

Remarks

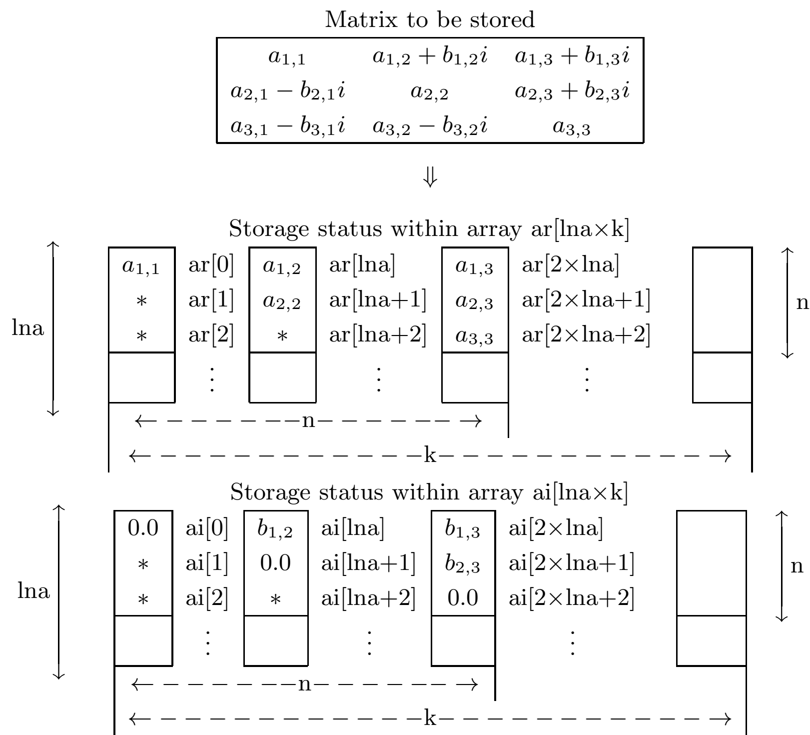
a. The asterisk ($*$) indicates an arbitrary value.

b. lna $\geq$ n and k $\geq$ n must hold.

Figure A−7 Real Symmetric Matrix (Two-dimensional Array Type, Lower Triangular Type) Storage mode

A.2.4 Hermitian matrix

(1) **Two-dimensional array type, real argument type, upper triangular type**
Upper triangular portions of the real and imaginary parts are stored in separate arrays.



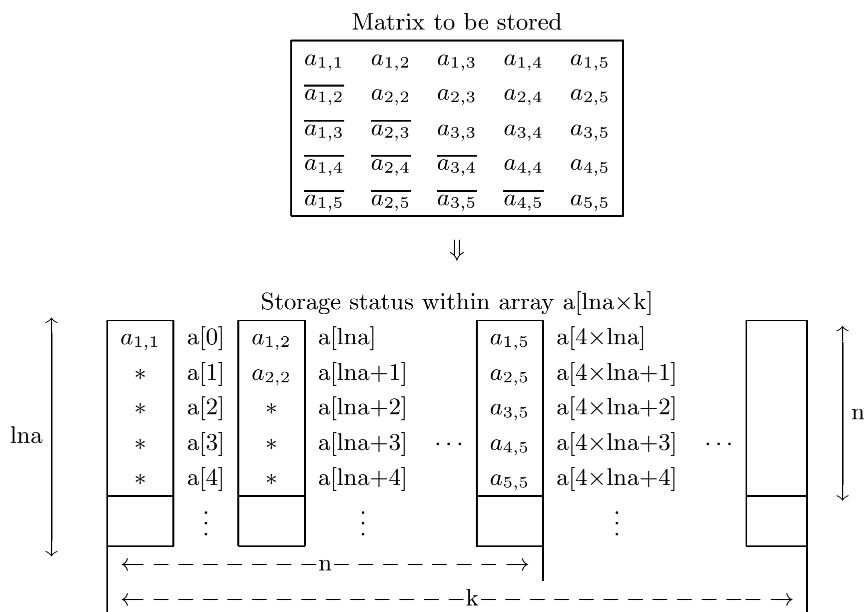
Remarks

a. The asterisk (∗) indicates an arbitrary value.

b. lna ≥ n and k ≥ n must hold.

