ADVANCED SCIENTIFIC LIBRARY
ASL C INTERFACE
User's Guide
<Basic Functions Vol.3>

PROPRIETARY NOTICE

PREFACE

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

The manuals corresponding to this product consist of seven volumes, which are divided into the chapters shown below. This manual describes the basic functions, volume 3.

Basic Functions Volume 1

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

Basic Functions Volume 2

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

Basic Functions Volume 3

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

Basic Functions Volume 4

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

Basic Functions Volume 5

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

Basic Functions Volume 6

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

Shared Memory Parallel Functions

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

Document Version 3.0.0-230301 for ASL, March 2023

Remarks

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

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

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

Contents

Chapter 1

INTRODUCTION

1.1 OVERVIEW

1.1.1 Introduction to The Advanced Scientific Library ASL C interface

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

Table 1−1 Classification of functions included in ASL

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

1.1.2 Distinctive Characteristics of ASL C interface

ASL C interface has the following distinctive characteristics.

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

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

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

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

1.2 KINDS OF LIBRARIES

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

Table 1−2 Kinds of libraries providing ASL C interface

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

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

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

1.3 ORGANIZATION

This section describes the organization of Chapters 2 and later.

1.3.1 Introduction

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

1.3.2 Organization of Function Description

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

(1) Function

(2) Usage

(3) Arguments and return value

(4) Restrictions

(5) Error indicator (Return Value)

(6) Notes

(7) Example

Each item is described according to the following principles.

1.3.3 Contents of Each Item

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

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

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

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

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

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

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

(a) Arguments and return value

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

(b) Type

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

I : Integer type

D : Double precision real

R : Real

Z : Double precision complex

C : Complex

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

(c) Size

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

1 : Indicates that argument is a variable.

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

(d) Input/Output

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

i. When only "Input" appears

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

ii. When only "Output" appears

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

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

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

iv. When "Work" appears

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

(e) Contents

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

- A sample Argument description follows.

