科学技術計算ライブラリ
ASL C言語インタフェース
ユーザーズガイド
＜基本機能編 第3分冊＞

はしがき

本書は, 科学技術計算ライブラリ ASL (Advanced Scientific Library) C 言語インタフェースの概念, 機能, 利用方法などについて説明したものです.
当製品に対応する説明書は 7 分冊からなっており, 構成は次のとおりです. このうち本書は, 基本機能第 3 分冊について記述したものです.

基本機能 第 1 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	格納モードの変換	配列データの格納モードの変換に関する関数のアルゴリズム, 使用方法および使用例の説明
3	基本行列演算	行列の基本演算に関する関数のアルゴリズム, 使用方法および使用例の説明
4	固有値・固有ベクトル	実行列, 複素行列, 実対称行列, エルミート行列, 実対称バンド行列, 実対称 3 重対角行列, 実対称スパース行列, エルミートスパース行列の標準固有値問題および実行列, 実対称行列, エルミート行列, 実対称バンド行列の一般化固有値問題に関する関数のアルゴリズム, 使用方法および使用例の説明

基本機能 第 2 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	連立 1 次方程式 (直接法)	実行列, 複素行列, 正値対称行列, 実対称行列, エルミート行列, 実バンド行列, 正値対称バンド行列, 実 3 重対角行列, 実上三角行列, 実下三角行列の連立 1 次方程式に関する関数のアルゴリズム, 使用方法および使用例の説明

基本機能 第 3 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	フーリエ変換とその応用	1 次元, 2 次元および 3 次元の複素ならびに実フーリエ変換, 1 次元, 2 次元および 3 次元の畳み込み, 相関, パワー・スペクトル解析, ウェーブレット変換およびラプラス逆変換に関する関数のアルゴリズム, 使用方法および使用例の説明

基本機能 第 4 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	微分方程式とその応用	〔常微分方程式初期値問題〕 連立高階, 陰的連立, 行列型, スティフ問題の連立高階, 連立 1 階, 高階常微分方程式 〔常微分方程式境界値問題〕 連立高階, 連立 1 階, 高階, 線形高階, 線形 2 階常微分方程式 〔積分方程式〕 第 2 種フレドホルム型, 第 1 種ボルテラ型積分方程式 〔偏微分方程式〕 2 次元および 3 次元の非同次ヘルムホルツ方程式 に関する関数のアルゴリズム, 使用方法および使用例の説明
3	数値微分	1 変数関数および多変数関数の数値微分に関する関数のアルゴリズム, 使用方法および使用例の説明
4	数値積分	有限区間, 半無限区間, 全無限区間, 2 次元有限区間, 多次元有限区間の数値積分に関する関数のアルゴリズム, 使用方法および使用例の説明
5	補間・近似	補間, 曲面補間, 最小二乗近似, 最小二乗曲面近似, チェビシェフ近似に関する関数のアルゴリズム, 使用方法および使用例の説明
6	スプライン関数	3 次スプライン, 双 3 次スプラインおよび B–スプラインを用いた補間, 平滑化, 数値微分, 数値積分に関する関数のアルゴリズム, 使用方法および使用例の説明

基本機能 第 5 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	特殊関数	ベッセル関数, 変形ベッセル関数, 球ベッセル関数, ベッセル関数に関連した関数, ガンマ関数, ガンマ関数に関連した関数, 楕円関数, 初等関数の不定積分, ルジャンドル陪関数, 直交多項式, その他の特殊関数に関する関数のアルゴリズム, 使用方法および使用例の説明
3	ソート・順位付け	ソート, 順位付けに関する関数の使用方法および使用例の説明
4	方程式の根	代数方程式, 非線形方程式, 連立非線形方程式の根に関する関数のアルゴリズム, 使用方法および使用例の説明
5	極値問題・最適化	制約なし関数の極小化, 制約なし関数二乗和の極小化, 制約付き 1 変数関数の極小化, 制約付き多変数関数の最小化, 最短路問題に関する関数のアルゴリズム, 使用方法および使用例の説明

基本機能 第 6 分冊

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	乱数の検定	一様乱数の検定, 分布乱数の検定に関する関数の使用方法および使用例の説明
3	確率分布	連続分布, 離散分布に関する関数の使用方法および使用例の説明
4	基礎統計量	基礎統計量, 分散共分散, 相関係数に関する関数の使用方法および使用例の説明
5	推定と検定	区間推定, 検定に関する関数の使用方法および使用例の説明
6	分散分析・実験計画	1 元配置, 2 元配置, 多元配置, 乱塊法, グレコ・ラテン方格法, 累積法に関する関数の使用方法および使用例の説明
7	ノンパラメトリック検定	χ^2 分布による検定, その他分布による検定に関する関数の使用方法および使用例の説明
8	多変量解析	主成分分析, 因子分析, 正準相関分析, 判別分析, クラスタ分析に関する関数の使用方法および使用例の説明
9	時系列分析	自己相関・相互相関, 自己共分散・相互共分散, 平滑化・需要予測に関する関数の使用方法および使用例の説明
10	回帰分析	線形回帰, 非線形回帰に関する関数の使用方法および使用例の説明

共有メモリ並列機能

章	タイトル	内　　容
1	使用の手引き	本説明書の構成, 各項目の見方, および使用上の制限事項などの説明
2	基本行列演算	実行列および複素行列の積を求める関数のアルゴリズム, 使用方法の説明
3	連立 1 次方程式 (直接法)	実行列, 複素行列, 実対称行列, エルミート行列の連立 1 次方程式（直接法）に関する関数のアルゴリズム, 使用方法および使用例の説明
4	連立 1 次方程式 (反復法)	実正値対称スパース行列, 実対称スパース行列, 実非対称スパース行列の連立 1 次方程式（反復法）に関する関数のアルゴリズム, 使用法および使用例の説明
5	固有値・固有ベクトル	実対称行列およびエルミート行列の固有値問題に関する関数のアルゴリズム, 使用方法および使用例の説明
6	フーリエ変換とその応用	1 次元, 2 次元および 3 次元の複素ならびに実フーリエ変換, 2 次元および 3 次元の畳み込み, 相関, パワー・スペクトル解析に関する関数のアルゴリズム, 使用方法および使用例の説明
7	ソート	ソートに関する関数の使用方法および使用例の説明

2023 年　3 月　ASL 付属説明書　3.0.0-230301

備考 (1) 本書に説明しているすべての機能は, プログラムプロダクトであり, ASL 1.1 に対応しています.

(2) 製品名などの固有名詞は, 各メーカーの登録商標または商標です.

(3) 本ライブラリは, 最新の数値計算技法を取り入れ, 開発されたものです. 従って, 最新の技術を維持する目的から, 改良または新しく追加された関数が, 既存の関数の機能を包含し, かつ, これまで以上の高速性能が得られる場合には, 既存の関数を削除することもあります.

目 次

第 1 章　使用の手引

1.1　概　説

1.1.1　科学技術計算ライブラリ ASL C 言語インタフェースの概要

科学技術計算ライブラリ ASL（Advanced Scientific Library）は, 数値解析プログラムの作成を強力に支援する数学ライブラリである. ASL では広範な数値解析分野で頻出するプログラムを提供しており, それらは VE(Vector Engine) 上で優れた実行速度と精度を実現するための高度な最適化が適用されている. 本マニュアルで説明する ASL C 言語インタフェースは, ASL を C および C++から利用するためのインタフェースライブラリである. ASL C 言語インタフェースを用いることによって, 難解な数値計算アルゴリズムの詳細に煩わされることなく高度な数値解析プログラムを作成することができ, 数値解析プログラム開発の生産性を大幅に改善することができる.

ASL C 言語インタフェースは, 基本機能, 共有メモリ並列機能で構成される. 機能分類と本マニュアルの分冊との対応を表 1–1 に示す.

表 1–1　ASL C 言語インタフェース の機能分類

機能分類	分冊
基本機能	第 1〜6 分冊
共有メモリ並列機能	第 7 分冊

1.1.2　ASL C 言語インタフェース の特長

ASL C 言語インタフェースの特長は, 次のとおりである.

(1) ハードウェア性能を十分発揮できるように設計しており, コンパイラの最適化機能を用いて作成した.

(2) 行列を扱う関数では, 行列の種類 (対称行列, エルミート行列など) に応じて最適に処理を行えるように, 専用の関数をそれぞれ提供している. 一般に, 専用の関数を用いて処理を行った方が, 処理性能を向上したり, 必要なメモリ容量を節約したりすることができる.

(3) 処理手順に従ってモジュール化を行い, コンポーネント関数ごとの信頼性向上に努めるとともに, システム全体の効率化, 信頼性向上を図った.