Figure A−8 Hermitian Matrix (Two-dimensional Array Type) (Real Argument Type) (Upper Triangular Type) Storage Mode

(2) **Two-dimensional array type, complex argument type, upper triangular type**



Remarks

a. The $\overline{x}$ indicates the complex conjugate of x.

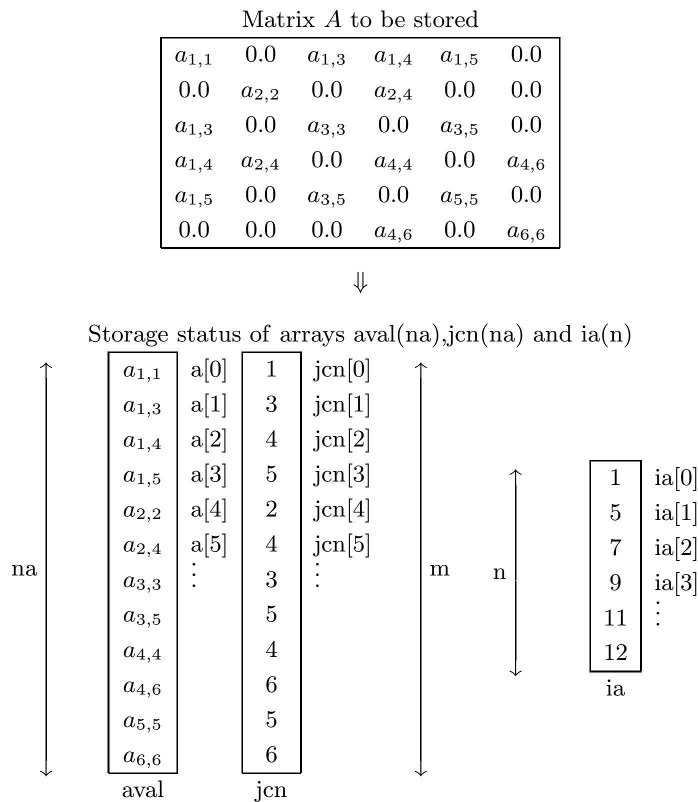
b. The asterisk $*$ indicates an arbitrary value.

c. lna $\geq$ n and k $\geq$ n must hold.

Figure A−9 Hermitian Matrix (Two-dimensional Array Type) (Complex Argument Type) (Upper Triangular Type) Storage Mode

A.2.5 Random sparse matrix (For symmetric matrix only)

(1) **Sparse format (Symmetric case)**



Remarks

a. m is the number of nonzero elements in the upper triangular part of the original matrix A including the diagonal.

b. Array aval contains the nonzero upper triangular elements of the original matrix A, stored sequentially beginning with the first row.

c. Array jcn contains the column numbers in the original matrix A of the elements stored in array aval.

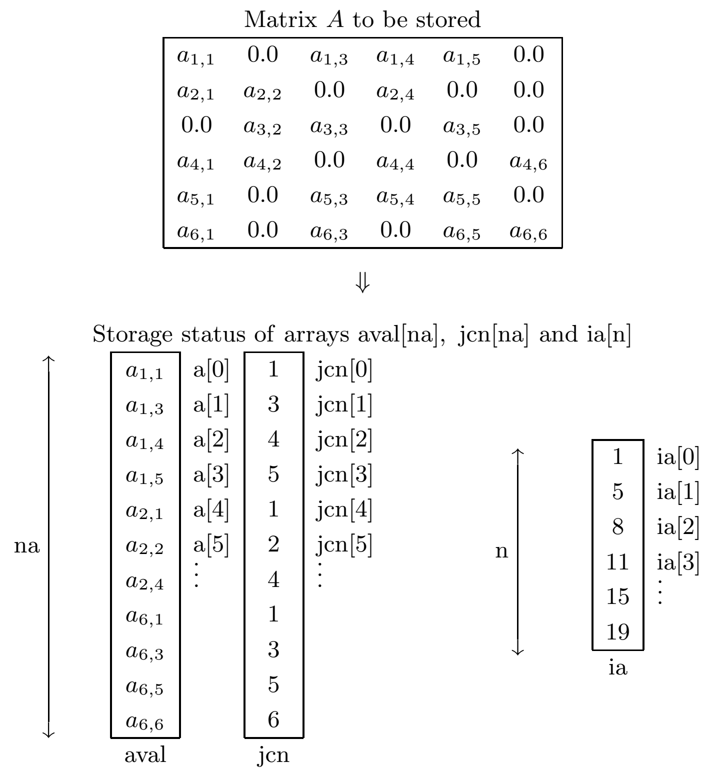
d. Array ia contains values equal to one added to the positions in array aval of the diagonal elements.