Example

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

Double precision:

```
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	Note {D∗ / R∗}	lna×n	Input	Real matrix A(two-dimensional array)
				Output	The matrix A decomposed into the matrix LU where U is a unit upper triangular matrix and L is a lower triangular matrix.
2	lna	I	1	Input	Adjustable dimension size of array a
3	n	I	1	Input	Order n of matrix A
4	ipvt	I*	n	Output	Pivoting information ipvt[i−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**

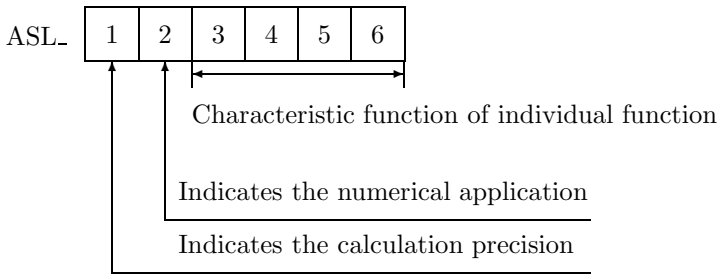
Here gives an example of how to use the function. Note that in some cases, multiple functions are combined in a single example. The output results are given in the 32-bit integer version, and may differ within the range of rounding error if the compiler or intrinsic functions are different.

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 basic functions consists of ten characters with a prefix "ASL_" and ⟨six alphanumeric characters⟩.

Figure 1−1 Function Name Components



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

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

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

"2" in Figure 1−1 : Currently, the following letters letterererererere are used to indicate the application field in the ASL C interface related products.

Letter	Application Field	Volume
a	Storage mode conversion	1
	Basic matrix algebra	1, 7
b	Simultaneous linear equations (direct method)	2, 7
c	Eigenvalues and eigenvectors	1, 7
f	Fourier transforms and their applications	3, 7
	Time series analysis	6
g	Spline function	4
h	Numeric integration	4
i	Special function	5
j	Random number tests	6
k	Ordinary differential equation (initial value problems)	4
l	Roots of equations	5
m	Extremum problems and optimization	5
n	Approximation and regression analysis	4, 6
o	Ordinary differential equations (boundary value problems), integral equations and partial differential equations	4
p	Interpolation	4
q	Numerical differentials	4

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

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

1.5 NOTES

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

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

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

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

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

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

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

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

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

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

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

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

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

Chapter 2

FOURIER TRANSFORMS AND THEIR APPLICATIONS

2.1 INTRODUCTION

This chapter describes functions that perform fast Fourier transforms, convolutions, correlations, power spectrum analysis, wavelet transforms, and inverse Laplace transform.

The following functions are provided for computing the discrete Fourier transform according to the data characteristics.

(1) Complex Fourier transform
Data values are complex numbers.

(2) Real Fourier transform
Data values are only real numbers.

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) one-dimensional data

(2) two-dimensional data

(3) three-dimensional data

The following functions are provided for computing the inverse Laplace transform according to the image function types.

(1) The case when image function is rational function.

(2) The case when image function is general function.

The following functions are provided for computing the wavelet transform according to the type of the base functions.

(1) When the base functions are Haar functions

(2) When the base function is a Mexican hat function

(3) When the base function is a French hat function

2.1.1 Notes

(1) You should choose the function group that best fits your input data type.

(2) In general, if your input data consists entirely of real numbers, you can use the real Fourier transform function.

(3) The sample input data n to which the transform is to be applied corresponds to a sample on a single period $[0, 2\pi]$.

(4) 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.

(5) In inverse Laplace transform of general function, you prepare function that calculates the imaginary part of image function $F(s)$. When $F(s)$ is a many valued function, care is required so that the correct branch is calculated within the function. In addition, $F(s)$ should satisfy the condition below.

$$\lim_{|s|\to\infty} F(s) = 0$$

2.1.2 Algorithms Used

2.1.2.1 One-Dimensional (Continuous) Fourier Transforms

The integral $F(\xi)$ defined by the following equation is called **the Fourier integral** of $f(x)$.

$$F(\xi) = a\int_{-\infty}^{\infty} f(x)e^{-\sqrt{-1}\xi x}dx$$

Here, a is a constant, and according to the Reference Bibliography, $a = 1$ is sometimes taken and $a = \frac{1}{2\pi}$ or $a = \frac{1}{\sqrt{2\pi}}$ is sometimes taken. Also, although i or j is normally used to represent the imaginary unit $\sqrt{-1}$, since this may be confused with the integers i and j used for subscripts, we have decided to use $\sqrt{-1}$.
If this integral exists for every value of the parameter ξ, $F(\xi)$ is called **the (continuous) Fourier transform** of the function $f(x)$. In this case, $F(\xi)$ is denoted by $\mathcal{F}\{f(x)\}$. For example, if time t is considered as x, then ξ corresponds to angular frequency ω, and if a component of the space coordinate $\boldsymbol{r}$ is considered as x, then ξ corresponds to a component of the wave vector $\boldsymbol{k}$. Also, when time t is considered as x, the frequency f defined by $f = \frac{\omega}{2\pi}$ is often used instead of angular frequency ω. To represent frequency, the symbol ν is used instead of the letter f. The quantity $\frac{k}{2\pi}$ in space corresponding to f is also called the spatial frequency. The inverse transform of $F(\xi)$ is defined as follows:

$$f(x) = \frac{1}{2\pi a}\int_{-\infty}^{\infty} F(\xi)e^{\sqrt{-1}\xi x}d\xi$$

A formal proof that this equation is the inverse transform is as follows.

$$\begin{aligned}\frac{1}{2\pi a}\int_{-\infty}^{\infty} F(\xi)e^{\sqrt{-1}\xi x}d\xi &= \frac{1}{2\pi a}\int_{-\infty}^{\infty}\left\{a\int_{-\infty}^{\infty} f(\eta)e^{-\sqrt{-1}\xi\eta}d\eta\right\}e^{\sqrt{-1}\xi x}d\xi \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(\eta)\left\{\int_{-\infty}^{\infty} e^{\sqrt{-1}\xi(x-\eta)}d\xi\right\}d\eta \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(\eta)2\pi\delta(x-\eta)d\eta \\ &= f(x)\end{aligned}$$

Here, $\delta(x)$ is the Dirac δ function defined as **distribution**.
Although the last equal sign holds when $f(x)$ is continuous, it will also hold when $f(x)$ is discontinuous if the value of $f(x)$ at a point of discontinuity $x = x_0$ is redefined as follows:

$$f(x_0) = \lim_{h\to+0}\frac{f(x_0+h)+f(x_0-h)}{2}$$

Although the continuous Fourier transform may be considered to exist for most functions that actually are encountered, care must be taken since it does not exist for every function (since it deals with an infinite integral). For information about the existence conditions for Fourier transforms, refer to a suitable technical text.
Also, note that the definitions of the Fourier transform (also called the Fourier forward transform) and its inverse transform (Fourier backward transform) are sometimes reversed. For example, when both time and space are considered, although a complex plane wave is represented by $e^{\sqrt{-1}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}$ by using the angular frequency ω and wave vector $\boldsymbol{k}$, in this case, it is convenient to reverse the signs of the power of the exponential functions in the definitions of the Fourier transform for time and the Fourier transform for space. In this case, the Fourier transform corresponds to the plane wave expansion of the function.
If $f(x)$ is a real function, changing the sign of the power of the exponential function corresponds to changing the sign of the imaginary part of the Fourier transform, which is obvious from the following **Euler's formula**.

$$e^{\sqrt{-1}x} = \cos x + \sqrt{-1}\sin x$$

The combination of $f(x)$ and $F(\xi)$ is called **the Fourier transform pair**.
The following **Parseval's Theorem** holds for a Fourier transform pair.

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \frac{1}{2\pi a} \int_{-\infty}^{\infty} |F(\xi)|^2 d\xi$$

This can be interpreted as assigning a correspondence between the total energy in the time (or space) domain and the frequency domain. In particular, if we consider a real time function as $f(x)(= f(t))$, to prevent the appearance of a negative frequency, this is modified as follows:

$$\int_{-\infty}^{\infty} \{f(t)\}^2 dt = \frac{1}{\pi a} \int_{0}^{\infty} |F(\omega)|^2 d\omega \quad [\because f(t) \in \boldsymbol{R} \Rightarrow F(-\omega) = F(\omega)^*]$$

For example, when the current flowing through an ideal resistor R (units: Ohms $[\Omega]$) in an electrical circuit is $i(t)$ (units: Amperes $[A]$), the electrical power corresponding to $|f(t)|^2$ becomes $R(i(t))^2$ (units: Watts $[W]$), and the amount of electrical power corresponding to $\int_{-\infty}^{\infty} |f(t)|^2 dt$ becomes $\int_{-\infty}^{\infty} R(i(t))^2 dt$ (units: Joules $[J]$). However, for the frequency spectrum $\hat{i}(f)$ of the current $i(t)$, the units should be set to $[A \cdot sec] = [A/Hz]$, and the size should be normalized so that the following relationship holds:

$$\int_{-\infty}^{\infty} \{R(i(t))^2\} dt = 2 \int_{0}^{\infty} R|\hat{i}(f)|^2 df$$

Of course, if you want to change the system of units, you should rewrite the constants according to the units used.
In many cases in a power spectrum analysis, **a full width half maximum (FWHM)** of the peak waveform that appears in the power spectrum is taken for the problem. In this case, the absolute value of the spectrum need not be specifically calculated, and a convenient **arbitrary unit** should be used.
If a real function $f(x)$ is a periodic function with period T_0, that is, for an arbitrary integer j, $f(x) = f(x + jT_0)$, the function can be expanded as follows in **a Fourier series**. (However, in a similar manner as described for a continuous Fourier transform, the function at a point of discontinuity is assumed to have as its value the average of the limit values from the left and right of that point.)

$$\begin{aligned} f(x) &= \frac{a_0}{2} + \sum_{j=1}^{\infty} \{a_j \cos(j\omega_0 x) + b_j \sin(j\omega_0 x)\} \\ &= \sum_{j=-\infty}^{\infty} c_j e^{\sqrt{-1} j\omega_0 x} \quad (\omega_0 = \frac{2\pi}{T_0}) \end{aligned}$$

Here, the expansion coefficients a_j and b_j (real numbers) and c_j (complex numbers), which are called **Fourier coefficients**, are given by the following equations.

$$\begin{aligned} a_j &= \frac{2}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} f(x) \cos(j\omega_0 x) dx \quad (j = 0, 1, 2, \cdots) \\ b_j &= \frac{2}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} f(x) \sin(j\omega_0 x) dx \quad (j = 1, 2, \cdots) \\ c_j &= \frac{1}{2}(a_j - \sqrt{-1} b_j) = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} f(x) e^{-\sqrt{-1} j\omega_0 x} dx \quad (j = 0, \pm 1, \pm 2, \cdots) \end{aligned}$$

When $f(x)$ is a complex-valued function (x is a real number), the Fourier coefficients a_j and b_j also become complex numbers. If we consider the function $g(x)$, which is defined by extracting only one period of $f(x)$ and setting it to 0 otherwise, that is $g(x)$ is as follows:

$$g(x) = \begin{cases} f(x) & |x| \leq \dfrac{T_0}{2} \\ 0 & \text{Otherwise} \end{cases}$$

then c_j is as follows:

$$\begin{aligned} c_j &= \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} f(x)e^{-\sqrt{-1}j\omega_0 x}dx \\ &= \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} g(x)e^{-\sqrt{-1}j\omega_0 x}dx \\ &= \frac{1}{T_0}\int_{-\infty}^{\infty} g(x)e^{-\sqrt{-1}j\omega_0 x}dx \\ &= \frac{1}{aT_0}G(\xi)_{\xi=j\omega_0} \end{aligned}$$

Therefore, except for the constant factor, the Fourier coefficients are equal to the value at $\xi = j\omega_0$ of the Fourier transform $G(\xi)$ of a function $(g(x))$ that limits the bandwidth of a periodic function $(f(x))$ to a single period. Conversely, the Fourier transform $(G(\xi)_{\xi=j\omega_0})$ at discrete points of a bandwidth-limited function $(g(x))$ can be calculated by expanding the function to a periodic function $(f(x))$ for which that bandwidth is assumed to be one period and obtaining its Fourier coefficients (c_j). The following sampling theorem can be used when sampling a continuous function.

Sampling theorem: For a time function $g(t)$ that is bandwidth limited by the frequency f_c, that is, the Fourier transform $G(\omega)$ of $g(t)$ takes nonzero values at $|\omega| \leq 2\pi f_c$, if the sampling interval is taken as $T = \frac{1}{2f_c}$, then $g(t)$ can be reconstructed from knowledge of only a sequence of sample values $\{g(iT)\}$ as follows.

$$g(t) = T\sum_{i=-\infty}^{\infty} g(iT)\frac{\sin 2\pi f_c(t-iT)}{\pi(t-iT)}$$

The frequency bandwidth $2f_c$ is called **the Nyquist sampling rate**.

Actually, when applying a discrete Fourier transform, although the sampling theorem conditions are rarely satisfied, sampling is performed based on the Nyquist sampling rate.
In **a discrete Fourier transform**, the integral in the Fourier series is square approximated as follows so that a computer can perform the calculation:

$$\begin{aligned} c_j &= \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} f(x)e^{-\sqrt{-1}j\frac{2\pi}{T_0}x}dx \\ &\simeq \frac{1}{T_0}\left\{\sum_{k=-(\lceil\frac{n}{2}\rceil-1)}^{-1} f_k e^{-\sqrt{-1}j\frac{2\pi}{T_0}\frac{kT_0}{n}} + \sum_{k=0}^{\lfloor\frac{n}{2}\rfloor} f_k e^{-\sqrt{-1}j\frac{2\pi}{T_0}\frac{kT_0}{n}}\right\}\frac{T_0}{n} \\ &= \frac{1}{n}\left\{\sum_{k=(\lfloor\frac{n}{2}\rfloor+1)}^{n-1} f_{k-n}e^{-2\pi\sqrt{-1}\frac{j(k-n)}{n}} + \sum_{k=0}^{\lfloor\frac{n}{2}\rfloor} f_k e^{-2\pi\sqrt{-1}\frac{jk}{n}}\right\} \\ &= \frac{1}{n}\left\{\sum_{k=0}^{n-1} f_k e^{-2\pi\sqrt{-1}\frac{jk}{n}}\right\} \end{aligned}$$

Here, $f_k = f(x)|_{x=\frac{kT_0}{n}}$ $(k = 0, \pm1, \pm2, \cdots)$. Also, $\lfloor x\rfloor$ represents the maximum integer that does not exceed x and $\lceil x\rceil$ represents the minimum integer greater than or equal to x. The final equation uses the fact that since $f(x)$ is a periodic function for which the period is T_0, then f_k is a discrete periodic function for which the period is n. From this equation, we see that c_j is also a discrete periodic function for which the period is n. Therefore, **in a discrete Fourier transform, n samples in a time (or space) domain and n corresponding samples of a frequency domain represent one period of a waveform in a time domain and frequency domain.** From

its periodicity, a discrete Fourier transform can also **be approximated by using the trapezoidal formula** for the integral in the Fourier series.

$$\frac{1}{n}\left\{\sum_{k=0}^{n-1} f_k e^{-2\pi\sqrt{-1}\frac{jk}{n}}\right\} = \frac{1}{n}\left[\sum_{k=0}^{n-1}\frac{1}{2}\left\{f_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} + f_{k+1}e^{-2\pi\sqrt{-1}\frac{j(k+1)}{n}}\right\}\right]$$
$$[\because\ f_n e^{-2\pi\sqrt{-1}\frac{jn}{n}} = f_0]$$

In this manual, unless specifically stated otherwise, the term "Fourier transform" indicates a "discrete Fourier transform." A discrete Fourier transform can be efficiently calculated by using **a Fast Fourier Transform (FFT) algorithm**. If the required number of multiplications of complex numbers in a discrete Fourier transform calculation for n data values is considered to be according to the definition, this number will be on the order of n^2. However, with a Fast Fourier Transform, when n can be factored as $n = r_1 r_2 \cdots r_m$, the number of complex multiplications is on the order of $n(r_1 + r_2 + \cdots + r_m)$. In particular, if $n = 2^m$, the number of multiplications is $2n \log_2 n$. Therefore, even if n normally does not get larger than this, the calculation efficiency when the calculation is performed using a Fast Fourier Transform algorithm is significantly different than when a Fast Fourier Transform algorithm is not used.

With a continuous Fourier transform, if the original waveform has a point of discontinuity, the partial sums of the Fourier series will not converge uniformly to the original waveform, and oscillations known as **the Gibbs phenomenon** will occur. However, no Gibbs phenomenon occurs with a discrete Fourier transform. (If the inverse transform is applied after a transform by a discrete Fourier transform, a series that matches the original series is obtained.) However, if a finite number of terms extracted from the high-order terms among the (infinite) Fourier coefficients corresponding to a continuous Fourier transform are given, the Gibbs phenomenon can be reproduced by performing **harmonic synthesis**. Also, for the inverse transform equation to exist for a continuous Fourier transform, the function value at a point of discontinuity must be defined as the average of the values on both sides (in the neighborhood of) that point. However, with a discrete Fourier transform, the data need not necessarily be selected in this way. (Normally, it is not practical to make the sampling interval sufficiently smaller to treat it as a neighborhood in the mathematical sense.)

2.1.2.2 Multidimensional (Continuous) Fourier Transforms

A Fourier transform can be expanded to a multidimensional case. For example, for a four-dimensional time space, it is defined as follows as a quadruple integral by using the position vector $\boldsymbol{r} = (x, y, z)$, wave vector $\boldsymbol{k} = (k_x, k_y, k_z)$, time t, and angular frequency ω.

$$\hat{f}(\boldsymbol{k}, \omega) = a\int_{-\infty}^{\infty} f(\boldsymbol{r}, t) e^{\sqrt{-1}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} d\boldsymbol{r}dt$$

The inverse transform $\hat{f}(\boldsymbol{k}, \omega)$ is represented by:

$$f(\boldsymbol{r}, t) = \frac{1}{(2\pi)^4 a}\int_{-\infty}^{\infty} \hat{f}(\boldsymbol{k}, \omega) e^{-\sqrt{-1}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)} d\boldsymbol{k}d\omega$$

Here, a is a constant, and according to the Reference Bibliography, $a = 1$ is sometimes taken and $a = \frac{1}{(2\pi)^4}$ is sometimes taken. Note that the definitions sometimes also reverses the signs of the power of the exponential functions. In a similar manner as described for the one-dimensional case, corresponding discrete Fourier transforms can be extended to multiple dimensions.

2.1.2.3 One-Dimensional Fourier Transform

The complex Fourier forward transform C_j for one period c_k $(k = 0, \cdots, n-1)$ of complex periodic data $\hat{c}_k$ satisfying $\hat{c}_k = \hat{c}_{k+n}$ for an arbitrary integer k is defined by the following equation.

$$C_j = \frac{1}{\alpha} \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

α is an arbitrary constant for which 1 or n normally is selected. At this time, the complex data after the transform C_j $(j = 0, \cdots, n-1)$ also corresponds to one period of complex periodic data $\hat{C}_j$ satisfying $\hat{C}_j = \hat{C}_{j+n}$ for an arbitrary integer j. The corresponding backward transform is as follows:

$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (k = 0, \cdots, n-1)$$

The following shows that this expression is the backward transform.

$$\begin{aligned}
\frac{1}{n} \sum_{j=0}^{n-1} (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} &= \frac{1}{n} \sum_{j=0}^{n-1} \left(\sum_{l=0}^{n-1} c_l e^{-2\pi\sqrt{-1}\frac{jl}{n}} \right) e^{2\pi\sqrt{-1}\frac{jk}{n}} \\
&= \frac{1}{n} \sum_{l=0}^{n-1} c_l \left\{ \sum_{j=0}^{n-1} e^{2\pi\sqrt{-1}\frac{j(k-l)}{n}} \right\} \\
&= \frac{1}{n} \sum_{l=0}^{n-1} c_l \{ n\delta_{k,l} \} \\
&= c_k
\end{aligned}$$

Here, the term

$$\sum_{j=0}^{n-1} e^{2\pi\sqrt{-1}\frac{j(k-l)}{n}} = n\delta_{k,l}$$

which is known as the orthogonal relationship for finite sums at selected points, can easily be proved by considering the sum of a geometric series for which the first term is 1, the common ratio is $r = e^{2\pi\sqrt{-1}\frac{(k-l)}{n}}$ and the number of terms is n. $\delta_{i,j}$, which is called the Kronecker delta, is defined as follows.

$$\delta_{i,j} = \begin{cases} 1 & (i = j) \\ 0 & (\text{otherwise}) \end{cases}$$

The discrete Fourier transform always exists if the discrete function takes values. Also, the result of applying a backward transform immediately after a forward transform will match the values of the original discrete function (excluding error in the numerical calculations). The orthogonal relationship for finite sums at selected points corresponds to the square approximation (or approximation by using the trapezoidal formula) of the following orthogonal relationship of the well known system of complex trigonometric functions $\{e^{\sqrt{-1}kt}\}$ (k: integer).

$$\frac{1}{2\pi} \int_0^{2\pi} e^{\sqrt{-1}kt} e^{\sqrt{-1}lt} dt = \delta_{k,l}$$

The functions in this manual obtain αC_j and nc_k, except for a constant factor, from the normal Fourier transform definition.

To correspond to the property that the domain of a function normally is symmetric to the left and right of the origin for a continuous Fourier transform, a half period of data should be considered as follows, shifted one period (**two-sided spectrum**):

$$\{\tilde{C}_k\}_{k=-(n-m-2),\cdots,-1,0,1,\cdots,m} = \{C_{m+1}, C_{m+2}, \cdots, C_{n-1}, C_0, C_1, \cdots, C_m\}$$

(Here, $m = \lfloor \frac{n}{2} \rfloor$ and $\lfloor x \rfloor$ represents the maximum integer not exceeding x.) At this time, C_0 is the element corresponding to zero frequency. Also, when a time function is specifically considered, to eliminate negative frequencies, this may also be considered as follows (**one-sided spectrum**):

$$\{\tilde{C}_k\}_{k=0,1,\cdots,m} = \begin{cases} \{C_0, 2C_1, \cdots, 2C_{m-1}, C_m\} & n\text{:Even number} \\ \{C_0, 2C_1, \cdots, 2C_m\} & n\text{:Odd number} \end{cases}$$

(1) **One-dimensional complex Fourier transforms and multiple one-dimensional complex Fourier transforms**

In this manual, complex Fourier transforms are divided into **one-dimensional complex Fourier transforms** and **multiple one-dimensional complex Fourier transforms** according to differences in the storage methods of the data for which the Fourier transform is calculated. For one-dimensional complex Fourier transforms, the data c_k and C_j are stored consecutively, and for multiple one-dimensional complex Fourier transforms, they are stored in fixed intervals. Multiple Fourier transforms can be used when calculating a normal multidimensional Fourier transform or a combination of Fourier transforms and other transforms.

(2) **One-dimensional real Fourier transforms and multiple one-dimensional real Fourier transforms**

When the data for which the Fourier forward transform is to be calculated is real, the following relationship is satisfied.

$$\begin{aligned} \alpha C^*_{n-j} &= \sum_{k=0}^{n-1} c^*_k e^{2\pi\sqrt{-1}\frac{(n-j)k}{n}} \\ &= \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad [\because e^{-2\pi k\sqrt{-1}} = 1] \\ &= \alpha C_j \end{aligned}$$

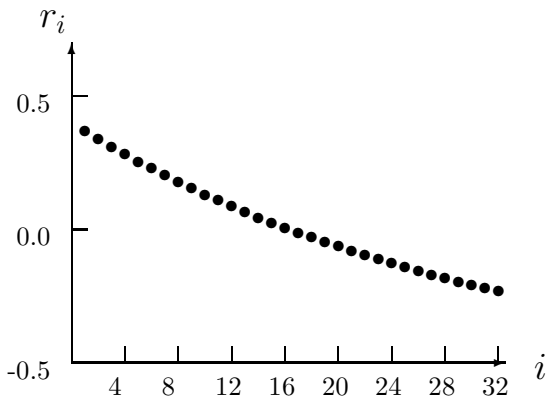
Here, z^* represents the conjugate complex number of the complex number z. In particular, C_0 is real and when n is even, then $C_{\frac{n}{2}}$ is also real.

Also, the backward transform is as follows.

$$\begin{aligned} nc_k &= \sum_{j=0}^{n-1} (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= \alpha(C_0 + (-1)^k \hat{C}_{\frac{n}{2}}) + \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} + \sum_{j=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= \alpha(C_0 + (-1)^k \hat{C}_{\frac{n}{2}}) + \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left\{ (\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} + (\alpha C_{n-j}) e^{2\pi\sqrt{-1}\frac{(n-j)k}{n}} \right\} \\ &= \alpha(C_0 + (-1)^k \hat{C}_{\frac{n}{2}}) + \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[(\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}} + \{(\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}}\}^* \right] \\ &= \alpha(C_0 + (-1)^k \hat{C}_{\frac{n}{2}}) + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{(\alpha C_j) e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= \alpha(C_0 + (-1)^k \hat{C}_{\frac{n}{2}}) + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[(\alpha \Re\{C_j\}) \cos(2\pi \frac{jk}{n}) - (\alpha \Im\{C_j\}) \sin(2\pi \frac{jk}{n}) \right] \end{aligned}$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x, and $\lceil x \rceil$ represents the minimum integer greater than or equal to x. Also, when n is odd, $\hat{C}_{\frac{n}{2}} = 0$, and when n is even, $\hat{C}_{\frac{n}{2}} = C_{\frac{n}{2}}$. Therefore, the

calculations for a Fourier transform can be executed using half the data for the case of general complex data (for c_k, only the real part, and for C_j, half of its period).
For example, when a complex Fourier transform is calculated for the real data r_i shown in Figure 2−1, the results will be as shown in Figure 2−2. However, with a real Fourier transform, the half periods of the real and imaginary parts, respectively, are calculated as shown in Figure 2−3. (Note that with a real Fourier transform, the real and imaginary parts of the transform results are stored in the array alternately.)



$$r_i = e^{-i/32.0} - 0.6 \ (i = 1, 2, \cdots, 32)$$

Figure 2−1 Data Before Fourier Transform

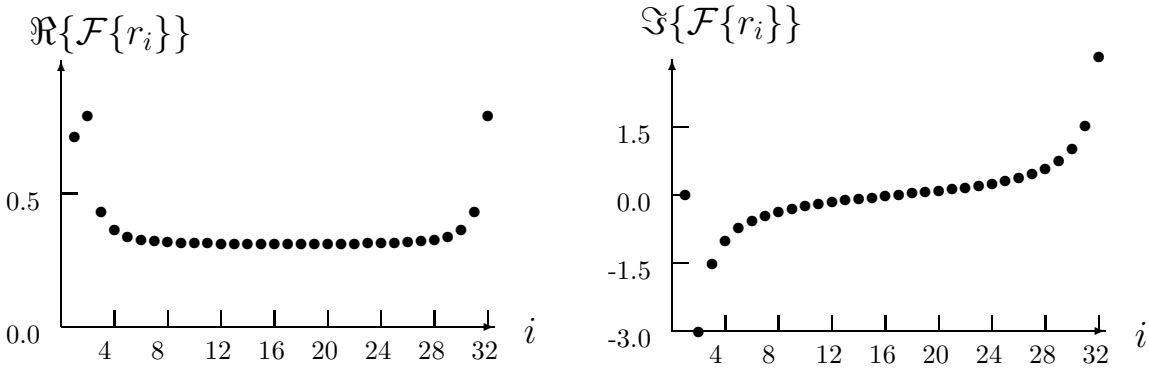
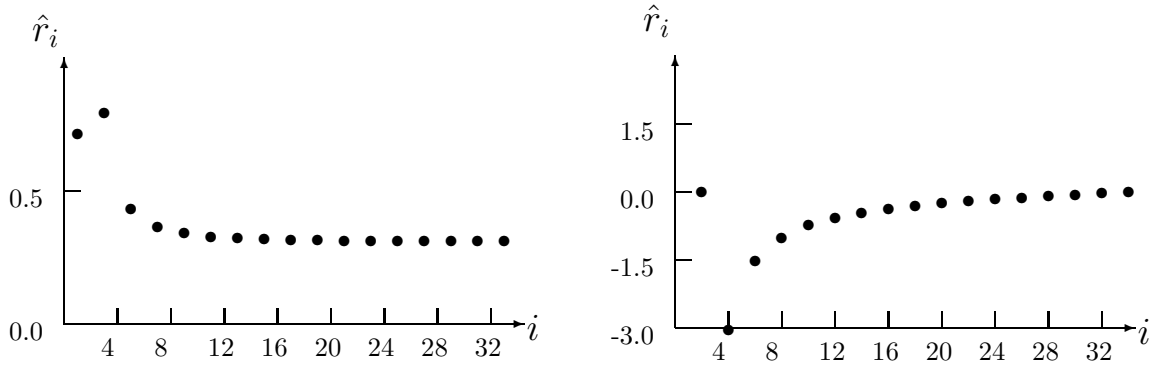


Figure 2−2 Data After Fourier Transform (For a Complex Fourier Transform)



$$\hat{r}_{2i-1} = \Re\{\mathcal{F}\{r_i\}\}, \quad \hat{r}_{2i} = \Im\{\mathcal{F}\{r_i\}\}, \quad (i = 1, 2, \cdots, 17)$$

Figure 2−3 Data After Fourier Transform (For a Real Fourier Transform)

Note that when n is even, the area required for the calculations may be conserved by defining $(C_0, C_{\frac{n}{2}})$ as a new complex number C_0. Also, In a similar manner as described for multiple one-dimensional complex Fourier transforms, data is stored in an array in fixed intervals for **multiple one-dimensional real Fourier transforms**.

2.1.2.4 Multidimensional Fourier Transforms

(1) **Two-dimensional complex Fourier transforms**

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.

$$C_{j_x,j_y} = \frac{1}{\alpha} \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} c_{k_x,k_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})}$$
$$(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$$

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:

$$c_{k_x,k_y} = \frac{1}{n_x n_y} \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} (\alpha C_{j_x,j_y}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

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.

(2) **Two-dimensional real Fourier transforms**

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).

(3) **Three-dimensional complex Fourier transforms**

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:

$$\begin{aligned}c_{k_x,k_y,j_z} &= \frac{1}{n_x n_y n_z}\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1}(\alpha C_{j_x,j_y,j_z}) e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)\end{aligned}$$

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.

(4) **Three-dimensional real Fourier transforms**

When the data for which the three-dimensional Fourier forward transform is to be calculated is real, the

following relationship is satisfied.

$$\begin{aligned}\alpha C^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c^*_{k_x,k_y,k_z} e^{2\pi\sqrt{-1}\{\frac{(n_x-j_x)k_x}{n_x}+\frac{(n_y-j_y)k_y}{n_y}+\frac{(n_z-j_z)k_z}{n_z}\}} \\ &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} c_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}\{\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z}\}} \\ &= \alpha C_{j_x,j_y,j_z}\end{aligned}$$

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).

2.1.2.5 Fast Fourier transform

(1) **Fast Fourier transform algorithm**
The **Temperton** algorithm, which is fast Fourier transform algorithm for arbitrary radices, is explained here.
The complex Fourier transform is represented by the following equation

$$C_k = \sum_{j=1}^{N} c_j W^{(j-1)(k-1)} \quad (k = 1, \cdots, N)$$

where $W = e^{-2\pi i/N}$, $i = \sqrt{-1}$ (N is the number of data).
This equation can be represented in the matrix form below.

$$X = W_n C, [W_n](j,k) = \omega^{(j-1)(k-1)}, \omega = \exp(-2\pi i/N)$$

If $N = n_1\ n_2\ n_3\ n_4\ \cdots\ n_k$ then

$$W_n = T_K\ T_{K-1}\ T_{K-2} \cdots\ T_2\ T_1.$$

If we let

$$l_1 = 1, l_{i+1} = n_i l_i, m_i = N/l_{i+1} \quad (i = 1, 2, \cdots k)$$

then

$$T_i = (P_{m_i}^{n_i}\ D_{m_i}^{n_i} \times I_{l_i})\ (W_{n_i} \times I_{N/n_i})$$

therefore,

$$\begin{aligned} \mathrm{X} &= W_n C \\ &= (P_1^{n_k}\ D_1^{n_k} \times I_{l_k})(W_{n_k} \times I_{N/n_k}) \cdots (P_{N/n_1}^{n_1}\ D_{N/n_1}^{n_1} \times I_1)(W_{n_1} \times I_{N/n_1})C. \end{aligned}$$

Example: the case of $N = 24$

$$N = 2 \times 3 \times 4$$

$$T_1 = (P_{12}^2 D_{12}^2 \times I_1)(W_2 \times I_{12})$$

$$T_2 = (P_4^3 D_4^3 \times I_2)$$

$$T_3 = (P_1^4 D_1^4 \times I_6)(W_4 \times I_6)$$

$$\begin{aligned} \mathrm{X} &= W_{24} C \\ &= T_3 T_2 T_1 C \\ &= (P_1^4 D_1^4 \times I_6)(W_4 \times I_6)(P_4^3 D_4^3 \times I_2)(W_3 \times I_8)(P_{12}^2 D_{12}^2 \times I_1)(W_2 \times I_{12})C \end{aligned}$$

(Notes)

(a) $\times$ means Kronecker matrix product. Suppose,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

then Kronecker matrix product is defined as follows:

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} e & f \\ g & h \end{bmatrix} & b\begin{bmatrix} e & f \\ g & h \end{bmatrix} \\ c\begin{bmatrix} e & f \\ g & h \end{bmatrix} & d\begin{bmatrix} e & f \\ g & h \end{bmatrix} \end{bmatrix}$$

(b) $P_{m_i}^{n_i}$ is a permutation matrix of order n_i, m_i.

(c) $D_{m_i}^{n_i}$ is a diagonal matrix of order n_i, m_i.

(d) I_r is a identity matrix of order r.

This transform is following flow. (C: input data, A: output data)

$LA = 1$
- $I = 1$, number of factors
- $IFAC \leftarrow$ Radix(2, 3, 4, 5, 6, 7, 8 or others)
- compute FFTs (Radix:$IFAC$)
- $M = N/IFAC$
- $I = 1$
- $J = 1$
- $JUMP = (IFAC - 1) \times LA$
  - $K = 0, M - LA, LA$
    - $L = 1, LA$
      - $A(J) = \Omega(K) \times (W(IFAC) \times C(I))$
    - $I = I + 1$
    - $J = J + 1$
  - $J = J + JUMP$

(Notes)
1. $W(IFAC)$: DFT(Discrete Fourier Transform) matrix order $IFAC$
2. $\Omega(K)$:$diag(trigs(1), trigs(K+1), trigs(2 \times K + 1), \cdots,$
   $trigs((IFAC - 1) \times K + 1))$
3. $trigs(K + 1) = \exp(2iK\pi/N), 0 \leq K \leq N - 1, i = \sqrt{-1}$

- $LA = LA \times IFAC$, exchange A and C

(2) **Enhancing speed for multiple one-dimensional complex Fourier transform**

The processing speed is enhanced as follows for a multiple one-dimensional complex Fourier transform that executes a one-dimensional complex Fourier transform for each of M data sequence of length N. Since the M data sequence are independent of one another, the ordinary one-dimensional complex Fourier transform loop can be replaced by a loop for executing M one-dimensional complex Fourier transforms. This enables you to achieve fast performance by easily avoiding the various types of problems that occur in ordinary one-dimensional complex fast Fourier transforms such as bank conflicts at definition time, input data, and rotational factor memory references. The extended function multiple one-dimensional complex Fourier transform program uses an algorithm that automatically applies the one-dimensional scan method so that the highest performance can be achieved even if the number of data sequence M is small.

(3) **Enhancing speed for real Fourier transforms**

You can obtain a real Fourier transform by treating the real input data values $\{r_k\}$ to which the transform is to be applied as complex input data values having zero as the imaginary part and computing a complex Fourier transform for those values. The result of the complex Fourier transform has the following conjugate symmetry relationship:

$$C_{N-j+2} = C_j^* \quad j = 2, \ \cdots, \ N$$

where, $*$ indicates the complex conjugate. In addition, the following equations indicate the relationship between the results $\{a_j\}$ and $\{b_j\}$ of the real Fourier transform and the results $\{C_j\}$ obtained from the complex Fourier transform.

$$a_j = (C_{j+1} + C_{N-j+1}) \quad j = 1, \ \cdots, \ \frac{N}{2} - 1$$

$$b_j = i(C_{j+1} - C_{N-j+1}) \quad j = 1, \ \cdots, \ \frac{N}{2} - 1$$

Therefore, if you compute a real Fourier transform by using a complex Fourier transform, you can obtain $\{a_j\}$ and $\{b_j\}$ by obtaining only $C_j (j = 1, \cdots, \frac{N}{2} + 1)$.
These functions take advantage of this to compute the optimum fast complex Fourier transform for the Vector Engine and obtain real Fourier coefficients.

2.1.2.6 One-Dimensional (Continuous) Convolutions and One-Dimensional (Continuous) Correlations

The integral $p(x)$ defined by the following equation is called **the convolution integral** of $f(x)$ and $g(x)$.

$$p(x) = \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi$$

The convolution integral of $f(x)$ and $g(x)$ is represented by $(f \times g)(x)$ or $f(x) \times g(x)$. The convolution integral is often called simply the "convolution." If we let the Fourier transforms of $f(x)$ and $g(x)$ be $\mathcal{F}\{f(x)\}$ and $\mathcal{F}\{g(x)\}$, respectively, then $(f \times g)(x)$ and $\mathcal{F}\{f(x)\}\mathcal{F}\{g(x)\}$ form a Fourier transform pair. That is, the following equations are satisfied.

$$\begin{aligned}
&\int_{-\infty}^{\infty} \left\{ \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi \right\} e^{-\sqrt{-1}\eta x} dx \\
&= \int_{-\infty}^{\infty} \left\{ \int_{-\infty}^{\infty} g(x - \xi) e^{-\sqrt{-1}\eta(x-\xi)} dx \right\} f(\xi) e^{-\sqrt{-1}\eta\xi} d\xi \\
&= \left\{ \int_{-\infty}^{\infty} f(x) e^{-\sqrt{-1}\eta x} dx \right\} \left\{ \int_{-\infty}^{\infty} g(x) e^{-\sqrt{-1}\eta x} dx \right\}
\end{aligned}$$

This relationship is called **the convolution theorem**. The convolution integral of $\mathcal{F}\{f(x)\}$ and $\mathcal{F}\{g(x)\}$ and $f(x)g(x)$ also form a Fourier transform pair (**frequency convolution theorem**). **Normally, the convolution integral is obtained by calculating the Fourier transforms of the two functions for which the convolution is applied and taking the inverse Fourier transform of the product of the two Fourier transforms. In particular, when a computer is used for the calculations, this method is extremely effective since efficient fast Fourier transform algorithms can be used.** Note that the convolution integral may be defined as a finite-interval integral as follows.

$$\hat{p}(x) = \int_{-a}^{a} f(\xi) g(x - \xi) d\xi$$

In particular, to associate it with **the Laplace transform**, it is defined as follows.

$$\tilde{p}(x) = \int_{0}^{x} x(\xi) h(x - \xi) d\xi$$

The convolution has the following properties.

(1) It satisfies the commutative law.
$f(x) \times g(x) = g(x) \times f(x)$

(2) It satisfies the associative law.
$f(x) \times (g(x) \times h(x)) = (f(x) \times g(x)) \times h(x)$

(3) It satisfies the distributive law.
$f(x) \times (g(x) + h(x)) = f(x) \times g(x) + f(x) \times h(x)$

One more important integral both theoretically and from the standpoint of actual applications is **the correlation integral**, which is defined by the following equation.

$$q(x) = \int_{-\infty}^{\infty} f(\xi) g(x + \xi) d\xi$$

If $f(x)$ is a real function and we let the Fourier transforms of $f(x)$ and $g(x)$ be $\mathcal{F}\{f(x)\}$ and $\mathcal{F}\{g(x)\}$, respectively, then the correlation integral of $f(x)$ and $g(x)$ and $\mathcal{F}\{f(x)\}^*\mathcal{F}\{g(x)\}$ form a Fourier transform pair (**correlation theorem**). That is, the following equations are satisfied.

$$\begin{aligned}
&\int_{-\infty}^{\infty}\left\{\int_{-\infty}^{\infty} f(\xi)g(x+\xi)d\xi\right\}e^{-\sqrt{-1}\eta x}dx \\
&= \int_{-\infty}^{\infty}\left\{\int_{-\infty}^{\infty} g(x+\xi)e^{-\sqrt{-1}\eta(x+\xi)}dx\right\}f(\xi)e^{\sqrt{-1}\eta\xi}d\xi \\
&= \left\{\int_{-\infty}^{\infty} f(x)^*e^{-\sqrt{-1}\eta x}dx\right\}^*\left\{\int_{-\infty}^{\infty} g(x)e^{-\sqrt{-1}\eta x}dx\right\} \\
&= \left\{\int_{-\infty}^{\infty} f(x)e^{-\sqrt{-1}\eta x}dx\right\}^*\left\{\int_{-\infty}^{\infty} g(x)e^{-\sqrt{-1}\eta x}dx\right\}
\end{aligned}$$

Here, z^* represents the conjugate complex number of the complex number z. **Note that the correlation integral does not satisfy the commutative law.** If $f(x)$ is an even function, the correlation integral matches the convolution integral.

$$\begin{aligned}
q(x) &= \int_{-\infty}^{\infty} f(\xi)g(x+\xi)d\xi \\
&= \int_{\infty}^{-\infty} f(-\eta)g(x-\eta)(-d\eta) \\
&= \int_{-\infty}^{\infty} f(\eta)g(x-\eta)d\eta \\
&= p(x)
\end{aligned}$$

Also, when $f(x)$ and $g(x)$ are the same function, the correlation integral is called **the autocorrelation function**, and when they are different, it is called **the cross correlation function**.

2.1.2.7 One-Dimensional Discrete Convolution and One-Dimensional Discrete Correlation

We can consider discrete convolution and discrete correlation corresponding to convolution and correlation, respectively, in a similar manner as we considered discrete Fourier transforms corresponding to continuous Fourier transforms. **Discrete convolution** is defined by the following equation as the approximation of continuous convolution by a square integral.

$$p(k) = \Delta x \sum_{i=0}^{m-1} f(i)g(k-i) \quad (k = 0, \cdots, m-1)$$

Here, Δx is the sampling interval. Also, $f(i)$, $g(j)$ and $p(k)$ are discrete functions having values defined only for integer values i, j and k. Since a computer is to be used for the calculations, the original function $f(x)$ or $g(x)$ of the real number x corresponding to $f(i)$ or $g(j)$ must be a function that is nonzero only on a specific finite interval or must be a periodic function. Also, the discrete function $f(i)$ or $g(j)$ is assumed to be a periodic function of period m satisfying the following relationship for an arbitrary integer k.

$$f(i) = f(i+km), g(i) = f(i+km) \ (i = 0, \cdots, m-1)$$

More specifically, the following distinction may be made. Discrete convolution for which the periodicity described above is not assumed is called **linear convolution**, and convolution for which the periodicity is assumed is called **circular convolution**. In the following, unless specifically stated otherwise, circular convolution will be considered as discrete convolution. Also, for the sake of explanation, the values that may become nonzero will be called "effective values". Now, if we assume the following:

$$f(i) = 0 \ (i = n_1, \cdots, m-1); \quad g(j) = 0 \ (j = n_2, \cdots, m-1)$$

that is, only n_1 $f(i)$ $(i = 0, \cdots, n_1 - 1)$ within one period of $f(i)$ and only n_2 $g(j)$ $(j = 1, \cdots, n_2 - 1)$ within one period of $g(j)$ are effective values, then if $m \geq n_1 + n_2 - 1$, the $n_1 + n_2 - 1$ $p(k)$ $(k = 0, \cdots, n_1 + n_2 - 2)$ within one period of $p(k)$ will be effective values. Therefore, if we take $m \geq n_1 + n_2 - 1$, the convolution can be calculated in relation to $f(i)$ and $g(j)$ without overlapping the data of the next period. That is, if only one period can be seen, the linear convolution and circular convolution results will match. If we consider circular convolution in particular, the square approximation of the integral matches the approximation by using the trapezoidal formula due to the periodicity, in a similar manner as described for the discrete Fourier transform.
For example, if we assume that the effective values of $f(i)$ are the three values $\{f(0), f(1), f(2)\}$ and the effective values of $g(j)$ are the two values $\{g(0), g(1)\}$, the corresponding effective values of $p(k)$ will be the following $3 + 2 - 1 = 4$ values:

$$\begin{array}{llll} p(0) = \Delta x(& f(0)g(0) & &) \\ p(1) = \Delta x(& f(0)g(1)+ & f(1)g(0) &) \\ p(2) = \Delta x(& & f(1)g(1)+ \quad f(2)g(0) &) \\ p(3) = \Delta x(& & f(2)g(1) &) \end{array}$$

However, to consider the correspondence with continuous convolution, rather than letting $m = n_1 + n_2 - 1$, we should consider $m = n_1 + n_2$ as the function values $p(n_1 + n_2 - 1) = 0$ added at the end. If we do this, then if we assume, for example, that the $f(i)$ are sample values that were sampled by dividing the interval $[0, a]$ into n_1 equal parts and the $g(j)$ are sample values that were sampled by dividing the interval $[0, b]$ into n_2 equal parts, the $p(k)$ can be considered to correspond to the convolution sample values when the interval $[0, a + b]$ is divided into $n_1 + n_2$ equal parts, and the sampling intervals will all match for $f(i)$, $g(j)$ and $p(k)$. If the numbers of effective values of $f(i)$ and $g(j)$ are uneven or too large, **sectioning** is performed, and a technique is used in which the interval is subdivided into several sections on which convolutions are obtained and then added together. There are two sectioning methods, which are called **the overlap-save method** and **the overlap-add method**. The functions in this manual provide functions that calculate the convolution by using the overlap-add method.
In a similar manner as described earlier for continuous functions, the following discrete convolution theorem between Fourier transforms and convolution holds for discrete functions.
If we let the discrete Fourier transforms of $f(i)$ $(i = 0, \cdots, m)$ and $g(j)$ $(j = 0, \cdots, m)$ be $F(i)$ $(i = 0, \cdots, m)$ and $G(j)$ $(j = 0, \cdots, m)$, respectively, then the discrete convolution of $f(i)$ and $g(j)$ and the product $F(j)G(j)$ $(j = 0, \cdots, m)$ form a Fourier transform pair (except for a constant factor). That is, the following relationship holds:

$$\begin{aligned} p(k) &= \Delta x \sum_{i=0}^{m-1} f(i)g(k-i) \\ &= \Delta x \sum_{i=0}^{m-1} \left\{ \frac{1}{m} \sum_{j=0}^{m-1} (\alpha F_j) e^{2\pi\sqrt{-1}\frac{ji}{m}} \right\} \left\{ \frac{1}{m} \sum_{l=0}^{m-1} (\alpha G_l) e^{2\pi\sqrt{-1}\frac{l(k-i)}{m}} \right\} \\ &= \Delta x \frac{\alpha}{m^2} \sum_{j=0}^{m-1} \sum_{l=0}^{m-1} (\alpha F_j G_l) e^{2\pi\sqrt{-1}\frac{lk}{m}} \left(\sum_{i=0}^{m-1} e^{2\pi\sqrt{-1}\frac{(j-l)i}{m}} \right) \\ &= \Delta x \frac{\alpha}{m^2} \sum_{j=0}^{m-1} \sum_{l=0}^{m-1} (\alpha F_j G_l) e^{2\pi\sqrt{-1}\frac{lk}{m}} (m\delta_{j,l}) \\ &= \Delta x \frac{\alpha}{m} \sum_{j=0}^{m-1} (\alpha F_j G_j) e^{2\pi\sqrt{-1}\frac{jk}{m}} \end{aligned}$$

Here, $\delta_{i,j}$, which is called the Kronecker delta, is defined as follows.

$$\delta_{i,j} = \begin{cases} 1 & (i = j) \\ 0 & (\text{otherwise}) \end{cases}$$

Discrete correlation, which is an approximation of continuous correlation by using square integration, is defined by the following equation.

$$q(k) = \Delta x \sum_{i=0}^{m-1} f(i)g(k+i) \quad (k = 0, \cdots, m-1)$$

Here, Δx is the sampling interval. Also, $f(i)$, $g(j)$ and $q(k)$ are discrete functions having values defined only for integer values i, j and k.
Note that when handling time series, a definition of correlation that differs in appearance may be used. For example, if the two time series x_i $(i = 1, 2, \cdots, n)$ and y_i $(i = 1, 2, \cdots, n)$ are given for which the number of samples is n, **the cross correlation coefficients** ${r_{xy}}^{(l)}$ and ${r_{yx}}^{(l)}$ are defined as functions of **lag** l $(l = 0, 1, \cdots, m-1;\ m \leq n)$ as follows.

$$\begin{aligned} {r_{xy}}^{(l)} &= \frac{{c_{xy}}^{(l)}}{\sqrt{{u_x}^{(l)}{v_y}^{(l)}}} \\ &= \frac{\displaystyle\sum_{i=1}^{n-l}(x_i - {\mu_x}^{(l)})(y_{i+l} - {\nu_y}^{(l)})}{\sqrt{\displaystyle\sum_{i=1}^{n-l}(x_i - {\mu_x}^{(l)})^2}\sqrt{\displaystyle\sum_{i=1}^{n-l}(y_{i+l} - {\nu_y}^{(l)})^2}} \end{aligned}$$

$$\begin{aligned} {r_{yx}}^{(l)} &= \frac{{c_{yx}}^{(l)}}{\sqrt{{u_y}^{(l)}{v_x}^{(l)}}} \\ &= \frac{\displaystyle\sum_{i=1}^{n-l}(y_i - {\mu_y}^{(l)})(x_{i+l} - {\nu_x}^{(l)})}{\sqrt{\displaystyle\sum_{i=1}^{n-l}(y_i - {\mu_y}^{(l)})^2}\sqrt{\displaystyle\sum_{i=1}^{n-l}(x_{i+l} - {\nu_x}^{(l)})^2}} \end{aligned}$$

Here, ${\mu_x}^{(l)}$, ${\nu_x}^{(l)}$, ${\mu_y}^{(l)}$ and ${\nu_y}^{(l)}$, which are defined by the following equations, represent the means of the first $n-l$ data and last $n-l$ data for x_i and y_i $(i = 1, 2, \cdots, n)$, respectively.

$${\mu_x}^{(l)} = \frac{\displaystyle\sum_{i=1}^{n-l} x_i}{(n-l)} \ (l = 0, 1, \cdots, m-1)$$

$${\nu_x}^{(l)} = \frac{\displaystyle\sum_{i=1}^{n-l} x_{i+l}}{(n-l)} \ (l = 0, 1, \cdots, m-1)$$

$${\mu_y}^{(l)} = \frac{\displaystyle\sum_{i=1}^{n-l} y_i}{(n-l)} \ (l = 0, 1, \cdots, m-1)$$

$${\nu_y}^{(l)} = \frac{\displaystyle\sum_{i=1}^{n-l} y_{i+l}}{(n-l)} \ (l = 0, 1, \cdots, m-1)$$

${c_{xy}}^{(l)}$ and ${c_{yx}}^{(l)}$ which are defined by the following equations, respectively, represent **the cross covariance**.

$$ {c_{xy}}^{(l)} = \frac{\sum_{i=1}^{n-l}(x_i - {\mu_x}^{(l)})(y_{i+l} - {\nu_y}^{(l)})}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

$$ {c_{yx}}^{(l)} = \frac{\sum_{i=1}^{n-l}(y_i - {\mu_y}^{(l)})(x_{i+l} - {\nu_x}^{(l)})}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

${u_x}^{(l)}$, ${v_x}^{(l)}$, ${u_y}^{(l)}$ and ${v_y}^{(l)}$, which are defined by the following equations, represent **the variance** of the first $n-l$ data and last $n-l$ data for x_i, y_i $(i = 1, 2, \cdots, n)$, respectively.

$$ {u_x}^{(l)} = \frac{\sum_{i=1}^{n-l}(x_i - {\mu_x}^{(l)})^2}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

$$ {v_x}^{(l)} = \frac{\sum_{i=1}^{n-l}(x_{i+l} - {\nu_x}^{(l)})^2}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

$$ {u_y}^{(l)} = \frac{\sum_{i=1}^{n-l}(y_i - {\mu_y}^{(l)})^2}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

$$ {v_y}^{(l)} = \frac{\sum_{i=1}^{n-l}(y_{i+l} - {\nu_y}^{(l)})^2}{(n-l)} \ (l = 0, 1, \cdots, m-1) $$

If we now define the standardized quantities $f(i)$, $g(i)$, $\hat{f}(i+l)$, and $\hat{g}(i+l)$, for the first $n-l$ data and last $n-l$ data for x_i and y_i $(i = 1, 2, \cdots, n)$, respectively, as follows:

$$ f(i) = \frac{x_{i+1} - {\mu_x}^{(l)}}{\sqrt{{u_x}^{(l)}}} $$
$$ \hat{f}(i+l) = \frac{x_{i+l+1} - {\nu_x}^{(l)}}{\sqrt{{v_x}^{(l)}}} $$
$$ g(i) = \frac{y_{i+1} - {\mu_y}^{(l)}}{\sqrt{{u_y}^{(l)}}} $$
$$ \hat{g}(i+l) = \frac{y_{i+l+1} - {\nu_y}^{(l)}}{\sqrt{{v_y}^{(l)}}} $$

then the following relationships hold:

$$ {r_{xy}}^{(l)} = \sum_{i=0}^{n-l-1} f(i)\hat{g}(i+l) $$
$$ {r_{yx}}^{(l)} = \sum_{i=0}^{n-l-1} g(i)\hat{f}(i+l) $$

Therefore, the definitions of ${r_{xy}}^{(l)}$ and ${r_{yx}}^{(l)}$ match the definition of the discrete correlation $q(k)$ (except for a constant factor). If x_i and y_i are the same time series, then ${r_{xx}}^{(l)}$ is called **the autocorrelation coefficient**,

and ${c_{xx}}^{(l)}$ is called **the autocovariance coefficient**.
When considering the statistical processing of time series, the following terms are used to distinguish between quantities and statistical estimators for samples.

Mean	→	Sample mean
Variance	→	Sample variance
Autocorrelation coefficient	→	Sample autocorrelation coefficient
Autocovariance	→	Sample autocovariance
Cross correlation coefficient	→	Sample cross correlation coefficient
Cross covariance	→	Sample cross covariance

Since a computer is to be used to calculate the discrete correlation $q(k)$, the original function $f(x)$ or $g(x)$ of the real number x corresponding to $f(i)$ or $g(j)$ must be a function that is nonzero only on a specific finite interval or must be a periodic function. Also, the discrete function $f(i)$ or $g(j)$ is assumed to be a periodic function of period m satisfying the following relationship for an arbitrary integer k.

$$f(i) = f(i + km), g(i) = g(i + km)\ (i = 0, \cdots, m - 1)$$

In a similar manner as described for discrete convolution, if we take $m \geq n_1 + n_2 - 1$, the correlation can be calculated in relation to $f(i)$ and $g(j)$ without overlapping the data of the next period. For example, if we assume that the effective values of $f(i)$ are the three values $\{f(0), f(1), f(2)\}$ and the effective values of $g(j)$ are the two values $\{g(0), g(1)\}$, the corresponding effective values of $q(k)$ will be the following $3 + 2 - 1 = 4$ values:

$$\begin{array}{lllll}
q(-2)(= q(m-2)) = & \Delta x(& & & f(2)g(0) \quad) \\
q(-1)(= q(m-1)) = & \Delta x(& & f(1)g(0)+ & f(2)g(1) \quad) \\
q(0) = & \Delta x(& f(0)g(0)+ & f(1)g(1) & \quad) \\
q(1) = & \Delta x(& f(0)g(1) & & \quad)
\end{array}$$

However, to consider the correspondence with continuous correlation, rather than letting $m = n_1 + n_2 - 1$, we should consider $m = n_1 + n_2$ as the function values $q(-n_1) = 0$ added first. If we do this, then if we assume, for example, that the $f(i)$ are sample values that were sampled by dividing the interval $[0, a]$ into n_1 equal parts and the $g(j)$ are sample values that were sampled by dividing the interval $[0, b]$ into n_2 equal parts, the $q(k)$ can be considered to correspond to the correlation sample values when the interval $[-a, b]$ is divided into $n_1 + n_2$ equal parts, and the sampling intervals will all match for $f(i)$, $g(j)$ and $q(k)$. If the numbers of effective values of $f(i)$ and $g(j)$ are uneven or too large, the discrete correlation can be calculated in a similar manner as described for discrete convolution, by performing sectioning and using the overlap-add method. In a similar manner as described earlier for continuous functions, the following discrete correlation theorem between Fourier transforms and correlation also holds for discrete functions.
If we let $f(i)$ be a real discrete function, and let the discrete Fourier transforms of $f(i)$ and $g(j)$ be $F(i)$ $(i = 0, \cdots, m)$ and $G(j)$ $(j = 0, \cdots, m)$, respectively, then the discrete correlation of $f(i)$ and $g(j)$ and the product $F(j)^*G(j)$ $(j = 0, \cdots, m)$ form a Fourier transform pair (except for a constant factor). That is, the following

relationship holds:

$$\begin{aligned}
q(k) &= \Delta x \sum_{i=0}^{m-1} f(i)g(k+i) \\
&= \Delta x \sum_{i=0}^{m-1} \left\{ \frac{1}{m} \sum_{j=0}^{m-1} (\alpha F(j))^* e^{-2\pi\sqrt{-1}\frac{ji}{m}} \right\}^* + \left\{ \frac{1}{m} \sum_{l=0}^{m-1} (\alpha G(l)) e^{2\pi\sqrt{-1}\frac{l(k+i)}{m}} \right\} \\
&= \Delta x \sum_{i=0}^{m-1} \left\{ \frac{1}{m} \sum_{j=0}^{m-1} (\alpha F(j))^* e^{-2\pi\sqrt{-1}\frac{ji}{m}} \right\} + \left\{ \frac{1}{m} \sum_{l=0}^{m-1} (\alpha G(l)) e^{2\pi\sqrt{-1}\frac{l(k+i)}{m}} \right\} \\
&= \Delta x \frac{\alpha}{m^2} \sum_{j=0}^{m-1} \sum_{l=0}^{m-1} (\alpha F(j)^* G(l)) e^{2\pi\sqrt{-1}\frac{lk}{m}} (\sum_{i=0}^{m-1} e^{2\pi\sqrt{-1}\frac{(l-j)i}{m}}) \\
&= \Delta x \frac{\alpha}{m^2} \sum_{j=0}^{m-1} \sum_{l=0}^{m-1} (\alpha F(j)^* G(l)) e^{2\pi\sqrt{-1}\frac{lk}{m}} (m\delta_{j,l}) \\
&= \Delta x \frac{\alpha}{m} \sum_{j=0}^{m-1} (\alpha F(j)^* G(j)) e^{2\pi\sqrt{-1}\frac{jk}{m}}
\end{aligned}$$

Here, $\delta_{i,j}$, which is called the Kronecker delta, is defined as follows.

$$\delta_{i,j} = \begin{cases} 1 & (i = j) \\ 0 & (\text{otherwise}) \end{cases}$$

The effective values of $q(k)$ are given by $q(k)$ $(k = 0, \cdots, n_2 - 1)$ and $q(-k) = q(m - k)$ $(k = 1, \cdots, n_1 - 1)$. However, since it is inconvenient to perform sectioning and other calculations directly in this form, the values $\hat{q}(k)$, which are shifted by $n_1 - 1$, should be calculated as defined by the following equation.

$$\hat{q}(k) = q(k - (n_1 - 1)) \quad (k = 0, \cdots, n_1 + n_2 - 2)$$

To calculate $\hat{q}(k)$, we will shift $g(k)$ instead of shifting $q(k)$. Now, $\hat{q}(k)$ is as follows:

$$\begin{aligned}
\hat{q}(k) &= q(k - (n_1 - 1)) \\
&= \Delta x \frac{\alpha}{m} \sum_{j=0}^{m-1} (\alpha F(j)^* G(j)) e^{2\pi\sqrt{-1}\frac{j(k-(n_1-1))}{m}} \\
&= \Delta x \frac{\alpha}{m} \sum_{j=0}^{m-1} (\alpha F(j)^* G(j) e^{-2\pi\sqrt{-1}\frac{j(n_1-1)}{m}}) e^{2\pi\sqrt{-1}\frac{jk}{m}} \\
&= \Delta x \frac{\alpha}{m} \sum_{j=0}^{m-1} (\alpha F(j)^* \hat{G}(j)) e^{2\pi\sqrt{-1}\frac{jk}{m}}
\end{aligned}$$

and, $\hat{G}(j)$ is as follows:

$$\begin{aligned}
\hat{G}(j) &= G(j) e^{-2\pi\sqrt{-1}\frac{j(n_1-1)}{m}} \\
&= \frac{1}{\alpha} \sum_{k=0}^{m-1} g(k) e^{-2\pi\sqrt{-1}\frac{j(k+(n_1-1))}{m}} \\
&= \frac{1}{\alpha} \sum_{k=0}^{m-1} g(k - (n_1 - 1)) e^{-2\pi\sqrt{-1}\frac{jk}{m}}
\end{aligned}$$

Therefore, if $g(j)$ is shifted in advance so that $\hat{g}(j)$ is defined as follows, then $\hat{q}(k)$ can be obtained directly.

$$\hat{g}(j + n_1 - 1) = g(j) \quad (j = 0, \cdots, n_2 - 1)$$

2.1.2.8 Multidimensional (Continuous) Convolution and Multidimensional (Continuous) Correlation

The convolution integral and correlation integral can be extended to multiple dimensions. For example, in three dimensions, the convolution and correlation of $f(x, y, z)$ and $g(x, y, z)$ are defined as follows as triple integrals.
Convolution:

$$p(x, y, z) = \int_{-\infty}^{\infty} f(\xi, \eta, \zeta) g(x - \xi, y - \eta, z - \zeta) d\xi d\eta d\zeta$$

Correlation:

$$q(x, y, z) = \int_{-\infty}^{\infty} f(\xi, \eta, \zeta) g(x + \xi, y + \eta, z + \zeta) d\xi d\eta d\zeta$$

In a similar manner as described for one-dimension, the following convolution theorem and correlation theorem hold.

$$\mathcal{F}\{p(x, y, z)\} = \mathcal{F}\{f(x, y, z)\}\mathcal{F}\{g(x, y, z)\}$$

$$\mathcal{F}\{q(x, y, z)\} = \mathcal{F}\{f(x, y, z)\}^* \mathcal{F}\{g(x, y, z)\} \quad (f(x, y, z) \in \boldsymbol{R})$$

Here, $\mathcal{F}\{f\}$ represents the Fourier transform of f. If $f(x, y, z)$ is a complex function in the correlation integral, then the negative frequency Fourier transform of $f(x, y, z)$ should be used instead of $\mathcal{F}\{f(x, y, z)\}^*$.

2.1.2.9 Power Spectrum

The quantity defined by $P(\xi) = |F(\xi)|^2$ for the Fourier integral $F(\xi)$ of $f(x)$, which is shown below,

$$F(\xi) = a \int_{-\infty}^{\infty} f(x) e^{-\sqrt{-1}\xi x} dx$$

is called the power spectrum (density function) of $f(x)$. Normally, the power spectrum is normalized so that Parseval's Theorem, which is shown below, holds.

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \frac{1}{2\pi a} \int_{-\infty}^{\infty} P(\xi) d\xi$$

As a quantity corresponding to the power spectrum of the continuous function $f(x)$, we consider **the raw periodogram** $p(j)$ for the discrete function $c(k)$ $(k = 0, 1, \cdots, n - 1)$ having period n. The discrete Fourier transform $C(j)$ of $c(i)$, which is shown below:

$$C(j) = \frac{1}{\alpha} \sum_{k=0}^{n-1} c(k) e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n - 1)$$

is used to define the periodogram $p(j)$ as follows:

$$p(j) = \beta |C(j)|^2$$

β is a suitable constant determined by the method of selecting the system of units. In the functions of this manual, $\alpha = 1$ and $\beta = \frac{1}{n^2}$ usually are selected. At this time, according to Parseval's Theorem, the total power corresponding to the time (or space) domain will be $\frac{\sum_{k=0}^{n-1} \{c(k)\}^2}{n}$. When $c(k)$ is a time series, **a two-sided power spectrum** and **a one-sided power spectrum** can be considered in a similar manner as described for a discrete Fourier transform (which is also called a Fourier spectrum). With a two-sided power spectrum, to correspond

to the property that the domain of a function normally is symmetric to the left and right of the origin for a continuous Fourier transform, a half period of data is considered as follows, shifted one period:

$$\{\tilde{p}(j)\}_{j=-(n-m-2),\cdots,-1,0,1,\cdots,m} = \{p(m+1), p(m+2), \cdots, p(n-1), p(0), p(1), \cdots, p(m)\}$$

(Here, $m = \lfloor \frac{n}{2} \rfloor$ and $\lfloor x \rfloor$ represents the maximum integer not exceeding x.) At this time, $p(0)$ is the element corresponding to zero frequency. The frequency corresponding to each periodogram $\tilde{p}(j)$ is $\frac{j}{nT}$ $(j = -(n-m-2), \cdots, -1, 0, 1, \cdots, m)$. Here, T is the sampling interval. With a one-sided spectrum, to eliminate negative frequencies, this may also be considered as follows:

$$\{\tilde{p}(j)\}_{j=0,1,\cdots,m} = \begin{cases} \{p(0), 2p(1), \cdots, 2p(m-1), p(m)\} & n\text{:Even number} \\ \{p(0), 2p(1), \cdots, 2p(m)\} & n\text{:Odd number} \end{cases}$$

The frequency corresponding to each periodogram $\tilde{p}(j)$ is $\frac{j}{nT}$ $(j = 0, 1, \cdots, m)$. The frequency interval $\frac{1}{nT}$ of sample points in the frequency domain of the discrete Fourier transform is also called **the resolution**.
Since the discrete Fourier transform is a square approximation (an approximation by using the trapezoidal formula may also be used) for the Fourier series, to raise the approximation precision, a larger number of data n must be taken. On the other hand, as described earlier, since the value of the Fourier series of a periodic function matches the continuous Fourier transform of a periodic function truncated at one of its periods, except for a constant factor, a periodogram can be expected to give a good approximation of the power spectrum for this kind of continuous function if the number of data n is sufficiently large. However, the original function reflected by the series used for the power spectrum estimate usually is not a periodic function, and even if it is periodic, it is not truncated exactly at one of its periods. A raw periodogram is treated as a discrete Fourier transform approximation of an autocorrelation function, from its definition. Figure 2−4 shows the calculation results of a raw periodogram and the discrete Fourier transform of the autocorrelation function for the discrete data $u_i = \cos(0.62\pi i) + \cos(0.14\pi i)$ $(i = 0, 1, \cdots, n-1;\ n = 50)$. Here, when calculating the autocorrelation function, the period is assumed to be $2n$, and $u_{n+1} = \cdots = u_{2n-1} = 0$ is assumed. For reference, the figure also shows the value of the resolution Δf when the sampling interval $T = \Delta t = 0.5[sec]$ is assumed. In this case, we can see that the power is concentrated at the portions corresponding to the frequency $0.14[Hz]$ and the frequency $0.62[Hz]$, and the expected results are given. Now, since the resolution is smaller for the discrete Fourier transform of the autocorrelation function than for the periodogram, it is assumed to be a more desirable form. However, note that the corresponding number of calculation points is double. Also, note that the discrete Fourier transform of the autocorrelation function is a real number. (The discrete Fourier transform of the more general cross correlation function is usually a complex number.)

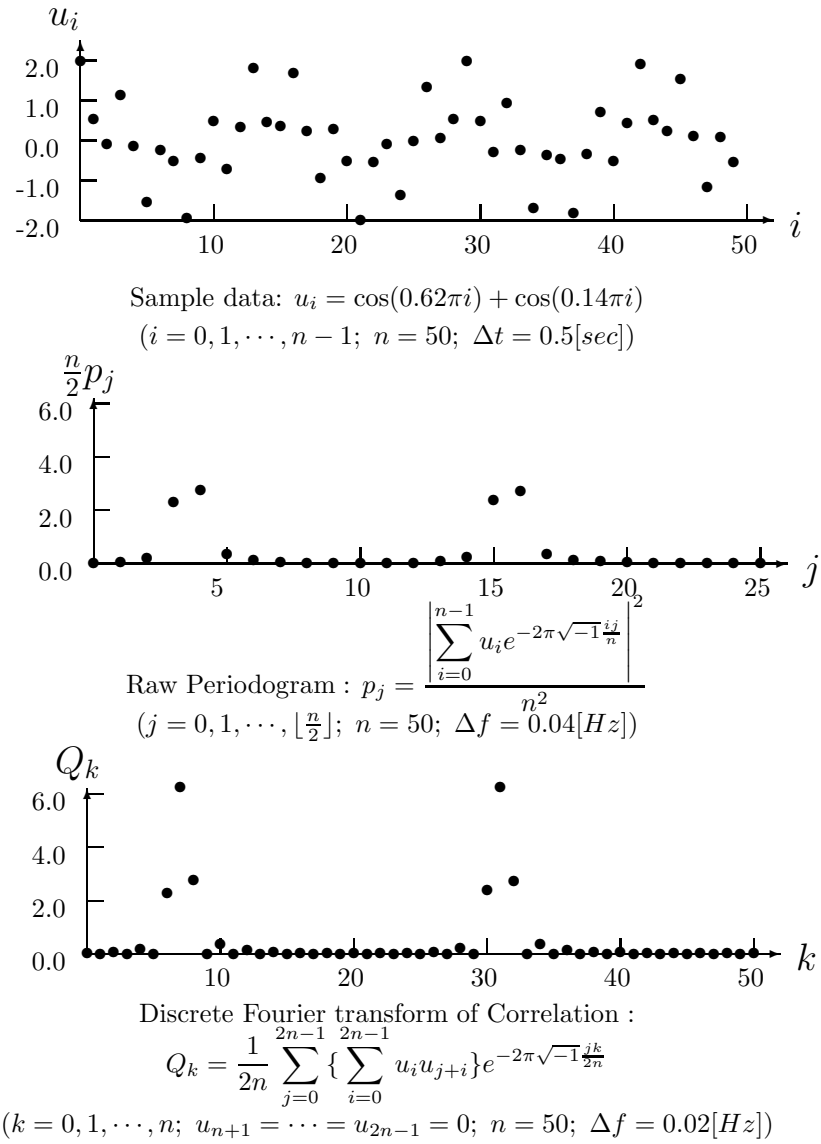


Figure 2−4 Periodogram and Fourier Transform of the Autocorrelation Function

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 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 is used. Along with this, the modified periodogram $\hat{p}$,

which modifies the periodogram definition as follows, usually is used.

$$\hat{p}(j) = \beta|\hat{C}(j)|^2$$

Here, $\hat{C}(j)$, which is a discrete Fourier transform having as values the original series $c(k)$ multiplied by the truncation function $w(k)$, is defined by the following equation.

$$\hat{C}(j) = \frac{1}{\alpha}\sum_{k=0}^{n-1} w(k)c(k)e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

$w(k)$ is also called **the data window**. The modified periodogram may also be defined as the convolution of the frequency domain using the Fourier transform $W(j)$ of $w(k)$. In this case, $W(j)$ may also be called the spectral window. The $w(k)$ corresponding to this is also called the lag window. Although the data window was originally proposed for sidelobe suppression in the frequency domain due to truncation, mathematically, the effect of the data window is the same as that of a smoothing expression in the frequency domain. Therefore, if the data window is selected appropriately, smoothing of the power spectrum can also be performed. In addition, when a time-series spectrum analysis is performed, time series data having a mean value of 0 often is considered. Since the mean value corresponds to the zero-frequency component of the Fourier transform, if the zero-frequency component after the transformation is cut, a similar effect can be expected. However, when the modified periodogram is used, since the mean value varies due to the multiplication by the data window, it is meaningless to set the mean value to zero in advance. Now, a difference of a constant factor occurs in the periodogram definition due to the multiplication by the truncation function $w(k)$. Originally, the total power was to be calculated in the time (or space) domain and this was to be corrected (according to Parseval's theorem) so that it matched the total power in the frequency domain. However, the computational cost for this kind of correction is not small. Also, it may be difficult to estimate the total power in the time (or space) domain. $\tilde{p}(j)$, which is defined by the following equation, may be used to correct the power according to a truncation function.

$$\tilde{p}(j) = \frac{\hat{p}(j)}{n\sum_{k=0}^{n-1}\{w(k)\}^2}$$

In this case, since the sum of the squares can be calculated analytically according to the truncation function used, the computational cost required to correct the power will not be as large. For the total power of the corresponding series, $\frac{\sum_{k=0}^{n-1}\{c(k)\}^2}{n}$ be the generalization when a square truncation is used.
Now, since the more the sidelobe of the spectral window $W(j)$ is suppressed, the more the spectral width of $W(j)$ increases, the estimated waveform of the power spectrum also will be blurred. 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. Some representative truncation functions are shown below.

$$w_j = \begin{cases} \sin^2(\pi u_j) & \text{(Hanning window)} \\ 1 - |2u_j - 1| & \text{(Bartlett window)} \\ 1 - (2u_j - 1)^2 & \text{(Welch window)} \\ \begin{cases} 16{u_j}^3 & 0 \le u_j < \frac{1}{4} \\ 1 - 6u_j(u_j - 1)^2 & \frac{1}{4} \le u_j \le \frac{1}{2} \\ 1 - 6u_j(u_{n-j+1} - 1)^2 & \frac{1}{2} \le u_j \le \frac{3}{4} \\ 16{u_{n-j+1}}^3 & \frac{3}{4} \le u_j < 1 \end{cases} & \text{(Parzen window)} \end{cases}$$

Here, $u_j = \frac{j}{n}$. If u_j is selected in this way, $w_0 = 0$. Therefore, since the component corresponding to c_0 will be invalid, the data windows may also be defined in a slightly modified form. The data windows are represented as

follows as time domain functions that are nonzero only for $|t| \leq 1$.

$$w(t) = \begin{cases} \dfrac{1+\cos \pi t}{2} = \cos^2 \dfrac{\pi t}{2} & \text{(Hanning window)} \\ 1 - |t| & \text{(Bartlett window)} \\ 1 - t^2 & \text{(Welch window)} \\ \left\{ \begin{array}{ll} 1 - 6t^2 + 6|t|^3 & |t| \leq \frac{1}{2} \\ 2(1-|t|)^3 & \frac{1}{2} \leq |t| \leq 1 \end{array} \right\} & \text{(Parzen window)} \end{cases}$$

Figure 2−5 shows graphs of these data windows (w_i) and the amplitudes ($|W_j|$) of their discrete Fourier transforms (W_j). To raise the resolution $\frac{1}{nT}$ of the discrete Fourier transform, you should increase the number of sample

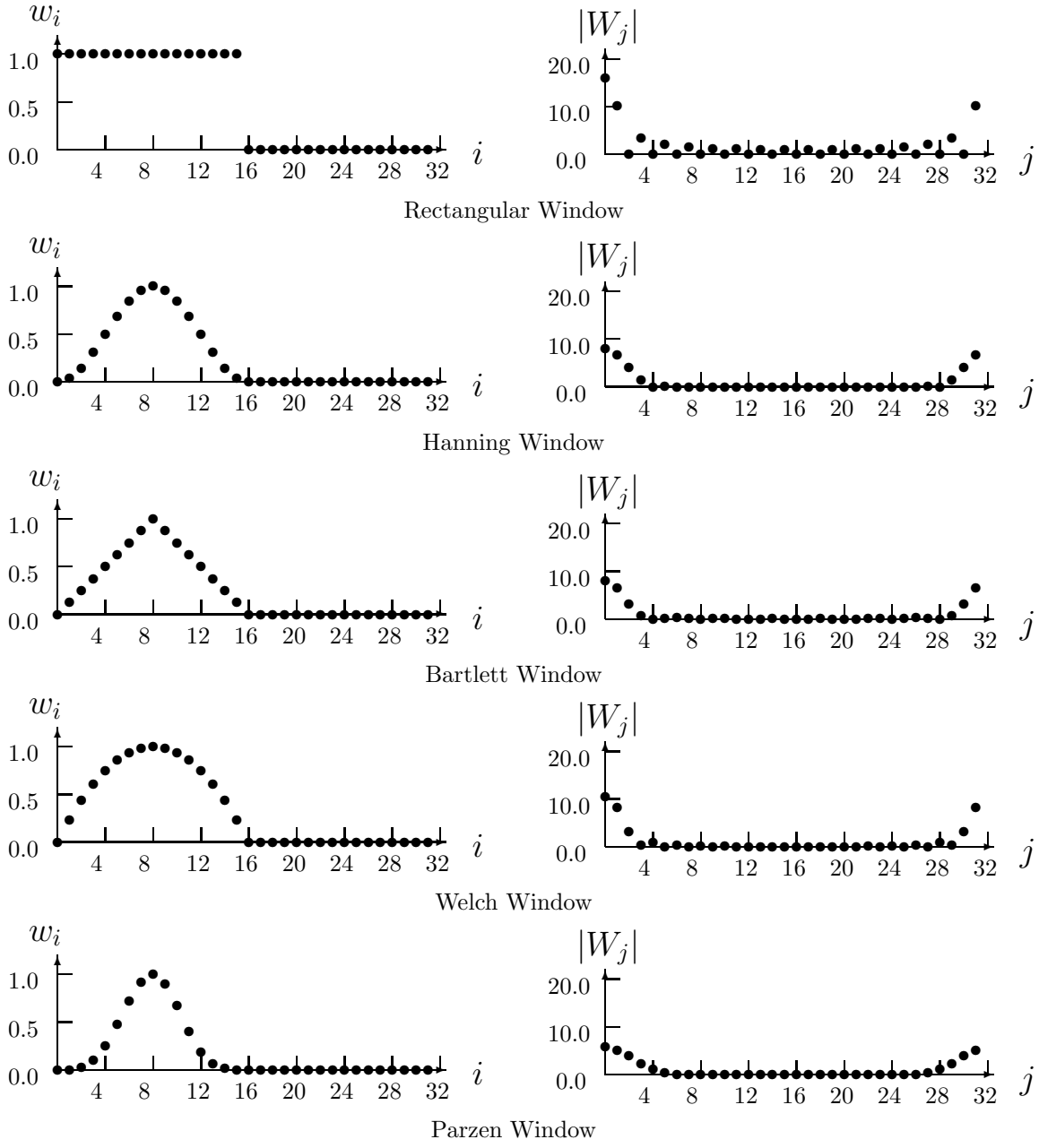


Figure 2−5 Data Windows and Absolute Values of Their Discrete Fourier Transforms

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 data 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 original series so that they overlap. For details, refer to the Reference Bibliography. Also, 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**.

2.1.2.10 Laplace Transform

(1) Laplace Transform

Given the function $f(t)$, the Laplace transforms consist of the regular transform defined by:

$$F(s) = \int_0^\infty f(t)e^{-st}dt \tag{2.1}$$

and the inverse transform, which is the following **Bromwich integral**:

$$f(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \tag{2.2}$$
$$\text{where}, \gamma > \gamma_0, \gamma_0 : \text{Abscissa of convergence}, i = \sqrt{-1}$$

$f(t)$, which is called **the original function**, and $F(s)$, which is called **the image function**, can be written as follows:

$$\text{Regular transform} \quad F(s) = \mathcal{L}\{f(t)\} \tag{2.3}$$
$$\text{Inverse transform} \quad f(t) = \mathcal{L}^{-1}\{F(s)\} \tag{2.4}$$

(a) The case when the following conditions hold for the image function $F(s)$

(I) When $\Re(s) > 0$, $F(s)$ is regular

(II) When $\Re(s) > 0$, $\lim_{s\to\infty} F(s) = 0$

(III) When $\Re(s) > 0$, $F(s^*) = F^*(s)$
($\Re(s)$ is real part of s; * is the complex conjugate symbol)

(i) Exponential function approximation

If the exponential function e^s can be approximated by a rational function having poles only for $\Re(s) > 0$, equation (2.2) can be reduced to an integral around this pole. Therefore, consider the following function expression as the exponential function approximation method.

$$E_{ec}(s,a) = \frac{e^a}{2\cosh(a-s)} \qquad (a > 0) \tag{2.5}$$

This expression can be rewritten in the following two ways.

$$E_{ec}(s,a) = \frac{e^a}{2}\sum_{n=-\infty}^{\infty}\frac{(-1)^{n_i}}{s-\{a+i(n-0.5)\pi\}} \tag{2.6}$$

$$E_{ec}(s,a) = e^s - e^{-2a}e^{3s} + e^{-4a}e^{5s} - \cdots \tag{2.7}$$

From equation (2.7), we know that $E_{ec}(s,a)$ is a good approximation of the exponential function when $a \gg \Re(s)$, and from equation (2.6), we know that the poles of $E_{ec}(s,a)$, which are all of order 1, are equally spaced on straight line $s = a$, and the residue is $\frac{(-1)^n ie^a}{2}$.

Let's define the function $f_{ec}(t,a)$ as follows by assigning $E_{ec}(st)$ in place of e^{st} in the Bromwich integral:

$$f_{ec}(t,a) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)E_{ec}(st,a)ds \qquad (0<\gamma<a) \tag{2.8}$$

We can rewrite this by using equation (2.6) and (2.7) as follows:

$$f_{ec}(t,a) = f(t) - e^{-2a}f(3t) + e^{-4a}f(5t) - \cdots \tag{2.9}$$

$$f_{ec}(t,a) = \frac{e^a}{t}\sum_{n=1}^{\infty}(-1)^n\Im\{F\Big(\frac{a}{t}+i\frac{(n-0.5)\pi}{t}\Big)\} \tag{2.10}$$

($\Im(s)$ is imaginary part of s)

(ii) Euler's transformation

To numerically calculate the sum of an infinite series like the one in equation (2.10), the series must be truncated at some appropriate number of terms N. This section explains how to use Euler's transformation to calculate the sum efficiently. Euler's transformation is a method of transforming the series on the left hand side of the following equation to the series on the right hand side:

$$\sum_{n=0}^{\infty} a_n = \sum_{p=0}^{\infty} 2^{-(p+1)}\Delta^p a_0 \tag{2.11}$$
$$\text{where, } \Delta a_0 = a_0 + a_1, \Delta^2 a_0 = \Delta a_0 + \Delta a_1, \cdots, \Delta^n a_0 = \Delta^{n-1}a_0 + \Delta^{n-1}a_1$$

$\Delta^n a_k$ is, which is the n-th difference, is defined by the following equation.

$$\Delta^n a_k = a_k - \begin{pmatrix} n \\ 1 \end{pmatrix} a_{k+1} + \begin{pmatrix} n \\ 2 \end{pmatrix} a_{k+2} - \cdots \pm \begin{pmatrix} n \\ n \end{pmatrix} a_{k+n} \tag{2.12}$$

It is known that when the following conditions are satisfied for a series that has be en transformed by Euler's transformation, it converges faster than the original series.

(A) $\sum_{n=0}^{\infty} a_n$ is alternating series, sequence of numbers $\{|a_n|\}$ monotonically approaches 0.

(B) $\frac{1}{2} < |\frac{a_{n+1}}{a_n}| \leq 1$

Since the effect of Euler's transformation is larger as $|\frac{a_{n+1}}{a_n}|$ is closer to 1, the first k terms of equation (2.11) are normally calculated, and if Euler's transformation is performed for terms $k+1$ and later, we have:

$$\sum_{n=0}^{p-1} 2^{-(n+1)}\Delta^n a_0 = 2^{-p}\sum_{q=1}^{p} A_{pq}a_{q-1}$$
$$\text{where, } A_{pp} = 1, A_{p,q-1} = A_{pq} + \begin{pmatrix} p+1 \\ q \end{pmatrix} \tag{2.13}$$

Therefore, the approximate value of $f(t)$ is calculated from the following equations:

$$f_{ec}^{kp} = \frac{e^a}{t}\left(\sum_{n=0}^{k-1} F_n + 2^{-p}\sum_{q=0}^{p-1} A_{pq}F_{k+q}\right)$$
$$F_n = (-1)^{n+1}\Im\{F\Big(\frac{a}{t}+i\frac{(n+0.5)\pi}{t}\Big)\} \tag{2.14}$$

In the actual calculation, the right hand site of equation (2.11) is truncated at term p. The truncation error at this time is given by:

$$R_p = \frac{1}{2^p}[\Delta^p a_0 + \Delta^p a_1 + \Delta^p a_1 + \cdots] \tag{2.15}$$

However, if the following additional condition also holds:

- $a_n = f(n)$ can be written, and $f^{(n)}(x)$, which is the differential coefficient of order po of $f(x)$, decreases monotonically as x increased with a fixed sign for positive values of x.

then the following relationship is confirmed:

$$|R_p(0)| < \frac{1}{2^p}|\Delta^p v_0| \tag{2.16}$$

[**Notes**]

When $F(s)$ has a factor like e^{-sx}, since F_n does not satisfy condition (1(a)iiA), precision may drop. In this time, the image function has characteristics described below.

- $f(t) = 0$ at $t < t_0$
- When $t = t_0$, $f(t)$or its derivative becomes discontinuous.

When it is known that $f(t) = 0$ at $t < t_0$, the t axis should be shifted by t_0 and the image function of:

$$g(t') = f(t' + t_0) \tag{2.17}$$

which is given by:

$$G(s) = \exp(t_0 s) \cdot F(s) \tag{2.18}$$

should be handled.

(b) When $F(s)$ has singular point for $\Re(s) > 0$

When the abscissa of convergence α of $F(s)$ is known, if we let $G(s)$ be defined as:

$$G(s) = F(s + b), b > \alpha \tag{2.19}$$

then $G(s)$ is regular for $\Re(s) > 0$. In addition, if we let the original functions of $F(s)$ and $G(s)$ be $f(t)$ and $g(t)$, we have the following relationship:

$$f(t) = e^{bt} g(t) \tag{2.20}$$

Therefore, when $\Re(s) > 0$, we can perform the calculations using the following equations.

$$\begin{aligned} f_{ec}^{kp}(t, a) &= e^{bt}\left(\frac{e^a}{t}\right)\sum F_n \\ F_n &= (-1)^n \Im\left(F\left[\frac{a}{t} + b + i\frac{(n-0.5\pi)}{t}\right]\right) \end{aligned} \tag{2.21}$$

[Determining the abscissa of convergence]

- When $F(s)$ is a rational function

  Express $F(s)$ as follows using the real coefficient polynomials $Q(s)$ and $P(s)$:

  $$F(s) = \frac{Q(s)}{P(s)} = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_{m+1}}{a_1 s^n + a_2 s^{n-1} + \cdots + a_{n+1}} \tag{2.22}$$

  If we let the roots of the denominator polynomial $P(s) = 0$ be $s_1, s_2, \cdots, s_n$, then the abscissa of convergence is given by the maximum value of the real part of the roots as follows:

  $$\alpha = \max \Re[s_k] \tag{2.23}$$

  If the real part of all roots of $P(s) = 0$ are negative, then the abscissa of convergence α will satisfy $\alpha < 0$, and $F(s)$ will be regular for $\Re(s) > 0$. Therefore, the following theorem is used to determine the sign of the abscissa of convergence. A necessary and sufficient condition for all roots of the polynomial $P(s)$ to be negative is that when the ratio of the sum of the odd terms and the sum of the even terms is expanded in a continued fraction as follows:

  $$\frac{a_1 s^n + a_3 s^{n-2} + \cdots}{a_2 s^{n-1} + a_4 s^{n-3} + \cdots} = h_1 s + \cfrac{1}{h_2 s + \cfrac{1}{h_3 s + \cfrac{1}{h_4 s + \cdots}}} \tag{2.24}$$

  the coefficients $h_1, h_2, \cdots$, are all positive numbers. A polynomial that satisfies this theorem is called a Hurwitz polynomial. The coefficients of the continued fraction $h_1, h_2, \cdots$, are calculated by using the Euclidean algorithm. If $F(s)$ has singular point for $\Re(s) > 0$, create the polynomial

$P(s+b)$ for an appropriate positive number b and use the decision method described above to decide whether or not it is a Hurwitz polynomial. Repeat this decision process while increasing b, and obtain the abscissa of convergence for b so that $F(s+b)$ becomes regular.

- When $F(s)$ is a general function
  When $F(s)$ is a general function, such as irrational function or distribution, since there is no effective method for determining whether or not $F(s)$ is regular for $\Re(s) > 0$, the user must specify the abscissa of convergence.

(c) When $F(s^*) \neq F^*(s)$
Let x be real number, and let $F_1(s)$ and $F_2(s)$ be defined as follows:

$$F_1(x) = 0.5[F(x) + F^*(x)] \tag{2.25}$$

$$F_2(x) = 0.5i[F(x) - F^*(x)] \tag{2.26}$$

Next, rewrite x as s. $F_1(s)$ and $F_2(s)$ satisfy conditions (1(a)i)-(1(a)iii). Therefore, if we obtain $f_1(t)$ and $f_2(t)$ as inverse transforms, then $f(t)$, which is defined as:

$$f(t) = f_1(t) - if_2(t) \tag{2.27}$$

becomes $\mathcal{L}^{-1}F(s)$.

(d) Determining parameter values

(i) Truncation term count $k+p$ of the sum of a series
For Euler's transformation to be effective and the truncation error evaluation expression (2.16) to be usable, at least $|F_n|$ must monotonically decrease. In general, $|F_n| = |\Im(F[\frac{a}{t} + i\frac{(n+0.5)\pi}{t}])$ varies in a complicated manner along with n. If we represent the maximum value of the imaginary part of a singular point of $F(s)$ by ω_m, then when $\frac{(n+0.5)\pi}{t} > \omega_m$, $|F_n|$ monotonically decreases. Therefore, the number of terms k in the normal sum in equation (2.16) must be increased in proportion to t so that $k > 0.5\frac{\omega_m}{\pi}t$. Actually, since k also is related to the value of a, let:

$$k = k_1 + k_2 t \tag{2.28}$$

and determine k_1 and k_2 so that the truncation error (2.16) become the desired value. When you want to obtain $f(t)$ in the range $t_1 \leq t \leq t_2$, one method of determining k_1 and k_2 is to first set $t = t_1$ and determine $k(t_1)$ for which the truncation error can be ignored, and then, in a similar manner, set $t = t_2$ and determine $k(t_2)$ for which the truncation error can be ignored. k_1 and k_2 are determined from:

$$\begin{cases} k_1 + k_2 t_1 = k(t_1) \\ k_1 + k_2 t_2 = k(t_2) \end{cases} \tag{2.29}$$

Degree p of Euler's transformation tend to nearly proportionate to exponent of calculation precision. Therefore, if required calculation precision is 10^{-d}, you should input d to parameter p.

(ii) Value of parameter a for approximating the exponential function using $E(s, a)$
Since we can obtain:

$$|f(t) - f_{ec}(t, a)| = |e^{-2a}f(3t) - e^{-4a}f(5t) + \cdots| \tag{2.30}$$

if we calculate $f_{ec}(3t, a)$ and $f_{ec}(5t, a)$ together with $f_{ec}(t, a)$, we can evaluate:

$$|f(t) - f_{ec}(t, a)| \simeq |e^{-2a}f(3t) - e^{-4a}f(5t) + \cdots| \tag{2.31}$$

Actually, if we assume $f(t)$ and $f(3t)$ are of the same order, e^{-2a} is considered to be the relative error, and then exponent of the relative error is nearly equal to a shown in the table below.

a	3	4	5	6
e^{-2a}	2.4×10^{-3}	3.4×10^{-4}	4.5×10^{-5}	6.2×10^{-6}

2.1.2.11 Wavelet transform

(1) Haar functions

Let the domain of the input data that is to be subject to the Wavelet transform be $[0, a]$. Let the Haar functions $H_{00}(x)$ and $H_{01}(x)$ be defined as follows:

$$H_{00}(x) = 1/\sqrt{a},\ 0 \leq x \leq a$$

$$H_{01}(x) = \begin{cases} 1/\sqrt{a} & \text{if } x \leq a/2 \\ -1/\sqrt{a} & \text{if } x > a/2 \end{cases}$$

For $H_{01}(x)$, create $H_{mn}(x)$ as follows. Divide the interval $[0, a]$ into 2^m intervals of equal length. Number the divided subintervals beginning with 1, starting at lowest values, and let the number of a subinterval be represented by n. If the subinterval is $[b1, b2]$, then $b1$ and $b2$ are as follows.

$$b1 = \frac{a}{2^m} \times (n-1)$$

$$b2 = \frac{a}{2^m} \times n$$

Let the Haar function in subinterval $[b1, b2]$ be defined as follows.

$$H_{mn}(x) = \begin{cases} \sqrt{2^m/a} & \text{if } x \leq (b1+b2)/2 \\ -\sqrt{2^m/a} & \text{if } x > (b1+b2)/2 \end{cases}$$

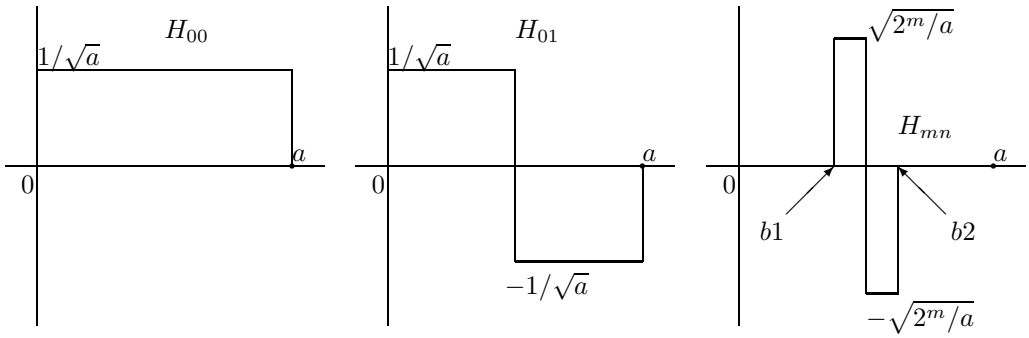
Function values are selected so that the interval $[0, a]$ is normalized. That is, they are selected so that the following equation is satisfied.

$$\int_0^a H_{mn}(x) \times H_{mn}(x)dx = 1$$

Also, note that the following must also be satisfied for different m and m' and n and n'.

$$\int_0^a H_{mn}(x) \times H_{m'n'}dx = 0$$

In other words, these Hmn are made into a system of orthonormal functions. Also , as you can see from the creation method described above, the numbers that can be specified as n for a given m are $n = 1, \ldots, 2^m$.



(2) Wavelet transform according to Haar functions

To perform a wavelet transform according to Haar functions means to integrate th e product of given input data and each Haar function over the existence range of the input data. If we suppose that the input data is a continuous function $y = f(x)$, the wavelet transform is defined as follows.

$$C_{mn} = \int_0^a f(x) \times H_{mn}(x)dx$$

To perform this calculation, we need the following information for the Haar function Hmn: (1) value y_H, (2) position x_0 of the rising point, (3) position x_1 of the positive/negative reversal point, and (4) position x_2 of the point where the value returns to zero. To create this information for the Haar function $H_{mn}(x)$ when the indexes (m, n) are given, this library provides the functions 2.18.1 $\left\{\begin{matrix}\text{ASL\_dfwth1}\\ \text{ASL\_rfwth1}\end{matrix}\right\}$. When the input data is a continuous function, the wavelet transform can be computed by numerical integration by using the results of these functions.

$$C_{mn} = \int_{x_0}^{x_1} f(x) \times y_H dx - \int_{x_1}^{x_2} f(x) \times y_H dx$$

When the input data is discrete, the wavelet transform is computed as follows. Assume that the range of the discrete input data values is "fixed within the range half way to the positions of the adjacent data." That is, if the data (x_i, y_i) is given, assume that the input data has y_i in the range $[\frac{1}{2}(x_{i-1}+x_i), \frac{1}{2}(x_i+x_{i+1})]$ for this data. Perform the integration described above for the input data that has become partially continuous in this way. If the sampling is performed with equal spacing dx and the sampling count is 2^k (where, k is a natural number), then for the $[b1, b2]$ of $H_{(k-1)n}$, input data can be made to exist at either $x = b1 + (b2 - b1)/4$ or $x = b2 - (b2 - b1)/4$. The Haar functions for this kind of input data are a complete system, and the input data can be completely represented by a linear combination of $H_{mn}, m = 0, 1, \cdots, (k-1), n = 1, 2, \cdots, 2^{k-1}$. For example, when there are 2= 2^1 data (x1,y1) and (x2,y2), C_{00}, which is shown as follows,

$$C_{00} = \int_{-(x2-x1)/2}^{x2+(x2-x1)/2} H_{00}(x + (x2 - x1)/2) \times ydx$$

is the mean value of the two data. However, since the Haar function is outside of the transform data existence range at both ends, the integration range is widened at both sides by $(b2 - b1)/2$. Furthermore, C_{01}, which is shown as follows,

$$C_{01} = \int_{-(x2-x1)/2}^{x2+(x2-x1)/2} H_{01}(x + (x2 - x1)/2) \times ydx$$

represents the discrepancy from the mean value of the two data. Since there are positive and negative differences having the same absolute value, this discrepancy can be completely represented by H_{01}. Therefore, the input data itself can be represented by using

$$y = C_{00} \times H_{00}(x) + C_{01} \times H_{01}$$

If the sampling is performed with equal spacing dx and the sampling count is 2^k (where, k is a natural number), the integration calculation becomes simpler since it can be completed without having to be concerned with the spacing at which individual input data exist. To enable this integration calculation to be completed easily, this library provides the functions 2.18.4 $\left\{\begin{matrix}\text{ASL\_dfwth2}\\ \text{ASL\_rfwth2}\end{matrix}\right\}$, which output array lr, which indicates the

positive, negative or zero values of H_{mn} in the interval $[0, 1]$, in an array of size $na = 2^k$, which is the same as the number of input data. When the existence range of the input data is assumed to be $[b1, b2]$, the integration range of the above calculation becomes $[b1 - dx/2, b2 + dx/2]$, and with $a = (b2 - b1) + dx$, C_{mn} is as follows.

$$C_{mn} = \sqrt{\frac{2^m}{a}} \sum \mathrm{lr[i-1]} \times \mathrm{y_i}$$

(3) Inverse wavelet transform according to Haar functions
The inverse transform for a Haar function wavelet transform reconstructs the original data for C_{mn}. If we let the original data be $f(x)$, the reconstruction of $f(x)$ is represented as follows.

$$f'(x) = \sum_{mn} C_{mn} \times H_{mn}(x)$$

When the original data does not consist of continuous values or when the sampling spacing is not equally spaced, the fact that $f'(x)$ does not match $f(x)$ can be seen from the fact that this sum does not create frequency values of at most 1/2 of the range having values when the Haar function is the maximum value m. When the sampling spacing is equally spaced and the sampling count is 2^k (where, k is a natural number), the original data can be regenerated by letting (maximum value m)=k.

(4) Mexican hat function
The Mexican hat function $\varphi_{MH}(x)$, which is used for a continuous Wavelet transform, is given as follows.

$$\varphi_{MH}(x) = (1 - 2x^2)e^{-x^2}$$

This function has values in the range $[-\infty, +\infty]$. Let the parameter corresponding to the frequency be a, let there be a shift of b in the x-axis direction, and let the base of the Wavelet transform be given as follows.

$$\phi_{MH}(x; a, b) = \frac{1}{\sqrt{(C)}} \varphi_{MH}(\frac{(x-b)}{a})$$

Here, let C be the normalization factor so that the following equation is satisfied.

$$\int_{-\infty}^{+\infty} \phi_{MH}(x; a, b)^2 dx = 1$$

From the following equations

$$\int_{-\infty}^{+\infty} e^{-x^2} dx = \sqrt{\pi}$$

$$\int_{-\infty}^{+\infty} x^2 e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$$

$$\int_{-\infty}^{+\infty} x^4 e^{-x^2} dx = \frac{3 \times \sqrt{\pi}}{4}$$

the normalization factor C is as follows.

$$C = a(1 - \frac{a^2}{2} + \frac{3a^4}{4})\sqrt{\frac{\pi}{2}}$$

For an arbitrary function $f(x)$, the Wavelet transform according to this function is given as follows.

$$\int_{-\infty}^{+\infty} \phi_{MH}(x; a, b) f(x) dx (W_{\phi_{MH}} f)(b, a) = \int_{-\infty}^{+\infty} \phi_{MH}(x; a, b) f(x) dx$$

(5) French hat function

The French hat function $\varphi_{FH}(x)$ is defined as follows.

$$\varphi_{FH}(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ -\frac{1}{2} & \text{if } 1 < |x| \leq 3 \\ 0 & otherwise \end{cases}$$

In a similar manner as described for the Mexican hat function, let the parameter corresponding to the frequency be a, let there be a shift of b in the x-axis direction, and let the base of the Wavelet transform be given as follows.

$$\phi_{FH}(x; a, b) = \frac{1}{\sqrt{(C)}}\varphi_{FH}(\frac{(x-b)}{a})$$

Here, let C be the normalization factor so that the following equation is satisfied.

$$\int_{-\infty}^{+\infty} \phi_{FH}(x; a, b)^2 dx = 1$$

$$C = 3a$$

For an arbitrary function $f(x)$, the Wavelet transform according to this function is given as follows.

$$(W_{\phi_{FH}} f)(b, a) = \int_{-\infty}^{+\infty} \phi_{FH}(x; a, b) f(x) dx$$

2.1.3 Reference Bibliography

(1) Brigham, E. Oran, "The Fast Fourier Transform", Prentice-Hall Inc. , (1974).

(2) Cochran, W. T. et al. , IEEE Trans. Audio and Electroacoustics, Vol.15. pp.45-55 (1967).

(3) Gentleman, W. M. and Sande, G. , AFIPS Conf. Proc. , Fall Joint Comput. Conf. , Vol. 29, pp. 563-578 (1966).

(4) Glassman, J. A. , IEEE Trans. Comput. , Vol. 19, pp. 105-116 (1970).

(5) Swarztrauber, P. N. , SIAM Rev. , Vol. 19, pp. 490-501 (1977).

(6) Temperton, C. , "Implementation of a Self-Sorting In-Place Prime-Factor FFT Algorithm", J. Comp. Phys., 58, 283 (1985).

(7) Temperton, C. , "Self-Sorting Mixed-Radix Fast Fourier Transforms", J. Comp. Phys. , 52, 1 (1983).

(8) Temperton, C. , "Fast Mixed-Radix Real Fourier Transforms", J. Comp. Phys. , 52, 340 (1983).

(9) Hosono, T. , "Numerical inversion of Laplace transform and some applications to wave optics", Radio Science, vol. 16, pp. 1015 (1981).

(10) Welch, P. D. , "The Use of the FFT for Estimation of Power Spectra: A Method Based on Averaging Over Short, Modified Periodograms", IEEE Trans. on Audio and Electroacoustics, Vol. AU-15, No. 2, pp. 70-73 (1967).

(11) Rader, C. M. , "An Improved Algorithm for High Speed Autocorrelation with Applications to Spectral Estimation", IEEE Trans. on Audio and Electroacoustics, Vol. AU-18, No. 4, pp. 439-442 (1970).

(12) Childers, D. G. (Ed.), "Modern Spectrum Analysis", IEEE Press (1978).

(13) Pease, M. C, An Adaption of the Fast Fourier Transform for Parallel Processing J. Assn. Comput. Mach., 15, 252 (1968); Stockham, T. G. , High Speed Convolution and Correlation AFIPS Conf. Proc. , 28, 229 (1966).

(14) Swarztrauber, P. N. , Vectorizing the FFTs Parallel Computations, 51 (1982).

(15) Singleton, R. C. , An Algorithm for Computing the Mixed Radix Fast Fourier Transform IEEE Trans. Audio and Electroacoust. , AU-17, 93 (1969); Singleton, R. C. , ALGOL Procedures for the Fast Fourier Transform Commun. ACM, 11, 773 (1968).

(16) Petersen, W. P. , Vector Fortran for Numerical Problems on CRAY-1 Commun. ACM, 26, 1008 (1983).

2.2 ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

2.2.1 [DEPRECATED]ASL_dfc1fb, ASL_rfc1fb One-Dimensional Complex Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

ASL_dfc1fb or ASL_rfc1fb computes the complex Fourier forward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

Backward transform

ASL_dfc1fb or ASL_rfc1fb computes the complex Fourier backward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

(2) **Usage**

Double precision:

ierr = ASL_dfc1fb (n, cr, ci, ld, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfc1fb (n, cr, ci, ld, isw, ifax, trigs, wk);

(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 input data values n (See Note (a))
2	cr	{D∗ / R∗}	ld	Input	Real part of input data c_k (See Note (b))
				Output	Real part of output results d_j (See Notes (b) and (c))
3	ci	{D∗ / R∗}	ld	Input	Imaginary part of input data c_k (See Note (b))
				Output	Real part of output results d_j (See Notes (b) and (c))
4	ld	I	1	Input	Size of array cr, ci
5	isw	I	1	Input	Processing switch(See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
6	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
7	trigs	{D∗ / R∗}	2 × n	Output	Work area. Trigonometric function table (See Note (d))
8	wk	{D∗ / R∗}	2 × n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

(b) $n \leq ld$

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

(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.	

(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) If we let the real and imaginary parts of the complex data c_k $(k = 0, \cdots, n-1)$ be $\Re\{c_k\}$ and $\Im\{c_k\}$, respectively, the c_k and elements of arrays cr and ci are associated as follows.

$$\begin{array}{llll} \Re\{c_0\} & \leftrightarrow \text{ cr[0]} & , \Im\{c_0\} & \leftrightarrow \text{ ci[0]} \\ \Re\{c_1\} & \leftrightarrow \text{ cr[1]} & , \Im\{c_1\} & \leftrightarrow \text{ ci[1]} \\ \cdots & \cdots \cdots & , \cdots & \cdots \cdots \\ \Re\{c_{n-1}\} & \leftrightarrow \text{ cr[n} - 1\text{]} & , \Im\{c_{n-1}\} & \leftrightarrow \text{ ci[n} - 1\text{]} \end{array}$$

Similarly, for the complex data $d_j (j = 0, \cdots, n-1)$.

(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 $c_k (k = 0, \cdots, n-1)$ be represented by $\hat{c}_k (k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{c}_k = nc_k \quad (k = 0, \cdots, n-1)$$

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 n, you should call this function once, and then use the after-initialization transform 2.2.2 $\left\{\begin{array}{l}\text{ASL\_dfc1bf}\\ \text{ASL\_rfc1bf}\end{array}\right\}$, 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 2.2.2 $\left\{\begin{array}{l}\text{ASL\_dfc1bf}\\ \text{ASL\_rfc1bf}\end{array}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.2.2 (7).

2.2.2 [DEPRECATED]ASL_dfc1bf, ASL_rfc1bf One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_dfc1bf or ASL_rfc1bf computes the complex Fourier forward transform (arbitrary radix) for the complex data $c_k (k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

Backward transform

ASL_dfc1bf or ASL_rfc1bf computes the complex Fourier backward transform (arbitrary radix) for the complex data $c_k (k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfc1bf  (n, cr, ci, ld, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfc1bf  (n, cr, ci, ld, isw, ifax, trigs, wk);
```