(4) 関数を利用した後の エラーインディケータ (戻り値) の番号が体系的に決めてあるので, エラー情報を把握しやすい.

1.2 ライブラリの種類

ASL C 言語インタフェース には, 32 ビット整数型ライブラリと 64 ビット整数型ライブラリがある. 32 ビット整数型ライブラリに含まれる関数の整数型の引数は, 32 ビット (4 バイト) 整数型である. 一方, 64 ビット整数型ライブラリに含まれる関数の整数型の引数は, 64 ビット (8 バイト) 整数型である. また, 関数の実数型の引数によって関数名が異なる. 関数名については, 1.4 を参照のこと.

表 1–2　ASL C 言語インタフェース で提供しているライブラリの種類

変数の大きさ (バイト)		引数の型宣言文	通称	ライブラリの種類
整数型	実数型			
4	8	int double	32 ビット整数型倍精度関数	32 ビット整数型ライブラリ (リンクオプション：-lasl_sequential)
4	4	int float	32 ビット整数型単精度関数	
8	8	long double	64 ビット整数型倍精度関数	64 ビット整数型ライブラリ (リンクオプション：-lasl_sequential_i64)
8	4	long float	64 ビット整数型単精度関数	

(注 1) 機能によっては, 4 種類全てをサポートしているとは限らない. その場合, 個別の説明の注意事項の欄に記述するので注意されたい.

(注 2) 64 ビット整数型ライブラリの関数を使用するプログラムをコンパイルする場合は, コンパイルオプション "`-DASL_LIB_INT64`" を指定しなければならない. (1.6 注意事項 (2) を参照のこと.)

1.3 マニュアルについて

ここでは本マニュアルの第 2 章以降の構成について述べる.
第 2 章以降は ASL で用いられる関数とその機能, 使用方法の説明を行う.

1.3.1 『概　要』

各章の第 1 節では, 概要として各関数の効果的な使用法, 採用した手法およびそのアルゴリズム, 注意事項などについて述べてある.

1.3.2 関数説明文の構成

各章の第 2 節では, 関数ごとに以下の順で説明している.

(1) 機能

(2) 使用法

(3) 引数と戻り値

(4) 制限条件

(5) エラーインディケータ (戻り値)

(6) 注意事項

(7) 使用例

各項目は次に述べる原則に従って記述されている.

1.3.3 各項目の内容

(1) **機 能**
この項目では, 関数の目的とする機能について簡単に述べてある.

(2) **使用法**
この項目では, 関数名とその引数の順序について記述してある.
引数の並べ方は, 原則として次のように決められている. なお, 引数がアドレス渡しの変数である場合には引数名の前に& を付加している.

ierr = 関数名 (入力引数, 入出力引数, 出力引数, isw, ワーク);

ここで, isw は処理の手順を指定するための入力引数であり, ierr は エラーインディケータ (戻り値) である. ただし, 入力引数と入出力引数の順序が逆の場合もある. さらに次の規則にしたがっている.

- 配列は重要度に応じてできるだけ左方によせる.
- 配列名に続けて配列の大きさをそえる. 同じ大きさをもつ配列が複数個あるときは, その最初の配列名に続けてその大きさを引数として与え, 2 番目以降の配列からは, その大きさは引数として与えない.

(3) **引　数**

(2) 項で記述された引数と戻り値について, 順番に説明されている. その形式は以下のように統一されている.

引数と戻り値	**型**	**大きさ**	**入出力**	**内　容**
(a)	(b)	**(c)**	**(d)**	**(e)**

(a) 引数と戻り値

引数と戻り値が記載されている.

(b) 型

引数と戻り値のデータの型を示す. 次の記号のいずれかに示されている.

I ：整数型

D ：倍精度実数型

R ：単精度実数型

Z ：倍精度複素数型

C ：単精度複素数型

整数型の引数には 64 ビット整数型と 32 ビット整数型とがある. 関数の整数型引数が 64 ビット整数型であるのか 32 ビット整数型であるのかは, その関数が 64 ビット整数型であるか 32 ビット整数型であるか, つまりライブラリの種類によって決められる (1.2 参照). ユーザプログラムにおいて引数の型を宣言する際は, 32 ビット整数型の引数は`int`, 64 ビット整数型の引数は`long` を用いて宣言する必要がある.

(c) 大きさ

指定された引数の必要な大きさを示す. 2 以上を指定した場合には, この関数を利用したプログラム側で, その必要な領域を確保しなければならない.

1　：変数であることを示す.

n　：要素が n 個の 1 次元配列であることを示す. この配列が指定された直後にその大きさを示す引数 n が定義される. ただし大きさ n が以前に定義された配列の大きさを規定している場合には省略される. このほかに数値のみにて指定する場合や, $3 \times n$ や $n + m$ のように, 積または和の形で表記する場合もある.

(d) 入出力

引数の内容説明が入力時であるか出力時であるかを示す.

i. 「入力」とだけある場合：

この関数を利用したプログラムに制御がもどったときに, 引数の入力時の情報は保存されている. 入力時の情報は特に断らない限り, 利用者が与えなければならない. なお, 引数が変数の場合には変数の値を渡す必要がある.

ii. 「出力」とだけある場合：

引数には, 関数内で計算された結果が出力される. 入力時には何も入れなくてよい. なお, 引数が変数の場合には変数のアドレスを渡す必要がある.

iii. 「入力」と「出力」の両方に説明がある場合：

関数に制御がわたる前と関数から制御がもどった後で, この引数の内容に変化がある場合である. 入力時の情報は特に断らない限り, 利用者が与えなければならない. なお, 引数が変数の場合には変数のアドレスを渡す必要がある.

iv. 「ワーク」とある場合：

関数内で演算を行うときに利用する領域であること示す. 関数を利用するプログラム側で, 指定された大きさの作業領域を確保しなければならない. なお, 次の計算に流用するために, 作業領域の内容を保存しておく必要がある場合がある.

(e) 内 容

入力時あるいは出力時に, 引数が保持している情報について説明される.

- 「引 数」の説明の例を次に示す.

例 実行列の LU 分解と条件数を求める関数 (ASL_dbgmlc, ASL_rbgmlc) の使用法は以下のとおりである.

倍精度関数:

```
ierr = ASL_dbgmlc  (a, lna, n, ipvt, & cond, w1);
```

単精度関数:

```
ierr = ASL_rbgmlc  (a, lna, n, ipvt, & cond, w1);
```

この場合の引数と戻り値の説明は次のようになる.

表 1−3 引数と戻り値の例

D:倍精度実数型 Z:倍精度複素数型
R:単精度実数型 C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内 容
1	a	{D∗ / R∗}注	lna×n	入 力	実行列 A(2 次元配列型)
				出 力	$A = LU$ と分解した時の単位上三角行列 U および下三角行列 L
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 A の次数 n
4	ipvt	I*	n	出 力	ピボッティング情報 ipvt[$i-1$]: i 段目の処理において行 i と交換した行の番号
5	cond	{D∗ / R∗}	1	出 力	条件の逆数
6	w1	{D∗ / R∗}	n	ワーク	作業領域
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

この関数を利用するには, まず, 引数として使用する配列 a, ipvt および w1 を, 呼び出し元の利用者プログラム側でアロケートする必要がある. それらはそれぞれ, {倍精度 / 単精度}注 実数型で大きさ lna × n, 整数型で大きさ n, {倍精度 / 単精度}実数型で大きさ n の配列である.

また, 64 ビット整数版を利用する場合には, 整数型引数 (lna,n,ipvt,ierr) はすべて`int` ではなく`long` を用いて宣言する必要がある.

注 ASL_dbgmlc のときには倍精度実数型 (D), ASL_rbgmlc のときには実数型 (R) で宣言することを意味する. 以下, 本文中で特に断らない限り中括弧 {} 等の使用法は, 同様の扱いとする.

この関数を使用するときには, a, lna および n にデータを格納しておかなければならない. 関数内では, 与えられた行列の LU 分解と条件数の算出が行われ, 結果が配列 a と変数 cond に格納される. また, 後続関数で利用するため, ピボッティング情報が ipvt に格納される.

ierr は, 入力データや処理途中の異常を利用者に知らせるための戻り値であり, 正常の場合は 0 にセットされる.

なお, w1 は関数内でのみ使用する作業領域であるので, 入力時および出力時の内容は特に意味をもたない.

(4) **制限条件**

関数の引数の制限範囲を明確にしてある.

(5) **エラーインディケータ (戻り値)**

各関数には, エラーインディケータが戻り値として設けられている. この エラーインディケータ (戻り値) は, ierr という変数名に統一されており, 引数と戻り値表の最後におかれている. 各関数は関数内でエラー検出を行い, その結果を ierr に設定する. ierr の値の意味は, 次の 5 段階に分かれている.