e. $n \leq m < na$ must hold.

Figure A−10 Storage of Random Symmetric Sparse Matrix (Sparse format)

A.2.6 Random sparse matrix

(1) Sparse format



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 A−11 Storage of Real Asymmetric Random Sparse Matrix (Sparse Format)

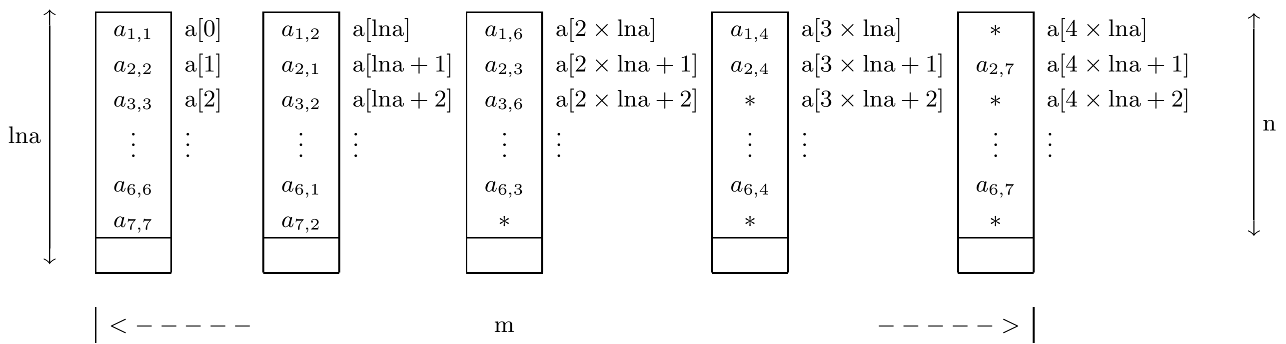
(2) **ELLPACK format**

Matrix A to be stored

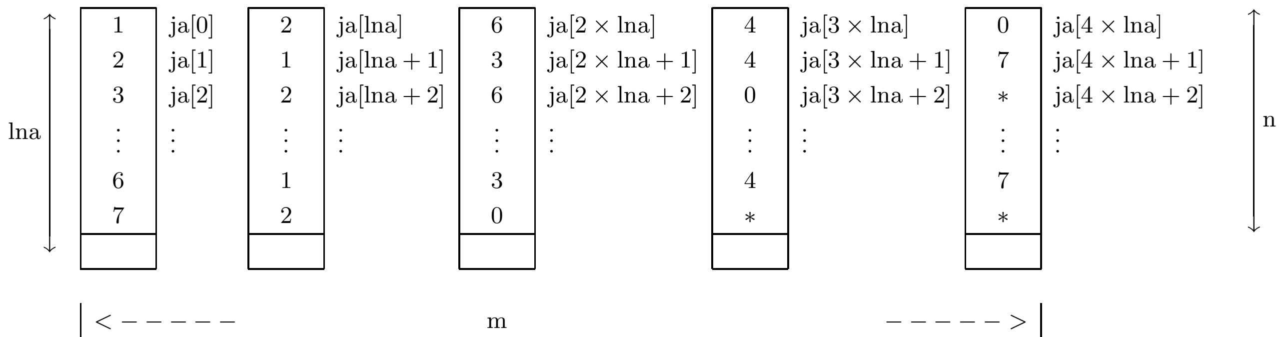
$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