(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 input data values n (See Note (a))
2	cr	{D∗ / R∗}	ld	Input	Real part of input data c_k (See Note (b))
				Output	Real part of output results d_j (See Notes (b) and (c))
3	ci	{D∗ / R∗}	ld	Input	Imaginary part of input data c_k (See Note (b))
				Output	Imaginary part of output results d_j (See Notes (b) and (c))
4	ld	I	1	Input	Size of array cr, ci
5	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
6	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
7	trigs	{D∗ / R∗}	2 × n	Input	Trigonometric function table (See Note (a))
8	wk	{D∗ / R∗}	2 × n	Work	Work area
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

(b) $n \leq ld$

(c) $isw \in \{1, -1\}$

(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.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data n after the including-initialization function 2.2.1 $\left\{\begin{matrix}\text{ASL\_dfc1fb}\\ \text{ASL\_rfc1fb}\end{matrix}\right\}$ 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) If we let the real and imaginary parts of the complex data c_k $(k = 0, \cdots, n-1)$ be $\Re\{c_k\}$ and $\Im\{c_k\}$, respectively, the c_k and elements of arrays cr and ci are associated as follows.

$$\begin{array}{llll} \Re\{c_0\} & \leftrightarrow \text{ cr[0]} & , \Im\{c_0\} & \leftrightarrow \text{ ci[0]} \\ \Re\{c_1\} & \leftrightarrow \text{ cr[1]} & , \Im\{c_1\} & \leftrightarrow \text{ ci[1]} \\ \cdots & \cdots \quad \cdots & , \cdots & \cdots \quad \cdots \\ \Re\{c_{n-1}\} & \leftrightarrow \text{ cr[n} - 1\text{]}, & \Im\{c_{n-1}\} & \leftrightarrow \text{ ci[n} - 1\text{]} \end{array}$$

Similarly, for the complex data $d_j (j = 0, \cdots, n-1)$.

(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 $c_k (k = 0, \cdots, n-1)$ be represented by $\hat{c}_k (k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{c}_k = nc_k \quad (k = 0, \cdots, n-1)$$

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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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 complex Fourier forward and backward transform using the following sequence of numbers as input data.

cr[0] = 3.000	ci[0] = 0.000
cr[1] = 2.786	ci[1] = 0.725
cr[2] = 2.300	ci[2] = 1.173
cr[3] = 1.792	ci[3] = 1.327
cr[4] = 1.381	ci[4] = 1.302
cr[5] = 1.080	ci[5] = 1.197
cr[6] = 0.865	ci[6] = 1.065
cr[7] = 0.711	ci[7] = 0.930
cr[8] = 0.600	ci[8] = 0.800
cr[9] = 0.519	ci[9] = 0.679
cr[10] = 0.459	ci[10] = 0.566
cr[11] = 0.415	ci[11] = 0.461
cr[12] = 0.383	ci[12] = 0.361
cr[13] = 0.360	ci[13] = 0.267
cr[14] = 0.345	ci[14] = 0.176
cr[15] = 0.336	ci[15] = 0.087

(b) Input data

Arrays cr and ci, n=16, ld=16, isw=1(Forward transform) and isw=−1(Backward transform).