表 1–4　エラーインディケータ (戻り値) の出力値区分

レベル	戻り値	意　　味	処　理　内　容
正　常	0	正常終了した.	結果は保証される.
警　告	1000〜2999	ある条件のもとで一応の処理が終了した.	条件付きで結果は保証される.
異　常	3000〜3499	引数が制限条件に違反したために処理が打ち切られた.	結果は保証されない.
	3500〜3999	得られた結果がある検定条件を満足しなかった.	得られた結果を返す (結果は保証されない).
	4000 以上	処理の途中で致命的なエラーが発見された. 通常は処理を打ち切る.	結果は保証されない.

(6) **注意事項**

関数を使用するときの注意点およびあいまいな点を明確にしてある.

(7) **使用例**

関数の使い方の一例を載せてある. なお複数の関数を組み合わせて一つの例としてある場合もあるので注意されたい. 出力結果は, 32 ビット整数版での結果であり, コンパイラや組み込み関数の変更などにより丸め誤差の範囲で異なる場合がある.

また, 64 ビット整数版ライブラリを利用する場合には`printf` や`scanf` に与える`long` 型の変換仕様は`%ld` でなければならない. 本説明書に記載されている使用例のプログラムはソースコードの形で「ASL ユーザーズガイド」に収録されている. 入力データも (もし存在する場合は)「ASL ユーザーズガイド」に収録されている. コンパイラを用いて使用例のソースコードから実行形式ファイルを作成する場合には, ライブラリ本体とリンクする必要がある.

1.4 関数名

ASL の基本機能の関数名は, 「ASL_」と 6 桁のアルファニューメリック記号の集まりである.
また, 関数名の各記号にはそれぞれ意味を持ち, 図 1−1 で表される. 利用時においては, 計算用途に合わせて関数名を指定する必要がある.

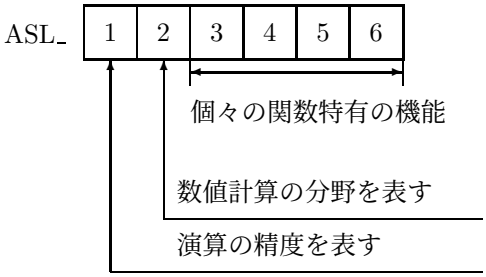


図 1−1　関数名の構成要素

図 1−1 の "1" ：演算の精度を表す. 基本機能編で使用される文字は, 次の 8 種類である.

d, w　倍精度実数型演算
r, v　単精度実数型演算
z, j　倍精度複素数型演算
c, i　単精度複素数型演算

ただし, 上記の複素数型とは必ずしも引数の型が複素数型であることを意味しない.

図 1−1 の "2" ：計算の分野を表す. 現在, ASL では次の文字が使用されている.

文字	計算の分野	分冊
a	格納モードの変換	1
	基本行列演算	1, 7
b	連立 1 次方程式 (直接法)	2, 7
c	固有値・固有ベクトル	1, 7
f	フーリエ変換とその応用	3, 7
	時系列分析	6
g	スプライン関数	4
h	数値積分	4
i	特殊関数	5
j	乱数の検定	6
k	常微分方程式初期値問題	4
l	方程式の根	5
m	極値問題・最適化	5
n	近似・回帰分析	4, 6
o	常微分方程式境界値問題, 積分方程式, 偏微分方程式	4
p	補間	4
q	数値微分	4

文字	計算の分野	分冊
s	ソート・順位付け	5, 7
x	基本行列演算	1
	連立 1 次方程式 (反復法)	7
1	確率分布	6
2	標本統計	6
3	推定と検定	6
4	分散分析・実験計画	6
5	ノンパラメトリック検定	6
6	多変量解析	6

図 1–1 の “3” ～ “6”： これらの文字で, 個々の関数に特有の機能を表す.

1.5 ASL C言語インタフェースの複素数型

ASL C 言語インタフェースの関数の引数が複素数型の場合, 必要なヘッダファイルをインクルードし, C99 の複素数型で宣言しなければならない.

表 1–7 ASL C 言語インタフェースの複素引数の型

言語	インクルードファイル	倍精度複素数型	単精度複素数型
C	complex.h	double _Complex	float _Complex
C++	ccomplex		

C 言語ならびに C++言語での使用例を以下に示す.

(1) C 言語での使用例 : ASL_zan1mm (< 基本機能第 1 分冊 > : 3.2.15 参照)
複素数型をdouble _Complex またはfloat _Complex で型宣言するために, asl.h の前にcomplex.h をインクルードしなければならない.

```
#include <complex.h>
#include <asl.h>

const int       mm=1000, nn=1000, nl=1000, lma=mm, lnb=nl, lmc=mm, isw=0;
double _Complex  a[lma*nn];
double _Complex  b[lnb*nl];
double _Complex  c[lmc*nl];
int             ierr;
    ～
ierr = ASL_zan1mm(a, lma, mm, nn, b, lnb, nl, c, lmc, isw);
```

(2) C++言語での使用例 : ASL_zan1mm (< 基本機能第 1 分冊 > : 3.2.15 参照)
複素数型をdouble _Complex またはfloat _Complex で型宣言するために, asl.h の前にccomplex をインクルードしなければならない.

```
#include <ccomplex>
#include <asl.h>

const int             mm=1000, nn=1000, nl=1000, lma=mm, lnb=nl, lmc=mm, isw=0;
double _Complex  a[lma*nn];
double _Complex  b[lnb*nl];
double _Complex  c[lmc*nl];
int                   ierr;
    ～
ierr = ASL_zan1mm(a, lma, mm, nn, b, lnb, nl, c, lmc, isw);
```

なお, 第 2 章以降の関数の使用例のプログラムは, C 言語から使用した例である.

1.6 注意事項

(1) ASL C 言語インタフェースを使用する場合は, ヘッダファイル `asl.h` をインクルードしなければならない. また, 複素数型を引数に持つ関数を使用する場合は, C 言語であれば`complex.h`, C++言語であれば`ccomplex` をインクルードしなければならない. 詳細は, 1.5 を参照のこと.

(2) ASL C 言語インタフェースの 64 ビット整数型ライブラリの関数を使用するプログラムをコンパイルする場合は, コンパイルオプション "`-DASL_LIB_I64`" を指定しなければならない. コンパイルオプション "`-DASL_LIB_I64`" を指定しない場合は, ヘッダファイル `asl.h` に記述されている 32 ビット整数型関数の関数プロトタイプ宣言が有効になり, 指定した場合は 64 ビット整数型関数の関数プロトタイプ宣言が有効になる.

(3) ASL C 言語インタフェースでは, ASL_ につづく ⟨6 文字⟩ という名前が ASL でリザーブされている.

(4) 64 ビット整数型ライブラリを使用する場合には, 整数型変数を "`long`" として, それ以外の場合には "`int`" として宣言すること.

(5) 単精度版ではなく, 倍精度版を標準として利用する方がよい. 精度が高いことに加え, 倍精度版の方が単精度版に比べて安定的に解が求まる場合 (特に固有値・固有ベクトル) が多い.

(6) 演算例外の抑止はメインプログラム側で行う必要がある. ASL C 言語インタフェースの関数では, コンパイラの演算例外の抑止に関して, ユーザのメインプログラムのコンパイルパラメータの指示に従うように設定してある.

(7) 扱う演算桁数を越える精度を期待することはできない. たとえば倍精度演算の (仮数部の) 演算桁数は 10 進 15 桁程度であるが, ここで数学的に 1 となるような値を計算した場合, 10^{-15} 程度の誤差は必ず発生する. これを抑制する方法として, 任意桁数演算のような多倍長演算のエミュレートが考えられるが, この場合, たとえば円周率のような定数や関数近似の定数なども都度計算する必要が生じるので, 通常の演算と比較して計算効率は悪くなる.

(8) 数学的に解が存在しないような問題の解を得ることはできない. たとえば, 数学的に特異な (または特異に近い) 行列を係数に持つ連立 1 次方程式の解を精度良く求めることは原理的にできない. なお, 数値計算上は, 数学的に特異な行列と特異に近い行列とを厳密に区別することはできない. もちろん, たとえば, 条件数の計算値が設定した基準値以上であれば特異とみなすというようなことはいつでも可能である.

(9) 浮動小数点例外 (オーバフローなど) をおこすようなデータを与えた場合, 正常な計算結果を期待することはできない. ただし, 反復計算で残差の加算等を行った場合に発生する浮動小数点アンダフローなどはこの限りではない.

(10) 数値計算で扱う問題 (特に反復法を計算手法とする問題) では, 与えるデータによっては解が精度良く求められない場合や全く求まらない場合がある. このような場合は, 問題自体を見直して, 解が求まるような問題に変更するなどの処置を講じる必要がある. たとえば, スパース行列を係数とする連立 1 次方程式を解く場合に, 専用の関数で解が得られないときでも, 密行列用の関数を用いることで解が得られる場合がある.