$\Downarrow$

Storage status of array a[lna×m]



Storage status of array ja[lna×m]



Remarks

a. n is order of Matrix A.

b. lna $\geq$ n must hold.

c. m is the column number of Array a, which contains the nonzero elements of Matrix A.

d. Array a should contain nonzero elements of Matrix A so that:
- Diagonal elements are stored in the first column.
- Nonzero elements in the lower triangular part and the upper triangular part are stored in the second though m-th columns, with the first one in the second column, one adjacent to the next in each row. Here, it is unnecessary that nonzero elements in each row are stored sequentially.
- Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

e. Array ja should contain the column indices in Matrix A of those elements that correspond to the elements contained in Array a.
For those rows in which m − 1 becomes greater than the number of nonzero elements in the lower and upper triangular part, value 0 should be stored in the right neighbor of the rightmost position of the region

in ja in which the column indices of nonzero elements in Matrix A are stored. Arbitrary values can be stored in the remaining positions that are marked with '$*$'.

Figure A−12 Storage format for Asymmetric Random Sparse Matrix (ELLPACK Format)

Appendix B

MACHINE CONSTANTS USED IN ASL C INTERFACE

B.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 B−1 Units for Determining Error

Single-precision	Double-precision
$2^{-23}(\simeq 1.19 \times 10^{-7})$	$2^{-52}(\simeq 2.22 \times 10^{-16})$

Remark: The unit for determining error ε, which is also called the machine ε, is usually defined as the smallest positive constant for which the calculation result of $1+\varepsilon$ differs from 1 in the corresponding floating point mode. Therefore, seeing the unit for determining error enables you to know the maximum number of significant digits of an operation (on the mantissa) in that floating point mode.

B.2 Maximum and Minimum Values of Floating Point Data

The table below shows maximum and minimum values of floating point data defined within ASL 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 B−2 Maximum and Minimum Values of Floating Point Data

	Single-precision	Double-precision
Maximum value	$2^{127}(2-2^{-23})$ $(\simeq 3.40 \times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80 \times 10^{308})$
Positive minimum value	2^{-126} $(\simeq 1.17 \times 10^{-38})$	2^{-1022} $(\simeq 2.23 \times 10^{-308})$
Negative maximum value	-2^{-126} $(\simeq -1.17 \times 10^{-38})$	-2^{-1022} $(\simeq -2.23 \times 10^{-308})$
Minimum value	$-2^{127}(2-2^{-23})$ $(\simeq -3.40 \times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80 \times 10^{308})$

Index

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_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_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_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

ASL_dfc3fb : Vol.3, 142
ASL_dfcmbf : Vol.3, 81
ASL_dfcmfb : Vol.3, 77
ASL_dfcn1d : Vol.3, 177
ASL_dfcn2d : Vol.3, 187
ASL_dfcn3d : Vol.3, 195
ASL_dfcr1d : Vol.3, 206
ASL_dfcr2d : Vol.3, 216
ASL_dfcr3d : Vol.3, 224
ASL_dfcrcs : Vol.6, 283
ASL_dfcrcz : Vol.6, 281
ASL_dfcrsc : Vol.6, 279
ASL_dfcvcs : Vol.6, 274
ASL_dfcvsc : Vol.6, 269
ASL_dfdped : Vol.6, 291
ASL_dfdpes : Vol.6, 289
ASL_dfdpet : Vol.6, 294
ASL_dflage : Vol.3, 273
ASL_dflara : Vol.3, 267
ASL_dfps1d : Vol.3, 235
ASL_dfps2d : Vol.3, 243
ASL_dfps3d : Vol.3, 252
ASL_dfr1bf : Vol.3, 71
ASL_dfr1fb : Vol.3, 67
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_dfrmfb : Vol.3, 101
ASL_dfwtff : Vol.3, 306
ASL_dfwtft : Vol.3, 308
ASL_dfwth1 : Vol.3, 277
ASL_dfwth2 : Vol.3, 289
ASL_dfwthi : Vol.3, 296
ASL_dfwthr : Vol.3, 280
ASL_dfwths : Vol.3, 284
ASL_dfwtht : Vol.3, 292
ASL_dfwtmf : Vol.3, 301
ASL_dfwtmt : Vol.3, 303
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_dgiccp : Vol.4, 469
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_dgidcy : Vol.4, 492
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