(c) Main program

```
/*      C interface example for ASL_dfc1fb , ASL_dfc1bf */

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

int main()
{
    int ld=16;
    int n;
    double *cr; double *ci;
    int isw;
    int ifax[20];   double *trigs;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "dfc1bf.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dfc1fb , ASL_dfc1bf ***\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", &n );

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf,%lf", &cr[i], &ci[i] );
    }

    printf( "\t  Real Part                  Imaginary Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n", i, cr[i], i, ci[i] );
    }

    fclose( fp );

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


    isw = 1;
    ierr = ASL_dfc1fb(n, cr, ci, ld, isw, ifax, trigs, wk);

    for( i=0 ; i<n ; i++ )
    {
        cr[i] /= n;
        ci[i] /= 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( i=0 ; i<n ; i++ )
    {
        printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n", i, cr[i], i, ci[i] );
    }

    isw = -1;
    ierr = ASL_dfc1bf(n, cr, ci, ld, isw, ifax, trigs, wk);


    printf( "\n\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  cr[%3d] = %8.3g         ci[%3d] = %8.3g\n", i, cr[i], i, ci[i] );
    }

    free( cr );
    free( ci );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_dfc1fb , ASL_dfc1bf ***

 ** Input **

  Real Part                  Imaginary Part
  cr[  0] =         3         ci[  0] =         0
  cr[  1] =      2.79         ci[  1] =     0.725
  cr[  2] =       2.3         ci[  2] =      1.17
  cr[  3] =      1.79         ci[  3] =      1.33
  cr[  4] =      1.38         ci[  4] =       1.3
  cr[  5] =      1.08         ci[  5] =       1.2
  cr[  6] =     0.865         ci[  6] =      1.06
  cr[  7] =     0.711         ci[  7] =      0.93
  cr[  8] =       0.6         ci[  8] =       0.8
  cr[  9] =     0.519         ci[  9] =     0.679
  cr[ 10] =     0.459         ci[ 10] =     0.566
  cr[ 11] =     0.415         ci[ 11] =     0.461
  cr[ 12] =     0.383         ci[ 12] =     0.361
  cr[ 13] =      0.36         ci[ 13] =     0.267
```

```
  cr[ 14] =    0.345         ci[ 14] =    0.176
  cr[ 15] =    0.336         ci[ 15] =    0.087

 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
  cr[  0] =     1.08         ci[  0] =    0.695
  cr[  1] =    0.583         ci[  1] =   -0.461
  cr[  2] =    0.208         ci[  2] =   -0.321
  cr[  3] =    0.115         ci[  3] =   -0.197
  cr[  4] =   0.0911         ci[  4] =   -0.126
  cr[  5] =   0.0854         ci[  5] =  -0.0826
  cr[  6] =   0.0839         ci[  6] =  -0.0541
  cr[  7] =   0.0835         ci[  7] =  -0.0325
  cr[  8] =   0.0834         ci[  8] =  -0.0144
  cr[  9] =   0.0834         ci[  9] =  0.00265
  cr[ 10] =   0.0833         ci[ 10] =   0.0197
  cr[ 11] =   0.0832         ci[ 11] =   0.0383
  cr[ 12] =   0.0833         ci[ 12] =   0.0609
  cr[ 13] =   0.0833         ci[ 13] =   0.0915
  cr[ 14] =   0.0834         ci[ 14] =     0.14
  cr[ 15] =   0.0834         ci[ 15] =    0.241

< Backward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
  cr[  0] =        3         ci[  0] = 1.11e-16
  cr[  1] =     2.79         ci[  1] =    0.725
  cr[  2] =      2.3         ci[  2] =     1.17
  cr[  3] =     1.79         ci[  3] =     1.33
  cr[  4] =     1.38         ci[  4] =      1.3
  cr[  5] =     1.08         ci[  5] =      1.2
  cr[  6] =    0.865         ci[  6] =     1.06
  cr[  7] =    0.711         ci[  7] =     0.93
  cr[  8] =      0.6         ci[  8] =      0.8
  cr[  9] =    0.519         ci[  9] =    0.679
  cr[ 10] =    0.459         ci[ 10] =    0.566
  cr[ 11] =    0.415         ci[ 11] =    0.461
  cr[ 12] =    0.383         ci[ 12] =    0.361
  cr[ 13] =     0.36         ci[ 13] =    0.267
  cr[ 14] =    0.345         ci[ 14] =    0.176
  cr[ 15] =    0.336         ci[ 15] =    0.087
```

2.3 ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

2.3.1 [DEPRECATED]ASL_zfc1fb, ASL_cfc1fb One-Dimensional Complex Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

ASL_zfc1fb or ASL_cfc1fb computes the complex Fourier forward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

Backward transform

ASL_zfc1fb or ASL_cfc1fb computes the complex Fourier backward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

(2) **Usage**

Double precision:

ierr = ASL_zfc1fb (n, c, ld, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc1fb (n, c, ld, isw, ifax, trigs, wk);

(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 input data values n (See Note (a))
2	c	{Z∗ / C∗}	ld	Input	Input data c_k (See Note (b))
				Output	Output results d_j (See Notes (b) and (c))
3	ld	I	1	Input	Size of array c
4	isw	I	1	Input	Processing switch (See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
5	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
6	trigs	{D∗ / R∗}	2 × n	Output	Trigonometric function table (See Note (d))
7	wk	{Z∗ / C∗}	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0

(b) n ≤ ld

(c) isw ∈ {0, 1, −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.	

(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 complex data $c_k(k = 0, \cdots, n-1)$ and elements of array c are associated as follows.

$$\begin{array}{lcl} c_0 & \leftrightarrow & \mathrm{c}[0] \\ c_1 & \leftrightarrow & \mathrm{c}[1] \\ \cdots & \cdots & \cdots \\ c_{n-1} & \leftrightarrow & \mathrm{c}[\mathrm{n}-1] \end{array}$$

Similarly, for the complex data $d_j(j = 0, \cdots, n-1)$.

(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 $c_k(k = 0, \cdots, n-1)$ be represented by $\hat{c}_k(k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{c}_k = nc_k \quad (k = 0, \cdots, n-1)$$

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 n, you should call this function once, and then use the after-initialization transform 2.3.2 $\left\{\begin{array}{l}\mathrm{ASL\_zfc1bf}\\ \mathrm{ASL\_cfc1bf}\end{array}\right\}$, 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 2.3.2 $\left\{\begin{array}{l}\mathrm{ASL\_zfc1bf}\\ \mathrm{ASL\_cfc1bf}\end{array}\right\}$.
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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.3.2 (7).

2.3.2 [DEPRECATED]ASL_zfc1bf, ASL_cfc1bf One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_zfc1bf or ASL_cfc1bf computes the complex Fourier forward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

Backward transform

ASL_zfc1bf or ASL_cfc1bf computes the complex Fourier backward transform (arbitrary radix) for the complex data $c_k(k = 0, \cdots, n-1)$.

$$d_j = \sum_{k=0}^{n-1} c_k e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1)$$

(2) **Usage**

Double precision:

ierr = ASL_zfc1bf (n, c, ld, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc1bf (n, c, ld, isw, ifax, trigs, wk);

(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 input data values n (See Note (a))
2	c	{Z∗ / C∗}	ld	Input	Input data c_k (See Note (b))
				Output	Output results d_j (See Notes (b) and (c))
3	ld	I	1	Input	Size of array c
4	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
5	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
6	trigs	{D∗ / R∗}	2 × n	Input	Trigonometric function table (See Note (a))
7	wk	{Z∗ / C∗}	n	Work	Work area
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0

(b) n ≤ ld

(c) isw ∈ {1, −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.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data n after the including-initialization function 2.3.1 {ASL_zfc1fb / ASL_cfc1fb} 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 $c_k(k = 0, \cdots, n-1)$ and elements of array c are associated as follows.

$$\begin{array}{lcl} c_0 & \leftrightarrow & \mathrm{c[0]} \\ c_1 & \leftrightarrow & \mathrm{c[1]} \\ \cdots & \cdots & \cdots \\ c_{n-1} & \leftrightarrow & \mathrm{c[n-1]} \end{array}$$

Similarly, for the complex data $d_j(j = 0, \cdots, n-1)$.

(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 $c_k(k = 0, \cdots, n-1)$ be represented by $\hat{c}_k(k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{c}_k = nc_k \quad (k = 0, \cdots, n-1)$$

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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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 complex Fourier forward and backward transform using the following sequence of numbers as input data.

$\mathrm{c[\ 0]} = 3.000 + \sqrt{-1} \times 0.000$
$\mathrm{c[\ 1]} = 2.786 + \sqrt{-1} \times 0.725$
$\mathrm{c[\ 2]} = 2.300 + \sqrt{-1} \times 1.173$
$\mathrm{c[\ 3]} = 1.792 + \sqrt{-1} \times 1.327$
$\mathrm{c[\ 4]} = 1.381 + \sqrt{-1} \times 1.302$
$\mathrm{c[\ 5]} = 1.080 + \sqrt{-1} \times 1.197$
$\mathrm{c[\ 6]} = 0.865 + \sqrt{-1} \times 1.065$
$\mathrm{c[\ 7]} = 0.711 + \sqrt{-1} \times 0.930$
$\mathrm{c[\ 8]} = 0.600 + \sqrt{-1} \times 0.800$
$\mathrm{c[\ 9]} = 0.519 + \sqrt{-1} \times 0.679$

$$c[10] = 0.459 + \sqrt{-1} \times 0.566$$
$$c[11] = 0.415 + \sqrt{-1} \times 0.461$$
$$c[12] = 0.383 + \sqrt{-1} \times 0.361$$
$$c[13] = 0.360 + \sqrt{-1} \times 0.267$$
$$c[14] = 0.345 + \sqrt{-1} \times 0.176$$
$$c[15] = 0.336 + \sqrt{-1} \times 0.087$$

(b) Input data

Arrays cr and ci, n=16, ld=16, isw=1(Forward transform) and isw=−1(Backward transform).

(c) Main program

```
/*      C interface example for ASL_zfc1fb , ASL_zfc1bf */

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

int main()
{
    int ld=16;
    int n;
    double _Complex *c;
    int isw;
    int ifax[20];   double *trigs;
    double _Complex *wk;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "zfc1bf.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zfc1fb , ASL_zfc1bf ***\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", &n );

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

    printf( "\t  Real Part                   Imaginary Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  creal(c[%3d]) = %8.3g         cimag(c[%3d]) = %8.3g\n", i, creal(c[i]), i, cimag(c[i]) );
    }

    fclose( fp );

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


    isw = 1;
    ierr = ASL_zfc1fb(n, c, ld, isw, ifax, trigs, wk);
```

```
    for( i=0 ; i<n ; i++ )
    {
        c[i] /= 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( i=0 ; i<n ; i++ )
    {
        printf( "\t  creal(c[%3d]) = %8.3g         cimag(c[%3d]) = %8.3g\n", i, creal(c[i]), i, cimag(c[i]) );
    }

    isw = -1;
    ierr = ASL_zfc1bf(n, c, ld, isw, ifax, trigs, wk);


    printf( "\n\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part                  Imaginary Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  creal(c[%3d]) = %8.3g         cimag(c[%3d]) = %8.3g\n", i, creal(c[i]), i, cimag(c[i]) );
    }

    free( c );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_zfc1fb , ASL_zfc1bf ***

 ** Input **

  Real Part                  Imaginary Part
  creal(c[  0]) =        3         cimag(c[  0]) =        0
  creal(c[  1]) =     2.79         cimag(c[  1]) =    0.725
  creal(c[  2]) =      2.3         cimag(c[  2]) =     1.17
  creal(c[  3]) =     1.79         cimag(c[  3]) =     1.33
  creal(c[  4]) =     1.38         cimag(c[  4]) =      1.3
  creal(c[  5]) =     1.08         cimag(c[  5]) =      1.2
  creal(c[  6]) =    0.865         cimag(c[  6]) =     1.06
  creal(c[  7]) =    0.711         cimag(c[  7]) =     0.93
  creal(c[  8]) =      0.6         cimag(c[  8]) =      0.8
  creal(c[  9]) =    0.519         cimag(c[  9]) =    0.679
  creal(c[ 10]) =    0.459         cimag(c[ 10]) =    0.566
  creal(c[ 11]) =    0.415         cimag(c[ 11]) =    0.461
  creal(c[ 12]) =    0.383         cimag(c[ 12]) =    0.361
  creal(c[ 13]) =     0.36         cimag(c[ 13]) =    0.267
  creal(c[ 14]) =    0.345         cimag(c[ 14]) =    0.176
  creal(c[ 15]) =    0.336         cimag(c[ 15]) =    0.087

 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
  creal(c[  0]) =     1.08         cimag(c[  0]) =    0.695
  creal(c[  1]) =    0.583         cimag(c[  1]) =   -0.461
  creal(c[  2]) =    0.208         cimag(c[  2]) =   -0.321
  creal(c[  3]) =    0.115         cimag(c[  3]) =   -0.197
  creal(c[  4]) =   0.0911         cimag(c[  4]) =   -0.126
  creal(c[  5]) =   0.0854         cimag(c[  5]) =  -0.0826
  creal(c[  6]) =   0.0839         cimag(c[  6]) =  -0.0541
  creal(c[  7]) =   0.0835         cimag(c[  7]) =  -0.0325
  creal(c[  8]) =   0.0834         cimag(c[  8]) =  -0.0144
  creal(c[  9]) =   0.0834         cimag(c[  9]) =  0.00265
  creal(c[ 10]) =   0.0833         cimag(c[ 10]) =   0.0197
  creal(c[ 11]) =   0.0832         cimag(c[ 11]) =   0.0383
  creal(c[ 12]) =   0.0833         cimag(c[ 12]) =   0.0609
  creal(c[ 13]) =   0.0833         cimag(c[ 13]) =   0.0915
  creal(c[ 14]) =   0.0834         cimag(c[ 14]) =     0.14
  creal(c[ 15]) =   0.0834         cimag(c[ 15]) =    0.241

< Backward Transform >
```

```
ierr =      0

Solution

  Real Part                    Imaginary Part
  creal(c[  0]) =         3         cimag(c[  0]) = 1.11e-16
  creal(c[  1]) =      2.79         cimag(c[  1]) =    0.725
  creal(c[  2]) =       2.3         cimag(c[  2]) =     1.17
  creal(c[  3]) =      1.79         cimag(c[  3]) =     1.33
  creal(c[  4]) =      1.38         cimag(c[  4]) =      1.3
  creal(c[  5]) =      1.08         cimag(c[  5]) =      1.2
  creal(c[  6]) =     0.865         cimag(c[  6]) =     1.06
  creal(c[  7]) =     0.711         cimag(c[  7]) =     0.93
  creal(c[  8]) =       0.6         cimag(c[  8]) =      0.8
  creal(c[  9]) =     0.519         cimag(c[  9]) =    0.679
  creal(c[ 10]) =     0.459         cimag(c[ 10]) =    0.566
  creal(c[ 11]) =     0.415         cimag(c[ 11]) =    0.461
  creal(c[ 12]) =     0.383         cimag(c[ 12]) =    0.361
  creal(c[ 13]) =      0.36         cimag(c[ 13]) =    0.267
  creal(c[ 14]) =     0.345         cimag(c[ 14]) =    0.176
  creal(c[ 15]) =     0.336         cimag(c[ 15]) =    0.087
```

2.4 ONE-DIMENSIONAL REAL FOURIER TRANSFORM

2.4.1 [DEPRECATED]ASL_dfr1fb, ASL_rfr1fb One-Dimensional Real Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

ASL_dfr1fb or ASL_rfr1fb obtains a half period of the Fourier forward transform (arbitrary radix) for the real data $r_k(k = 0, \cdots, n-1)$.

$$c_j = \sum_{k=0}^{n-1} r_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j} = c_j$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_j(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$ for n complex data $c_j(j = 0, \cdots, n-1)$ satisfying $c^*_{n-j} = c_j$, ASL_dfr1fb or ASL_rfr1fb obtains the Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_k &= \sum_{j=0}^{n-1} c_j e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_0 + (-1)^k \hat{c}_{\frac{n}{2}} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{c_j e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_0 + (-1)^k \hat{c}_{\frac{n}{2}} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[\Re\{c_j\} \cos(2\pi \frac{jk}{n}) - \Im\{c_j\} \sin(2\pi \frac{jk}{n}) \right] \\ & \quad (k = 0, \cdots, n-1) \end{aligned}$$

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, $\hat{c}_{\frac{n}{2}} = 0$, and when n is even, $\hat{c}_{\frac{n}{2}} = c_{\frac{n}{2}}$.

(2) **Usage**

Double precision:

ierr = ASL_dfr1fb (n, r, ld, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfr1fb (n, r, ld, isw, ifax, trigs, wk);

(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 data values n (See Note (a))
2	r	{D∗ / R∗}	ld	Input	Input data r_k (Forward transform) or c_j (Backward transform) (See Note (b))
				Output	Output results c_j (Forward transform), or r_k (Backward transform) (See Notes (b) and (c)).
3	ld	I	1	Input	Size of array r
4	isw	I	1	Input	Processing switch(See Note (d)) isw= 0:Initialization only isw= 1:Forward transform isw=−1:Backward transform
5	ifax	I*	20	Output	Factorization results and number of factors (See Note (d))
6	trigs	{D∗ / R∗}	n	Output	Trigonometric function table (See Note (d))
7	wk	{D∗ / R∗}	See Contents	Work	Work area **Size:** n+1, where n is an odd, or n+2, where n is an even.
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

(b) $n + 1 \leq ld$, where n is an odd, or
$n + 2 \leq ld$, where n is an even.

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

(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.	

(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 $r_k (k = 0, \cdots, n-1)$ and elements of array r are associated as follows.

$$\begin{array}{lcl} r_0 & \leftrightarrow & \mathrm{r[0]} \\ r_1 & \leftrightarrow & \mathrm{r[1]} \\ \cdots & \cdots & \cdots \\ r_{n-1} & \leftrightarrow & \mathrm{r[n-1]} \end{array}$$

When computing the backward transform, if n(=n) is odd, then r[n] = 0, and when n is even, then r[n] = r[n+1] = 0. Also, when entering the real data $r_k (k = 0, \cdots, n-1)$ into array r, the corresponding zeros need not be specifically stored in elements r[n] and following.
If we let the real and imaginary parts of the complex data c_j $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$ be $\Re\{c_j\}$ and $\Im\{c_j\}$, respectively, the c_j and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{array}{lcl} \Re\{c_0\} & \leftrightarrow & \mathrm{r[0]} \\ \Im\{c_0\} & \leftrightarrow & \mathrm{r[1]} \\ \Re\{c_1\} & \leftrightarrow & \mathrm{r[2]} \\ \Im\{c_1\} & \leftrightarrow & \mathrm{r[3]} \\ \cdots & \cdots & \cdots \\ \Re\{c_{\lfloor \frac{n}{2} \rfloor}\} & \leftrightarrow & \mathrm{r[m-2]} \\ \Im\{c_{\lfloor \frac{n}{2} \rfloor}\} & \leftrightarrow & \mathrm{r[m-1]} \quad (\mathrm{m = n+1[n:Odd] \text{ or } n+2[n:Even]}) \end{array}$$

However, when n is odd, m=n+1 is assumed, and when n is even, m=n+2 is assumed. From the properties of a real Fourier transform, when n is odd, $\Im\{c_0\} = 0$, and when n is even, $\Im\{c_0\} = \Im\{c_{\frac{n}{2}}\} = 0$. 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 $(j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1)$ 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.

$$c_{n-j} = c_j^*$$

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 data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_k(k = 0, \cdots, n-1)$ be represented by $\hat{r}_k(k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{r}_k = nr_k \quad (k = 0, \cdots, n-1)$$

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 n, you should call this function once, and then use the after-initialization transform 2.4.2 $\left\{\begin{matrix}\text{ASL\_dfr1bf}\\ \text{ASL\_rfr1bf}\end{matrix}\right\}$, 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 2.4.2 $\left\{\begin{matrix}\text{ASL\_dfr1bf}\\ \text{ASL\_rfr1bf}\end{matrix}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.4.2 (7).

2.4.2 [DEPRECATED]ASL_dfr1bf, ASL_rfr1bf One-Dimensional Real Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_dfr1bf or ASL_rfr1bf obtains a half period of the Fourier forward transform (arbitrary radix) for the real data $r_k (k = 0, \cdots, n-1)$.

$$c_j = \sum_{k=0}^{n-1} r_k e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationship.

$$c^*_{n-j} = c_j$$

Here, z^* represents the conjugate complex number of the complex number z.

Backward transform

Given the half period $c_j (j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$ for n complex data $c_j (j = 0, \cdots, n-1)$ satisfying $c^*_{n-j} = c_j$, ASL_dfr1bf or ASL_rfr1bf obtains the Fourier backward transform (arbitrary radix) defined as follows.

$$\begin{aligned} r_k &= \sum_{j=0}^{n-1} c_j e^{2\pi\sqrt{-1}\frac{jk}{n}} \\ &= c_0 + (-1)^k \hat{c}_{\frac{n}{2}} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \Re\{c_j e^{2\pi\sqrt{-1}\frac{jk}{n}}\} \\ &= c_0 + (-1)^k \hat{c}_{\frac{n}{2}} + 2 \sum_{j=1}^{\lceil \frac{n}{2} \rceil - 1} \left[\Re\{c_j\} \cos(2\pi \frac{jk}{n}) - \Im\{c_j\} \sin(2\pi \frac{jk}{n}) \right] \\ &\quad (k = 0, \cdots, n-1) \end{aligned}$$

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, $\hat{c}_{\frac{n}{2}} = 0$, and when n is even, $\hat{c}_{\frac{n}{2}} = c_{\frac{n}{2}}$.

(2) **Usage**

Double precision:

```
ierr = ASL_dfr1bf  (n, r, ld, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfr1bf  (n, r, ld, isw, ifax, trigs, wk);
```

(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 data values n (See Note (a))
2	r	{D∗ / R∗}	ld	Input	Input data r_k (Forward transform) or c_j (Backward transform) (See Note (b)).
				Output	Output results c_j (Forward transform), or r_k (Backward transform) (See Notes (b) and (c)).
3	ld	I	1	Input	Size of array r
4	isw	I	1	Input	Processing switch isw= 1:Forward transform isw=−1:Backward transform
5	ifax	I*	20	Input	Factorization results and number of factors (See Note (a))
6	trigs	{D∗ / R∗}	n	Input	Trigonometric function table (See Note (a))
7	wk	{D∗ / R∗}	See Contents	Work	Work area **Size:** n + 1, where n is an odd, or n + 2, where n is an even.
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $n > 0$

(b) $n + 1 \leq ld$, where n is an odd, or
$n + 2 \leq ld$, where n is an even.

(c) $isw \in \{1, -1\}$

(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.	

(6) **Notes**

(a) This function can be used to repeatedly compute the transform for the same number of data n after the including-initialization function 2.4.1 $\left\{\begin{matrix}\text{ASL\_dfr1fb}\\ \text{ASL\_rfr1fb}\end{matrix}\right\}$ 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 $r_k(k = 0, \cdots, n-1)$ and elements of array r are associated as follows.

$$\begin{array}{lcl} r_0 & \leftrightarrow & \text{r[0]} \\ r_1 & \leftrightarrow & \text{r[1]} \\ \cdots & \cdots & \cdots \\ r_{n-1} & \leftrightarrow & \text{r[n}-1\text{]} \end{array}$$

When computing the backward transform, if n(=n) is odd, then r[n] = 0, and when n is even, then r[n] = r[n+1] = 0. Also, when entering the real data $r_k(k = 0, \cdots, n-1)$ into array r, the corresponding zeros need not be specifically stored in elements r[n] and following.
If we let the real and imaginary parts of the complex data c_j $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$ be $\Re\{c_j\}$ and $\Im\{c_j\}$, respectively, the c_j and elements of array r are associated as follows. Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

$$\begin{array}{lcl} \Re\{c_0\} & \leftrightarrow & \text{r[0]} \\ \Im\{c_0\} & \leftrightarrow & \text{r[1]} \\ \Re\{c_1\} & \leftrightarrow & \text{r[2]} \\ \Im\{c_1\} & \leftrightarrow & \text{r[3]} \\ \cdots & \cdots & \cdots \\ \Re\{c_{\lfloor \frac{n}{2} \rfloor}\} & \leftrightarrow & \text{r[m}-2\text{]} \\ \Im\{c_{\lfloor \frac{n}{2} \rfloor}\} & \leftrightarrow & \text{r[m}-1\text{]} \quad (\text{m = n+1[n:Odd] or n+2[n:Even]}) \end{array}$$

However, when n is odd, m=n+1 is assumed, and when n is even, m=n+2 is assumed. From the properties of a real Fourier transform, when n is odd, $\Im\{c_0\} = 0$, and when n is even, $\Im\{c_0\} = \Im\{c_{\frac{n}{2}}\} = 0$. 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 $c_j \quad (j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1)$ 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.

$$c_{n-j} = c_j^*$$

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 data. For example, if we let the data obtained by computing the backward transform immediately following the forward transform for the real data $r_k(k = 0, \cdots, n-1)$ be represented by $\hat{r}_k(k = 0, \cdots, n-1)$, then the following relationship holds.

$$\hat{r}_k = nr_k \quad (k = 0, \cdots, n-1)$$

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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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 one-dimensional real Fourier forward and backward transforms using the following sequence of numbers as input data.
r[0] = 2.000
r[1] = 1.503
r[2] = 1.000
r[3] = 0.665
r[4] = 0.500
r[5] = 0.452
r[6] = 0.478
r[7] = 0.553
r[8] = 0.667
r[9] = 0.815
r[10] = 1.000
r[11] = 1.227
r[12] = 1.500
r[13] = 1.808
r[14] = 2.094
r[15] = 2.214

(b) Input data
Array r, n=16, ld=18, isw=1(Forward transform) and isw=−1 (Backward transform).

(c) Main program

```
/*      C interface example for ASL_dfr1fb , ASL_dfr1bf */

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

int main()
{
    int ld=18;
    int n;
    double *r;
    int ifax[20];
    double *trigs;
    double *wk;
    int isw;
    int ierr;
    int i;
    FILE *fp;

    fp = fopen( "dfr1bf.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dfr1fb , ASL_dfr1bf ***\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", &n );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &r[i] );
    }

    printf( "\t  Real Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  r[%3d] = %8.3g\n", i, r[i] );
    }

    fclose( fp );

    printf( "\n     ** Output **\n" );
    isw =  1;
    ierr = ASL_dfr1fb(n, r, ld, isw, ifax, trigs, wk);
    for( i=0 ; i<n+2 ; i++ )
    {
        r[i] /= n;
    }

    printf( "\n\t< Forward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n\tSolution\n" );
    printf( "\t  Real Part                     Imaginary Part\n" );
    for( i=0 ; i<n+2 ; i = i+2 )
    {
        printf( "\t  r[%3d] = %8.3g\t\t  r[%3d] = %8.3g\n", i, r[i], i+1, r[i+1] );
    }

    isw = -1;
    ierr = ASL_dfr1bf(n, r, ld, isw, ifax, trigs, wk);

    printf( "\n\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

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

```
    printf( "\t  Real Part\n" );
    for( i=0 ; i<n+2 ; i++ )
    {
        printf( "\t  r[%3d] = %8.3g\n", i, r[i] );
    }

    free( r );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) Output results

```
*** ASL_dfr1fb , ASL_dfr1bf ***

 ** Input **

  Real Part
  r[  0] =         2
  r[  1] =       1.5
  r[  2] =         1
  r[  3] =     0.665
  r[  4] =       0.5
  r[  5] =     0.452
  r[  6] =     0.478
  r[  7] =     0.553
  r[  8] =     0.667
  r[  9] =     0.815
  r[ 10] =         1
  r[ 11] =      1.23
  r[ 12] =       1.5
  r[ 13] =      1.81
  r[ 14] =      2.09
  r[ 15] =      2.21

 ** Output **

< Forward Transform >
ierr =      0

Solution
  Real Part                        Imaginary Part
  r[  0] =      1.15       r[  1] =         0
  r[  2] =     0.309       r[  3] =     0.268
  r[  4] =    0.0829       r[  5] =    0.0719
  r[  6] =    0.0222       r[  7] =    0.0192
  r[  8] =   0.00594       r[  9] =   0.00506
  r[ 10] =   0.00156       r[ 11] =   0.00139
  r[ 12] = 0.000454       r[ 13] = 0.000357
  r[ 14] = 0.000103       r[ 15] = 0.000104
  r[ 16] = 0.000125       r[ 17] =         0

< Backward Transform >
ierr =      0

Solution
  Real Part
  r[  0] =         2
  r[  1] =       1.5
  r[  2] =         1
  r[  3] =     0.665
  r[  4] =       0.5
  r[  5] =     0.452
  r[  6] =     0.478
  r[  7] =     0.553
  r[  8] =     0.667
  r[  9] =     0.815
  r[ 10] =         1
  r[ 11] =      1.23
  r[ 12] =       1.5
  r[ 13] =      1.81
  r[ 14] =      2.09
  r[ 15] =      2.21
  r[ 16] =         0
  r[ 17] =         0
```

2.5 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

2.5.1 [DEPRECATED]ASL_dfcmfb, ASL_rfcmfb Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

ASL_dfcmfb or ASL_rfcmfb 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_dfcmfb or ASL_rfcmfb 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_dfcmfb (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfcmfb (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0
m > 0

(b) incn > 0
incm > 0

(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 ∈ {0, 1, −1}

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 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.	

(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.

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{cr[incn} * \mathrm{k} + \mathrm{incm} * (\mathrm{l}-1)]$$
$$\Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{ci[incn} * \mathrm{k} + \mathrm{incm} * (\mathrm{l}-1)]$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{cr[k} + \mathrm{n} * (\mathrm{l}-1)], \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{ci[k} + \mathrm{n} * (\mathrm{l}-1)]$$

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:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{cr[(l-1)} + \mathrm{m} * \mathrm{k}], \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \mathrm{ci[(l-1)} + \mathrm{m} * \mathrm{k}]$$

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 2.5.2 $\begin{Bmatrix} \text{ASL\_dfcmbf} \\ \text{ASL\_rfcmbf} \end{Bmatrix}$, 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 2.5.2 $\begin{Bmatrix} \text{ASL\_dfcmbf} \\ \text{ASL\_rfcmbf} \end{Bmatrix}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.5.2 (7).

2.5.2 [DEPRECATED]ASL_dfcmbf, ASL_rfcmbf Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_dfcmbf or ASL_rfcmbf 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_dfcmbf or ASL_rfcmbf 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_dfcmbf (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfcmbf (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n $>$ 0
m $>$ 0

(b) incn $>$ 0
incm $>$ 0

(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 $\in \{1, -1\}$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 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.	

(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 2.5.1 $\left\{\begin{matrix}\text{ASL\_dfcmfb}\\ \text{ASL\_rfcmfb}\end{matrix}\right\}$ 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) 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.

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \text{cr[incn} * \text{k} + \text{incm} * (\text{l} - 1)]$$
$$\Im\{c_{k,l}\} \quad \leftrightarrow \quad \text{ci[incn} * \text{k} + \text{incm} * (\text{l} - 1)]$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \text{cr[k} + \text{n} * (\text{l} - 1)], \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \text{ci[k} + \text{n} * (\text{l} - 1)]$$

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:

$$\Re\{c_{k,l}\} \quad \leftrightarrow \quad \text{cr[(l} - 1) + \text{m} * \text{k}], \quad \Im\{c_{k,l}\} \quad \leftrightarrow \quad \text{ci[(l} - 1) + \text{m} * \text{k]}$$

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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (backward transform).

(c) Main program

```
/*      C interface example for ASL_dfcmfb , ASL_dfcmbf */

#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 ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dfcmfb , ASL_dfcmbf ***\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", &n, &m, &incn, &incm );

    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\n", incm );

    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_dfcmfb(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

    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_dfcmbf(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk);

    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_dfcmfb , ASL_dfcmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 9

  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
```

2.6 MULTIPLE ONE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

2.6.1 [DEPRECATED]ASL_zfcmfb, ASL_cfcmfb Multiple One-Dimensional Complex Fourier Transforms (Include Initialization)

(1) **Function**

Forward transform

ASL_zfcmfb or ASL_cfcmfb 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_zfcmfb or ASL_cfcmfb 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_zfcmfb (n, m, c, incn, incm, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfcmfb (n, m, c, incn, incm, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0
m > 0

(b) incn > 0
incm > 0

(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 ∈ {0, 1, −1}

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 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.	

(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.

$$c_{k,l} \leftrightarrow \mathrm{c[incn} * \mathrm{k} + \mathrm{incm} * (\mathrm{l} - 1)]$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[k} + \mathrm{n} * (\mathrm{l} - 1)]$$

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:

$$c_{k,l} \leftrightarrow \mathrm{c[(l} - 1) + \mathrm{m} * \mathrm{k}]$$

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 2.6.2 $\left\{\begin{matrix}\text{ASL\_zfcmbf}\\ \text{ASL\_cfcmbf}\end{matrix}\right\}$, 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 2.6.2 $\left\{\begin{matrix}\text{ASL\_zfcmbf}\\ \text{ASL\_cfcmbf}\end{matrix}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.6.2 (7).

2.6.2 [DEPRECATED]ASL_zfcmbf, ASL_cfcmbf Multiple One-Dimensional Complex Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_zfcmbf or ASL_cfcmbf 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_zfcmbf or ASL_cfcmbf 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_zfcmbf (n, m, c, incn, incm, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfcmbf (n, m, c, incn, incm, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0
m > 0

(b) incn > 0
incm > 0

(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}

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	n was equal to 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.	

(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 2.6.1 $\left\{\begin{matrix}\text{ASL\_zfcmfb}\\\text{ASL\_cfcmfb}\end{matrix}\right\}$ 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 $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array c are associated as follows.

$$c_{k,l} \leftrightarrow \mathrm{c[incn} * \mathrm{k} + \mathrm{incm} * (\mathrm{l} - 1)]$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$c_{k,l} \leftrightarrow \mathrm{c[k} + \mathrm{n} * (\mathrm{l} - 1)]$$

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:

$$c_{k,l} \leftrightarrow \mathrm{c[(l} - 1) + \mathrm{m} * \mathrm{k}]$$

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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (backward transform).

(c) Main program

```
/*      C interface example for ASL_zfcmfb , ASL_zfcmbf */
```

```
#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 ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_zfcmfb , ASL_zfcmbf ***\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", &n, &m, &incn, &incm );

    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\n", incm );

    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_zfcmfb(n, m, c, incn, incm, isw, ifax, trigs, wk);

    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_zfcmbf(n, m, c, incn, incm, isw, ifax, trigs, wk);

    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_zfcmfb , ASL_zfcmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 9

  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
```

2.7 MULTIPLE ONE-DIMENSIONAL REAL FOURIER TRANSFORM

2.7.1 [DEPRECATED]ASL_dfrmfb, ASL_rfrmfb Multiple One-Dimensional Real Fourier Transforms (Including Initialization)

(1) **Function**

Forward transform

ASL_dfrmfb or ASL_rfrmfb 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_dfrmfb or ASL_rfrmfb 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_dfrmfb  (n, m, r, incn, incm, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfrmfb  (n, m, r, incn, incm, isw, ifax, trigs, wk);
```

(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 (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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0
m > 0

(b) incn > 0
incm > 0

(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 ∈ {0, 1, −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.	

(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.

$$r_{k,l} \leftrightarrow \mathrm{r[incn} * \mathrm{k} + \mathrm{incm} * (\mathrm{l} - 1)]$$

For example, if we let incn=1 and incm=n, then the associations are as follows:

$$r_{k,l} \leftrightarrow \mathrm{r[k} + \mathrm{n} * (\mathrm{l} - 1)]$$

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:

$$r_{k,l} \leftrightarrow \mathrm{r[(l} - 1) + \mathrm{m} * \mathrm{k}]$$

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.

$$\begin{aligned} \Re\{c_{j,l}\} &\leftrightarrow \text{r[incn} * (2\text{j}) + \text{incm} * (\text{l} - 1)] \\ \Im\{c_{j,l}\} &\leftrightarrow \text{r[incn} * (2\text{j} + 1) + \text{incm} * (\text{l} - 1)] \end{aligned}$$

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) 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 2.7.2 $\left\{\begin{matrix}\text{ASL\_dfrmbf} \\ \text{ASL\_rfrmbf}\end{matrix}\right\}$, 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 2.7.2 $\left\{\begin{matrix}\text{ASL\_dfrmbf} \\ \text{ASL\_rfrmbf}\end{matrix}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.7.2 (7).

2.7.2 [DEPRECATED]ASL_dfrmbf, ASL_rfrmbf Multiple One-Dimensional Real Fourier Transforms (After Initialization)

(1) **Function**

Forward transform

ASL_dfrmbf or ASL_rfrmbf 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_dfrmbf or ASL_rfrmbf 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_dfrmbf  (n, m, r, incn, incm, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfrmbf  (n, m, r, incn, incm, isw, ifax, trigs, wk);
```

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0
m > 0

(b) incn > 0
incm > 0

(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 ∈ {1, −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.	

(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 2.7.1 $\left\{\begin{matrix}\text{ASL\_dfrmfb}\\ \text{ASL\_rfrmfb}\end{matrix}\right\}$ 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 $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ and elements of array r are associated as follows.

$r_{k,l} \leftrightarrow$ r[incn ∗ k + incm ∗ (l − 1)]

For example, if we let incn=1 and incm=n, then the associations are as follows:

$r_{k,l} \leftrightarrow$ 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:

$r_{k,l} \leftrightarrow$ 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.

$$\begin{aligned} \Re\{c_{j,l}\} &\leftrightarrow \text{r[incn} * (2\text{j}) + \text{incm} * (\text{l} - 1)] \\ \Im\{c_{j,l}\} &\leftrightarrow \text{r[incn} * (2\text{j} + 1) + \text{incm} * (\text{l} - 1)] \end{aligned}$$

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 $c_{j,l}$ $(j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (Backward transform).