(11) 解が複数ある問題を解く場合, 実行するマシンや OS, 用いるコンパイラ等で実行結果が見掛け上異なる場合がある. たとえば, 固有値問題を解いた場合に得られる固有ベクトルがこれに相当する.

(12) "[非推奨]" と表示のある関数は, 今後廃止予定の機能である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを利用されたい.

第 2 章　フーリエ変換とその応用

2.1　概 要

本章では高速フーリエ変換 FFT(Fast Fourier Transform), 畳み込み, 相関, パワー・スペクトル解析, ウェーブレット変換およびラプラス逆変換を行う関数について説明する.

離散型フーリエ変換では, 入力データの性質に応じて, 次のような関数が用意されている.

(1) 複素フーリエ変換
　入力データが複素数である.

(2) 実フーリエ変換
　入力データが実数のみである.

本章で扱う離散型フーリエ変換は, 高速フーリエ変換を用いて実現している. 高速フーリエ変換は, 入力データの等分数 (n) がどのような素数を基数としても変換が可能であるが, 大きな素数からなる数列では演算効率が減少する. 従って等分数 (n) は小さな基数 (2, 3, 5 など) に因数分解できる数であることが望ましい.

畳み込み, 相関, パワー・スペクトル解析では, 次のようなデータを扱う関数が 用意されている.

(1) 1 次元データ

(2) 2 次元データ

(3) 3 次元データ

ラプラス逆変換では, 像関数の型に応じて次のような関数が用意されている.

(1) 像関数が有理関数の場合

(2) 像関数が一般関数の場合

ウェーブレット変換では, 基底関数の型に応じて次のような関数が用意されている.

(1) 基底関数が Haar 関数である場合.

(2) 基底関数がメキシカンハット関数である場合.

(3) 基底関数がフレンチハット関数である場合.

2.1.1　使用上の注意

(1) 利用者は入力データのタイプにより適切な関数群を選べばよい.

(2) 一般に入力データが実数の場合は, 実フーリエ変換の関数が利用できる.

(3) 変換の対象となる入力データは, 1 周期分 $[0, 2\pi]$ の入力データである.

(4) 利用者はまず初期化を含む変換の関数を呼び出す必要がある. この関数では, 三角関数テーブルの作成や因数分解がなされるので, 同じ入力データ数 n で同じフーリエ変換を続ける場合には, これ以後, ifax, trigs の内容を保存し, 初期化後の変換関数を利用することにより効率のよい処理ができる.

(5) 一般関数のラプラス逆変換では, 像関数 $F(s)$ の虚数部を計算する関数を利用者が用意する. $F(s)$ が多価関数の場合は, 関数の中で正しい枝を計算する様に注意しなければならない. また, 像関数 $F(s)$ は無限遠で収束することが望ましい. ($\lim_{|s|\to\infty} F(s) = 0$)

2.1.2 使用しているアルゴリズム

2.1.2.1 1 次元 (連続) フーリエ変換

次式で定義される積分 $F(\xi)$ を $f(x)$ の**フーリエ積分 (Fourier integral)** という.

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

ここで a は定数であり, 文献によって, $a=1$ ととる場合もあれば, $a=\frac{1}{2\pi}$ や $a=\frac{1}{\sqrt{2\pi}}$ ととる場合もある. なお, 虚数単位 $\sqrt{-1}$ を表すために通常 i や j が用いられるが, 添え字等に用いる整数 i, j と紛らわしいのであえて $\sqrt{-1}$ を用いることにする.
もしこの積分がパラメータ ξ のあらゆる値に対して存在するとき $F(\xi)$ を関数 $f(x)$ の **(連続) フーリエ変換 (Fourier transform)** と呼ぶ. この場合, $F(\xi)$ は $\mathcal{F}\{f(x)\}$ と表記される. 例えば, x として時間 t を考える場合には ξ は角周波数または角振動数 (angular frequency) ω に対応し, x として空間座標 $\boldsymbol{r}$ の成分を考える場合には ξ は波数ベクトル (wave vector) $\boldsymbol{k}$ の成分に対応する. また, x として時間 t を考える場合には角周波数 ω の代わりに, $f=\frac{\omega}{2\pi}$ で定義される周波数または振動数 (frequency) f も良く用いられる. なお, 振動数を表す場合, 文字 f の代わりに文字 ν が利用される. f に対応する空間での量 $\frac{k}{2\pi}$ は空間周波数と呼ばれることもある. $F(\xi)$ の逆変換は

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

と定義される. この式が逆変換であることの形式的な証明は以下のとおり.

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

ここで, $\delta(x)$ は**超関数 (distribution)** として定義される Dirac の δ 関数である.
最後の等号は $f(x)$ が連続であるときに成立するが, 不連続点 $x=x_0$ での関数 $f(x)$ の値を

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

と再定義すれば成立する. なお, 連続フーリエ変換は実際上あらわれる大部分の関数に対して存在すると考えて良いが, (無限積分を扱うので) あらゆる関数について存在するわけではないということに注意しておく必要がある. フーリエ変換の存在条件については専門書を参照されたい.
また, フーリエ変換 (以降フーリエ順変換と呼ぶ) とその逆変換の定義を逆にする場合もあるので注意が必要である. 例えば, 時間と空間の両方を考える場合, 複素平面波は角周波数 ω と波数ベクトル $\boldsymbol{k}$ を用いて $e^{\sqrt{-1}(\boldsymbol{k}\cdot\boldsymbol{r}-\omega t)}$ と表せるが, この場合, 時間についてのフーリエ変換と空間についてのフーリエ変換とでフーリエ変換の定義における指数関数のベキの符号を逆にしておく方が都合が良い. なお, この場合, フーリエ変換は関数の平面波展開に相当する.
$f(x)$ が実関数であれば, 次の**オイラーの公式 (Euler's formula)** からも明らかなように指数関数のベキの符号の変更はフーリエ変換の虚部の符号を変更することに対応する.

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

$f(x)$ と $F(\xi)$ を組みにして**フーリエ変換対 (Fourier transform pair)** と呼ぶ.
フーリエ変換対に対して次のような**パーセバルの定理 (Parseval's Theorem)** が成立する.

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

これは, 時間 (または空間) 領域と周波数領域での全エネルギーの対応を与えるものであると解釈される. 特に $f(x)(=f(t))$ として実時間関数を考える場合には, 負の周波数の出現を避けるために

$$\int_{-\infty}^{\infty}\{f(t)\}^2dt=\frac{1}{\pi a}\int_0^{\infty}|F(\omega)|^2d\omega \quad [\because f(t)\in \boldsymbol{R}\Rightarrow F(-\omega)=F(\omega)^*]$$

と変形する.

たとえば, 電気回路で, 理想抵抗 R(単位:オーム $[\Omega]$) に流れる電流が $i(t)$(単位:アンペア $[A]$) の場合, $|f(t)|^2$ に対応する電力は $R(i(t))^2$(単位:ワット $[W]$), $\int_{-\infty}^{\infty}|f(t)|^2dt$ に対応する電力量は $\int_{-\infty}^{\infty}R(i(t))^2dt$(単位:ジュール $[J]$) となるが, 電流 $i(t)$ の周波数スペクトル $\hat{i}(f)$ についてはその単位を $[A\cdot sec]=[A/Hz]$ として, その大きさを,

$$\int_{-\infty}^{\infty}\{R(i(t))^2\}dt=2\int_0^{\infty}R|\hat{i}(f)|^2df$$

が成立するように正規化すればよい. もちろん, 単位系を変えたい場合には, 用いる単位に応じて, 定数をかければよい. なお, 多くの場合, パワー・スペクトル解析では, パワー・スペクトルに現れるピーク波形の**半値全幅 (full width half maximum; FWHM)** 等を問題にするので, このような場合には, 特にスペクトルの絶対値を換算する必要はなく, 便利な**任意単位 (arbitrary unit)** を用いればよい.