ASL_dibinx : Vol.5, 129
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
ASL_dibsin : Vol.5, 153
ASL_dibsjn : Vol.5, 147
ASL_dibskn : Vol.5, 156
ASL_dibsyn : Vol.5, 150
ASL_dibymx : Vol.5, 93
ASL_dibynx : Vol.5, 87
ASL_dieii1 : Vol.5, 221
ASL_dieii2 : Vol.5, 223
ASL_dieii3 : Vol.5, 226
ASL_dieii4 : Vol.5, 228
ASL_digig1 : Vol.5, 199
ASL_digig2 : Vol.5, 202
ASL_diicos : Vol.5, 261
ASL_diierf : Vol.5, 281
ASL_diisin : Vol.5, 259
ASL_dileg1 : Vol.5, 285
ASL_dileg2 : Vol.5, 288
ASL_dimtce : Vol.5, 306
ASL_dimtse : Vol.5, 309
ASL_diopc2 : Vol.5, 302
ASL_diopch : Vol.5, 300
ASL_diopgl : Vol.5, 304
ASL_diophe : Vol.5, 298
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
ASL_dizbsl : Vol.5, 114
ASL_dizbsn : Vol.5, 108
ASL_dizbyn : Vol.5, 111
ASL_dizglw : Vol.5, 293
ASL_djtecc : Vol.6, 33
ASL_djteex : Vol.6, 29
ASL_djtegm : Vol.6, 45
ASL_djtegu : Vol.6, 37
ASL_djtelg : Vol.6, 49
ASL_djteno : Vol.6, 25
ASL_djteun : Vol.6, 20
ASL_djtewe : Vol.6, 41
ASL_dkfncs : Vol.4, 72
ASL_dkhncs : Vol.4, 78
ASL_dkinct : Vol.4, 55
ASL_dkmncn : Vol.4, 84
ASL_dksnca : Vol.4, 49
ASL_dksncs : Vol.4, 43
ASL_dkssca : Vol.4, 65
ASL_dlarha : Vol.5, 388
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_dmclaf : Vol.5, 490
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_dmcqlm : Vol.5, 538
ASL_dmcqsn : Vol.5, 531
ASL_dmcusn : Vol.5, 479
ASL_dmsp11 : Vol.5, 567
ASL_dmsp1m : Vol.5, 558
ASL_dmspmm : Vol.5, 563
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_dncbpo : Vol.4, 392
ASL_dndaao : Vol.4, 362
ASL_dndanl : Vol.4, 372
ASL_dndapo : Vol.4, 367
ASL_dngapl : Vol.4, 386
ASL_dnlnma : Vol.6, 605
ASL_dnlnrg : Vol.6, 592
ASL_dnlnrr : Vol.6, 598
ASL_dnnlgf : Vol.6, 617
ASL_dnnlpo : Vol.6, 611
ASL_dnrapl : Vol.4, 379
ASL_dofnnf : Vol.4, 115
ASL_dofnnv : Vol.4, 108
ASL_dohnlv : Vol.4, 136
ASL_dohnnf : Vol.4, 129
ASL_dohnnv : Vol.4, 122
ASL_doief2 : Vol.4, 149
ASL_doiev1 : Vol.4, 153
ASL_dolnlv : Vol.4, 143
ASL_dopdh2 : Vol.4, 157
ASL_dopdh3 : Vol.4, 165
ASL_dosnnf : Vol.4, 100
ASL_dosnnv : Vol.4, 91
ASL_dpdapn : Vol.4, 347
ASL_dpdopl : Vol.4, 343
ASL_dpgopl : Vol.4, 358
ASL_dplopl : Vol.4, 351
ASL_dqfodx : Vol.4, 182
ASL_dqmogx : Vol.4, 186