(c) Main program

```
/*      C interface example for ASL_dfrmfb , ASL_dfrmbf */

#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 ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_dfrmfb , ASL_dfrmbf ***\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", &n, &m, &incn, &incm );

    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\n", incm );

    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_dfrmfb(n, m, r, incn, incm, isw, ifax, trigs, wk);

    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_dfrmbf(n, m, r, incn, incm, isw, ifax, trigs, wk);

    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_dfrmfb , ASL_dfrmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 12

  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
```

2.8 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

2.8.1 [DEPRECATED]ASL_dfc2fb, ASL_rfc2fb Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_dfc2fb or ASL_rfc2fb 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_dfc2fb or ASL_rfc2fb 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_dfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1

(b) nx ≤ lx
ny ≤ ly

(c) isw ∈ {0, 1, −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.	

(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.

$$\Re\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{cr[k_x + lx * k_y]}$$
$$\Im\{c_{k_x,k_y}\} \quad \leftrightarrow \quad \mathrm{ci[k_x + lx * k_y]}$$

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) 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 2.8.2 $\left\{\begin{matrix}\text{ASL\_dfc2bf}\\ \text{ASL\_rfc2bf}\end{matrix}\right\}$, 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 2.8.2 $\left\{\begin{matrix}\text{ASL\_dfc2bf}\\ \text{ASL\_rfc2bf}\end{matrix}\right\}$.
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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.8.2 (7).

2.8.2 [DEPRECATED]ASL_dfc2bf, ASL_rfc2bf Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_dfc2bf or ASL_rfc2bf 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_dfc2bf or ASL_rfc2bf 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_dfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

(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 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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1

(b) nx $\leq$ lx
ny $\leq$ ly

(c) isw $\in \{1, -1\}$

(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.	

(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 2.8.1 $\left\{\begin{matrix}\text{ASL\_dfc2fb}\\ \text{ASL\_rfc2fb}\end{matrix}\right\}$ 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) 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.

$$\begin{aligned}\Re\{c_{k_x,k_y}\} &\leftrightarrow \text{cr}[\text{k}_\text{x} + \text{lx} * \text{k}_\text{y}]\\ \Im\{c_{k_x,k_y}\} &\leftrightarrow \text{ci}[\text{k}_\text{x} + \text{lx} * \text{k}_\text{y}]\end{aligned}$$

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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (backward transform).

(c) Main program

```
/*      C Interface example for ASL_dfc2fb , ASL_dfc2bf */

#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 ierr;
    int i,j;

    printf( "    *** ASL_dfc2bf , ASL_dfc2bf ***\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 );

    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( "\t(cr[ix][iy], ci[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_dfc2fb(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

    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( "\t(cr[ix][iy],ci[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_dfc2bf(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

    printf( "\t< Backward Transform >\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\t(cr[ix][iy],ci[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_dfc2bf , ASL_dfc2bf ***

 ** Input **
nx =      5
ny =      4
(cr[ix][iy], ci[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],ci[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],ci[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)
```

2.9 TWO-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

2.9.1 [DEPRECATED]ASL_zfc2fb, ASL_cfc2fb Two-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_zfc2fb or ASL_cfc2fb 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_zfc2fb or ASL_cfc2fb 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_zfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1

(b) nx $\leq$ lx
ny $\leq$ ly

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

(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.	

(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 2.9.2 $\left\{\begin{matrix}\text{ASL\_zfc2bf}\\ \text{ASL\_cfc2bf}\end{matrix}\right\}$, 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 2.9.2 $\left\{\begin{matrix}\text{ASL\_zfc2bf}\\ \text{ASL\_cfc2bf}\end{matrix}\right\}$.
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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.9.2 (7).

2.9.2 [DEPRECATED]ASL_zfc2bf, ASL_cfc2bf Two-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_zfc2bf or ASL_cfc2bf 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_zfc2bf or ASL_cfc2bf 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_zfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1

(b) nx ≤ lx
ny ≤ ly

(c) isw ∈ {1, −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.	

(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 2.9.1 $\left\{\begin{matrix}\text{ASL\_zfc2fb}\\ \text{ASL\_cfc2fb}\end{matrix}\right\}$ 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (Backward transform).

(c) Main program

```
/*      C Interface example for ASL_zfc2fb , ASL_zfc2bf */

#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 ierr;
    int i,j;

    printf( "    *** ASL_zfc2fb , ASL_zfc2bf ***\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+2*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 );

    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_zfc2fb(nx, ny, c, lx, ly, isw, ifax, trigs, wk);
```

```
    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_zfc2bf(nx, ny, c, lx, ly, isw, ifax, trigs, wk);

    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_zfc2fb , ASL_zfc2bf ***

 ** Input **
nx =      5
ny =      4
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)
```

2.10 TWO-DIMENSIONAL REAL FOURIER TRANSFORM

2.10.1 [DEPRECATED]ASL_dfr2fb, ASL_rfr2fb Two-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_dfr2fb or ASL_rfr2fb 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_dfr2fb or ASL_rfr2fb 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_dfr2fb  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfr2fb  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

(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 (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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx $>$ 1
ny $>$ 1

(b) nx + 1 $\leq$ lx, where nx is an odd, or
nx + 2 $\leq$ lx, where nx is an even.

(c) ny $\leq$ ly

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

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b) or (c) was not satisfied.	
3020	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 $\mathrm{r[nx + lx * k_y]} = \mathrm{r[nx + 1 + lx * k_y]} = 0$. 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.

$$\begin{aligned} \Re\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + lx * j_y]} \\ \Im\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * j_y]} \end{aligned}$$

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.

$$\begin{aligned} c^*_{n_x - j_x, n_y - j_y} \quad &= c_{j_x,j_y} \\ c^*_{n_x - j_x, j_y} \quad &= c_{j_x,n_y - j_y} \end{aligned}$$

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 2.10.2 $\left\{\begin{matrix}\text{ASL\_dfr2bf}\\ \text{ASL\_rfr2bf}\end{matrix}\right\}$, 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 2.10.2 $\left\{\begin{matrix}\text{ASL\_dfr2bf}\\ \text{ASL\_rfr2bf}\end{matrix}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.10.2 (7).

2.10.2 [DEPRECATED]ASL_dfr2bf, ASL_rfr2bf Two-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_dfr2bf or ASL_rfr2bf 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_dfr2bf or ASL_rfr2bf 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_dfr2bf  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfr2bf  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1

(b) nx + 1 ≤ lx, where nx is an odd, or
nx + 2 ≤ lx, where nx is an even.

(c) ny ≤ ly

(d) isw ∈ {1, −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) or (c) was not satisfied.	
3020	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 2.10.1 $\left\{\begin{matrix}\text{ASL\_dfr2fb}\\ \text{ASL\_rfr2fb}\end{matrix}\right\}$ 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.

$$\begin{aligned}\Re\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + lx * j_y]}\\ \Im\{c_{j_x,j_y}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * j_y]}\end{aligned}$$

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.

$$\begin{aligned}c^*_{n_x-j_x,n_y-j_y} &= c_{j_x,j_y}\\ c^*_{n_x-j_x,j_y} &= c_{j_x,n_y-j_y}\end{aligned}$$

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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (Backward transform).

(c) Main program

```
/*      C Interface example for ASL_dfr2fb , ASL_dfr2bf */

#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 ierr;
    int i,j;

    printf( "    *** ASL_dfr2fb , ASL_dfr2bf ***\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 );

    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_dfr2fb(nx, ny, r, lx, ly, isw, ifax, trigs, wk);


    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_dfr2bf(nx, ny, r, lx, ly, isw, ifax, trigs, wk);

    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_dfr2fb , ASL_dfr2bf ***

 ** Input **
nx =      6
ny =      4
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
```

2.11 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (REAL ARGUMENT TYPE)

2.11.1 [DEPRECATED]ASL_dfc3fb, ASL_rfc3fb Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_dfc3fb or ASL_rfc3fb 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_dfc3fb or ASL_rfc3fb 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_dfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

(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 (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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1
nz > 1

(b) nx $\leq$ lx
ny $\leq$ ly
nz $\leq$ lz

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

(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.	

(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.

$$\Re\{c_{k_x,k_y,k_z}\} \quad \leftrightarrow \quad \mathrm{cr[k_x + lx * (k_y + ly * k_z)]}$$
$$\Im\{c_{k_x,k_y,k_z}\} \quad \leftrightarrow \quad \mathrm{ci[k_x + lx * (k_y + ly * k_z)]}$$

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 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 2.11.2 $\begin{Bmatrix}\text{ASL\_dfc3bf}\\ \text{ASL\_rfc3bf}\end{Bmatrix}$, 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 2.11.2 $\begin{Bmatrix}\text{ASL\_dfc3bf}\\ \text{ASL\_rfc3bf}\end{Bmatrix}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.11.2 (7).

2.11.2 [DEPRECATED]ASL_dfc3bf, ASL_rfc3bf Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_dfc3bf or ASL_rfc3bf 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_dfc3bf or ASL_rfc3bf 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_dfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_rfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1
nz > 1

(b) nx $\leq$ lx
ny $\leq$ ly
nz $\leq$ lz

(c) isw $\in \{1, -1\}$

(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.	

(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 2.11.1 $\left\{\begin{matrix}\text{ASL\_dfc3fb}\\ \text{ASL\_rfc3fb}\end{matrix}\right\}$ 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) 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (Backward transform).

(c) Main program

```
/*      C Interface example for ASL_dfc3fb , ASL_dfc3bf */

#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 ierr;
    int i,j,k;

    printf( "    *** ASL_dfc3fb , ASL_dfc3bf ***\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 );

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_dfc3fb(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

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_dfc3bf(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

    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_dfc3fb , ASL_dfc3bf ***

 ** Input **
nx =      5
ny =      4
nz =      3
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)
```

2.12 THREE-DIMENSIONAL COMPLEX FOURIER TRANSFORM (COMPLEX ARGUMENT TYPE)

2.12.1 [DEPRECATED]ASL_zfc3fb, ASL_cfc3fb Three-Dimensional Complex Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_zfc3fb or ASL_cfc3fb 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_zfc3fb or ASL_cfc3fb 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_zfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx $>$ 1
ny $>$ 1
nz $>$ 1

(b) nx $\leq$ lx
ny $\leq$ ly
nz $\leq$ lz

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

(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.	

(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**, 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 2.12.2 $\begin{Bmatrix}\text{ASL\_zfc3bf}\\ \text{ASL\_cfc3bf}\end{Bmatrix}$, 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 2.12.2 $\begin{Bmatrix}\text{ASL\_zfc3bf}\\ \text{ASL\_cfc3bf}\end{Bmatrix}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.12.2 (7).

2.12.2 [DEPRECATED]ASL_zfc3bf, ASL_cfc3bf Three-Dimensional Complex Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_zfc3bf or ASL_cfc3bf 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_zfc3bf or ASL_cfc3bf 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_zfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

Single precision:

ierr = ASL_cfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1
nz > 1

(b) nx ≤ lx
ny ≤ ly
nz ≤ lz

(c) isw ∈ {1, −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.	

(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 2.12.1 $\left\{\begin{matrix}\text{ASL\_zfc3fb}\\ \text{ASL\_cfc3fb}\end{matrix}\right\}$ 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.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{c[k_x + lx * (k_y + ly * k_z)]}$$

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**, 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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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) and isw=−1 (Backward transform).

(c) Main program

```
/*      C Interface example for ASL_zfc3fb , ASL_zfc3bf */

#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 ierr;
    int i,j,k;

    printf( "    *** ASL_zfc3fb , ASL_zfc3bf ***\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 );

    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_zfc3fb(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

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_zfc3bf(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

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_zfc3fb , ASL_zfc3bf ***

 ** Input **
nx =      5
ny =      4
nz =      3
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)
```

2.13 THREE-DIMENSIONAL REAL FOURIER TRANSFORM

2.13.1 [DEPRECATED]ASL_dfr3fb, ASL_rfr3fb Three-Dimensional Real Fourier Transform (Including Initialization)

(1) **Function**

Forward transform

ASL_dfr3fb or ASL_rfr3fb obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$.

$$\begin{aligned} c_{j_x,j_y,j_z} &= \sum_{k_x=0}^{n_x-1}\sum_{k_y=0}^{n_y-1}\sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1) \end{aligned}$$

Here, $\lfloor x\rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

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 $n_xn_yn_z$ 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_dfr3fb or ASL_rfr3fb 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_dfr3fb  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfr3fb  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1
nz > 1

(b) nx $+ 1 \leq$ lx, where nx is an odd, or
nx $+ 2 \leq$ lx, where nx is an even.

(c) ny $\leq$ ly

(d) nz $\leq$ lz

(e) lx should be even number.

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

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b), (c) or (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	
3030	Restriction (f) 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.

$$\begin{aligned} \Re\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + lx * (j_y + ly * j_z)]} \\ \Im\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * (j_y + ly * j_z)]} \end{aligned}$$

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.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

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.

$$\begin{aligned} &\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1) \end{aligned}$$

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 2.13.2 $\left\{\begin{matrix}\text{ASL\_dfr3bf}\\ \text{ASL\_rfr3bf}\end{matrix}\right\}$, 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 2.13.2 $\left\{\begin{matrix}\text{ASL\_dfr3bf}\\ \text{ASL\_rfr3bf}\end{matrix}\right\}$.
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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(g) **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 2.13.2 (7).

2.13.2 [DEPRECATED]ASL_dfr3bf, ASL_rfr3bf Three-Dimensional Real Fourier Transform (After Initialization)

(1) **Function**

Forward transform

ASL_dfr3bf or ASL_rfr3bf obtains a half period of the three-dimensional Fourier forward transform (arbitrary radix) for the three-dimensional real data $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x-1;\ k_y = 0, \cdots, n_y-1;\ k_z = 0, \cdots, n_z-1)$.

$$\begin{aligned} c_{j_x,j_y,j_z} &= \sum_{k_x=0}^{n_x-1} \sum_{k_y=0}^{n_y-1} \sum_{k_z=0}^{n_z-1} r_{k_x,k_y,k_z} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y-1;\ j_z = 0, \cdots, n_z-1) \end{aligned}$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. The remaining half period is obtained from the following relationships.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

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_dfr3bf or ASL_rfr3bf 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_dfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

Single precision:

```
ierr = ASL_rfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) nx > 1
ny > 1
nz > 1

(b) nx $+ 1 \leq$ lx, where nx is an odd, or
nx $+ 2 \leq$ lx, where nx is an even.

(c) ny $\leq$ ly

(d) nz $\leq$ lz

(e) lx should be even number.

(f) isw $\in \{1, -1\}$

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (b), (c) or (d) was not satisfied.	
3020	Restriction (e) was not satisfied.	
3030	Restriction (f) 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 2.13.1 $\left\{\begin{matrix}\text{ASL\_dfr3fb} \\ \text{ASL\_rfr3fb}\end{matrix}\right\}$ 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 r[nx + lx $*$ (k$_\mathrm{y}$ + ly $*$ k$_\mathrm{z}$)] $= 0$, and when nx is even, then r[nx + lx $*$ (k$_\mathrm{y}$ + ly $*$ k$_\mathrm{z}$)] $=$ r[nx + 1 + lx $*$ (k$_\mathrm{y}$ + ly $*$ k$_\mathrm{z}$)] $= 0$. 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.

$$\begin{aligned}\Re\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + lx * (j_y + ly * j_z)]} \\ \Im\{c_{j_x,j_y,j_z}\} \quad &\leftrightarrow \quad \mathrm{r[2 * j_x + 1 + lx * (j_y + ly * j_z)]}\end{aligned}$$

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.

$$\begin{aligned} c^*_{n_x-j_x,n_y-j_y,n_z-j_z} &= c_{j_x,j_y,j_z} \\ c^*_{n_x-j_x,j_y,j_z} &= c_{j_x,n_y-j_y,n_z-j_z} \\ c^*_{n_x-j_x,n_y-j_y,j_z} &= c_{j_x,j_y,n_z-j_z} \end{aligned}$$

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.

$$\begin{aligned} &\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1) \end{aligned}$$

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) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(f) **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

$$\begin{aligned} &r_{k_x,k_y,k_z} = \frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1) \end{aligned}$$

as input data.

(b) Input data

Array r, nx=6, ny=4, nz=3, lx=10, ly=5, lz=3, isw=1 (Forward transform) and isw=−1 (Backward transform).

(c) Main program

```
/*      C Interface example for ASL_dfr3fb , ASL_dfr3bf */

#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 ierr;
    int i,j,k;

    printf( "    *** ASL_dfr3fb , ASL_dfr3bf ***\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 );

    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_dfr3fb(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);

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_dfr3bf(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);

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_dfr3fb , ASL_dfr3bf ***

 ** Input **
nx =      6
ny =      4
nz =      3
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
```

2.14 CONVOLUTIONS

2.14.1 ASL_dfcn1d, ASL_rfcn1d One-Dimensional Convolutions

(1) **Function**

Given the two discrete functions $f(i)$ and $g(j)$ of period m satisfying:

$$f(i) = f(i + km), g(i) = g(i + km) \ (i = 0, \cdots, m - 1)$$

for an arbitrary integer k, where:

$$f(i) = 0 \ (i = n_1, \cdots, m - 1); \quad g(j) = 0 \ (j = n_2, \cdots, m - 1)$$

ASL_dfcn1d or ASL_rfcn1d calculates the discrete convolution $p(k)$ $(k = 0, \cdots, m - 1)$ defined as follows:

$$p(k) = \sum_{i=0}^{m-1} f(i)g(k - i) = \sum_{i=0}^{m-1} g(i)f(k - i) \quad (k = 0, \cdots, m - 1)$$

Here, $m = \min(n_1 + n_2 - 1, M)$ and M is an arbitrary integer satisfying $M \geq \max(n_1, n_2)$. The real Fourier transform of $p(k)$ can also be obtained.

(2) **Usage**

Double precision:

```
ierr = ASL_dfcn1d  (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);
```

Single precision:

```
ierr = ASL_rfcn1d  (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);
```

(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	n1	I	1	Input	Number of effective data n_1 for discrete function $f(i)$
2	n2	I	1	Input	Number of effective data n_2 for discrete function $g(j)$
3	r1	{D∗ / R∗}	ld1	Input	Values of discrete function $f(i)$ (See Notes (a) and (b))
				Output	When isw $\geq$ 1, result of real Fourier transform of discrete function $f(i)$ (period M)
4	ld1	I	1	Input	Size of array r1
5	r2	{D∗ / R∗}	ld2	Input	Values of discrete function $g(j)$ (See Notes (a) and (b))
				Output	Value of discrete function $p(k)$ or its real Fourier transform (See Notes (a) and (c))
6	ld2	I	1	Input	Size of array r2
7	m	I	1	Input	Parameter M corresponding to the period m of discrete functions $f(i)$, $g(j)$, and $p(k)$ (See Note (d))
8	isw	I	1	Input	Processing switch (See Notes (a) and (e)) 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. isw= 3: Calculate the convolution according to the sectioning FFT method.
9	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 20 (When isw $\geq$ 1)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
10	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	See Contents	Work	Work area **Size**: n2 (When isw= 0) $2 \times \text{m} + 1$ (When isw= 1 or 2 and m is odd) $2 \times \text{m} + 2$ (When isw= 1 or 2 and m is even) $2 \times \text{m} + \text{n1}$ (When isw= 3 and m is odd) $2 \times \text{m} + \text{n1} + 1$ (When isw= 3 and m is even)
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\text{isw} \in \{0, 1, 2, 3\}$

(b) $\text{n1} > 1$

(c) $\text{n2} > 1$

(d) $\text{m} \geq \max(\text{n1}, \text{n2})$

(e) When $\text{isw} = 0$:
$\text{ld1} \geq \text{n1}$
When $\text{isw} > 0$ and m is odd:
$\text{ld1} \geq \text{m} + 1$
When $\text{isw} > 0$ and m is even:
$\text{ld1} \geq \text{m} + 2$

(f) When $\text{isw} = 0$:
$\text{ld2} \geq \text{m}$
When $\text{isw} > 0$ and m is odd:
$\text{ld2} \geq \text{m} + 1$
When $\text{isw} > 0$ and m is even:
$\text{ld2} \geq \text{m} + 2$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	$\text{m} < \text{n1} + \text{n2} - 1$.	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.	

(6) **Notes**

(a) If the number of effective data for one of the functions for which the convolution is to be calculated is extremely large compared to the number of effective data for the other, then sectioning should be used

to divide the larger number of data into equal parts (by adding zeros at the end if necessary) and this function should be applied repeatedly to calculate the discrete convolution efficiently. In addition, the required amount of memory will be smaller.

For example, to calculate the discrete convolution of the two series $\{u_1, u_2, \cdots, u_k\}$ (number of effective data k) and $\{v_1, \cdots, v_{pq}\}$ (number of effective data pq ($pq \gg k$)), first set isw=1, n1=k, n2=q, m $\geq$ n1 + n2 − 1, r1=$\{u_1, u_2, \cdots, u_k\}$, and r2=$\{v_1, v_2, \cdots, v_q\}$ and apply this function. As a result, the first q values of the convolution to be calculated are obtained as the first q elements at the beginning of array r2.

Next, change isw and r2 to isw=3 and r2=$\{v_{q+1}, \cdots, v_{2q}\}$ and apply this function with the contents of the other arguments unchanged. As a result, the next q values of the convolution to be calculated are obtained as the first q elements at the beginning of array r2. Then, continue to perform the calculations in a similar manner while sequentially shifting the values set in r2. The convolution calculated for the last repetition, that is, the convolution calculated when r2=$\{v_{(p-1)q+1}, \cdots, v_{pq}\}$ is set, gives the last $2q - 1$ elements of the convolution to be calculated. (However, when the series $\{v_j\}$ is not a finite waveform, the last $q - 1$ elements are indeterminate).

(b) The values of the discrete functions $f(i)$ and $g(j)$ are stored in arrays r1 and r2, respectively, as follows. However, when isw = 3 is set, values are only stored in r2, and the contents of r1 are used directly (See Note (a)).

$f(0)$	$\rightarrow$	r1[0]
$f(1)$	$\rightarrow$	r1[1]
$\cdots$	$\cdots$	$\cdots$
$f(n_1 - 1)$	$\rightarrow$	r1[n1 − 1]

$g(0)$	$\rightarrow$	r2[0]
$g(1)$	$\rightarrow$	r2[1]
$\cdots$	$\cdots$	$\cdots$
$g(n_2 - 1)$	$\rightarrow$	r2[n2 − 1]

No values need be entered in elements r1[n1] and after of array r1 and in elements r2[n2] and after of array r2. Also, in particular, when isw = 3 is set, the elements in r2[n2] and after must not be changed because they are used in the calculation.

(c) The values of the discrete convolution $p(k)$ are obtained in array r2 as follows.

$p(0)$	$\rightarrow$	r2[0]
$p(1)$	$\rightarrow$	r2[1]
$\cdots$	$\cdots$	$\cdots$
$p(M - 1)$	$\rightarrow$	r2[m − 1]

When m is odd, r2[m] is 0.0, and when m is even, r2[m] and r2[m+1] are each 0.0. Also, when sectioning is performed, the first n2 data usually are meaningful as convolution data (See Note (a)).

When isw=2 is set to obtain the real Fourier transform $P(j)$ of the discrete convolution $p(k)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$P(j) = \frac{1}{M} \sum_{k=0}^{M-1} p(k) e^{-2\pi\sqrt{-1}\frac{jk}{M}} \quad (j = 0, \cdots, \lfloor \frac{M}{2} \rfloor)$$

the following associations are made:

$$\begin{array}{lcl}
\Re\{P(0)\} & \leftrightarrow & \text{r2[0]} \\
\Im\{P(0)\} & \leftrightarrow & \text{r2[1]} \\
\Re\{P(1)\} & \leftrightarrow & \text{r2[2]} \\
\Im\{P(1)\} & \leftrightarrow & \text{r2[3]} \\
\cdots & \cdots & \cdots \\
\Re\{P(\lfloor\frac{M}{2}\rfloor)\} & \leftrightarrow & \text{r2[l} - 2\text{]} \\
\Im\{P(\lfloor\frac{M}{2}\rfloor)\} & \leftrightarrow & \text{r2[l} - 1\text{]} \quad (\text{l} = \text{m}+1\text{[m: Odd] or m}+2\text{[m: Even]})
\end{array}$$

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:

$$P(M - j) = P(j)^*$$

(Here, z^* represents the conjugate complex number of the complex number z.)

(d) If m $\geq$ n1 + n2 $-$ 1 is set, the convolution can be calculated without causing an overlap with the convolution of the next period. When m $>$ n1 + n2 $-$ 1, values that match 0.0 within the error range are stored in element n1 + n2 and following. When isw=0, m = n1 + n2 $-$ 1 should be set. When isw $\geq$ 1, the calculations can be performed more efficiently by setting a value for m 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 n1=n2=145, then when isw=0, m = 289(=17^2) should be set. However, when isw $\geq$ 1, it is usually more efficient to set m = 300(=$2^2 \times 3 \times 5^2$), m = 320(=$2^6 \times 5$), m = 384(=$2^7 \times 3$) or the like.

(e) **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 m, the calculations should be performed by setting isw=0.

(f) 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 $f(i)$ and $g(j)$ are the intervals $[i_0, i_0 + n_1 - 1]$ and $[j_0, j_0 + n_2 - 1]$, respectively, let $\hat{f}(i)$ and $\hat{g}(j)$ be defined as follows:

$$\hat{f}(i) = f(i - i_0), \quad \hat{g}(j) = g(j - j_0)$$

and apply this function to $\hat{f}(i)$ and $\hat{g}(j)$. Let $\hat{p}(k)$ represent the result that was obtained, and the convolution $p(k)$ of the original functions $f(i)$ and $g(j)$ is given as follows:

$$p(k) = \hat{p}(k + (i_0 + j_0))$$

That is, the desired results are obtained if you shift $f(i)$ and $g(j)$ in the negative direction 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.

(g) The sampling interval 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 $p(n_1 + n_2 - 1) = 0$ and consider $n_1 + n_2$ data of $p(k)$ $(k = 0, 1, \cdots, n_1 + n_2 - 1)$. In this case, the coordinate 0 element usually is associated with $p(0)$. However,

When isw=0, then ld1=n1, ld2=m and nwk=n2
When isw=1 or 2, if m is odd, then ld1=ld2=m+1 and nwk = 2 × m + 1,
and if m is even, then ld1=ld2=m+2 and nwk = 2 × m + 2.
When isw=3, if m is odd, then ld1=ld2=m+1 and nwk = 2 × m + n1,
and if m is even, then ld1=ld2=m+2 and nwk = 2 × m + n1 + 1.

(h) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(7) **Example**

(a) Problem
Use the sampling interval Δx to discretize the two finite waveforms defined by the following equations and calculate the discrete convolution.

$$f(x) = \begin{cases} x & 0 \le x \le a \\ 0 & \text{Otherwise} \end{cases}$$

$$g(x) = \begin{cases} b - x & 0 \le x \le b \\ 0 & \text{Otherwise} \end{cases}$$

Remarks:
The continuous convolution $p(x) = (f \times g)(x)$ of $f(x)$ and $g(x)$ is as follows:

$$p(x) = \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi = \begin{cases} G(0, x, x) & 0 \le x \le a \\ G(0, a, x) & a \le x \le b \\ G(x - b, a, x) & b \le x \le a + b \\ 0 & \text{Otherwise} \end{cases}$$

Here, $G(\alpha, \beta, x)$ is as follows:

$$\begin{aligned} G(\alpha, \beta, x) &= \left[\frac{\xi^2}{6} (3(b - x) + 2\xi) \right]_{\alpha}^{\beta} \\ &= \frac{\xi^2}{6} (3(b - x) + 2\xi) \Big|_{\xi=\beta} - \frac{\xi^2}{6} (3(b - x) + 2\xi) \Big|_{\xi=\alpha} \end{aligned}$$

When $a = 2$ and $b = 3$ are set, the values $f(i\Delta x)$, $g(i\Delta x)$ and $p(i\Delta x)$ obtained by sampling $f(x)$, $g(x)$, and $p(x) = (f \times g)(x)$ with $\Delta x = 0.1$ are graphed as follows. The values $p(i)\Delta x$, which are the discrete convolution calculated by this function multiplied by Δx are also shown for reference. They match the continuous convolution pretty well for a small number of samples.

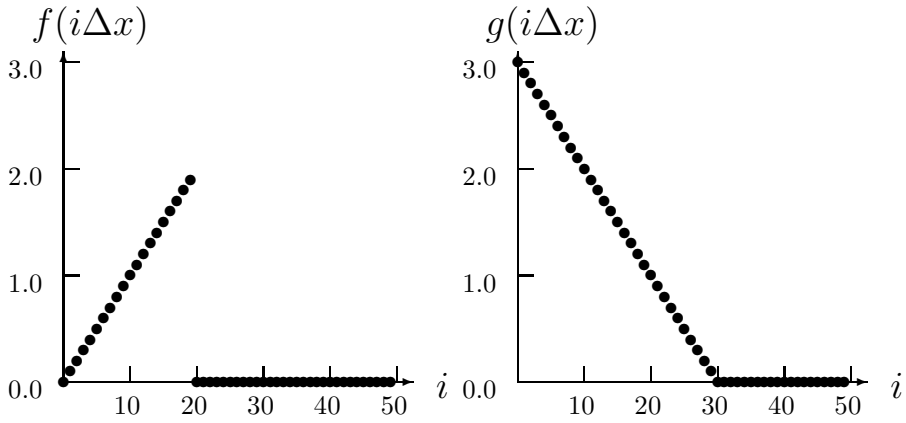


Figure 2−6

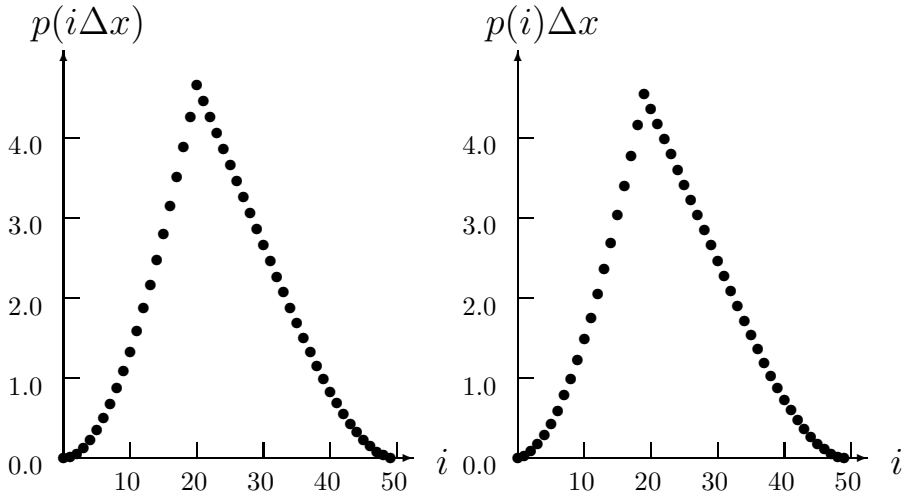


Figure 2−7

The program also calculates the continuous convolution for reference.

(b) Input data

Sampling data

r1[i − 1] = f((i − 1)Δx) (i = 1, 2, ⋯, n1) and

r2[j − 1] = g((j − 1)Δx) (j = 1, 2, ⋯, n2).

Here, $\Delta x = 0.1$.

n1 = $\frac{a}{\Delta x}$, n2 = $\frac{b}{\Delta x}$, m and isw.

(c) Main program

```
/*      C interface example for ASL_dfcn1d */

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

#ifdef __STDC__
double f(double tau, double t, double b)
#else
double f(tau, t, b)
double tau, t, b;
#endif
{
    return tau*tau*(0.5*(b-t)+tau/3.0);
}
```

```
int main()
{
    int n1;
    int n2;
    double *r1;
    int m0=100;
    int ld1=m0+2;
    double *r2;
    int ld2=m0+2;
    int niwk=20;
    int m;
    int isw;
    int *iwk;
    double *wk;
    int ierr;

    int i;
    double *cr,t,dt;
    double a,b;


    printf( "    *** ASL_dfcn1d ***\n" );
    printf( "\n     ** Input **\n\n" );

    r1 = ( double * )malloc((size_t)( sizeof(double) * ld1 ));
    if( r1 == NULL )
    {
        printf( "no enough memory for array r1\n" );
        return -1;
    }
    r2 = ( double * )malloc((size_t)( sizeof(double) * ld2 ));
    if( r2 == NULL )
    {
        printf( "no enough memory for array r2\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * (2*m0+2) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    cr = ( double * )malloc((size_t)( sizeof(double) * ld2 ));
    if( cr == NULL )
    {
        printf( "no enough memory for array cr\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
    isw=1;
    dt=0.1;
    a=2.0;
    b=3.0;
    n1=(int) ((a+0.5*dt)/dt);
    n2=(int) ((b+0.5*dt)/dt);
    m=50;

    printf( "\t isw = %6d\n\t n1 = %6d\n\t n2 = %6d\n\t m = %6d\n\n",
        isw, n1, n2, m);

    for( i=0 ; i<n1 ; i++ )
    {
        t=i*dt;
        r1[i]=t;
    }
    for( i=0 ; i<n2 ; i++ )
    {
        t=i*dt;
        r2[i]=b-t;
    }

    printf( "\tData(r1,r2)\n" );
    printf( "\t  i    r1[i]    r2[i]\n" );
/* ASSUME n2>n1 */
    for( i=0 ; i<n1 ; i++ )
    {
        printf( "\t%3d %8.3g %8.3g\n", i, r1[i], r2[i] );
    }
    for( i=n1 ; i<n2 ; i++ )
    {
        printf( "\t%3d          %8.3g\n", i, r2[i] );
    }

    ierr = ASL_dfcn1d(n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

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

```
    for( i=0 ; i<n1 ; i++ )
    {
        t=i*dt;
        cr[i]=f(t,t,b);
    }
    for( i=n1 ; i<n2 ; i++ )
    {
        t=i*dt;
        cr[i]=f(a,t,b);
    }
    for( i=n2 ; i<n1+n2 ; i++ )
    {
        t=i*dt;
        cr[i]=f(a,t,b)-f(t-b,t,b);
    }

    printf( "\tConvolution\n" );
    printf( "\t     i  r2[i]  r2[i]*dt     cr[i]\n" );
    for( i=0 ; i<n1+n2 ; i++ )
    {
        printf( "\t%3d %9.4lf %9.4lf %9.4lf\n",
             i, r2[i], r2[i]*dt, cr[i] );
    }
    free( iwk );
    free( cr );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) Output results

```
*** ASL_dfcn1d ***

 ** Input **

 isw =      1
 n1 =     20
 n2 =     30
 m =     50

Data(r1,r2)
  i    r1[i]    r2[i]
  0        0        3
  1      0.1      2.9
  2      0.2      2.8
  3      0.3      2.7
  4      0.4      2.6
  5      0.5      2.5
  6      0.6      2.4
  7      0.7      2.3
  8      0.8      2.2
  9      0.9      2.1
 10        1        2
 11      1.1      1.9
 12      1.2      1.8
 13      1.3      1.7
 14      1.4      1.6
 15      1.5      1.5
 16      1.6      1.4
 17      1.7      1.3
 18      1.8      1.2
 19      1.9      1.1
 20                 1
 21               0.9
 22               0.8
 23               0.7
 24               0.6
 25               0.5
 26               0.4
 27               0.3
 28               0.2
 29               0.1

 ** Output **

ierr =      0
Convolution
     i  r2[i]  r2[i]*dt     cr[i]
  0    0.0000    0.0000    0.0000
  1    0.3000    0.0300    0.0148
  2    0.8900    0.0890    0.0587
  3    1.7600    0.1760    0.1305
  4    2.9000    0.2900    0.2293
  5    4.3000    0.4300    0.3542
  6    5.9500    0.5950    0.5040
  7    7.8400    0.7840    0.6778
  8    9.9600    0.9960    0.8747
  9   12.3000    1.2300    1.0935
 10   14.8500    1.4850    1.3333
 11   17.6000    1.7600    1.5932
```

12	20.5400	2.0540	1.8720
13	23.6600	2.3660	2.1688
14	26.9500	2.6950	2.4827
15	30.4000	3.0400	2.8125
16	34.0000	3.4000	3.1573
17	37.7400	3.7740	3.5162
18	41.6100	4.1610	3.8880
19	45.6000	4.5600	4.2718
20	43.7000	4.3700	4.6667
21	41.8000	4.1800	4.4667
22	39.9000	3.9900	4.2667
23	38.0000	3.8000	4.0667
24	36.1000	3.6100	3.8667
25	34.2000	3.4200	3.6667
26	32.3000	3.2300	3.4667
27	30.4000	3.0400	3.2667
28	28.5000	2.8500	3.0667
29	26.6000	2.6600	2.8667
30	24.7000	2.4700	2.6667
31	22.8000	2.2800	2.4668
32	20.9100	2.0910	2.2680
33	19.0400	1.9040	2.0712
34	17.2000	1.7200	1.8773
35	15.4000	1.5400	1.6875
36	13.6500	1.3650	1.5027
37	11.9600	1.1960	1.3238
38	10.3400	1.0340	1.1520
39	8.8000	0.8800	0.9882
40	7.3500	0.7350	0.8333
41	6.0000	0.6000	0.6885
42	4.7600	0.4760	0.5547
43	3.6400	0.3640	0.4328
44	2.6500	0.2650	0.3240
45	1.8000	0.1800	0.2292
46	1.1000	0.1100	0.1493
47	0.5600	0.0560	0.0855
48	0.1900	0.0190	0.0387
49	0.0000	0.0000	0.0098

2.14.2 ASL_dfcn2d, ASL_rfcn2d 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:

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

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_dfcn2d or ASL_rfcn2d calculates the discrete convolution $p(k_x, k_y)$ defined as follows:

$$\begin{aligned} p(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x - i_x, k_y - i_y) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} g(j_x, j_y) f(k_x - j_x, k_y - j_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

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_dfcn2d  (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);
```

Single precision:

```
ierr = ASL_rfcn2d  (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);
```

(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 $g(j_x, j_y)$

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
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	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	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	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	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	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	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×my+max(lx1×ly1, lx2×ly2) (When isw ≥ 1)
16	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\text{isw} \in \{0, 1, 2\}$

(b) $\text{nx1} > 1$
$\text{ny1} > 1$

(c) $\text{nx2} > 1$
$\text{ny2} > 1$

(d) $\text{mx} \geq \max(\text{nx1}, \text{nx2})$
$\text{my} \geq \max(\text{ny1}, \text{ny2})$

(e) When $\text{isw} = 0$:
$\text{lx1} \geq \text{nx1}$
$\text{ly1} \geq \text{ny1}$
When $\text{isw} > 0$ and mx is odd:
$\text{lx1} \geq \text{mx} + 1$
$\text{ly1} \geq \text{my}$
When $\text{isw} > 0$ and mx is even:
$\text{lx1} \geq \text{mx} + 2$
$\text{ly1} \geq \text{my}$

(f) When $\text{isw} = 0$:
$\text{lx2} \geq \text{mx}$
$\text{ly2} \geq \text{my}$
When $\text{isw} > 0$ and mx is odd:
$\text{lx2} \geq \text{mx} + 1$
$\text{ly2} \geq \text{my}$
When $\text{isw} > 0$ and mx is even:
$\text{lx2} \geq \text{mx} + 2$
$\text{ly2} \geq \text{my}$

(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.	