実関数 $f(x)$ が周期 T_0 の周期関数すなわち, 任意の整数 j について $f(x)=f(x+jT_0)$ であれば関数を以下のように**フーリエ級数 (Fourier series)** に展開できる (ただし, 連続フーリエ変換の場合と同様, 不連続点で関数はその点の左右両極限値の平均値を値としてとるものとする).

$$\begin{aligned}f(x) &= \frac{a_0}{2}+\sum_{j=1}^{\infty}\{a_j\cos(j\omega_0x)+b_j\sin(j\omega_0x)\}\\ &= \sum_{j=-\infty}^{\infty}c_je^{\sqrt{-1}j\omega_0x}\quad(\omega_0=\frac{2\pi}{T_0})\end{aligned}$$

ここで, 展開係数 a_j, b_j(実数) ならびに c_j(複素数) は**フーリエ係数 (Fourier coefficients)** と呼ばれ, 次式で与えられる.

$$\begin{aligned}a_j &= \frac{2}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}f(x)\cos(j\omega_0x)dx\quad(j=0,1,2,\cdots)\\ b_j &= \frac{2}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}f(x)\sin(j\omega_0x)dx\quad(j=1,2,\cdots)\\ c_j &= \frac{1}{2}(a_j-\sqrt{-1}b_j)=\frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}f(x)e^{-\sqrt{-1}j\omega_0x}dx\quad(j=0,\pm1,\pm2,\cdots)\end{aligned}$$

なお, $f(x)$ が複素値関数 (x は実数) の場合には, フーリエ係数 a_j, b_j も複素数となる. $f(x)$ の 1 周期分だけを取りだし, 残りを 0 とした関数 $g(x)$ すなわち

$$g(x)=\begin{cases}f(x) & |x|\le\dfrac{T_0}{2}\\ 0 & \text{それ以外}\end{cases}$$

を考えると,

$$\begin{aligned}c_j &= \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}f(x)e^{-\sqrt{-1}j\omega_0x}dx\\ &= \frac{1}{T_0}\int_{-\frac{T_0}{2}}^{\frac{T_0}{2}}g(x)e^{-\sqrt{-1}j\omega_0x}dx\\ &= \frac{1}{T_0}\int_{-\infty}^{\infty}g(x)e^{-\sqrt{-1}j\omega_0x}dx\\ &= \frac{1}{aT_0}G(\xi)_{\xi=j\omega_0}\end{aligned}$$

したがって定数因子を除けばフーリエ係数は周期関数 ($f(x)$) を 1 周期分で帯域制限した関数 ($g(x)$) のフーリエ変換 $G(\xi)$ の $\xi = j\omega_0$ での値に等しい. 逆に, 帯域制限された関数 ($g(x)$) の離散点でのフーリエ変換 ($G(\xi)_{\xi=j\omega_0}$) は関数をその帯域を 1 周期とする周期関数 ($f(x)$) に拡張し, そのフーリエ係数 (c_j) を求めることによって計算できる. なお, 連続関数の標本化を行うに当たっては次の標本化定理がある.

標本化定理 (sampling theorem): 周波数 f_c で帯域制限された時間関数 $g(t)$ の場合, すなわち, $g(t)$ のフーリエ変換 $G(\omega)$ が $|\omega| \leq 2\pi f_c$ でのみ非ゼロ値をとる場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下のように標本値列 $\{g(iT)\}$ だけの知識から $g(t)$ を復元できる.

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

周波数帯域幅 $2f_c$ は**ナイキストの標本化周波数 (Nyquist sampling rate)** と呼ばれる.

現実には, 標本化定理の条件が満たされることはほとんどないが, 離散フーリエ変換の応用においては, ナイキストの標本化周波数をもとに標本化が行われる.

離散フーリエ変換 (discrete Fourier transform) では, 計算機で計算可能となるように, フーリエ級数における積分を以下のように方形近似して計算する.

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

ここで, $f_k = f(x)|_{x=\frac{kT_0}{n}}$ $(k = 0, \pm 1, \pm 2, \cdots)$ である. また $\lfloor x \rfloor$ は x を超えない最大の整数, $\lceil x \rceil$ は x 以上の最小の整数をそれぞれ表す. 最後の等式では $f(x)$ が周期 T_0 を周期とする周期関数であるから, f_k は n を周期とする離散周期関数となるという事実を用いている. この式から c_j も n を周期とする離散周期関数であることがわかる. したがって, **離散フーリエ変換では, 時間 (または空間) 領域での n 個の標本と周波数領域の n 個の対応する標本とが時間領域と周波数領域における波形の 1 周期を表す.** なお, 離散フーリエ変換はその周期性から, フーリエ級数における積分の**台形公式による近似**にもなっている.

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

本書では, 以下特に断らない限り, フーリエ変換は「離散フーリエ変換」を指すものとする. 離散フーリエ変換は**高速フーリエ変換 (Fast Fourier Transform; FFT)** アルゴリズムを用いることで効率良く計算できる. データ数 n の離散フーリエ変換の計算に必要な複素数の乗算回数は, 定義通り考えれば, n^2 のオーダとなるが, 高速フーリエ変換では, $n = r_1 r_2 \cdots r_m$ と因数分解できるとき, $n(r_1 + r_2 + \cdots + r_m)$ のオーダになる. 特に, $n = 2^m$ であれば, 乗算回数は $2n \log_2 n$ となる. したがって, 通常 n がそれほど大きくなくても, 高速フーリエ変換アルゴリズムで計算を行う場合とそうでない場合の計算効率の違いは大きい.

なお, 連続フーリエ変換では, もとの波形が不連続点を持てば, フーリエ級数の部分和はもとの波形に一様には収束せず, **ギブスの現象 (Gibbs phenomenon)** として知られる振動が発生するが, 離散フーリエ変換そのものにはギブスの現象はない. (離散フーリエ変換では変換後, 逆変換を行えば, もとの系列に一致した系列が得られる.) ただし, 連続フーリエ変換に対応する (無限個の) フーリエ係数のうち高次の項を打ち切った有限個が与えられた場合, **調和合成**

(harmonic synthesis) を行うことによって, ギブスの現象を再現することはできる. また, 連続フーリエ変換では, 逆変換式が成立するためには, 不連続点における関数値を, その点の両側 (近傍) の値の平均値と定義しなければならなかったが, 離散フーリエ変換では, 必ずしもこのようにデータを選ぶ必要性はない (サンプリング間隔をそれが数学的な意味で近傍とみなせるほど十分小さくすることは, 通常, 現実的でない).

2.1.2.2 多次元 (連続) フーリエ変換

フーリエ変換は多次元の場合に拡張することができる. 例えば, 4 次元時空間の場合には位置ベクトル $\boldsymbol{r} = (x, y, z)$, 波数ベクトル (wave vector) $\boldsymbol{k} = (k_x, k_y, k_z)$, 時間 t, 角周波数 ω を用いて 4 重積分として以下のように定義される.

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

$\hat{f}(\boldsymbol{k}, \omega)$ の逆変換は

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

と表せる. ここで a は定数であり, 文献によって, $a = 1$ ととる場合もあれば, $a = \frac{1}{(2\pi)^4}$ ととる場合もある. なお, 指数関数のベキの符号を逆にした定義をとる場合もあるので注意されたい. 1 次元の場合と同様に対応する離散フーリエ変換を多次元にも拡張できる.

2.1.2.3 1 次元フーリエ変換

任意の整数 k に対して $\hat{c}_k = \hat{c}_{k+n}$ を満たす複素周期データ $\hat{c}_k$ の 1 周期分 c_k $(k = 0, \cdots, n-1)$ に対して, 複素フーリエ順変換 C_j は次式で定義される.

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

α は任意の定数であり通常 1 や n が選ばれる. このとき変換後の複素データ C_j $(j = 0, \cdots, n-1)$ も任意の整数 j に対して $\hat{C}_j = \hat{C}_{j+n}$ を満たす複素周期データ $\hat{C}_j$ の 1 周期分に相当する. 対応する逆変換は

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

である. この式が逆変換であることは,

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

からわかる. ここで,

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

は**選点直交関係**として知られ, 初項 1, 公比 $r = e^{2\pi\sqrt{-1}\frac{(k-l)}{n}}$, 項数 n の等比級数の和を考えれば容易に証明できる. $\delta_{i,j}$ はクロネッカーのデルタで以下のように定義される.

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

離散フーリエ変換は, 離散関数が値を取ればいつでも存在し, また, 順変換に引き続き逆変換を行った結果は (数値計算上の誤差を除いて) もとの離散関数の値に一致する. なお, 選点直交関係は良く知られた複素三角関数系 $\{e^{\sqrt{-1}kt}\}$(k: 整数) の直交関係

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

の方形近似 (または台形公式による近似) に対応する. なお, 本書の関数では通常フーリエ変換の定義から定数倍を除いた αC_j, nc_k を求める.
連続フーリエ変換では, 通常関数の変域を原点に対して左右対称に取るので, これと対応させる場合には

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

(ただし, $m = \lfloor \frac{n}{2} \rfloor$ であり, $\lfloor x \rfloor$ は x を超えない最大の整数を表す) として半周期分のデータを 1 周期ずらして考えた方がよい (**両側スペクトル (two-sided spectrum)**). このとき C_0 がゼロ周波数に対応する要素である. また, 特に時間関数を考える場合には, 負の周波数を避けるために,

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

として考える場合もある (**片側スペクトル (one-sided spectrum)**).