ASL_dqmohx : Vol.4, 190
ASL_dqmojx : Vol.4, 194
ASL_dsmgon : Vol.5, 348
ASL_dsmgpa : Vol.5, 352
ASL_dssta1 : Vol.5, 331
ASL_dssta2 : Vol.5, 335
ASL_dsstpt : Vol.5, 344
ASL_dsstra : Vol.5, 340
ASL_dxa005 : Vol.1, 47
ASL_gam1hh : SMP Functions(*) , 49
ASL_gam1hm : SMP Functions, 44
ASL_gam1mh : SMP Functions, 39
ASL_gam1mm : SMP Functions, 34
ASL_gan1hh : SMP Functions, 66
ASL_gan1hm : SMP Functions, 62
ASL_gan1mh : SMP Functions, 58
ASL_gan1mm : SMP Functions, 54
ASL_gbhesl : SMP Functions, 156
ASL_gbheud : SMP Functions, 161
ASL_gbhfsl : SMP Functions, 149
ASL_gbhfud : SMP Functions, 154
ASL_gbhpsl : SMP Functions, 133
ASL_gbhpud : SMP Functions, 139
ASL_gbhrsl : SMP Functions, 141
ASL_gbhrud : SMP Functions, 147
ASL_gcgjaa : SMP Functions, 302
ASL_gcgjan : SMP Functions, 307
ASL_gcgkaa : SMP Functions, 309
ASL_gcgkan : SMP Functions, 314
ASL_gcgraa : SMP Functions, 294
ASL_gcgran : SMP Functions, 299
ASL_gcheaa : SMP Functions, 249
ASL_gchean : SMP Functions, 253
ASL_gchesn : SMP Functions, 261
ASL_gchess : SMP Functions, 255
ASL_gchraa : SMP Functions, 233
ASL_gchran : SMP Functions, 238
ASL_gchrsn : SMP Functions, 246
ASL_gchrss : SMP Functions, 240
ASL_gfc2bf : SMP Functions, 371
ASL_gfc2fb : SMP Functions, 367
ASL_gfc3bf : SMP Functions, 401
ASL_gfc3fb : SMP Functions, 397
ASL_gfcmbf : SMP Functions, 338
ASL_gfcmfb : SMP Functions, 334
ASL_ham1hh : SMP Functions, 49
ASL_ham1hm : SMP Functions, 44
ASL_ham1mh : SMP Functions, 39
ASL_ham1mm : SMP Functions, 34
ASL_han1hh : SMP Functions, 66
ASL_han1hm : SMP Functions, 62