(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.

$$f(i_x, i_y) \quad \leftrightarrow \quad \mathrm{r1[i_x + lx1 * i_y]}$$
$$g(j_x, j_y) \quad \leftrightarrow \quad \mathrm{r2[j_x + lx2 * j_y]}$$

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.

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * k_y]}$$

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):

$$P(j_x, j_y) = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} p(k_x, k_y) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y}\right)}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor)$$

the following associations are made:

$$\Re\{P(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + lx2 * j_y]}$$
$$\Im\{P(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + 1 + lx2 * j_y]}$$

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:

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * k_y]}$$

and values that match 0.0 within the error range are stored in elements corresponding to $k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1$; $k_y = 0, \cdots, \mathrm{my} - 1$ or $k_x = 0, \cdots, \mathrm{mx} - 1$; $k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1$. When isw=0, mx = nx1 + nx2 − 1 and my = ny1 + ny2 − 1 should be set. When isw ≥ 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 ≥ 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.
This procedure is available for i_y, j_y, and k_y as well.

(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 is usually associated with $p(0, 0)$, and:

- when isw=0,
  then lx1 = nx1, ly1 = ny1, lx2 = mx, ly2 = my, and
  nwk = nx2 × ny2 (when nx2 is odd) or
  nwk = (nx2 + 1) × 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 nwk = mx + (lx1 + 2) × my.

(g) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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

$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * \mathrm{i_y}] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1)$ and

$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * \mathrm{j_y}] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1)$

Here, $\Delta = 0.5$.

nx1, ny1, nx2, ny2, mx, my and isw.

(c) Main program

```
/*      C interface example for ASL_dfcn2d */

#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;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0;
    double xg=2.0,yg=2.0;


    printf( "    *** ASL_dfcn2d ***\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_dfcn2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
         r2, lx2, ly2, mx, my, isw, iwk, wk);

    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_dfcn2d ***

 ** 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
```

2.14.3 ASL_dfcn3d, ASL_rfcn3d 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:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

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_dfcn3d or ASL_rfcn3d calculates the discrete convolution $p(k_x, k_y, k_z)$ defined as follows:

$$\begin{aligned} p(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x - i_x, k_y - i_y, k_z - i_z) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} \sum_{j_z=0}^{m_z-1} g(j_x, j_y, j_z) f(k_x - j_x, k_y - j_y, k_z - j_z) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

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_dfcn3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

Single precision:

```
ierr = ASL_rfcn3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

(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	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 > 1
ny1 > 1
nz1 > 1

(c) nx2 > 1

ny2 > 1
nz2 > 1

(d) mx ≥ max(nx1, nx2)
my ≥ max(ny1, ny2)
mz ≥ max(nz1, nz2)

(e) When isw = 0 :
lx1 ≥ nx1
ly1 ≥ ny1
lz1 ≥ nz1
When isw > 0 and mx is odd:
lx1 ≥ mx + 1 and lx1 is even
ly1 ≥ my
lz1 ≥ mz
When isw > 0 and mx is even:
lx1 ≥ mx + 2 and lx1 is even
ly1 ≥ my
lz1 ≥ mz

(f) When isw = 0 :
lx2 ≥ mx
ly2 ≥ ny2
lz2 ≥ nz2
When isw > 0 and mx is odd :
lx2 ≥ mx + 1 and lx2 is even
ly2 ≥ my
lz2 ≥ mz
When isw > 0 and mx is even :
lx2 ≥ mx + 2 and lx2 is even
ly2 ≥ my
lz2 ≥ mz

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1 + nx2 − 1, 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.	

(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):

$$P(j_x, j_y, j_z) = \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} p(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)}$$
$$\left(j_x = 0, \cdots, \left\lfloor \frac{M_x}{2} \right\rfloor;\ j_y = 0, \cdots, \left\lfloor \frac{M_y}{2} \right\rfloor; j_z = 0, \cdots, \left\lfloor \frac{M_z}{2} \right\rfloor\right)$$

the following associations are made:

$$\Re\{P(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + lx2 * (j_y + ly2 * j_z)]}$$
$$\Im\{P(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + 1 + lx2 * (j_y + ly2 * j_z)]}$$

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:

$$P(M_x - j_x, M_y - j_y, M_z - j_z)^* = P(j_x, j_y, j_z)$$
$$P(M_x - j_x, j_y, j_z)^* = P(j_x, M_y - j_y, M_z - j_z)$$
$$P(M_x - j_x, M_y - j_y, j_z)^* = P(j_x, j_y, M_z - j_z)$$

(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:

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

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 $k_x = 0, \cdots, \mathrm{mx} - 1$; $k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1$; $k_z = 0, \cdots, \mathrm{mz} - 1$ 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.
This procedure is available for i_y, j_y, and k_y and i_z, j_z, and k_z as well.

(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 $p(0, 0, 0)$, and

- 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.

(g) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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

$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * (\mathrm{i_y} + \mathrm{ly1} * \mathrm{i_z})] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta, \mathrm{i_z}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1;\ \mathrm{i_z} = 0, 1, \cdots, \mathrm{nz1} - 1)$ and

$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1;\ \mathrm{j_z} = 0, 1, \cdots, \mathrm{nz2} - 1)$.

Here, $\Delta = 0.5$.

nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz and isw.

(c) Main program

```
/*      C interface example for ASL_dfcn3d */

#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;
    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_dfcn3d ***\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_dfcn3d(nx1, ny1, nz1, nx2, ny2, nz2,
    r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
    mx, my, mz, isw, iwk, wk);

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_dfcn3d ***

 ** 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
```

2.15 CORRELATIONS

2.15.1 ASL_dfcr1d, ASL_rfcr1d One-Dimensional Correlations

(1) **Function**

Given the two discrete functions $f(i)$ and $g(j)$ of period m satisfying:

$$f(i) = f(i + km), g(i) = g(i + km) \ (i = 0, \cdots, m - 1)$$

for an arbitrary integer k, where:

$$f(i) = 0 \ (i = n_1, \cdots, m - 1); \quad g(j) = 0 \ (j = n_2, \cdots, m - 1)$$

ASL_dfcr1d or ASL_rfcr1d calculates the quantity $\tilde{q}(k)$ $(k = 0, \cdots, m - 1)$ obtained by shifting the discrete correlation $q(k)$ $(k = 0, \cdots, m - 1)$, which is defined as follows:

$$q(k) = \sum_{i=0}^{m-1} f(i)g(k + i) \quad (k = 0, \cdots, m - 1)$$

by $n_1 - 1$ in the positive direction. $\tilde{q}(k)$ is defined as follows:

$$\tilde{q}(k) = q(k - (n_1 - 1)) \quad (k = 0, \cdots, m - 1)$$

Here, $m = \min(n_1 + n_2 - 1, M)$ and M is an arbitrary integer satisfying $M \geq \max(n_1, n_2)$. The real Fourier transform of the discrete correlation $q(k)$ can also be obtained.

(2) **Usage**

Double precision:

ierr = ASL_dfcr1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

Single precision:

ierr = ASL_rfcr1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

(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	n1	I	1	Input	Number of effective data n_1 for discrete function $f(i)$
2	n2	I	1	Input	Number of effective data n_2 for discrete function $g(j)$
3	r1	{D∗ / R∗}	ld1	Input	Values of discrete function $f(i)$ (See Notes (a) and (b))
				Output	When isw $\geq$ 1, result of real Fourier transform of discrete function $f(i)$ (period M)
4	ld1	I	1	Input	Size of array r1
5	r2	{D∗ / R∗}	ld2	Input	Values of discrete function $g(j)$ (See Notes (a) and (b))
				Output	Value of discrete function $\tilde{q}(k)$ or its real Fourier transform (See Notes (a) and (c))
6	ld2	I	1	Input	Size of array r2
7	m	I	1	Input	Parameter M corresponding to the period m of discrete functions $f(i)$, $g(j)$, and $\tilde{q}(k)$ (See Note (d))
8	isw	I	1	Input	Processing switch (See Notes (a) and (e)) 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. isw= 3: Calculate the correlation according to the sectioning FFT method.
9	iwk	I*	See Contents	Work	Work area **Size**: 0 (When isw= 0) 20 (When isw $\geq$ 1)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
10	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	See Contents	Work	Work area **Size**: n2 (When isw= 0) $2 \times \text{m} + 1$ (When isw= 1 or 2 and m is odd) $2 \times \text{m} + 2$ (When isw= 1 or 2 and m is even) $2 \times \text{m} + \text{n1}$ (When isw= 3 and m is odd) $2 \times \text{m} + \text{n1} + 1$ (When isw= 3 and m is even)
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\text{isw} \in \{0, 1, 2, 3\}$

(b) $\text{n1} > 1$

(c) $\text{n2} > 1$

(d) $\text{m} \geq \max(\text{n1}, \text{n2})$

(e) When $\text{isw} = 0$:
$\text{ld1} \geq \text{n1}$
When $\text{isw} > 0$ and m is odd:
$\text{ld1} \geq \text{m} + 1$
When $\text{isw} > 0$ and m is even:
$\text{ld1} \geq \text{m} + 2$

(f) When $\text{isw} = 0$:
$\text{ld2} \geq \text{m}$
When $\text{isw} > 0$ and m is odd:
$\text{ld2} \geq \text{m} + 1$
When $\text{isw} > 0$ and m is even:
$\text{ld2} \geq \text{m} + 2$

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	$\text{m} < \text{n1} + \text{n2} - 1$.	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.	

(6) **Notes**

(a) If the number of effective data for one of the functions for which the correlation is to be calculated is extremely large compared to the number of effective data for the other, then sectioning should be used

to divide the larger number of data into equal parts (by adding zeros at the end if necessary) and this function should be applied repeatedly to calculate the discrete correlation efficiently. In addition, the required amount of memory will be smaller.

For example, to calculate the discrete correlation of the two series $\{u_1, u_2, \cdots, u_k\}$ (number of effective data k) and $\{v_1, \cdots, v_{pq}\}$ (number of effective data pq ($pq \gg k$)), first set isw=1, n1=k, n2=q, m $\geq$ n1 + n2 − 1, r1=$\{u_1, u_2, \cdots, u_k\}$, and r2=$\{v_1, v_2, \cdots, v_q\}$ and apply this function. As a result, the first q values of the correlation to be calculated are obtained as the first q elements at the beginning of array r2.

Next, change isw and r2 to isw=3 and r2=$\{v_{q+1}, \cdots, v_{2q}\}$ and apply this function with the contents of the other arguments unchanged. As a result, the next q values of the correlation to be calculated are obtained as the first q elements at the beginning of array r2. Then, continue to perform the calculations in a similar manner while sequentially shifting the values set in r2. The correlation calculated for the last repetition, that is, the correlation calculated when r2=$\{v_{(p-1)q+1}, \cdots, v_{pq}\}$ is set, gives the last $2q - 1$ elements of the correlation to be calculated. (However, when the series $\{v_j\}$ is not a finite waveform, the last $q - 1$ elements are indeterminate).

(b) The values of the discrete functions $f(i)$ and $g(j)$ are stored in arrays r1 and r2, respectively, as follows. However, when isw = 3 is set, values are only stored in r2, and the contents of r1 are used directly (See Note (a)).

$f(0)$	$\rightarrow$	r1[0]
$f(1)$	$\rightarrow$	r1[1]
$\cdots$	$\cdots$	$\cdots$
$f(n_1 - 1)$	$\rightarrow$	r1[n1 − 1]

$g(0)$	$\rightarrow$	r2[0]
$g(1)$	$\rightarrow$	r2[1]
$\cdots$	$\cdots$	$\cdots$
$g(n_2 - 1)$	$\rightarrow$	r2[n2 − 1]

No values need be entered in elements r1[n1] and after of array r1 and in elements r2[n2] and after of array r2. Also, in particular, when isw = 3 is set, the elements in r2[n2] and after must not be changed because they are used in the calculation.

(c) The values of the discrete correlation $\tilde{q}(k)$ are obtained in array r2 as follows.

$\tilde{q}(0)$	$\rightarrow$	r2[0]
$\tilde{q}(1)$	$\rightarrow$	r2[1]
$\cdots$	$\cdots$	$\cdots$
$\tilde{q}(M - 1)$	$\rightarrow$	r2[m − 1]

When m is odd, r2[m] is 0.0, and when m is even, r2[m] and r2[m+1] are each 0.0. Also, when sectioning is performed, the first n2 data usually are meaningful as correlation data (See Note (a)).

When isw=2 is set to obtain the real Fourier transform $Q(j)$ of the discrete correlation $q(k)$, which is defined as follows ($\lfloor x \rfloor$ represents the maximum integer that does not exceed x):

$$Q(j) = \frac{1}{M} \sum_{k=0}^{M-1} q(k) e^{-2\pi\sqrt{-1}\frac{jk}{M}} \quad (j = 0, \cdots, \lfloor \frac{M}{2} \rfloor)$$

the following associations are made:

$$\begin{array}{lcl}
\Re\{Q(0)\} & \leftrightarrow & \text{r2[0]} \\
\Im\{Q(0)\} & \leftrightarrow & \text{r2[1]} \\
\Re\{Q(1)\} & \leftrightarrow & \text{r2[2]} \\
\Im\{Q(1)\} & \leftrightarrow & \text{r2[3]} \\
\cdots & \cdots & \cdots \\
\Re\{Q(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \text{r2[l} - 2\text{]} \\
\Im\{Q(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \text{r2[l} - 1\text{]} \quad (\text{l} = \text{m}+1\text{[m: Odd] or m}+2\text{[m: Even]})
\end{array}$$

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:

$$Q(M-j) = Q(j)^*$$

(Here, z^* represents the conjugate complex number of the complex number z.) Now, $Q(j)$ can be thought of as the estimate of the cross spectrum of the original two functions for which the correlation is to be calculated. In this case, M should be thought of as $M = n_1 + n_2$. In particular, if the original two functions for which the correlation is to be calculated are the same function, $Q(j)$ corresponds to the raw Fourier periodogram (estimate of the power spectrum), and $Q(j)$ is a real number.

(d) If m $\geq$ n1 + n2 $-$ 1 is set, the correlation can be calculated without causing an overlap with the correlation of the next period. When m $>$ n1 + n2 $-$ 1, values that match 0.0 within the error range are stored in element n1 + n2 and following. When isw=0, m = n1 + n2 $-$ 1 should be set. When isw $\geq$ 1, the calculations can be performed more efficiently by setting a value for m 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 n1=n2=145, then when isw=0, m = 289(=17^2) should be set. However, when isw $\geq$ 1, it is usually more efficient to set m = 300(=$2^2 \times 3 \times 5^2$), m = 320(=$2^6 \times 5$), m = 384(=$2^7 \times 3$) or the like.

(e) **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 m, the calculations should be performed by setting isw=0.

(f) 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 $f(i)$ and $g(j)$ are the intervals $[i_0, i_0 + n_1 - 1]$ and $[j_0, j_0 + n_2 - 1]$, respectively, let $\hat{f}(i)$ and $\hat{g}(j)$ be defined as follows:

$$\hat{f}(i) = f(i - i_0), \quad \hat{g}(j) = g(j - j_0)$$

and apply this function to $\hat{f}(i)$ and $\hat{g}(j)$. Let $\tilde{q}(k)$ represent the result that was obtained, and the correlation $q(k)$ of the original functions $f(i)$ and $g(j)$ is given as follows:

$$q(k) = \tilde{q}(k - (j_0 - i_0) + (n_1 - 1))$$

Therefore, even when $i_0 = j_0 = 0$, to consider the correlation $q(k)$ that conforms to the normal definition, you must consider shifting the result by $n_1 - 1$ in the negative direction after applying this function, or if you shift $f(i)$ and $g(j)$ in the negative direction 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.

(g) The sampling interval 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 $q(-n_1) = \tilde{q}(-1) = 0$ and consider $n_1 + n_2$ data of $q(k)$ $(k = -n_1, \cdots, -1, 0, 1, \cdots, n_2 - 1)$. Of course, this is the same as letting $q(n_1 + n_2) = \tilde{q}(n_2) = 0$ and considering $q(k)$ $(k = -(n_1 - 1), \cdots, -1, 0, 1, \cdots, n_2)$. In this case, the coordinate 0 element usually is associated with $q(0)$ and

- when isw=0, ld1=n1, ld2=m and nwk=n2
- when isw=1 or 2,
  ld1=ld2=m+1 and nwk = 2 × m + 1 (when m is odd), or
  ld1=ld2=m+2 and nwk = 2 × m + 2 (when m is even).
- when isw=3,
  ld1=ld2=m+1 and nwk = 2 × m + n1 (when m is odd), or
  ld1=ld2=m+2 and nwk = 2 × m + n1 + 1 (when m is even).

(h) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(7) **Example**

(a) Problem

Use the sampling interval Δx to discretize the two finite waveforms defined by the following equations and calculate the discrete correlation.

$$f(x) = \begin{cases} x & 0 \le x \le a \\ 0 & \text{Otherwise} \end{cases}$$

$$g(x) = \begin{cases} b - x & 0 \le x \le b \\ 0 & \text{Otherwise} \end{cases}$$

Remarks:

The continuous correlation $q(x)$ of $f(x)$ and $g(x)$ is as follows:

$$q(x) = \int_{-\infty}^{\infty} f(\xi) g(x + \xi) d\xi = \begin{cases} G(-x, a, x) & -a \le x \le 0 \\ G(0, a, x) & 0 \le x \le b - a \\ G(0, b - x, x) & b - a \le x \le b \\ 0 & \text{Otherwise} \end{cases}$$

Here, $G(\alpha, \beta, x)$ is as follows:

$$\begin{aligned} G(\alpha, \beta, x) &= \left[\frac{\xi^2}{6} (3(b - x) - 2\xi) \right]_{\alpha}^{\beta} \\ &= \frac{\xi^2}{6} (3(b - x) - 2\xi) \Big|_{\xi=\beta} - \frac{\xi^2}{6} (3(b - x) - 2\xi) \Big|_{\xi=\alpha} \end{aligned}$$

When $a = 2$ and $b = 3$ are set, the values $f(i\Delta x)$, $g(i\Delta x)$ and $q(i\Delta x)$ obtained by sampling $f(x)$, $g(x)$ and $q(x)$ with $\Delta x = 0.1$ are graphed as follows. The values $q(i)\Delta x$ which are the discrete correlation calculated by this function multiplied by Δx are also shown for reference. They match the continuous correlation pretty well for a small number of samples. The program also calculates the continuous correlation for reference.

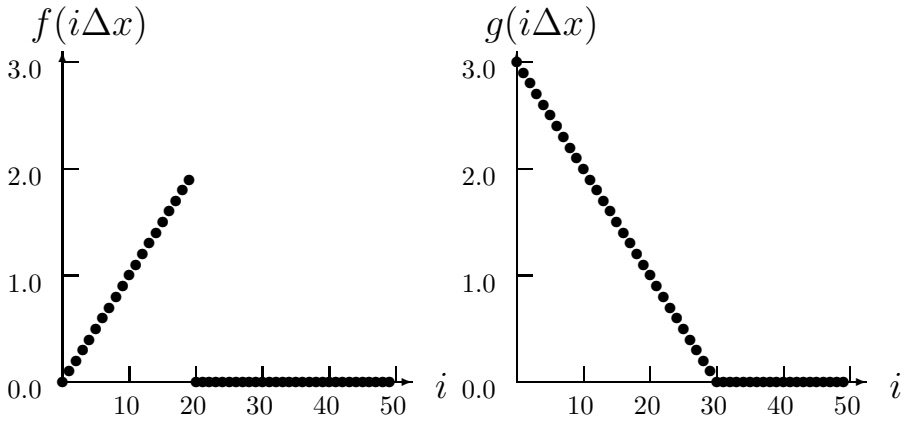


Figure 2–8

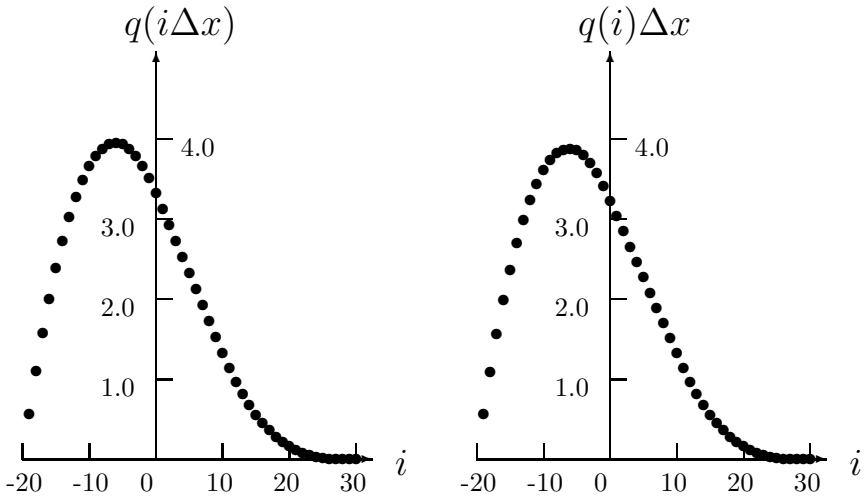


Figure 2–9

(b) Input data

Sampling data

r1[i − 1] = f((i − 1)Δx) (i = 1, 2, ⋯, n1) and

r2[j − 1] = g((j − 1)Δx) (j = 1, 2, ⋯, n2).

Here, $\Delta x = 0.1$.

n1 = $\frac{a}{\Delta x}$, n2 = $\frac{b}{\Delta x}$, m and isw.

(c) Main program

```
/*      C interface example for ASL_dfcr1d */

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

#ifdef __STDC__
double f(double tau, double t, double b)
#else
double f(tau, t, b)
double tau, t, b;
#endif
{
    return tau*tau*(0.5*(b-t)-tau/3.0);
}
```

```
int main()
{
    int n1;
    int n2;
    double *r1;
    int m0=100;
    int ld1=m0+2;
    double *r2;
    int ld2=m0+2;
    int niwk=20;
    int m;
    int isw;
    int *iwk;
    double *wk;
    int ierr;

    int i;
    double *cr,t,dt;
    double a,b;


    printf( "    *** ASL_dfcr1d ***\n" );
    printf( "\n     ** Input **\n\n" );

    r1 = ( double * )malloc((size_t)( sizeof(double) * ld1 ));
    if( r1 == NULL )
    {
        printf( "no enough memory for array r1\n" );
        return -1;
    }
    r2 = ( double * )malloc((size_t)( sizeof(double) * ld2 ));
    if( r2 == NULL )
    {
        printf( "no enough memory for array r2\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * (2*m0+2) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    cr = ( double * )malloc((size_t)( sizeof(double) * ld2 ));
    if( cr == NULL )
    {
        printf( "no enough memory for array cr\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
    isw=1;
    dt=0.1;
    a=2.0;
    b=3.0;
    n1=(int) ((a+0.5*dt)/dt);
    n2=(int) ((b+0.5*dt)/dt);
    m=50;

    printf( "\t isw = %6d\n\t n1 = %6d\n\t n2 = %6d\n\t m = %6d\n\n",
        isw, n1, n2, m);

    for( i=0 ; i<n1 ; i++ )
    {
        t=i*dt;
        r1[i]=t;
    }
    for( i=0 ; i<n2 ; i++ )
    {
        t=i*dt;
        r2[i]=b-t;
    }

    printf( "\tData(r1,r2)\n" );
    printf( "\t  i    r1[i]    r2[i]\n" );
/* ASSUME n2>n1 */
    for( i=0 ; i<n1 ; i++ )
    {
        printf( "\t%3d %8.3g %8.3g\n", i, r1[i], r2[i] );
    }
    for( i=n1 ; i<n2 ; i++ )
    {
        printf( "\t%3d          %8.3g\n", i, r2[i] );
    }

    ierr = ASL_dfcr1d(n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

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

```
    for( i=0 ; i<n1 ; i++ )
    {
        t=(i-n1+1)*dt;
        cr[i]=f(a,t,b)-f(-t,t,b);
    }
    for( i=n1 ; i<n2 ; i++ )
    {
        t=(i-n1+1)*dt;
        cr[i]=f(a,t,b);
    }
    for( i=n2 ; i<n1+n2 ; i++ )
    {
        t=(i-n1+1)*dt;
        cr[i]=f(b-t,t,b);
    }

    printf( "\tCorrelation\n" );
    printf( "\ti-n1+1  r2[i]  r2[i]*dt     cr[i]\n" );
    for( i=0 ; i<n1+n2 ; i++ )
    {
        printf( "\t%3d %9.4lf %9.4lf %9.4lf\n",
             i-n1+1, r2[i], r2[i]*dt, cr[i] );
    }
    free( iwk );
    free( cr );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) Output results

```
*** ASL_dfcr1d ***

 ** Input **

 isw =      1
 n1 =     20
 n2 =     30
 m =     50

Data(r1,r2)
  i    r1[i]    r2[i]
  0        0        3
  1      0.1      2.9
  2      0.2      2.8
  3      0.3      2.7
  4      0.4      2.6
  5      0.5      2.5
  6      0.6      2.4
  7      0.7      2.3
  8      0.8      2.2
  9      0.9      2.1
 10        1        2
 11      1.1      1.9
 12      1.2      1.8
 13      1.3      1.7
 14      1.4      1.6
 15      1.5      1.5
 16      1.6      1.4
 17      1.7      1.3
 18      1.8      1.2
 19      1.9      1.1
 20                 1
 21               0.9
 22               0.8
 23               0.7
 24               0.6
 25               0.5
 26               0.4
 27               0.3
 28               0.2
 29               0.1

 ** Output **

ierr =      0
Correlation
i-n1+1  r2[i]  r2[i]*dt     cr[i]
-19    5.7000    0.5700    0.5752
-18   10.9100    1.0910    1.1013
-17   15.6400    1.5640    1.5795
-16   19.9000    1.9900    2.0107
-15   23.7000    2.3700    2.3958
-14   27.0500    2.7050    2.7360
-13   29.9600    2.9960    3.0322
-12   32.4400    3.2440    3.2853
-11   34.5000    3.4500    3.4965
-10   36.1500    3.6150    3.6667
 -9   37.4000    3.7400    3.7968
 -8   38.2600    3.8260    3.8880
```

-7	38.7400	3.8740	3.9412
-6	38.8500	3.8850	3.9573
-5	38.6000	3.8600	3.9375
-4	38.0000	3.8000	3.8827
-3	37.0600	3.7060	3.7938
-2	35.7900	3.5790	3.6720
-1	34.2000	3.4200	3.5182
0	32.3000	3.2300	3.3333
1	30.4000	3.0400	3.1333
2	28.5000	2.8500	2.9333
3	26.6000	2.6600	2.7333
4	24.7000	2.4700	2.5333
5	22.8000	2.2800	2.3333
6	20.9000	2.0900	2.1333
7	19.0000	1.9000	1.9333
8	17.1000	1.7100	1.7333
9	15.2000	1.5200	1.5333
10	13.3000	1.3300	1.3333
11	11.4000	1.1400	1.1432
12	9.6900	0.9690	0.9720
13	8.1600	0.8160	0.8188
14	6.8000	0.6800	0.6827
15	5.6000	0.5600	0.5625
16	4.5500	0.4550	0.4573
17	3.6400	0.3640	0.3662
18	2.8600	0.2860	0.2880
19	2.2000	0.2200	0.2218
20	1.6500	0.1650	0.1667
21	1.2000	0.1200	0.1215
22	0.8400	0.0840	0.0853
23	0.5600	0.0560	0.0572
24	0.3500	0.0350	0.0360
25	0.2000	0.0200	0.0208
26	0.1000	0.0100	0.0107
27	0.0400	0.0040	0.0045
28	0.0100	0.0010	0.0013
29	0.0000	0.0000	0.0002
30	0.0000	0.0000	0.0000

2.15.2 ASL_dfcr2d, ASL_rfcr2d 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:

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

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_dfcr2d or ASL_rfcr2d 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:

$$\begin{aligned} q(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x + i_x, k_y + i_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

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:

$$\begin{aligned} \tilde{q}(k_x, k_y) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1)) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

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_dfcr2d  (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);
```

Single precision:

```
ierr = ASL_rfcr2d  (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);
```

(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	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	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	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	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))
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)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
15	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	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×my+max(lx1×ly1, lx2×ly2) (When isw ≥ 1)
16	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\text{isw} \in \{0, 1, 2\}$

(b) $\text{nx1} > 1$
$\text{ny1} > 1$

(c) $\text{nx2} > 1$
$\text{ny2} > 1$

(d) $\text{mx} \geq \max(\text{nx1}, \text{nx2})$
$\text{my} \geq \max(\text{ny1}, \text{ny2})$

(e) When $\text{isw} = 0$:
$\text{lx1} \geq \text{nx1}$
$\text{ly1} \geq \text{ny1}$
When $\text{isw} > 0$ and mx is odd:
$\text{lx1} \geq \text{mx} + 1$
$\text{ly1} \geq \text{my}$
When $\text{isw} > 0$ and mx is even:
$\text{lx1} \geq \text{mx} + 2$
$\text{ly1} \geq \text{my}$

(f) When $\text{isw} = 0$:
$\text{lx2} \geq \text{mx}$
$\text{ly2} \geq \text{my}$
When $\text{isw} > 0$ and mx is odd:
$\text{lx2} \geq \text{mx} + 1$
$\text{ly2} \geq \text{my}$
When $\text{isw} > 0$ and mx is even:
$\text{lx2} \geq \text{mx} + 2$
$\text{ly2} \geq \text{my}$

(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.	

(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.

$$f(i_x, i_y) \quad \leftrightarrow \quad \mathrm{r1[i_x + lx1 * i_y]}$$
$$g(j_x, j_y) \quad \leftrightarrow \quad \mathrm{r2[j_x + lx2 * j_y]}$$

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 correlation $\tilde{q}(k_x, k_y)$ and the elements of array r2 are associated as follows.

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * k_y]}$$

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 $Q(j_x, j_y)$ 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):

$$Q(j_x, j_y) = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} q(k_x, k_y) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y}\right)}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor)$$

the following associations are made:

$$\Re\{Q(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + lx2 * j_y]}$$
$$\Im\{Q(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + 1 + lx2 * j_y]}$$

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:

$$Q(M_x - j_x, M_y - j_y)^* = Q(j_x, j_y)$$
$$Q(M_x - j_x, j_y)^* = Q(j_x, M_y - j_y)$$

(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.
This procedure is available for i_y, j_y, and k_y as well.

(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 $q(0, 0)$, and

- when isw=0,
  lx1 = nx1, ly1 = ny1, lx2 = mx, ly2 = my, and

nwk = nx2 × ny2 (when nx2 is odd) or
nwk = (nx2 + 1) × ny2 (when nx2 is even)

- when isw ≥ 1,
  lx1=lx2=mx+1 (when mx is odd) or
  lx1=lx2=mx+2 (when mx is even),
  ly1=ly2=my, and nwk = mx + (lx1 + 2) × my.

(g) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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
$r1[i_x + lx1 * i_y] = f(i_x\Delta, i_y\Delta)$ $(i_x = 0, 1, \cdots, nx1 - 1;\ i_y = 0, 1, \cdots, ny1 - 1)$ and
$r2[j_x + lx2 * j_y] = g(j_x\Delta, j_y\Delta)$ $(j_x = 0, 1, \cdots, nx2 - 1;\ j_y = 0, 1, \cdots, ny2 - 1)$.
Here, $\Delta = 0.5$.
nx1, ny1, nx2, ny2, mx, my and isw.

(c) Main program

```
/*      C interface example for ASL_dfcr2d */

#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;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0;
    double xg=2.0,yg=2.0;


    printf( "    *** ASL_dfcr2d ***\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_dfcr2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
     r2, lx2, ly2, mx, my, isw, iwk, wk);

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_dfcr2d ***

 ** 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
```

2.15.3 ASL_dfcr3d, ASL_rfcr3d 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:

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ & \quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

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_dfcr3d or ASL_rfcr3d 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:

$$\begin{aligned} q(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x + i_x, k_y + i_y, k_z + i_z) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

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:

$$\begin{aligned} \tilde{q}(k_x, k_y, k_z) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1), k_z - (n_z^{(f)} - 1)) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

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 $q(k_x, k_y, k_z)$ can also be obtained.

(2) **Usage**

Double precision:

```
ierr = ASL_dfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

Single precision:

```
ierr = ASL_rfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

(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

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 $\tilde{q}(k_x, k_y, k_z)$ (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 $\tilde{q}(k_x, k_y, k_z)$ (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	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, 1, 2\}$

(b) nx1 > 1
ny1 > 1
nz1 > 1

(c) nx2 > 1

ny2 > 1
nz2 > 1

(d) mx ≥ max(nx1, nx2)
my ≥ max(ny1, ny2)
mz ≥ max(nz1, nz2)

(e) When isw = 0 :
lx1 ≥ nx1
ly1 ≥ ny1
lz1 ≥ nz1
When isw > 0 and mx is odd:
lx1 ≥ mx + 1, lx1 is even
ly1 ≥ my
lz1 ≥ mz
When isw > 0 and mx is even:
lx1 ≥ mx + 2, lx1 is even
ly1 ≥ my
lz1 ≥ mz

(f) When isw = 0)
lx2 ≥ mx
ly2 ≥ ny2
lz2 ≥ nz2
When isw > 0 and mx is odd)
lx2 ≥ mx + 1, lx2 is even
ly2 ≥ my
lz2 ≥ mz
When isw > 0 and mx is even:
lx2 ≥ mx + 2, lx2 is even
ly2 ≥ my
lz2 ≥ mz

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	mx < nx1 + nx2 − 1, 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.	

(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 $\tilde{q}(k_x, k_y, k_z)$ and the elements of array r2 are associated as follows.

$$\tilde{q}(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

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 $Q(j_x, j_y, j_z)$ 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):

$$Q(j_x, j_y, j_z) = \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} q(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor;\ j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor)$$

the following associations are made:

$$\Re\{Q(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + lx2 * (j_y + ly2 * j_z)]}$$
$$\Im\{Q(j_x, j_y, j_z)\} \quad \leftrightarrow \quad \mathrm{r2[2 * j_x + 1 + lx2 * (j_y + ly2 * j_z)]}$$

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 mx $\geq$ nx1 + nx2 $-$ 1 and my $\geq$ ny1 + ny2 $-$ 1 and mz $\geq$ nz1 + nz2 $-$ 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 or mz $>$ nz1 + nz2 $-$ 1, the following correspondences are made:

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2[k_x + lx2 * (k_y + ly2 * k_z)]}$$

and values that match 0.0 within the error range are stored in elements corresponding to k_x = nx1 + nx2 $-$ 1, $\cdots$, mx $-$ 1; $k_y = 0, \cdots,$ my $-$ 1; $k_z = 0, \cdots,$ mz $-$ 1 or $k_x = 0, \cdots,$ mx $-$ 1; $k_y =$

ny1 + ny2 − 1, ⋯ , my − 1; $k_z = 0, \cdots, mz - 1$ or $k_x = 0, \cdots, mx - 1$; $k_y = 0, \cdots, my - 1$; $k_z =$ 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) o 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.
This procedure is available for i_y, j_y, and k_y and i_z, j_z, and k_z as well.

(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

$$\begin{aligned} q(-n_x^{(f)}, k_y, k_z) &= \tilde{q}(-1, k_y, k_z) &= 0 \\ q(k_x, -n_y^{(f)}, k_z) &= \tilde{q}(k_x, -1, k_z) &= 0 \\ q(k_x, k_y, -n_z^{(f)}) &= \tilde{q}(k_x, k_y, -1) &= 0 \end{aligned}$$

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

$$\begin{aligned} q(n_x^{(f)} + n_x^{(g)}, k_y, k_z) &= \tilde{q}(n_x^{(g)}, k_y, k_z) &= 0 \\ q(k_x, n_y^{(f)} + n_y^{(g)}, k_z) &= \tilde{q}(k_x, n_y^{(g)}, k_z) &= 0 \\ q(k_x, k_y, n_z^{(f)} + n_z^{(g)}) &= \tilde{q}(k_x, k_y, n_y^{(g)}) &= 0 \end{aligned}$$

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 is usually associated with $q(0, 0, 0)$, and

- when isw=0,
  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.