(1) **1 次元複素フーリエ変換と多重 1 次元複素フーリエ変換**
本書では, フーリエ変換を行うデータの格納法の違いにより **1 次元複素フーリエ変換**と**多重 1 次元複素フーリエ変換**をわけて考える. **1 次元複素フーリエ変換**ではデータ c_k または C_j は配列に連続に格納するのに対して, **多重 1 次元複素フーリエ変換**では一定の間隔をおいて格納する. 多重フーリエ変換は通常多次元フーリエ変換やフーリエ変換と他の変換との混合による多次元変換を行う場合に利用できる.

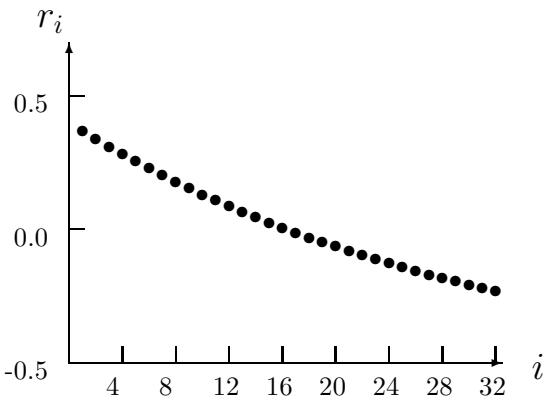
(2) **1 次元実フーリエ変換と多重 1 次元実フーリエ変換**
フーリエ順変換を行うデータが実数の場合,

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

を満たす. ただし, z^* は複素数 z の共役複素数を表す. 特に, C_0 は実数で, n が偶数のときは, $C_{\frac{n}{2}}$ も実数となる. また, 逆変換は

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数, $\lceil x\rceil$ は x 以上の最小の整数をそれぞれ表す. また, n が奇数のとき $\hat{C}_{\frac{n}{2}}=0$, n が偶数のとき $\hat{C}_{\frac{n}{2}}=C_{\frac{n}{2}}$. したがって, フーリエ変換は一般の複素データの場合の半分のデータ (c_k についてはその実部のみ, C_j についてはその半周期分) で計算を実行できる.
たとえば, 図 2–1 のような実数データ r_i を複素フーリエ変換すると図 2–2 のようになるが, 実フーリエ変換では図 2–3 のように実部と虚部それぞれ半周期分が計算される. (なお, 実フーリエ変換では通常, 変換結果の実部と虚部を配列に交互に格納するので注意のこと.)



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

図 2–1　フーリエ変換前のデータ

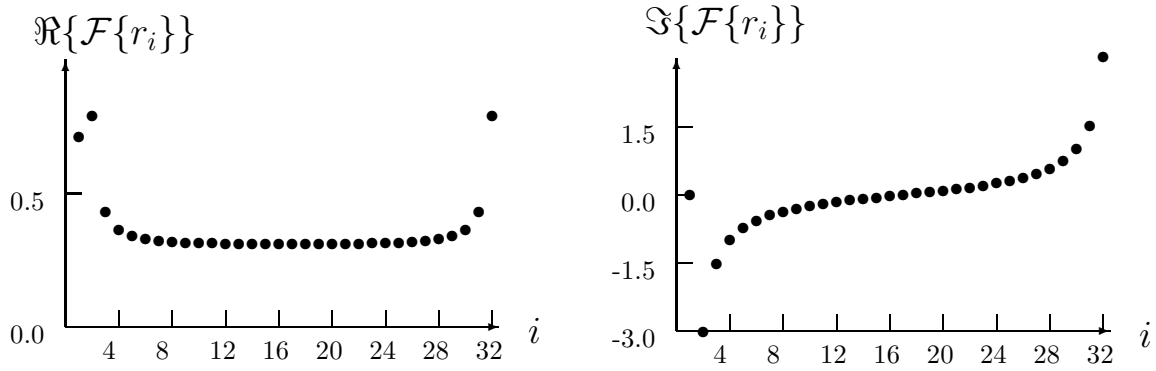
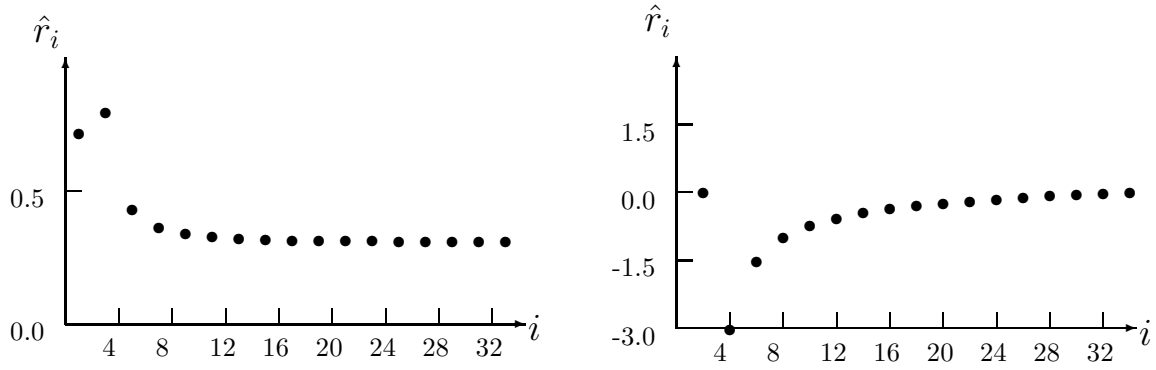


図 2–2　フーリエ変換後のデータ (複素フーリエ変換の場合)



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

図 2–3　フーリエ変換後のデータ (実フーリエ変換の場合)

なお, n が偶数のとき, $(C_0, C_{\frac{n}{2}})$ を新しい複素数 C_0 と定義して計算に必要な領域を節約する場合があるので注意されたい. また, **多重 1 次元複素フーリエ変換**の場合と同様**多重 1 次元実フーリエ変換**ではデータを配列に一定の間隔をおいて格納する.

2.1.2.4　多次元フーリエ変換

(1) **2 次元複素フーリエ変換**

任意の整数 k_x, k_y に対して $\hat{c}_{k_x,k_y} = \hat{c}_{k_x+n_x,k_y+n_y}$ を満たす複素多重周期データ $\hat{c}_{k_x,k_y}$ の 1 周期分 c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 複素フーリエ順変換 C_{j_x,j_y} は次式で定義される.

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

α は任意の定数であり通常 1 や $n_x n_y$ が選ばれる. このとき変換後の複素データ C_{j_x,j_y} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ も任意の整数 j_x, j_y に対して $\hat{C}_{j_x,j_y} = \hat{C}_{j_x+n_x,j_y+n_y}$ を満たす複素多重周期データ $\hat{C}_{j_x,j_y}$ の 1 周

期分に相当する. 対応する逆変換は

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

である. なお, 本書の関数では通常フーリエ変換の定義から定数倍を除いた $\alpha C_{j_x,j_y}$, $n_x n_y c_{k_x,k_y}$ を求める.

(2) **2 次元実フーリエ変換**

2 次元フーリエ順変換を行うデータが実数の場合,

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

を満たす. ただし, z^* は複素数 z の共役複素数を表す. 特に, $C_{0,0}$ は実数で, n_x と n_y が偶数のときは, $C_{\frac{n_x}{2},\frac{n_y}{2}}$ も実数となる.

同様に

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

したがって, フーリエ変換は一般の複素データの場合の半分のデータ (c_{k_x,k_y} についてはその実部のみ, C_{j_x,j_y} については j_x または j_y どちらかに対してその半周期分) で計算を実行できる.

(3) **3 次元複素フーリエ変換**

任意の整数 k_x, k_y, k_z に対して $\hat{c}_{k_x,k_y,k_z}=\hat{c}_{k_x+n_x,k_y+n_y,k_z+n_z}$ を満たす複素多重周期データ $\hat{c}_{k_x,k_y,k_z}$ の 1 周期分 c_{k_x,k_y,k_z} $(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ に対して, 複素フーリエ順変換 C_{j_x,j_y,j_z} は次式で定義される.

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

α は任意の定数であり通常 1 や $n_x n_y n_z$ が選ばれる. このとき変換後の複素データ C_{j_x,j_y,j_z} $(j_x=0,\cdots,n_x-1;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ も任意の整数 j_x, j_y, j_z に対して $\hat{C}_{j_x,j_y,j_z}=\hat{C}_{j_x+n_x,j_y+n_y,j_z+n_z}$ を満たす複素多重周期データ $\hat{C}_{j_x,j_y,j_z}$ の 1 周期分に相当する. 対応する逆変換は

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

である. なお, 本書の関数では通常フーリエ変換の定義から定数倍を除いた $\alpha C_{j_x,j_y,j_z}$, $n_x n_y n_z c_{k_x,k_y,k_z}$ を求める.