(*) SMP Functions = Shared Memory Parallel Processing Functions

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_qam1mm : SMP Functions, 18
ASL_qam1mt : SMP Functions, 22
ASL_qam1mu : SMP Functions, 14
ASL_qam1tm : SMP Functions, 26
ASL_qam1tt : SMP Functions, 30
ASL_qbgmlc : SMP Functions, 90
ASL_qbgmlu : SMP Functions, 88
ASL_qbgmsl : SMP Functions, 83
ASL_qbgmsm : SMP Functions, 78
ASL_qbsnsl : SMP Functions, 126
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_qcgkan : SMP Functions, 292
ASL_qcgsaa : SMP Functions, 264
ASL_qcgsan : SMP Functions, 270
ASL_qcgssn : SMP Functions, 279
ASL_qcgsss : SMP Functions, 272
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_r1ddgo : Vol.6, 168
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_r2bamo : Vol.6, 215
ASL_r2bams : Vol.6, 204
ASL_r2basm : Vol.6, 223
ASL_r2ccma : Vol.6, 249
ASL_r2ccmt : Vol.6, 243
ASL_r2ccpr : Vol.6, 256
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_r3iesr : Vol.6, 342
ASL_r3iesu : Vol.6, 326
ASL_r3ietc : Vol.6, 333
ASL_r3ieva : Vol.6, 330
ASL_r3tscd : Vol.6, 380
ASL_r3tsme : Vol.6, 357
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_r3tsva : Vol.6, 369
ASL_r41wr1 : Vol.6, 397
ASL_r42wr1 : Vol.6, 417
ASL_r42wrm : Vol.6, 409
ASL_r42wrn : Vol.6, 403
ASL_r4bi01 : Vol.6, 477
ASL_r4gl01 : Vol.6, 472
ASL_r4mu01 : Vol.6, 452
ASL_r4mwrf : Vol.6, 426
ASL_r4mwrm : Vol.6, 439
ASL_r4rb01 : Vol.6, 468
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_r5tesg : Vol.6, 501
ASL_r5tesp : Vol.6, 513
ASL_r5tewl : Vol.6, 505
ASL_r6clan : Vol.6, 571
ASL_r6clda : Vol.6, 576
ASL_r6clds : Vol.6, 565
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
ASL_rcsjss : Vol.1, 311
ASL_rcsmaa : Vol.1, 204
ASL_rcsman : Vol.1, 208
ASL_rcsmee : Vol.1, 217
ASL_rcsmen : Vol.1, 222
ASL_rcsmsn : Vol.1, 215
ASL_rcsmss : Vol.1, 210
ASL_rcsrss : Vol.1, 303
ASL_rcstaa : Vol.1, 283
ASL_rcstan : Vol.1, 287
ASL_rcstee : Vol.1, 296
ASL_rcsten : Vol.1, 301
ASL_rcstsn : Vol.1, 294
ASL_rcstss : Vol.1, 289
ASL_rfasma : Vol.6, 285
ASL_rfc1bf : Vol.3, 50
ASL_rfc1fb : Vol.3, 46
ASL_rfc2bf : Vol.3, 117
ASL_rfc2fb : Vol.3, 113
ASL_rfc3bf : Vol.3, 146
ASL_rfc3fb : Vol.3, 142
ASL_rfcmbf : Vol.3, 81
ASL_rfcmfb : Vol.3, 77
ASL_rfcn1d : Vol.3, 177
ASL_rfcn2d : Vol.3, 187
ASL_rfcn3d : Vol.3, 195
ASL_rfcr1d : Vol.3, 206
ASL_rfcr2d : Vol.3, 216
ASL_rfcr3d : Vol.3, 224
ASL_rfcrcs : Vol.6, 283
ASL_rfcrcz : Vol.6, 281
ASL_rfcrsc : Vol.6, 279
ASL_rfcvcs : Vol.6, 274
ASL_rfcvsc : Vol.6, 269
ASL_rfdped : Vol.6, 291
ASL_rfdpes : Vol.6, 289
ASL_rfdpet : Vol.6, 294
ASL_rflage : Vol.3, 273
ASL_rflara : Vol.3, 267
ASL_rfps1d : Vol.3, 235
ASL_rfps2d : Vol.3, 243
ASL_rfps3d : Vol.3, 252
ASL_rfr1bf : Vol.3, 71
ASL_rfr1fb : Vol.3, 67
ASL_rfr2bf : Vol.3, 136
ASL_rfr2fb : Vol.3, 132
ASL_rfr3bf : Vol.3, 169
ASL_rfr3fb : Vol.3, 164
ASL_rfrmbf : Vol.3, 106
ASL_rfrmfb : Vol.3, 101
ASL_rfwtff : Vol.3, 306
ASL_rfwtft : Vol.3, 308
ASL_rfwth1 : Vol.3, 277
ASL_rfwth2 : Vol.3, 289
ASL_rfwthi : Vol.3, 296
ASL_rfwthr : Vol.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
ASL_rhnenl : Vol.4, 250
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_rnnlgf : Vol.6, 617
ASL_rnrapl : Vol.4, 379
ASL_rofnnf : Vol.4, 115
ASL_rofnnv : Vol.4, 108
ASL_rohnlv : Vol.4, 136
ASL_rohnnf : Vol.4, 129
ASL_rohnnv : Vol.4, 122
ASL_roief2 : Vol.4, 149
ASL_roiev1 : Vol.4, 153
ASL_rolnlv : Vol.4, 143
ASL_ropdh2 : Vol.4, 157
ASL_ropdh3 : Vol.4, 165
ASL_rosnnf : Vol.4, 100
ASL_rosnnv : Vol.4, 91
ASL_rpdapn : Vol.4, 347
ASL_rpdopl : Vol.4, 343
ASL_rpgopl : Vol.4, 358
ASL_rplopl : Vol.4, 351
ASL_rqfodx : Vol.4, 182
ASL_rqmogx : Vol.4, 186
ASL_rqmohx : Vol.4, 190
ASL_rqmojx : Vol.4, 194
ASL_rsmgon : Vol.5, 348
ASL_rsmgpa : Vol.5, 352
ASL_rssta1 : Vol.5, 331
ASL_rssta2 : Vol.5, 335
ASL_rsstpt : Vol.5, 344
ASL_rsstra : Vol.5, 340
ASL_rxa005 : Vol.1, 47
ASL_vibh0x : Vol.5, 173
ASL_vibh1x : Vol.5, 176
ASL_vibhy0 : Vol.5, 179
ASL_vibhy1 : Vol.5, 182
ASL_vibi0x : Vol.5, 117
ASL_vibi1x : Vol.5, 123
ASL_vibj0x : Vol.5, 72
ASL_vibj1x : Vol.5, 78
ASL_vibk0x : Vol.5, 120
ASL_vibk1x : Vol.5, 126
ASL_viby0x : Vol.5, 75
ASL_viby1x : Vol.5, 81
ASL_vidbey : Vol.5, 314
ASL_vieci1 : Vol.5, 215
ASL_vieci2 : Vol.5, 218
ASL_viejac : Vol.5, 230
ASL_viejep : Vol.5, 244
ASL_viejte : Vol.5, 247
ASL_viejzt : Vol.5, 241
ASL_vienmq : Vol.5, 234
ASL_viepai : Vol.5, 250
ASL_vierfc : Vol.5, 278
ASL_vierrf : Vol.5, 275
ASL_viethe : Vol.5, 238
ASL_vigamx : Vol.5, 193