(g) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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

$r1[i_x + lx1 * (i_y + ly1 * i_z)] = f(i_x\Delta, i_y\Delta, i_z\Delta)$

($i_x = 0, 1, \cdots, nx1 - 1$; $i_y = 0, 1, \cdots, ny1 - 1$; $i_z = 0, 1, \cdots, nz1 - 1$) and

$r2[j_x + lx2 * (j_y + ly2 * j_z)] = g(j_x\Delta, j_y\Delta, j_z\Delta)$

($j_x = 0, 1, \cdots, nx2 - 1$; $j_y = 0, 1, \cdots, ny2 - 1$; $j_z = 0, 1, \cdots, nz2 - 1$).

Here, $\Delta = 0.5$.

nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz and isw.

(c) Main program

```
/*      C interface example for ASL_dfcr3d */

#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;
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_dfcr3d ***\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_dfcr3d(nx1, ny1, nz1, nx2, ny2, nz2,
        r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
        mx, my, mz, isw, iwk, wk);

    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_dfcr3d ***

 ** 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

2.16 POWER SPECTRUM ANALYSIS

2.16.1 ASL_dfps1d, ASL_rfps1d One-Dimensional Fourier Periodograms

(1) **Function**

ASL_dfps1d or ASL_rfps1d obtains the (modified) Fourier periodogram of the series u_j $(j = 0, \cdots, n-1)$. The Fourier periodogram p_k is defined by the following equation.

$$p_k = \frac{\left| \sum_{j=0}^{n-1} w_j u_j e^{-2\pi\sqrt{-1}\frac{jk}{n}} \right|^2}{n\beta} \quad (k = 0, 1, \cdots, \lfloor \frac{n}{2} \rfloor)$$

Here, $\lfloor x \rfloor$ represents the maximum integer that does not exceed x. w_j is the truncation function (data window). For a raw Fourier periodogram, $w_j = 1$ $(j = 0, \cdots, n-1)$ and $\beta = n$ are set, and for a modified periodogram β is set as follows:

$$\beta = \begin{cases} \sum_{j=0}^{n-1} {w_j}^2 & \text{(when a power modification expression according to a data window is used)} \\ n & \text{(Otherwise)} \end{cases}$$

The periodogram p_k corresponds to a half period (period n) of a two-sided power spectrum, and the remainder is obtained from the relationship $p_{-k} = p_k$. Also, the total power of the corresponding series is as follows.

$$\frac{\sum_{j=0}^{n-1} \{u_j\}^2}{n}$$

(2) **Usage**

Double precision:

ierr = ASL_dfps1d (n, r, ld, isw, iwk, wk);

Single precision:

ierr = ASL_rfps1d (n, r, ld, isw, iwk, wk);

(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	Length n of series u_j (See Note (d))
2	r	{D∗ / R∗}	ld	Input	Values of series u_j (See Note (a))
				Output	Values of Fourier periodogram p_k of series u_j (See Notes (b) and (c))
3	ld	I	1	Input	Size of array r
4	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.
5	iwk	I*	20	Work	Work area
6	wk	{D∗ / R∗}	n+ld	Work	Work area When isw=±1, enter the values of the user-defined data window. (See Note (e))
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) n > 1

(c) When n is an odd:
ld $\geq$ n + 1
When n is an even:
ld $\geq$ n + 2

(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.	
4000	When isw = 1, the user-defined data window was $w_j = 0 \quad (j = 0, \cdots, n-1)$.	

(6) **Notes**

(a) The values of the series u_j are stored in array r as follows.

$u_0 \quad \rightarrow \quad$ r[0]
$u_1 \quad \rightarrow \quad$ r[1]
$\cdots \quad \cdots \quad \cdots$
$u_{n-1} \quad \rightarrow \quad$ r[n − 1]

No values need be entered in elements r[n] and after of array r.

(b) The values of the Fourier periodogram p_k are obtained in array r as follows.

$p_0 \quad \rightarrow \quad$ r[0]
$p_1 \quad \rightarrow \quad$ r[1]
$\cdots \quad \cdots \quad \cdots$
$p_{\lfloor \frac{n+1}{2} \rfloor - 1} \quad \rightarrow \quad$ r[$\lfloor \frac{n+1}{2} \rfloor - 1$]

$\lfloor x \rfloor$ represents the maximum integer that does not exceed x.

(c) The Fourier periodogram p_k that is obtained corresponds to a half period of a two-sided power spectrum (when the negative frequencies are considered), and the corresponding frequencies ξ_k are given by $\xi_k = \frac{k}{n\Delta x}$ (where $\Delta x$is the sampling interval). At this time, the components corresponding to $-\xi_k$ will be p_k. The Fourier periodogram $\hat{p}_k$ corresponding to a one-sided power spectrum is obtained by setting $\hat{p}_0 = p_0;\ \hat{p}_k = 2p_k\ (k = 1, 2, \cdots, m-1)$. However, when n is even, $m = \frac{n}{2}$ and $\hat{p}_m = p_m$ are set, and when n is odd, $m = \frac{n+1}{2}$ is set.

(d) The calculations can be performed more efficiently by setting the length n of the series u_j 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 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. When the number of data cannot be increased, adjust n 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.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{isw= } \pm 2 \text{ (Hanning window)} \\ 1 - |2v_j - 1| & \text{isw= } \pm 3 \text{ (Bartlett window)} \\ 1 - (2v_j - 1)^2 & \text{isw= } \pm 4 \text{ (Welch window)} \\ \begin{cases} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{cases} & \text{isw= } \pm 5 \text{ (Parzen window)} \end{cases}$$

Here, $v_j = \frac{j}{n}$. Therefore, when the data windows shown above are used, the first element u_0 of the series u_j does not affect the calculation of the modified periodogram. To avoid this situation, you should specify a value for n that is larger by 1 than the length of the series for which you actually want to calculate the periodogram and set the effective data so that it starts at u_1. The data windows are represented as follows as time (or space) domain functions that are nonzero only for $|x| \leq 1$.

$$w(x) = \begin{cases} \dfrac{1+\cos \pi x}{2} = \cos^2 \dfrac{\pi x}{2} & \text{Hanning window} \\ 1 - |x| & \text{Bartlett window} \\ 1 - x^2 & \text{Welch window} \\ \left\{ \begin{array}{ll} 1 - 6x^2 + 6|x|^3 & |x| \leq \frac{1}{2} \\ 2(1-|x|)^3 & \frac{1}{2} \leq |x| \leq 1 \end{array} \right\} & \text{Parzen window} \end{cases}$$

Also, to use user-defined data window values w_j, set isw = ± 1, set the values in work array wk as follows:

$$\text{wk[j]} = \text{w}_\text{j} \quad (\text{j} = 0, \cdots, \text{n} - 1)$$

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, when n is odd ld=n+1, and when n is even, ld=n+2.

(i) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(7) **Example**

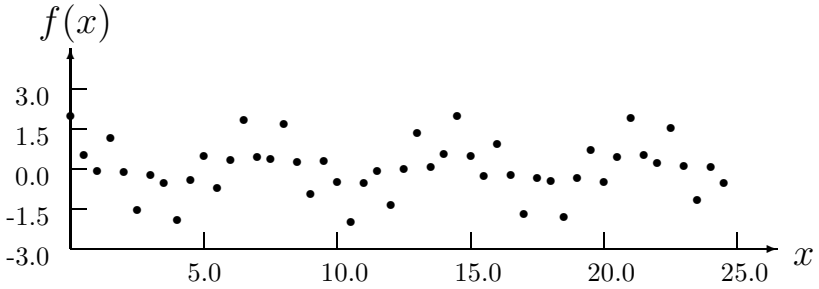
(a) Problem

Use the sampling interval Δx to discretize the waveform defined by the following equation and estimate the power spectrum by calculating the Fourier periodogram.

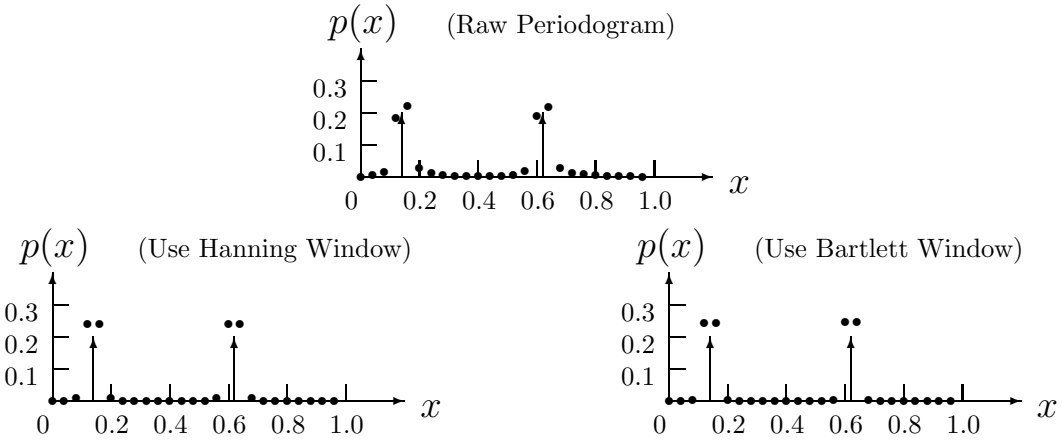
$$f(x) = \cos 2\pi f_1 x + \cos 2\pi f_2 x$$

Remarks:

If $f_1 = 0.62$ and $f_2 = 0.14$ are set and $f(x)$ is sampled on the interval $[0, 25)$ with $\Delta x = 0.5$, the values are graphed as follows. Although, according the to sampling theorem, sampling should be done in more detail depending on the objective, even this degree of sampling enables you to see tendencies concerning differences due to the selection of the data window.



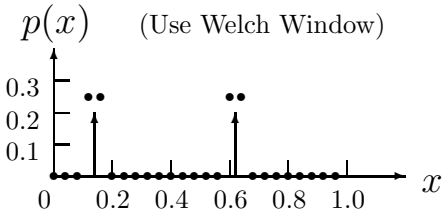
Also, the corresponding Fourier periodogram is graphed as follows (the upward pointing arrows are the signal frequency). Since a frequency for which the discontinuity increases due to truncation is deliberately used as the signal frequency, the leakage increases in the raw periodogram.

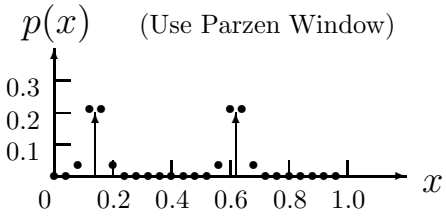


(b) Input data

Sampling data

r[j − 1] = f((j − 1)Δx) (j = 1, 2, ⋯, n).





Here, $\Delta x = 0.5$.

n and isw.

(c) Main program

```
/*      C interface example for ASL_dfps1d */

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

int main()
{
    int n0=50,isw0=4;
    int n;
    double *r;
    int ld=n0+2;
    int isw;
    int *iwk;
    int niwk=20;
    double *wk;
    int ierr;
    int i,m,nd2,is;
    double *p,t,dt,f0,f1,f2;

    printf( "    *** ASL_dfps1d ***\n" );
    printf( "\n     ** Input **\n\n" );

    r = ( double * )malloc((size_t)( sizeof(double) * (ld*(isw0+2)) ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * (n0+ld) ));
    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;
    }

    n=n0;
    printf( "\t isw=0, 2 to %6d\n", isw0+1 );
    printf( "\t n=%6d\n\n", n );

    dt=0.5;
    f0=1.0/(2.0*dt);
    f1=0.62*f0;
    f2=0.14*f0;
    nd2=(int) (n+1)/2;
    p[isw0+1]=0.0;
    for( i=0 ; i<n ; i++ )
    {
        t=(double) i*dt;
        t=cos(2.0*M_PI*f1*t)+cos(2.0*M_PI*f2*t);
        r[i+ld*(isw0+1)]=t;
        p[isw0+1] += (t*t);
    }
    p[isw0+1] /= (double) n;
    printf( "\tTime series data\n");
    printf( "\t  i      time      r[i]\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%3d %9.4lf %9.4lf\n", i, i*dt, r[i+ld*(isw0+1)] );
    }
    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( i=0 ; i<n ; i++ )
            r[i+ld*isw]=r[i+ld*(isw0+1)];
        if (isw != 0)
            is=isw+1;

        ierr = ASL_dfps1d(n, &r[ld*isw], ld, is, iwk, wk);

        /* For one-sided power spectral densities */
        if (n%2==0)
            m=nd2-1;
        else
            m=nd2;
        for( i=1 ; i<m ; i++ )
            r[i+ld*isw] *=2.0;
        p[isw]=0.0;
        for( i=0 ; i<nd2 ; i++ )
            p[isw] += r[i+ld*isw];
    }

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\t(Modified) periodogram/");
    printf( "one-sided power spectrum estimation\n" );
    printf( "\t  i     Freq.     Raw    Hanning Bartlett");
    printf( "   Welch   Parzen\n" );
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "\t%3d %9.4lf", i, (double) i/(dt*n) );
        for( isw=0 ; isw<=isw0 ; isw++ )
            printf( "%9.4lf", r[i+ld*isw] );
        printf( "\n");
    }
    printf( "\n\tFrequency domain power\n" );
    printf( "\t                " );
    printf( "  Raw    Hanning Bartlett   Welch   Parzen\n" );
    printf( "\t             " );
    for( isw=0 ; isw<=isw0 ; isw++ )
        printf( "%9.4lf", p[isw] );
    printf( "\n");

    free( iwk );
    free( p );
    free( wk );
    free( r );

    return 0;
}
```

(d) Output results

```
*** ASL_dfps1d ***

 ** Input **

 isw=0, 2 to      5
 n=    50

Time series data
  i      time      r[i]
  0    0.0000    2.0000
  1    0.5000    0.5367
  2    1.0000   -0.0915
  3    1.5000    1.1535
  4    2.0000   -0.1246
  5    2.5000   -1.5388
  6    3.0000   -0.2389
  7    3.5000   -0.5163
  8    4.0000   -1.9219
  9    4.5000   -0.4359
 10    5.0000    0.5000
 11    5.5000   -0.7190
 12    6.0000    0.3484
 13    6.5000    1.8266
 14    7.0000    0.4563
 15    7.5000    0.3633
 16    8.0000    1.6976
 17    8.5000    0.2428
 18    9.0000   -0.9391
 19    9.5000    0.2888
 20   10.0000   -0.5000
 21   10.5000   -1.9803
 22   11.0000   -0.5428
 23   11.5000   -0.0860
 24   12.0000   -1.3556
 25   12.5000    0.0000
```

```
 26   13.0000    1.3556
 27   13.5000    0.0860
 28   14.0000    0.5428
 29   14.5000    1.9803
 30   15.0000    0.5000
 31   15.5000   -0.2888
 32   16.0000    0.9391
 33   16.5000   -0.2428
 34   17.0000   -1.6976
 35   17.5000   -0.3633
 36   18.0000   -0.4563
 37   18.5000   -1.8266
 38   19.0000   -0.3484
 39   19.5000    0.7190
 40   20.0000   -0.5000
 41   20.5000    0.4359
 42   21.0000    1.9219
 43   21.5000    0.5163
 44   22.0000    0.2389
 45   22.5000    1.5388
 46   23.0000    0.1246
 47   23.5000   -1.1535
 48   24.0000    0.0915
 49   24.5000   -0.5367
Time domain power =   1.0000
Signal frequency =   0.6200,    0.1400

 ** Output **

ierr =      0
(Modified) periodogram/one-sided power spectrum estimation
  i     Freq.      Raw    Hanning Bartlett   Welch   Parzen
  0    0.0000   0.0016   0.0000   0.0000   0.0000   0.0000
  1    0.0400   0.0051   0.0001   0.0002   0.0000   0.0006
  2    0.0800   0.0166   0.0094   0.0026   0.0003   0.0369
  3    0.1200   0.1841   0.2408   0.2437   0.2494   0.2116
  4    0.1600   0.2211   0.2398   0.2446   0.2498   0.2121
  5    0.2000   0.0286   0.0096   0.0029   0.0003   0.0373
  6    0.2400   0.0117   0.0002   0.0004   0.0000   0.0006
  7    0.2800   0.0068   0.0000   0.0001   0.0000   0.0000
  8    0.3200   0.0047   0.0000   0.0000   0.0000   0.0000
  9    0.3600   0.0036   0.0000   0.0000   0.0000   0.0000
 10    0.4000   0.0032   0.0000   0.0000   0.0000   0.0000
 11    0.4400   0.0033   0.0000   0.0001   0.0000   0.0000
 12    0.4800   0.0042   0.0000   0.0001   0.0000   0.0000
 13    0.5200   0.0072   0.0002   0.0004   0.0000   0.0006
 14    0.5600   0.0197   0.0096   0.0031   0.0003   0.0373
 15    0.6000   0.1906   0.2403   0.2463   0.2496   0.2121
 16    0.6400   0.2177   0.2401   0.2462   0.2497   0.2121
 17    0.6800   0.0285   0.0096   0.0030   0.0003   0.0373
 18    0.7200   0.0122   0.0002   0.0004   0.0000   0.0006
 19    0.7600   0.0075   0.0000   0.0001   0.0000   0.0000
 20    0.8000   0.0054   0.0000   0.0000   0.0000   0.0000
 21    0.8400   0.0044   0.0000   0.0000   0.0000   0.0000
 22    0.8800   0.0038   0.0000   0.0000   0.0000   0.0000
 23    0.9200   0.0034   0.0000   0.0000   0.0000   0.0000
 24    0.9600   0.0016   0.0000   0.0000   0.0000   0.0000

Frequency domain power
                  Raw    Hanning Bartlett   Welch   Parzen
                0.9968   1.0000   0.9943   0.9999   0.9991
```

2.16.2 ASL_dfps2d, ASL_rfps2d Two-Dimensional Fourier Periodograms

(1) **Function**

ASL_dfps2d or ASL_rfps2d 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_dfps2d (nx, ny, r, lx, ly, isw, iwk, wk);

Single precision:

ierr = ASL_rfps2d (nx, ny, r, lx, ly, isw, iwk, wk);

(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**: nx + 2 × ny + lx × ly
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx > 1
ny > 1

(c) When nx is an odd:
lx $\geq$ nx + 1
ly $\geq$ ny
When nx is an even:
lx $\geq$ nx + 2
ly $\geq$ ny

(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.	
4000	When isw = 1, the user-defined data window was $w_{j_x}^{(x)} = 0 \quad (j_x = 0, \cdots, n_x - 1)$.	
4010	When isw = 1, the user-defined data window was $w_{j_y}^{(y)} = 0 \quad (j_y = 0, \cdots, n_y - 1)$.	

(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 $w_{j_x}^{(x)}$ and $j = j_y$ and $n = n_y$ are set for $w_{j_y}^{(y)}$. 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 $w_{j_x}^{(x)}$ and $w_{j_y}^{(y)}$, set isw = ± 1, set the values in work array wk as follows:

$$\mathrm{wk[j_x]} = \mathrm{w_{j_x}^{(x)}} \ \ (\mathrm{j_x} = 0, \cdots, \mathrm{n_x} - 1), \mathrm{wk[n_x + j_y]} = \mathrm{w_{j_y}^{(y)}} \ \ (\mathrm{j_y} = 0, \cdots, \mathrm{n_y} - 1)$$

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.

(i) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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
$\mathrm{r}[\mathrm{j_x} + \mathrm{lx} * \mathrm{j_y}] = \mathrm{f}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny} - 1)$.
Here, $\Delta = 0.5$.
nx, ny and isw.

(c) Main program

```
/*      C interface example for ASL_dfps2d */

#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;
double *p,t,tx,ty,dt,dfx,dfy,f0,f1,f2;

printf( "    *** ASL_dfps2d ***\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_dfps2d(nx, ny, &r[lx*ly*isw], lx, ly, is, iwk, wk);
    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_dfps2d ***

 ** 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
```

2.16.3 ASL_dfps3d, ASL_rfps3d Three-Dimensional Fourier Periodograms

(1) **Function**

ASL_dfps3d or ASL_rfps3d 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:

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1} (w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1} (w_{j_y}^{(y)})^2)(\sum_{j_z=0}^{n_z-1} (w_{j_z}^{(z)})^2) & \text{(when a power modification expression according to a data window is used)} \\ n_x n_y n_z & \text{(Otherwise)} \end{cases}$$

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_dfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk);

Single precision:

ierr = ASL_rfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk);

(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	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx > 1
ny > 1
nz > 1

(c) When nx is an odd:
lx $\geq$ nx $+ 1$, lx is even
ly $\geq$ ny
lz $\geq$ nz
When nx is an even:
lx $\geq$ nx $+ 2$, lx is even
ly $\geq$ ny
lz $\geq$ nz

(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.	
4000	When isw $= 1$, the user-defined data window was $w_{j_x}^{(x)} = 0 \quad (j_x = 0, \cdots, n_x - 1)$.	
4010	When isw $= 1$, the user-defined data window was $w_{j_y}^{(y)} = 0 \quad (j_y = 0, \cdots, n_y - 1)$.	
4020	When isw $= 1$, the user-defined data window was $w_{j_z}^{(z)} = 0 \quad (j_z = 0, \cdots, n_z - 1)$.	

(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 $w_{j_x}^{(x)}$, $j = j_y$ and $n = n_y$ are set for $w_{j_y}^{(y)}$, and $j = j_z$ and $n = n_z$ are set for $w_{j_z}^{(z)}$. 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 $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ and $w_{j_z}^{(z)}$, set isw = ± 1, set the values in work array wk as follows:

$$\mathrm{wk[j_x]} = \mathrm{w_{j_x}^{(x)}} \quad (\mathrm{j_x} = 0, \cdots, \mathrm{n_x} - 1), \mathrm{wk[n_x + j_y]} = \mathrm{w_{j_y}^{(y)}} \quad (\mathrm{j_y} = 0, \cdots, \mathrm{n_y} - 1),$$
$$\mathrm{wk[n_x + n_y + j_z]} = \mathrm{w_{j_z}^{(z)}} \quad (\mathrm{j_z} = 0, \cdots, \mathrm{n_z} - 1)$$

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)
ly=ny and lz=nz.

(i) This function is not thread-safe in the sequential version and the MPI version of the libraries without OpenMP.

(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

$r[j_x + lx * (j_y + ly * j_z)] = f(j_x\Delta, j_y\Delta, j_z\Delta)$
$(j_x = 0, 1, \cdots, nx - 1;\ j_y = 0, 1, \cdots, ny - 1;\ j_z = 0, 1, \cdots, nz - 1).$
Here, $\Delta = 0.5$.

nx, ny, nz and isw.

(c) Main program

```
/*      C interface example for ASL_dfps3d */

#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;
    double *p,t,tx,ty,tz,dt,dfx,dfy,dfz,f0,f1,f2,f3;

    printf( "    *** ASL_dfps3d ***\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_dfps3d(nx, ny, nz, &r[lx*ly*lz*isw],
         lx, ly, lz, is, iwk, wk);
    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_dfps3d ***

     ** 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
```

2.17 LAPLACE TRANSFORM

2.17.1 ASL_dflara, ASL_rflara
Inverse Laplace Transform (Rational Function)

(1) **Function**

ASL_dflara or ASL_rflara obtains the inverse Laplace transform:

$$f(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \qquad (0 < t < \infty)$$

of the rational function:

$$F(s) = \frac{Q(s)}{P(s)} = \frac{q_1 s^{nq} + q_2 s^{nq-1} + \cdots + q_{nq}s + q_{nq+1}}{p_1 s^{np} + p_2 s^{np-1} + \cdots + p_{np}s + p_{np+1}}$$

$(np \leq nq; p_1, \cdots, p_{np+q}, q_1, \cdots, q_{nq+1}$: real number)

(2) **Usage**

Double precision:

```
ierr = ASL_dflara  (p, np, q, nq, t, n, a, ip, k1, k2, &r, f, er, isw, w1);
```

Single precision:

```
ierr = ASL_rflara  (p, np, q, nq, t, n, a, ip, k1, k2, &r, f, er, isw, w1);
```

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	p	{D∗ / R∗}	np+1	Input	Coefficients of denominator polynomial $P(s)$
2	np	I	1	Input	Degree of denominator polynomial$P(s)$
3	q	{D∗ / R∗}	nq+1	Input	Coefficients of numerator polynomial $Q(s)$
4	nq	I	1	Input	Degree of numerator polynomial $Q(s)$
5	t	{D∗ / R∗}	n	Input	Calculation points of original function $f(t)$
6	n	I	1	Input	Dimension of array t
7	a	{D / R}	1	Input	Value of a for determining approximation error (See 2.1.2(1) (d))
8	ip	I	1	Input	Degree p of Euler's transformation (See 2.1.2(1) (d))
9	k1	{D / R}	1	Input	Parameter k_1 for determining truncation term count (See 2.1.2(1) (d))
10	k2	{D / R}	1	Input	Parameter k_2 for determining truncation term count (See 2.1.2(1) (d))
11	r	{D∗ / R∗}	1	Input Output	isw=1: 0.0 is output to r. isw=2: Input Abscissa of convergence which is known. isw=3: Output Abscissa of convergence which is calculated. (See Note (b))
12	f	{D∗ / R∗}	n	Output	Value of original function$f(t)$ for each t[i] (i=0, ···, n−1)
13	er	{D∗ / R∗}	n	Output	Truncation errors calculating each f[i] (i=0, ···, n−1)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
14	isw	I	1	Input	isw=1: When function $F(s)$ is regular for $\Re(s) > 0$. isw=2: Input abscissa of convergence to r. (when $F(s)$ is irregular for $\Re(s) > 0$ and abscissa of convergence is known.) isw=3: Output Abscissa of convergence to r. (when Abscissa of convergence is unknown.)
15	w1	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	2 × (ip + np + 1)	Work	Work area
16	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0

(b) t[i] > 0.0 $\quad (i = 0, \cdots, \text{n} - 1)$

(c) p[0] > 0.0

(d) 0 < nq ≤ np

(e) a > 0.0

(f) ip > 0

(g) k1 > 0.0
k2 ≥ 0.0

(h) r ≥ 0.0

(i) isw ∈ {1, 2, 3}

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	t[i] was equal to 0.0. $(i = 0, \cdots, \text{n} - 1)$	See Notes (a) and (b).
1100	t[i] was less than 0.0. $(i = 0, \cdots, \text{n} - 1)$	Processing of t[i] is aborted and processing continues of t[i + 1] and later.
2000	r was less than 0.0. (If isw = 2.)	Set 0.0 to r and processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3020	Restriction (d) was not satisfied.	
3030	Restriction (e) was not satisfied.	
3040	Restriction (f) was not satisfied.	
3050	Restriction (g) was not satisfied.	
3060	Restriction (i) was not satisfied.	

(6) **Notes**

(a) You can easily control error by changing the values of a, ip, k1 and k2.

(b) If isw is equal to 1, r←0.0 is performed.
If isw is equal to 2, when $F(s)$ is regular for $\Re(s) > \alpha$, input γ to r, where γ is greater than or equal to α. If r is less than or equal to 0.0, then 0.0 is used as the value of r.
If isw is equal to 3, abscissa of convergence γ_0 is calculated and is used as the value of r.

(c) When np is equal to nq, $F(s)$ can be decomposed as follows:

$$F(s) = \frac{Q(s)}{P(s)} = \frac{q_1}{p_1} + G(s)$$

$$\text{where, } G(s) = \frac{o_2 s^{np-1} + o_3 s^{np-2} + \cdots + o_{np-1}}{p_1 s^{np} + p_2 s^{np-1} + \cdots + p_{np+1}}$$

If we let $g(t)$ represent the inverse transform of $G(s)$, then:

$$f(t) = \frac{q_1}{p_1}\delta(t) + g(t)$$

Here, $\delta(t)$ is the Dirac δ-function.
Therefore,

$$f(t) = \begin{cases} \text{Maximum value} & (t = 0) \\ g(t) & (t > 0) \end{cases} .$$

(d) When t[i−1]=0.0 is assigned, the value of $f(0)$ is calculated from the formula $f(0) = [sF(s)]_{s=\infty}$.

$$f(0) = \begin{cases} \text{Maximum value} & (np = nq) \\ \frac{q_1}{p_1} & (np = nq + 1) \\ 0 & (np > nq + 1) \end{cases}$$

(7) **Example**

(a) Problem
Obtain the Laplace transform $f(t)$ of the following rational function that is regular for $\Re(s) > 0$:

$$F(s) = \frac{1}{s+1}$$

for t =1.0, 2.0, 3.0, 4.0 and 5.0.

(b) Input data
p={1, 1}, np=1, q=1, nq=0, n=5, t={1.0, 2.0, 3.0, 4.0, 5.0}, k1=10.0, k2=0.0, ip=10, a=10.0 and isw=1.

(c) Main program

```
/*      C interface example for ASL_dflara */

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

int main()
{
    double *q;
    int nq;
    double *p;
    int np;
    double *t;
    int n;
    double a;
    int ip;
    double k1;
    double k2;
```

```
double r;
double *ff;
double *e;
int isw;
double *w1;
int ierr;
int i;
FILE *fp;

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

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

fscanf( fp, "%d", &np );
fscanf( fp, "%d", &nq );
fscanf( fp, "%d", &n );
fscanf( fp, "%lf", &a );
fscanf( fp, "%d", &ip );
fscanf( fp, "%lf", &k1 );
fscanf( fp, "%lf", &k2 );
fscanf( fp, "%d", &isw );

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

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

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

ff = ( double * )malloc((size_t)( sizeof(double) * n ));
if( ff == NULL )
{
    printf( "no enough memory for array ff\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) * 2 * (np+ip+1) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\tnp= %6d\n", np );
for( i=0 ; i<np+1 ; i++ )
{
    fscanf( fp, "%lf", &p[i] );
    printf( "\tp[ %2d ]=  %8.3g\n", i, p[i] );
}

printf( "\n\tnq= %6d\n", nq );
for( i=0 ; i<nq+1 ; i++ )
{
    fscanf( fp, "%lf", &q[i] );
    printf( "\tq[ %2d ]=  %8.3g\n", i, q[i] );
}

printf( "\n\tn= %6d\n", n );
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &t[i] );
    printf( "\tt[ %2d ]=  %8.3g\n", i, t[i] );
```

```
    }

    printf( "\n\ta  = %8.3g\n", a );
    printf( "\tip = %6d\n", ip );
    printf( "\tk1 = %8.3g\n", k1 );
    printf( "\tk2 = %8.3g\n", k2 );
    printf( "\tisw= %6d\n", isw );

    fclose( fp );

    ierr = ASL_dflara(p, np, q, nq, t, n, a, ip, k1, k2, &r, ff, e, isw, w1);

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

    printf( "\tSolution \n" );
    printf( "\t     i         t[i]        ff[i]         e[i] \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%6d %12.5g %12.5g %12.5g\n", i, t[i], ff[i], e[i]);
    }

    free( p );
    free( q );
    free( t );
    free( ff );
    free( e );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_dflara ***

 ** Input **

np=       1
p[  0 ]=         1
p[  1 ]=         1

nq=       0
q[  0 ]=         1

n=       5
t[  0 ]=         1
t[  1 ]=         2
t[  2 ]=         3
t[  3 ]=         4
t[  4 ]=         5

a  =       10
ip =     10
k1 =       10
k2 =        0
isw=       1

 ** Output **

ierr =       0

Solution
     i         t[i]        ff[i]         e[i]
     0            1      0.36788  -1.5402e-06
     1            2      0.13534   -1.637e-06
     2            3     0.049788  -1.6359e-06
     3            4     0.018317  -1.5501e-06
     4            5    0.0067389   -1.397e-06
```

2.17.2 ASL_dflage, ASL_rflage
Inverse Laplace Transform (General Function)

(1) **Function**

ASL_dflage or ASL_rflage obtains the inverse Laplace transform:

$$f(t) = \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \qquad (0 < t < \infty)$$

of the general function $F(s)$.

(2) **Usage**

Double precision:

ierr = ASL_dflage (fi, t, n, a, ip, k1, k2, r, f, er, w1);

Single precision:

ierr = ASL_rflage (fi, t, n, a, ip, k1, k2, r, f, er, w1);

(3) **Arguments and Return Value**

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

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
1	fi	{D / R}	−	Input	Name of function that calculates the imaginary part of image function $F(s)$
2	t	{D∗ / R∗}	n	Input	Calculation points of original function $f(t)$
3	n	I	1	Input	Dimension of array t
4	a	{D / R}	1	Input	Value of a for determining approximation error (See 2.1.2(1)(d))
5	ip	I	1	Input	Degree p of Euler's transformation (See 2.1.2(1)(d))
6	k1	{D / R}	1	Input	Parameter k_1 for determining truncation term count (See 2.1.2(1)(d))
7	k2	{D / R}	1	Input	Parameter k_2 for determining truncation term count (See 2.1.2(1)(d))
8	r	{D / R}	1	Input	Abscissa of convergence
9	f	{D∗ / R∗}	n	Output	Value of original function$f(t)$ for each t[i] (i=0, ···, n−1)

No.	Argument and Return Value	Type	Size	Input/ Output	Contents
10	er	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	n	Output	Truncation errors calculating each f[i] (i=0, $\cdots$, n−1)
11	w1	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	2×(ip+2)	Work	Work area
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) n > 0

(b) t[i] > 0.0 $\quad (i = 0, \cdots, \text{n} - 1)$

(c) a > 0.0

(d) ip > 0

(e) k1 > 0.0
k2 ≥ 0.0

(f) r ≥ 0.0

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

ierr value	Meaning	Processing
0	Normal termination.	
1000	t[i] was less than or equal to 0.0. $(i = 0, \cdots, \text{n} - 1)$	Processing of t[i] is aborted and processing continues of t[i + 1] and later.
2000	r was less than 0.0.	Set 0.0 to r and processing continues.
3000	Restriction (a) was not satisfied.	Processing is aborted.
3010	Restriction (c) was not satisfied.	
3020	Restriction (d) was not satisfied.	
3030	Restriction (e) was not satisfied.	

(6) **Notes**

(a) The functions fi is created as follows.