(4) **3 次元実フーリエ変換**

3 次元フーリエ順変換を行うデータが実数の場合,

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

を満たす. ただし, z^* は複素数 z の共役複素数を表す. 特に, $C_{0,0,0}$ は実数で, n_x, n_y, n_z すべてが偶数のときは, $C_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}$ も実数となる.
同様に

$$C^*_{n_x-j_x,j_y,j_z} = C_{j_x,n_y-j_y,n_z-j_z}$$

等の関係が成立する. すなわち, j_x, j_y, j_z のすべてについて次の対応関係を用いて置き換えを行ったものは置き換え前のものと互いに複素共役の関係にある.

$$\begin{aligned} j_x &\leftrightarrow n_x - j_x \\ j_y &\leftrightarrow n_y - j_y \\ j_z &\leftrightarrow n_z - j_z \end{aligned}$$

したがって, フーリエ変換は一般の複素データの場合の半分のデータ (c_{k_x,k_y,k_z} についてはその実部のみ, C_{j_x,j_y,j_z} については j_x または j_y または j_z のどれかに対してその半周期分) で計算を実行できる.

2.1.2.5 高速フーリエ変換

(1) **高速フーリエ変換のアルゴリズム**

ここでは高速フーリエ変換の任意基数の Temperton のアルゴリズムを説明する.
1 次元フーリエ変換を次式で定義する.

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

$k = 1, \cdots, N$(N:入力データ数), $W = e^{-2\pi i/N}, i = \sqrt{-1}$ ($1/N$ は省略する).

ここで, フーリエ変換を行列形式で考える. 変換式は以下のように表せる.

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

いま, データ数 $N = n_1 \ \ n_2 \ \ n_3 \ \ n_4 \ \ \cdots \ \ n_k$ ならば

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

と表せる.

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

とすると

$$T_i = (P^{n_i}_{m_i} \ \ D^{n_i}_{m_i} \times I_{l_i}) \ \ (W_{n_i} \times I_{N/n_i})$$

ゆえに

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

例えば, $N = 24$ の場合,

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

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

$$T_2 = (P_4^3 D_4^3 \times I_2)(W_3 \times I_8)$$

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

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

(注)

(a) $\times$ はクロネッカー積を表し, たとえば,

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

に対して,

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

という演算で定義される.

(b) $P_{m_i}^{n_i}$ はオーダ n_i, m_i の交換行列

(c) $D_{m_i}^{n_i}$ はオーダ n_i, m_i の対角行列

(d) I_r はオーダ r の単位行列

この計算は, 次のような流れになっている. (C:入力データ, A:出力データとする.)

$LA = 1$
- $I = 1$, 基数の個数
- $IFAC \leftarrow$ 基数 (2, 3, 4, 5, 6, 7, 8, または任意)
- 基数 $IFAC$ の FFT 計算
- $M = N/IFAC$
- $I = 1$
- $J = 1$
- $JUMP = (IFAC - 1) \times LA$
  - $K = 0, M - LA, LA$
    - $L = 1, LA$
    - $A(J) = \Omega(K) \times (W(IFAC) \times C(I))$
    - $I = I + 1$
    - $J = J + 1$
  - $J = J + JUMP$

(注)
1. $W(IFAC)$:オーダ $IFAC$ の DFT 配列
2. $\Omega(K)$:$diag(trigs(1), trigs(K+1), trigs(2 \times K + 1), \cdots, trigs((IFAC - 1) \times K + 1))$
3. $trigs(K+1) = \exp(2iK\pi/N), 0 \leq K \leq N-1, i = \sqrt{-1}$

- $LA = LA \times IFAC$, A と C を変換

(2) **多重 1 次元複素フーリエ変換について**

長さ N からなる M 組のデータ列のそれぞれに 1 次元複素フーリエ変換を実行する多重 1 次元複素フーリエ変換では, M 組のデータ列は各々独立であるため, M 回の 1 次元複素フーリエ変換の実行のためのループと通常の 1 次元複素フーリエ変換のループを入れ替えることが可能である. これにより, 通常の 1 次元複素高速フーリエ変換において生じる種々の問題 (入力データ, 回転因子のメモリ参照, 定義時のバンクコンフリクト, ループ長の減少など) を容易に避けることができ, 高速性能が得られる.

(3) **実フーリエ変換について**

実フーリエ変換は, 変換の対象となる実入力データ $\{r_k\}$ を虚部が零である複素入力データとみなし, それに対して複素フーリエ変換を行えば得られる. この場合, 複素フーリエ変換の結果は, 次のような共役関係 (conjugate symmetry) がある.

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

ここで, $*$ は, 複素共役を表わす. また, 実フーリエ変換の結果の $\{a_j\}$ $\{b_j\}$ と, 複素フーリエ変換による結果の $\{C_j\}$ との関連は, 次式で示される.

$a_0 = 2C_1$, $a_{N/2} = 2C_{N/2+1}$

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

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

従って, 実フーリエ変換を複素フーリエ変換により行う場合には, C_j $(j = 1, \cdots, \frac{N}{2} + 1)$ だけを求めれば $\{a_j\}$ と $\{b_j\}$ を求められることがわかる.

2.1.2.6 1 次元 (連続) 畳み込みと 1 次元 (連続) 相関

次式で定義される積分 $p(x)$ を $f(x)$ と $g(x)$ との**畳み込み積分 (convolution integral)** という.

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

$f(x)$ と $g(x)$ との畳み込み積分は $(f \times g)(x)$ や $f(x) \times g(x)$ と表される．なお, 畳み込み積分はしばしば単に「畳み込み」と呼ばれる．$f(x)$ と $g(x)$ のフーリエ変換をそれぞれ $\mathcal{F}\{f(x)\}$, $\mathcal{F}\{g(x)\}$ とすると $(f \times g)(x)$ と $\mathcal{F}\{f(x)\}\mathcal{F}\{g(x)\}$ はフーリエ変換対をなす．すなわち

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

この関係は**畳み込み定理 (convolution theorem)** と呼ばれる．また, $\mathcal{F}\{f(x)\}$ と $\mathcal{F}\{g(x)\}$ の畳み込み積分と $f(x)g(x)$ もフーリエ変換対をなす (**周波数畳み込み定理 (frequency convolution theorem)**)．**通常, 畳み込み積分は, 畳み込む 2 つの関数のフーリエ変換を計算し, 2 つのフーリエ変換の積の逆フーリエ変換を行うことによって求める．特に計算機を用いて計算する場合には, 効率がよい高速フーリエ変換アルゴリズムを利用できるのでこの方法は非常に有効である．** なお, 畳み込み積分は次のように有限区間積分として定義される場合があるので注意が必要である．

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

特に **Laplace 変換**と対応づける場合には, 次のように定義する．

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

畳み込みには次のような性質がある．

(1) 交換則 (commutative law) を満たす．
$f(x) \times g(x) = g(x) \times f(x)$

(2) 結合則 (associative law) を満たす．
$f(x) \times (g(x) \times h(x)) = (f(x) \times g(x)) \times h(x)$

(3) 分配則 (distributive law) を満たす．
$f(x) \times (g(x) + h(x)) = f(x) \times g(x) + f(x) \times h(x)$

理論ならびに実用的な応用面で重要なもう 1 つの積分として次式で定義される**相関積分 (correlation integral)** がある．

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

$f(x)$ が実関数であり, $f(x)$ と $g(x)$ のフーリエ変換をそれぞれ $\mathcal{F}\{f(x)\}$, $\mathcal{F}\{g(x)\}$ とすると $f(x)$ と $g(x)$ の相関積分と $\mathcal{F}\{f(x)\}^* \mathcal{F}\{g(x)\}$ はフーリエ変換対をなす (**相関定理 (correlation theorem)**)．すなわち

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

ただし, z^* は複素数 z の共役複素数を表す. **なお, 相関積分は交換則を満たさないので, 注意する必要がある.** $f(x)$ が偶関数であれば相関積分は畳み込み積分に一致する.

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

また, 相関積分は $f(x)$ と $g(x)$ が同じ関数のとき特に**自己相関関数 (autocorrelation function)** と呼ばれ, 異なるとき**相互相関関数 (cross correlation function)** と呼ばれる.

2.1.2.7　1 次元離散畳み込みと 1 次元離散相関

連続フーリエ変換に対応して離散フーリエ変換を考えるのと同様連続畳み込みと連続相関のそれぞれに対応して離散畳み込みと離散相関を考えることができる. **離散畳み込み**は連続畳み込みを方形積分で近似したものとして次式で定義される.