ASL_vigbet : Vol.5, 212
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_viifsi : Vol.5, 263
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_wbtcls : Vol.2, 297
ASL_wbtcsl : Vol.2, 292
ASL_wbtdls : Vol.2, 288
ASL_wbtdsl : Vol.2, 284
ASL_wibh0x : Vol.5, 173
ASL_wibh1x : Vol.5, 176
ASL_wibhy0 : Vol.5, 179
ASL_wibhy1 : Vol.5, 182
ASL_wibi0x : Vol.5, 117
ASL_wibi1x : Vol.5, 123
ASL_wibj0x : Vol.5, 72
ASL_wibj1x : Vol.5, 78
ASL_wibk0x : Vol.5, 120
ASL_wibk1x : Vol.5, 126
ASL_wiby0x : Vol.5, 75
ASL_wiby1x : Vol.5, 81
ASL_widbey : Vol.5, 314
ASL_wieci1 : Vol.5, 215
ASL_wieci2 : Vol.5, 218
ASL_wiejac : Vol.5, 230
ASL_wiejep : Vol.5, 244
ASL_wiejte : Vol.5, 247
ASL_wiejzt : Vol.5, 241
ASL_wienmq : Vol.5, 234
ASL_wiepai : Vol.5, 250
ASL_wierfc : Vol.5, 278
ASL_wierrf : Vol.5, 275
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_wiicnc : Vol.5, 272
ASL_wiicnd : Vol.5, 270
ASL_wiidaw : Vol.5, 268
ASL_wiiexp : Vol.5, 253
ASL_wiifco : Vol.5, 265
ASL_wiifsi : Vol.5, 263
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, 89
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