```
double FORTRAN fi(Complex_d *s)
{
        return (fi(*s));
}
```

In addition, this function evaluates the value of imaginary part of function $F(s)$ in the range:

$$\Re(s) > 0.0, \Im(s) > 0.0$$

When $F(s)$ is a many valued function, care is required so that the correct branch is calculated within the function fi.

(b) You can easily control error by changing the values of a, ip, k1 and k2.

(c) When $F(s)$ is regular for $\Re(s) > \alpha$, input γ to r, where γ is greater than or equal toα. If r is less than of equal to 0.0, then 0.0 is used as the value of r.

(7) **Example**

(a) Problem

Obtain the Laplace transform $f(t)$ of the following function that is regular for $\Re(s) > 0$:

$$F(s) = e^{-\sqrt{s}}$$

for t =1.0, 2.0, 3.0, 4.0 and 5.0.

(b) Input data

function name: fi.

t={1.0, 2.0, 3.0, 4.0, 5.0}, n=5, k1=10.0, k2=0.0, ip=10, a=10 and r=0.0.

(c) Main program

```
/*      C interface example for ASL_dflage */

#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <asl.h>
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double fi(double _Complex *s)
#else
double fi(s)
double _Complex *s;
#endif
{
    double _Complex y,z;
    double xi;
    y = csqrt(*s);
    if( creal(y) > 0.0 )
    {
        y = -y;
    }
    z = cexp(y);
    xi = cimag(z);
    return xi;
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *t;
    int n;
    double a;
    int ip;
    double k1;
    double k2;
    double r;
    double *ff;
    double *e;
    double *w1;
    int ierr;
    int i;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%d",  &ip );
    fscanf( fp, "%lf", &k1 );
    fscanf( fp, "%lf", &k2 );
    fscanf( fp, "%lf", &r );

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

```
        return -1;
    }

    ff = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( ff == NULL )
    {
        printf( "no enough memory for array ff\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) * 2 * (ip+2) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

    printf( "\tn= %6d\n", n );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &t[i] );
        printf( "\tt[ %2d ]=  %8.3g\n", i, t[i] );
    }


    printf( "\n\ta  = %8.3g\n", a );
    printf( "\tip = %6d\n", ip );
    printf( "\tk1 = %8.3g\n", k1 );
    printf( "\tk2 = %8.3g\n", k2 );
    printf( "\tr  = %8.3g\n", r );

    fclose( fp );

    ierr = ASL_dflage(fi, t, n, a, ip, k1, k2, r, ff, e, w1);

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

    printf( "\tSolution \n" );
    printf( "\t     i         t[i]        ff[i]         e[i] \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%6d %12.5g %12.5g %12.5g\n", i, t[i], ff[i], e[i]);
    }

    free( t );
    free( ff );
    free( e );
    free( w1 );

    return 0;
}
```

(d) Output results

```
*** ASL_dflage ***

 ** Input **

n=      5
t[  0 ]=         1
t[  1 ]=         2
t[  2 ]=         3
t[  3 ]=         4
t[  4 ]=         5

a  =       10
ip =     10
k1 =       10
k2 =        0
r  =        0

 ** Output **

ierr =      0
Solution
     i         t[i]        ff[i]         e[i]
     0            1       0.2197  -4.0346e-06
     1            2     0.088017  -1.5425e-06
     2            3     0.049949  -7.5899e-07
     3            4     0.033126  -4.5184e-07
     4            5     0.024001  -3.0216e-07
```

2.18 WAVELET TRANSFORM

2.18.1 ASL_dfwth1, ASL_rfwth1 Haar Function Generation

(1) **Function**

The ASL_dfwth1 or ASL_rfwth1 generates the following Haar function, which is required for a one-dimensional Wavelet transform, on the interval $[0, a](a \geq 0.0)$.

$$H_{mn}(x) = \begin{cases} \sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1) \leq x \leq \frac{a}{2^m}(n-1/2) \\ -\sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1/2) < x \leq \frac{a}{2^m}n \\ 0 & \text{Otherwise} \end{cases}$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwth1  (a, m, n, &c, &bl, &bm, &br);
```

Single precision:

```
ierr = ASL_rfwth1  (a, m, n, &c, &bl, &bm, &br);
```

(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}	1	Input	Upper bound a of interval $[0, a]$ on which Haar function $H_{mn}(x)$ is to be generated.
2	m	I	1	Input	Generation m of Haar function $H_{mn}(x)$.
3	n	I	1	Input	Index n of Haar function $H_{mn}(x)$ (See Note (a)).
4	c	{D∗ / R∗}	1	Output	Absolute value $\sqrt{\frac{2^m}{a}}$ of Haar function $H_{mn}(x)$ on interval $[\frac{a}{2^m}(n-1), \frac{a}{2^m}n]$.
5	bl	{D∗ / R∗}	1	Output	Smaller position $\frac{a}{2^m}(n-1)$ at which Haar function $H_{mn}(x)$ rises.
6	bm	{D∗ / R∗}	1	Output	Position $\frac{a}{2^m}(n-1/2)$ at which value of Haar function $H_{mn}(x)$ changes from positive to negative.
7	br	{D∗ / R∗}	1	Output	Larger position $\frac{a}{2^m}n$ at which Haar function $H_{mn}(x)$ rises.
8	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $a > 0$

(b) $m \geq 0$

(c) $n \leq 2^m$

(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.	

(6) **Notes**

(a) This function does not assign a constant term. The fixed value $1/\sqrt{a}$ required on the interval $[0, a]$ in addition to the values assigned by this function for the Haar function used as the base. If the input data is continuous or the sampled intervals are irregular, this function is used in a Wavelet transform. If the spacing with which the transformed data was sampled is fixed and there are 2^k sampled data (where k is a natural number), the Haar function for the Wavelet transform can be generated by using the simpler function 2.18.4 $\left\{\begin{matrix}\text{ASL\_dfwth2}\\ \text{ASL\_rfwth2}\end{matrix}\right\}$.

(7) **Example**

(a) Problem
When $a = 2$, compute the Haar function $H_{34}(x)$.

(b) Input data

$$a = 2, m = 3 \text{and} n = 4.$$

(c) Main program

```
/*      C interface example for ASL_dfwth1 */

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

int main()
{
    double a;
    int m;
    int n;
    double c;
    double bl;
    double bm;
    double br;
    int ierr;
    FILE *fp;

    fp = fopen( "dfwth1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dfwth1 ***\n" );
    printf( "\n     ** Input **\n\n" );

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

```
        printf( "\ta = %8.3g m = %6d n = %6d\n", a, m, n );

        fclose( fp );

        ierr = ASL_dfwth1(a, m, n, &c, &bl, &bm, &br);

        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n\n", ierr );
        printf( "\t   c = %8.3g\n", c );
        printf( "\t  bl = %8.3g\n", bl );
        printf( "\t  bm = %8.3g\n", bm );
        printf( "\t  br = %8.3g\n", br );

        return 0;
    }
```

(d) Output results

```
*** ASL_dfwth1 ***

 ** Input **

a =        1 m =      2 n =      2

 ** Output **

ierr =      0

   c =        2
  bl =     0.25
  bm =    0.375
  br =      0.5
```

2.18.2 ASL_dfwthr, ASL_rfwthr
Wavelet Transform According to Haar Functions

(1) **Function**

When the spacing at which the input data $\{(x_i, f(x_i))\}$ is sampled is not equally spaced or when the number of sampled data is not 2^k (where k is a natural number), the ASL_dfwthr or ASL_rfwthr computes the following Wavelet transform according to Haar functions.

$$C_{mn} = \int_0^a f(x) H_{mn}(x) dx$$

(2) **Usage**

Double precision:

ierr = ASL_dfwthr (xd, yd, nd, mr, nr, dr, imr, inr);

Single precision:

ierr = ASL_rfwthr (xd, yd, nd, mr, nr, dr, imr, inr);

(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	xd	{D∗ / R∗}	nd	Input	Set of input data x-coordinates $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	Input	Set of input data function values $\{f(x_i)\}$.
3	nd	I	1	Input	Number of input data N.
4	mr	I	1	Input	Maximum value of index m of Haar functions $H_{mn}(x)$ used for Wavelet transform
5	nr	I	1	Input	Maximum value of index n of Haar functions $H_{mn}(x)$ used for Wavelet transform. Haar functions up to index n in generation m are used for Wavelet transform.
6	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	Output	Wavelet transform C_{mn}. (See Notes (a) and (b))
7	imr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Output	Index m information of Wavelet transform C_{mn}, which is stored in dr (See Note (b))
8	inr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Output	Index n information of Wavelet transform C_{mn}, which is stored in dr. (See Note (b))
9	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{mr} \geq 0$

(b) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(c) $\mathrm{nd} \leq 100000$

(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.	

(6) **Notes**

(a) The mean value of the input data $\{f(x_i)\}$ is stored in dr[0].

(b) The Wavelet transform C_{mn} values are stored in dr[i]($i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1$). The corresponding index m values are stored in imr[$i-1$] and the corresponding index n values are stored in inr[$i-1$].

(7) **Example**

(a) Problem

Compute the Wavelet transform up to $m = 4$ and $n = 2$ using the input data which is obtained by the equally spaced sampling in the interval $[0, 1]$ for the function

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}.$$

(b) Input data

Input data $\{(x_i, f(x_i))\}, \mathrm{nd} = 10, \mathrm{mr} = 4 \mathrm{and} \mathrm{nr} = 2.$

(c) Main program

```
/*      C interface example for ASL_dfwthr */

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

int main()
{
    double *xd;
    double *yd;
    int nd;
    int mr;
    int nr;
    double *dr;
    int *imr;
    int *inr;
    int ierr;
    int i,numresult;
    FILE *fp;

    nd = 10;
    mr = 3;
    nr = 8;
    numresult = (1<<mr) - 1 + nr;
    fp = fopen( "dfwthr.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

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

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

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

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

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

    printf( "\tnd = %6d mr = %6d nr = %6d\n\n", nd, mr, nr );

    printf( "\t    i      xd    yd\n\n" );
    for( i=0 ; i<nd ; i++ )
    {
        fscanf( fp, "%lf %lf", &xd[i], &yd[i] );
        printf( "\t%6d %8.3g %8.3g\n", i, xd[i], yd[i] );
    }

    fclose( fp );

    ierr = ASL_dfwthr(xd, yd, nd, mr, nr, dr, imr, inr);

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

    printf( "\t    dr      imr    inr\n\n" );
    for( i=0 ; i<numresult ; i++ )
    {
        printf( "\t%8.3g %6d %6d\n", dr[i+1], imr[i], inr[i] );
    }

    free( xd );
    free( yd );
    free( dr );
    free( imr );
    free( inr );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwthr ***

 ** Input **

nd =     10 mr =      3 nr =      8

    i      xd    yd

     0      0.1     1.28
     1      0.2     1.33
     2      0.3     1.33
     3      0.4     1.28
     4      0.5      0.5
     5      0.6   -0.278
     6      0.7   -0.334
     7      0.8   -0.334
     8      0.9   -0.278
     9        1      0.5

 ** Output **

ierr =      0

    dr      imr    inr
```

```
   0.638      0      1
   0.128      1      1
 -0.0621      1      2
-0.00585      2      1
   0.167      2      2
  0.0117      2      3
 -0.0908      2      4
-0.00827      3      1
4.14e-17      3      2
 0.00414      3      3
   0.029      3      4
 0.00207      3      5
       0      3      6
-0.00621      3      7
  -0.116      3      8
```

2.18.3 ASL_dfwths, ASL_rfwths
Inverse Wavelet Transform According to Haar Functions

(1) **Function**

When the spacing at which the input data $\{(x_i, f(x_i))\}$ is sampled is not equally spaced or when the number of sampled data is not 2^k (where k is a natural number), the ASL_dfwths or ASL_rfwths computes the approximation of $f(x)$ according to the following inverse Wavelet transform for the Wavelet transform C_{mn} according to Haar functions.

$$\sum_{m,n} C_{mn} H_{mn}(x)$$

(2) **Usage**

Double precision:

ierr = ASL_dfwths (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr);

Single precision:

ierr = ASL_rfwths (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr);

(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}	1	Input	Difference of maximum and minimum values of input data x-coordinates. Data is regenerated on interval $[0, a]$.
2	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	Input	Wavelet transform C_{mn}. (See Note (a))
3	mr	I	1	Input	Maximum value of m of Haar functions $H_{mn}(x)$ that were used for Wavelet transform.
4	nr	I	1	Input	Maximum value of n of Haar functions $H_{mn}(x)$ that were used for Wavelet transform.
5	mr2	I	1	Input	Maximum value of index m of Haar functions $H_{mn}(x)$ to be used for inverse Wavelet transform.
6	nr2	I	1	Input	Maximum value of index n of Haar functions $H_{mn}(x)$ to be used for inverse Wavelet transform.
7	imr	I*	2^{mr}+nr−1	Input	Index m information of Wavelet transform C_{mn}. (See Note (a))
8	inr	I*	2^{mr}+nr−1	Input	Index n information of Wavelet transform C_{mn}. (See Note (a))
9	fr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	Output	Values of $f(x)$ that were regenerated from Wavelet transform C_{mn}. (See Note (b))
10	xr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	Output	Values of x-coordinates of $f(x)$ that were regenerated from Wavelet transform C_{mn}. (See Note (b))
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{mr} \geq 0$

(b) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(c) $\mathrm{mr2} \leq \mathrm{mr}$

(d) $\mathrm{nr2} \leq 2^{\mathrm{mr2}}$

(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) The Wavelet transform C_{mn} values are stored in $\mathrm{dr}[i](i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$. The corresponding index m values are stored in $\mathrm{imr}[i-1]$ and the corresponding index n values are stored in $\mathrm{inr}[i-1]$.

(b) The values stored in fr are approximations of the original data, and the corresponding x-coordinate values, which are stored in xr, are equally spaced.

(7) **Example**

(a) Problem
Compute the inverse Wavelet transform up to $m = 4$ and $n = 2$ using the Wavelet transform computed in the Example of 2.18.2 $\left\{\begin{matrix}\mathrm{ASL\_dfwthr}\\ \mathrm{ASL\_rfwthr}\end{matrix}\right\}$ as the input data.

(b) Input data

$$\text{Wavelet transform } \{C_{mn}\}, \mathrm{a} = 1, \mathrm{mr} = 3, \mathrm{nr} = 8, \mathrm{mr2} = 3 \mathrm{and} \mathrm{nr2} = 8.$$

(c) Main program

```
/*      C interface example for ASL_dfwths */

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

int main()
{
    double a;
    double *dr;
    int mr;
    int nr;
    int mr2;
    int nr2;
    int *imr;
    int *inr;
    double *fr;
    double *xr;
    int ierr;
    int i,numdata,numresult;
    FILE *fp;

    mr = 3;
    nr = 8;
    mr2 = 3;
    nr2 = 8;
    numdata = (1<<mr) - 1 + nr;
    numresult = 1<<(mr2+1);
    fp = fopen( "dfwths.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

    fscanf( fp, "%lf", &a );

    dr = ( double * )malloc((size_t)( sizeof(double) * (numdata+1) ));
    if( dr == NULL )
    {
```

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

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

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

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

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

    printf( "\ta   = %8.3g\n", a  );
    printf( "\tmr  = %6d nr  = %6d\n", mr, nr  );
    printf( "\tmr2 = %6d nr2 = %6d\n\n", mr2, nr2  );

    dr[0] = 0.0;
    printf( "\tdr[0]   = %8.3g\n\n", dr[0]  );

    printf( "\t     dr     imr     inr\n\n" );
    for( i=0 ; i<numdata ; i++ )
    {
        fscanf( fp, "%lf %d %d", &dr[i+1], &imr[i], &inr[i] );
        printf( "\t%8.3g    %6d  %6d\n", dr[i+1], imr[i], inr[i] );
    }

    fclose( fp );

    ierr = ASL_dfwths(a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr);

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

    printf( "\t     xr      fr\n\n" );
    for( i=0 ; i<numresult ; i++ )
    {
        printf( "\t%8.3g %8.3g\n", xr[i], fr[i] );
    }

    free( dr );
    free( imr );
    free( inr );
    free( fr );
    free( xr );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwths ***

 ** Input **

a   =        1
mr  =      3 nr  =      8
mr2 =      3 nr2 =      8

dr[0]   =        0

     dr     imr     inr

   0.638         0       1
   0.128         1       1
 -0.0621         1       2
-0.00585         2       1
   0.167         2       2
  0.0117         2       3
 -0.0908         2       4
-0.00827         3       1
4.14e-17         3       2
 0.00414         3       3
   0.029         3       4
```

```
 0.00207          3        5
       0          3        6
-0.00621          3        7
  -0.116          3        8

 ** Output **

ierr =       0

      xr       fr

  0.0313    0.785
  0.0938    0.832
   0.156    0.832
   0.219    0.832
   0.281    0.802
   0.344    0.779
   0.406    0.205
   0.469    0.041
   0.531   -0.697
   0.594   -0.709
   0.656    -0.75
   0.719    -0.75
   0.781    -0.75
   0.844   -0.715
   0.906   -0.697
   0.969   -0.041
```

2.18.4 ASL_dfwth2, ASL_rfwth2
Haar Function Generation (Equally Spaced Sampling Data)

(1) **Function**

When the spacing with which the data to be subject to the Wavelet transform was sampled is fixed and there are $N = 2^k$ sampled data (where k is a natural number), the ASL_dfwth2 or ASL_rfwth2 function generates the following Haar function, which is required for a one-dimensional Wavelet transform, on the interval $[0, 1]$.

$$H_{mn}(x) = \begin{cases} \sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1) \le x \le \frac{a}{2^m}(n-1/2) \\ -\sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1/2) < x \le \frac{a}{2^m}n \\ 0 & \text{Otherwise} \end{cases}$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwth2  (na, m, n, &c, lr);
```

Single precision:

```
ierr = ASL_rfwth2  (na, m, n, &c, lr);
```

(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	na	I	1	Input	Number of input data N.
2	m	I	1	Input	Generation m of Haar function $H_{mn}(x)$.
3	n	I	1	Input	Index n of Haar function $H_{mn}(x)$. (See Note (a))
4	c	{ D∗ / R∗ }	1	Output	Value of Haar function $H_{mn}(x)$.
5	lr	I*	na	Output	Sign of value of Haar function $H_{mn}(x)$. When the Haar function is positive, 1 is stored as the value. When the Haar function is negative, -1 is stored as the value. When the Haar function is zero, 0 is stored as the value.
6	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) na $= 2^k$ (k must be a natural number less than or equal to 20.)

(b) $0 \le$ m $\le k$

(c) n $\le 2^{\mathrm{m}}$

(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.	

(6) **Notes**

(a) This function does not assign a constant term. The fixed value 1 is required on the interval $[0, 1]$ in addition to the values assigned by this function for the Haar function used as the base.

(b) If the data to be transformed is continuous or the sampled intervals are irregular, the function 2.18.1 $\left\{\begin{matrix}\text{ASL\_dfwth1}\\ \text{ASL\_rfwth1}\end{matrix}\right\}$ must be used to generate the Haar functions.

(7) **Example**

(a) Problem
For 16 x-coordinates that are equally spaced in the interval $[0, 1]$, compute the Haar function $H_{34}(x)$.

(b) Input data

$$\text{na} = 16, \text{m} = 3 \text{and} \text{n} = 4.$$

(c) Main program

```
/*      C interface example for ASL_dfwth2 */

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

int main()
{
    int na;
    int m;
    int n;
    double c;
    int *lr;
    int ierr;
    int i;
    FILE *fp;

    na = 16;
    fp = fopen( "dfwth2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dfwth2 ***\n" );
    printf( "\n     ** Input **\n\n" );

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

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

    printf( "\tna = %6d m = %6d n = %6d\n", na, m, n );

    fclose( fp );

    ierr = ASL_dfwth2(na, m, n, &c, lr);

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

```
    printf( "\t   c = %8.3g\n\n", c );

    printf( "\t     i      lr\n\n" );
    for( i=0 ; i<na ; i++ )
    {
        printf( "\t%6d %6d\n", i, lr[i] );
    }

    free( lr );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwth2 ***

 ** Input **

na =     16 m =      3 n =      8

 ** Output **

ierr =      0

   c =     2.83

     i      lr

     0      0
     1      0
     2      0
     3      0
     4      0
     5      0
     6      0
     7      0
     8      0
     9      0
    10      0
    11      0
    12      0
    13      0
    14      1
    15     -1
```

2.18.5 ASL_dfwtht, ASL_rfwtht
Wavelet Transform According to Haar Functions (Equally Spaced Sampling Data)

(1) **Function**

When the spacing at which the input data $\{(x_i, f(x_i))\}$ is sampled is equally spaced and the number of sampled data is 2^k (where k is a natural number), the ASL_dfwtht or ASL_rfwtht computes the following Wavelet transform according to Haar functions.

$$C_{mn} = \int_0^a f(x) H_{mn}(x) dx$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwtht  (xd, yd, nd, mr, nr, &a, dr, imr, inr, iwk);
```

Single precision:

```
ierr = ASL_rfwtht  (xd, yd, nd, mr, nr, &a, dr, imr, inr, iwk);
```

(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	xd	{D∗ / R∗}	nd	Input	Set of input data x-coordinates $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	Input	Set of input data function values $\{f(x_i)\}$.
3	nd	I	1	Input	Number of input data N.
4	mr	I	1	Input	Maximum value of index m of Haar functions H_{mn} used for Wavelet transform.
5	nr	I	1	Input	Maximum value of index n of Haar functions H_{mn} used for Wavelet transform.
6	a	{D∗ / R∗}	1	Output	Upper bound a of integration interval $[0, a]$ used for Wavelet transform.
7	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	Output	Wavelet transform C_{mn}. (See Notes (a) and (b))
8	imr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Output	Index m information of Wavelet transform C_{mn}, which is stored in dr. (See Note (b))
9	inr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Output	Index n information of Wavelet transform C_{mn}, which is stored in dr. (See Note (b))
10	iwk	I*	nd	Work	Work area
11	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{nd} = 2^k$ (k must be a natural number less than or equal to 20.)

(b) $0 \leq \mathrm{mr} \leq k - 1$

(c) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(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.	

(6) **Notes**

(a) The mean value of the input data $\{f(x_i)\}$ is stored in dr[0].

(b) The Wavelet transform C_{mn} values are stored in dr[i]($i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1$). The corresponding index m values are stored in imr[$i - 1$] and the corresponding index n values are stored in inr[$i - 1$].

(7) **Example**

(a) Problem

Compute the Wavelet transform up to $m = 4$ and $n = 2$ using the input data which is obtained by the equally spaced sampling for the function

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}.$$

(b) Input data

Input data $\{(x_i, f(x_i))\}, \mathrm{nd} = 16, \mathrm{mr} = 3 \mathrm{and} \mathrm{nr} = 8.$

(c) Main program

```
/*      C interface example for ASL_dhwtht */

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

int main()
{
    double *xd;
    double *yd;
    int nd;
    int mr;
    int nr;
    double a;
    double *dr;
    int *imr;
    int *inr;
    int *iwk;
    int ierr;
    int i,numresult;
    FILE *fp;

    nd = 16;
    mr = 3;
    nr = 8;
    numresult = (1<<mr) - 1 + nr;
    fp = fopen( "dfwtht.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

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

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

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

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

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

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

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

    printf( "\tnd = %6d mr = %6d nr = %6d\n\n", nd, mr, nr );

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

    fclose( fp );

    ierr = ASL_dfwtht(xd, yd, nd, mr, nr, &a, dr, imr, inr, iwk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\t   a = %8.3g\n\n", a );

    printf( "\t      dr     imr    inr\n\n" );
    for( i=0 ; i<numresult ; i++ )
    {
        printf( "\t%8.3g %6d %6d\n", dr[i+1], imr[i], inr[i] );
    }

    free( xd );
    free( yd );
    free( dr );
    free( imr );
    free( inr );
    free( iwk );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwtht ***

 ** Input **

nd =     16 mr =      3 nr =      8

     xd       yd

  0.0625     1.07
   0.125     1.35
   0.188     1.35
    0.25      1.3
   0.313     1.35
```

```
   0.375     1.35
   0.438     1.07
     0.5      0.5
   0.563  -0.0675
   0.625   -0.349
   0.688   -0.347
    0.75     -0.3
   0.813        0
   0.875        0
   0.938        0
       1      0.5

 ** Output **

ierr =      0

   a =         1

      dr     imr    inr

   0.618       0      1
  0.0707       1      1
  -0.138       1      2
 -0.0289       2      1
   0.141       2      2
  0.0289       2      3
 -0.0625       2      4
 -0.0497       3      1
 0.00837       3      2
-0.00021       3      3
     0.1       3      4
  0.0497       3      5
-0.00837       3      6
       0       3      7
 -0.0884       3      8
```

2.18.6 ASL_dfwthi, ASL_rfwthi
Inverse Wavelet Transform According to Haar Functions (Equally Spaced Sampling Data)

(1) **Function**

When the spacing at which the input data $\{(x_i, f(x_i))\}$ is sampled is equally spaced and the number of sampled data is 2^k (where k is a natural number), the ASL_dfwthi or ASL_rfwthi computes $f(x)$ according to the following inverse Wavelet transform for the Wavelet transform C_{mn} according to Haar functions.

$$\sum_{m,n} C_{mn} H_{mn}(x)$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwthi  (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr, iwk);
```

Single precision:

```
ierr = ASL_rfwthi  (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr, iwk);
```

(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}	1	Input	Value obtained when data sampling spacing is added to difference of maximum and minimum values of input data x-coordinates. Data is regenerated on interval $[0, \mathrm{a}]$.
2	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	Input	Wavelet transform C_{mn}. (See Note (a))
3	mr	I	1	Input	Maximum value of m of Haar functions H_{mn} that were used for Wavelet transform
4	nr	I	1	Input	Maximum value of n of Haar functions H_{mn} that were used for Wavelet transform
5	mr2	I	1	Input	Maximum value of index m of Haar functions $H_{mn}(x)$ to be used for inverse Wavelet transform.
6	nr2	I	1	Input	Maximum value of index n of Haar functions $H_{mn}(x)$ to be used for inverse Wavelet transform.
7	imr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Input	Index m information of Wavelet transform C_{mn}. (See Note (a))
8	inr	I*	$2^{\mathrm{mr}}+\mathrm{nr}-1$	Input	Index n information of Wavelet transform C_{mn}. (See Note (a))
9	fr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	Output	Values of $f(x)$ that were regenerated from Wavelet transform C_{mn}. (See Note (b))
10	xr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	Output	Values of x-coordinates of $f(x)$ that were regenerated from Wavelet transform C_{mn}. (See Note (b))
11	iwk	I*	$2^{\mathrm{mr2}+1}$	Work	Work area.
12	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) $\mathrm{mr} \geq 0$

(b) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(c) $\mathrm{mr2} \leq \mathrm{mr}$

(d) $\mathrm{nr2} \leq 2^{\mathrm{mr2}}$

(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) The Wavelet transform C_{mn} values are stored in $\mathrm{dr}[i](i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$. The corresponding index m values are stored in $\mathrm{imr}[i-1]$ and the corresponding index n values are stored in $\mathrm{inr}[i-1]$.

(b) The values stored in fr are approximations of the original data, and the corresponding x-coordinate values, which are stored in xr, are equally spaced.

(7) **Example**

(a) Problem
Compute the inverse Wavelet transform up to $m = 4$ and $n = 2$ using the Wavelet transform computed in the Example of 2.18.5 $\left\{\begin{matrix}\mathrm{ASL\_dfwtht}\\ \mathrm{ASL\_rfwtht}\end{matrix}\right\}$ as the input data.

(b) Input data

Wavelet transform $\{C_{mn}\}, \mathrm{a} = 1, \mathrm{mr} = 3, \mathrm{nr} = 8, \mathrm{mr2} = 3\mathrm{andnr2} = 8.$

(c) Main program

```
/*      C interface example for ASL_dfwthi */

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

int main()
{
    double a;
    double *dr;
    int mr;
    int nr;
    int mr2;
    int nr2;
    int *imr;
    int *inr;
    double *fr;
    double *xr;
    int *iwk;
    int ierr;
    int i,numdata,numresult;
    FILE *fp;

    mr = 3;
    nr = 8;
    mr2 = 3;
    nr2 = 8;
    numdata = (1<<mr)-1+nr;
    numresult = 1<<(mr2+1);
    fp = fopen( "dfwthi.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

    fscanf( fp, "%lf", &a );

    dr = ( double * )malloc((size_t)( sizeof(double) * (numdata + 1 )));
    if( dr == NULL )
```

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

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

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

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

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

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

    printf( "\ta = %8.3g\n", a );
    printf( "\tmr  = %6d nr  = %6d\n", mr, nr );
    printf( "\tmr2 = %6d nr2 = %6d\n\n", mr2, nr2 );

    dr[0] = 0.0;
    printf( "\tdr[0] = %8.3g\n\n", dr[0] );

    printf( "\t    dr       imr   inr\n\n" );
    for( i=0 ; i<numdata ; i++ )
    {
        fscanf( fp, "%lf %d %d", &dr[i+1], &imr[i], &inr[i] );
        printf( "\t%8.3g %6d %6d\n", dr[i+1], imr[i], inr[i] );
    }

    fclose( fp );

    ierr = ASL_dfwthi(a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr, iwk);

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

    printf( "\t   xr         fr\n\n" );
    for( i=0 ; i<numresult ; i++ )
    {
        printf( "\t%8.3g %8.3g\n", xr[i], fr[i] );
    }

    free( dr );
    free( imr );
    free( inr );
    free( fr );
    free( xr );
    free( iwk );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwthi ***

 ** Input **

a =        1
mr  =      3 nr  =      8
mr2 =      3 nr2 =      8

dr[0] =        0

    dr       imr   inr

    58.6       0      1
  0.0707       1      1
```

```
    81.9      1      2
-0.0289       2      1
   0.141      2      2
  0.0289      2      3
   -38.2      2      4
 -0.0497      3      1
 0.00837      3      2
-0.00021      3      3
     0.1      3      4
  0.0497      3      5
-0.00837      3      6
   -1.22      3      7
   -55.2      3      8

 ** Output **

ierr =      0

   xr          fr

  0.0313     58.6
  0.0938     58.8
   0.156     58.8
   0.219     58.8
   0.281     58.8
   0.344     58.8
   0.406     58.6
   0.469       58
   0.531     57.4
   0.594     57.1
   0.656     57.1
   0.719     57.2
   0.781     -254
   0.844     -247
   0.906     -254
   0.969       58
```

2.18.7 ASL_dfwtmf, ASL_rfwtmf Mexican Hut Function Computation

(1) **Function**

The ASL_dfwtmf or ASL_rfwtmf uses the following Mexican hat function to compute the Wavelet transform base shown below.

$$\varphi_{MH}(x) = (1 - 2x^2)e^{-x^2}$$

$$\phi_{MH}(x; a, b) = \frac{1}{\sqrt{C}} \left(\frac{x - b}{a} \right)$$

Here,

$$C = a \left(1 - \frac{a^2}{2} + \frac{3a^4}{4} \right) \sqrt{\frac{\pi}{2}}$$

is the normalization factor.

(2) **Usage**

Double precision:

```
ierr = ASL_dfwtmf  (a, b, x, &v);
```

Single precision:

```
ierr = ASL_rfwtmf  (a, b, x, &v);
```

(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}	1	Input	Frequency parameter a of base $\phi_{MH}(x; a, b)$ for Wavelet transform according to Mexican hat function.
2	b	{D / R}	1	Input	Shift parameter b of base $\phi_{MH}(x; a, b)$ for Wavelet transform according to Mexican hat function.
3	x	{D / R}	1	Input	Variable value x.
4	v	{D∗ / R∗}	1	Output	Value of base $\phi_{MH}(x; a, b)$ for Wavelet transform according to Mexican hat function
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) a > 0

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

(7) **Example**

(a) Problem

Compute the base $\phi_{MH}(4; 2, 3)$ for a Wavelet transform according to a Mexican hat function.

(b) Input data

$a = 4.0, b = 2.0 and x = 3.0$.

(c) Main program

```
/*      C interface example for ASL_dfwtmf */

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

int main()
{
    double a;
    double b;
    double x;
    double v;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &x );

    printf( "\ta = %8.3g b = %8.3g c = %8.3g\n", a, b, x );

    fclose( fp );

    ierr = ASL_dfwtmf(a, b, x, &v);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\t   v = %8.3g\n", v );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwtmf ***

 ** Input **

a =        4 b =        2 c =        3

 ** Output **

ierr =      0

   v =     28.6
```

2.18.8 ASL_dfwtmt, ASL_rfwtmt
Wavelet Transform According to Mexican Hut Functions

(1) **Function**

For the set $\{(x_i, f(x_i))\}$ $(i = 1, 2, \cdots, n)$ of n x-coordinates and function values $f(x)$ that were given as input data, the ASL_dfwtmt or ASL_rfwtmt computes the following Wavelet transform according to a Mexican hat function.

$$(W_{\phi_{MH}} f)(b, a) = \int_{-\infty}^{\infty} \phi_{MH}(x; a, b) f(x) dx$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwtmt  (xd, yd, nd, a, b, &c);
```

Single precision:

```
ierr = ASL_rfwtmt  (xd, yd, nd, a, b, &c);
```

(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	xd	{D∗ / R∗}	nd	Input	Set of input data x-coordinates $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	Input	Set of input data function values $\{f(x_i)\}$.
3	nd	{D / R}	1	Input	Number of input data n.
4	a	{D / R}	1	Input	Wavelet transform frequency parameter a
5	b	{D / R}	1	Input	Wavelet transform shift parameter b
6	c	{D∗ / R∗}	1	Output	Wavelet transform value $(W_{\phi_{MH}} f)(b, a)$.
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) a > 0

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

(7) **Example**

(a) Problem

Compute the Wavelet transform according to a Mexican hat function $(W_{\phi_{MH}} f)(2,3)$ using the input data which is obtained by the equally spaced sampling for the function

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}.$$

(b) Input data

Input data $\{(x_i, f(x_i))\}, \mathrm{a} = 3 \mathrm{and} \mathrm{b} = 2.$

(c) Main program

```
/*      C interface example for ASL_dfwtmt */

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

int main()
{
    double *xd;
    double *yd;
    int nd;
    double a;
    double b;
    double c;
    int ierr;
    int i;
    FILE *fp;

    nd = 10;
    fp = fopen( "dfwtmt.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );

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

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

    printf( "\tnd = %6d a = %8.3g b = %8.3g\n", nd, a, b );

    printf( "\t  xd    yd\n" );
    for( i=0 ; i<nd ; i++ )
    {
        fscanf( fp, "%lf %lf", &xd[i], &yd[i] );
        printf( "%8.3g %8.3g\n", xd[i], yd[i] );
    }
```

```
    fclose( fp );

    ierr = ASL_dfwtmt(xd, yd, nd, a, b, &c);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\t   c = %8.3g\n", c );

    free( xd );
    free( yd );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwtmt ***

 ** Input **

nd =     10 a =        3 b =        2
  xd    yd
 0.1     1.28
 0.2     1.33
 0.3     1.33
 0.4     1.28
 0.5      0.5
 0.6   -0.278
 0.7   -0.334
 0.8   -0.334
 0.9   -0.278
   1      0.5

 ** Output **

ierr =      0

   c =     4.21
```

2.18.9 ASL_dfwtff, ASL_rfwtff
French Hut Function Computation

(1) **Function**

The ASL_dfwtff or ASL_rfwtff uses the following French hat function to compute the Wavelet transform base shown below.

$$\varphi_{FH}(x) = \begin{cases} 1 & -1 \leq x < 1 \\ -\frac{1}{2} & -3 \leq x < -1 \text{ or } 1 \leq x < 3 \\ 0 & \text{Otherwise} \end{cases}$$

$$\phi_{FH}(x; a, b) = \frac{1}{\sqrt{3a}} \varphi\left(\frac{x-b}{a}\right)$$

(2) **Usage**

Double precision:

ierr = ASL_dfwtff (a, b, x, &v);

Single precision:

ierr = ASL_rfwtff (a, b, x, &v);

(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}	1	Input	Frequency parameter a of base $\phi_{FH}(x; a, b)$ for Wavelet transform according to French hat function.
2	b	{D / R}	1	Input	Shift parameter b of base $\phi_{FH}(x; a, b)$ for Wavelet transform according to French hat function.
3	x	{D / R}	1	Input	Variable value x
4	v	{D∗ / R∗}	1	Output	Value of base $\phi_{FH}(x; a, b)$ for Wavelet transform according to French hat function.
5	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) a > 0

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

(7) **Example**

(a) Problem

Compute the base $\phi_{FH}(1.5; 2, 1)$ for a Wavelet transform according to a French hat function.

(b) Input data

$$\mathrm{x} = 1.5, \mathrm{a} = 2 \mathrm{and} \mathrm{b} = 1.$$

(c) Main program

```
/*      C interface example for ASL_dfwtff */

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

int main()
{
    double a;
    double b;
    double x;
    double v;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );
    fscanf( fp, "%lf", &x );

    printf( "\ta = %8.3g b = %8.3g x = %8.3g\n", a, b, x );

    fclose( fp );

    ierr = ASL_dfwtff(a, b, x, &v);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\t   v = %8.3g\n", v );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwtff ***

 ** Input **

a =        2 b =        1 x =      1.5

 ** Output **

ierr =      0

   v =        0
```

2.18.10 ASL_dfwtft, ASL_rfwtft
Wavelet Transform According to French Hut Function

(1) **Function**

For the set $\{(x_i, f(x_i))\}$ $(i = 1, 2, \cdots, n)$ of n x-coordinates and function values $f(x)$ that were given as input data, the ASL_dfwtft or ASL_rfwtft computes the following Wavelet transform according to a French hat function.

$$(W_{\phi_{FH}} f)(b, a) = \int_{-\infty}^{\infty} \phi_{FH}(x; a, b) f(x) dx$$

(2) **Usage**

Double precision:

```
ierr = ASL_dfwtft  (xd, yd, nd, a, b, &c);
```

Single precision:

```
ierr = ASL_rfwtft  (xd, yd, nd, a, b, &c);
```

(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	xd	{D∗ / R∗}	nd	Input	Set of input data x-coordinates $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	Input	Set of input data function values $\{f(x_i)\}$.
3	nd	{D / R}	1	Input	Number of input data n
4	a	{D / R}	1	Input	Wavelet transform frequency parameter a
5	b	{D / R}	1	Input	Wavelet transform shift parameter a
6	c	{D∗ / R∗}	1	Output	Wavelet transform value $(W_{\phi_{FH}} f)(b, a)$.
7	ierr	I	1	Output	Error indicator (Return Value)

(4) **Restrictions**

(a) a > 0

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

ierr value	Meaning	Processing
0	Normal termination.	
3000	Restriction (a) was not satisfied.	Processing is aborted.

(6) **Notes**

None

(7) **Example**

(a) Problem

Compute the Wavelet transform according to a French hat function $(W_{\phi_{FH}} f)(2,3)$ using the input data which is obtained by the equally spaced sampling for the function

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}.$$

(b) Input data

Input data $\{(x_i, f(x_i))\}$, a = 2andb = 1.

(c) Main program

```
/*      C interface example for ASL_dfwtft */

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

int main()
{
    double *xd;
    double *yd;
    int nd;
    double a;
    double b;
    double c;
    int ierr;
    int i;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nd );
    fscanf( fp, "%lf", &a );
    fscanf( fp, "%lf", &b );

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

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

    printf( "\t   nd = %6d\n", nd );
    printf( "\t   a  = %8.3g\n", a );
    printf( "\t   b  = %8.3g\n\n", b );

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

    fclose( fp );

    ierr = ASL_dfwtft(xd, yd, nd, a, b, &c);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\t   c = %8.3g\n\n", c );
```

```
    free( xd );
    free( yd );

    return 0;
}
```

(d) Output results

```
*** ASL_dfwtft ***

 ** Input **

   nd =     10
   a  =      2
   b  =      1

      xd       yd

     0.1     1.28
     0.2     1.33
     0.3     1.33
     0.4     1.28
     0.5      0.5
     0.6   -0.278
     0.7   -0.334
     0.8   -0.334
     0.9   -0.278
       1      0.5

 ** Output **

ierr =      0

   c =  -0.0125
```

Appendix A

MACHINE CONSTANTS USED IN ASL C INTERFACE

A.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 A−1 Units for Determining Error

Single-precision	Double-precision
$2^{-23}(\simeq 1.19 \times 10^{-7})$	$2^{-52}(\simeq 2.22 \times 10^{-16})$

Remark: The unit for determining error ε, which is also called the machine ε, is usually defined as the smallest positive constant for which the calculation result of $1+\varepsilon$ differs from 1 in the corresponding floating point mode. Therefore, seeing the unit for determining error enables you to know the maximum number of significant digits of an operation (on the mantissa) in that floating point mode.

A.2 Maximum and Minimum Values of Floating Point Data

The table below shows maximum and minimum values of floating point data defined within ASL 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 A−2 Maximum and Minimum Values of Floating Point Data

	Single-precision	Double-precision
Maximum value	$2^{127}(2-2^{-23})$ $(\simeq 3.40 \times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80 \times 10^{308})$
Positive minimum value	2^{-126} $(\simeq 1.17 \times 10^{-38})$	2^{-1022} $(\simeq 2.23 \times 10^{-308})$
Negative maximum value	-2^{-126} $(\simeq -1.17 \times 10^{-38})$	-2^{-1022} $(\simeq -2.23 \times 10^{-308})$
Minimum value	$-2^{127}(2-2^{-23})$ $(\simeq -3.40 \times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80 \times 10^{308})$

Index

(*) SMP Functions = Shared Memory Parallel Processing Functions