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

ここで Δx は標本化間隔である. また $f(i)$, $g(j)$, $p(k)$ は整数値 i, j, k に対してだけ値が定義される離散関数である. なお, 計算機で計算することを前提としているので $f(i)$ や $g(j)$ に対応するもとの実数 x の関数 $f(x)$ や $g(x)$ は特定の有限区間でのみ非ゼロとなる関数または周期関数である必要がある. また, 離散関数 $f(i)$, $g(j)$ は任意の整数 k に対して

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

を満たす周期 m の周期関数とする.
なお, 厳密には, 上述の周期性を仮定しない離散畳み込みを**線形畳み込み (linear convolution)**, 周期性を仮定する畳み込みを**巡回畳み込み (circular convolution)** と呼び区別する場合がある. 以下特に断らない限り, 離散畳み込みとして巡回畳み込みを考える. また, 説明のために, ゼロ以外の値となり得る値のことを「有効値」と呼ぶことにする. いま,

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

すなわち, $f(i)$ の 1 周期分のうち $f(i)$ $(i = 0, \cdots, n_1 - 1)$ の n_1 個だけが有効値をとり, $g(j)$ についても 1 周期分のうち $g(j)$ $(j = 1, \cdots, n_2 - 1)$ の n_2 個だけが有効値をとるとすると, $m \geq n_1 + n_2 - 1$ であれば $p(k)$ は 1 周期分のうち $p(k)$ $(k = 0, \cdots, n_1 + n_2 - 2)$ の $n_1 + n_2 - 1$ 個が有効値をとる. したがって, $m \geq n_1 + n_2 - 1$ ととれば, $f(i)$ と $g(j)$ に関して次の周期のデータとの重なりを起こさずに畳み込みを計算できる. すなわち, 1 周期分だけをみれば線形畳み込みと巡回畳み込みの結果は一致する. 特に巡回畳み込みを考える場合には, 離散フーリエ変換の場合と同様, その周期性により積分の方形近似は台形公式による近似と一致している.
たとえば, $f(i)$ の有効値が $\{f(0), f(1), f(2)\}$ の 3 個, $g(j)$ の有効値が $\{g(0), g(1)\}$ の 2 個とすると, 対応する $p(k)$ の有効値は

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

の $3+2-1=4$ 個となる. ただし, 連続畳み込みとの対応を考える場合には, $m=n_1+n_2-1$ とするよりも関数値 $p(n_1+n_2-1)=0$ を最後に追加して $m=n_1+n_2$ として考えた方がよい. このようにすれば, 例えば, $f(i)$ が区間 $[0,a]$ を n_1 等分して標本化した標本値で $g(j)$ が区間 $[0,b]$ を n_2 等分して標本化した標本値であるとした場合に, $p(k)$ が区間 $[0,a+b]$ を n_1+n_2 等分した場合の畳み込みの標本値に対応すると考えることができ, 標本化間隔は $f(i)$, $g(j)$, $p(k)$ で全て一致する. なお, $f(i)$ と $g(j)$ の有効値の数が不揃いな場合や, 大きすぎる場合には, **区分化 (sectioning)** を行い, 幾つかの区間に分けて離散畳み込みを求め加算するという手法がとられる. 区分化には**重複保持法 (overlap-save method)** や**重複加算法 (overlap-add method)** があるが, 本書の関数では重複加算法を利用して畳み込みを計算する機能を提供している.

連続関数の場合と同様, 離散関数の場合にもフーリエ変換と畳み込みとの間には以下のような離散畳み込み定理が成立する.

$f(i)$ $(i=0,\cdots,m)$ と $g(j)$ $(j=0,\cdots,m)$ の離散フーリエ変換をそれぞれ $F(i)$ $(i=0,\cdots,m)$, $G(j)$ $(j=0,\cdots,m)$ とすると, $f(i)$ と $g(j)$ との離散畳み込みと積 $F(j)G(j)$ $(j=0,\cdots,m)$ は (定数倍を除いて) フーリエ変換対をなす. すなわち,

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

が成立する. なお, $\delta_{i,j}$ はクロネッカーのデルタで以下のように定義される.

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

離散相関は連続相関を方形積分で近似したものとして次式で定義される.

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

ここで Δx は標本化間隔である. また $f(i)$, $g(j)$, $q(k)$ は整数値 i, j, k に対してだけ値が定義される離散関数である. 特に時系列を扱う場合, 相関の定義として見掛け上異なる定義が採用されている場合があるので注意が必要である. 例えば, 標本数が n の 2 つの時系列 x_i $(i=1,2,\cdots,n)$ y_i $(i=1,2,\cdots,n)$ が与えられている場合, **ラグ (lag)** l $(l=0,1,\cdots,m-1;\ m\leq n)$ の関数として, **相互相関係数 (cross correlation coefficient)** ${r_{xy}}^{(l)}$, ${r_{yx}}^{(l)}$ は

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

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

で定義される. ここで, ${\mu_x}^{(l)}$, ${\nu_x}^{(l)}$, ${\mu_y}^{(l)}$, ${\nu_y}^{(l)}$ は x_i, y_i $(i = 1, 2, \cdots, n)$ それぞれに対して, 前から $n - l$ 個のデータと後ろから $n - l$ 個のデータについての平均,

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

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

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

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

を表し, ${c_{xy}}^{(l)}$, ${c_{yx}}^{(l)}$ はそれぞれ次式で定義される**相互共分散 (cross covariance)**

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

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

${u_x}^{(l)}$, ${v_x}^{(l)}$, ${u_y}^{(l)}$, ${v_y}^{(l)}$ はそれぞれ次式で定義される x_i, y_i $(i = 1, 2, \cdots, n)$ それぞれに対して, 前から $n - l$ 個のデータと後ろから $n - l$ 個のデータについての**分散 (variance)** とする.

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

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

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

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

いま,

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

として x_i, y_i $(i = 1, 2, \cdots, n)$ の前から $n-l$ 個のデータと後ろから $n-l$ 個のデータについての標準化した量 $f(i)$, $g(i)$, $\hat{f}(i+l)$, $\hat{g}(i+l)$ を定義すれば

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

が成立する．したがって, ${r_{xy}}^{(l)}$ と ${r_{yx}}^{(l)}$ の定義は (定数倍を除いて) 離散相関 $q(k)$ の定義と一致する．なお, 今の場合, x_i と y_i とが同じ時系列であれば ${r_{xx}}^{(l)}$ は**自己相関係数 (autocorrelation coefficient)**, ${c_{xx}}^{(l)}$ は**自己共分散 (autocovariance coefficient)** と呼ばれる．
また, 時系列の統計処理を考える場合には, 標本についての量と統計的な推定量とを区別するために以下の用語を用いる．

平均	$\rightarrow$	標本平均
分散	$\rightarrow$	標本分散
自己相関係数	$\rightarrow$	標本自己相関係数
自己共分散	$\rightarrow$	標本自己共分散
相互相関係数	$\rightarrow$	標本相互相関係数
相互共分散	$\rightarrow$	標本相互共分散

離散相関 $q(k)$ は, 計算機で計算することを前提としているので $f(i)$ や $g(j)$ に対応するもとの実数 x の関数 $f(x)$ や $g(x)$ は特定の有限区間でのみ非ゼロとなる関数または周期関数である必要がある．また, 離散関数 $f(i)$, $g(j)$ は任意の整数 k に対して

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

を満たす周期 m の周期関数とする．
離散畳み込みの場合と同様, $m \geq n_1 + n_2 - 1$ ととれば, $f(i)$ と $g(j)$ に関して次の周期のデータとの重なりを起こさずに相関を計算できる．たとえば, $f(i)$ の有効値が $\{f(0), f(1), f(2)\}$ の 3 個, $g(j)$ の有効値が $\{g(0), g(1)\}$ の 2 個とすると, 対応する $q(k)$ の有効値は

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

の $3+2-1=4$ 個となる．ただし, 連続相関との対応を考える場合には, $m = n_1 + n_2 - 1$ とするよりも関数値 $q(-n_1) = 0$ を最初に追加して $m = n_1 + n_2$ として考えた方がよい．このようにすれば, 例えば, $f(i)$ が区間 $[0, a]$ を n_1 等分して標本化した標本値で $g(j)$ が区間 $[0, b]$ を n_2 等分して標本化した標本値であるとした場合に, $q(k)$ が区間 $[-a, b]$ を $n_1 + n_2$ 等分した場合の相関の標本値に対応すると考えることができ, 標本化間隔は $f(i)$, $g(j)$, $q(k)$ で全て

一致する. なお, $f(i)$ と $g(j)$ の有効値の数が不揃いな場合や, 大きすぎる場合には, 離散畳み込みの場合と同様**重複加算法 (overlap-add method)** を利用して区分化を行って計算することができる. 連続関数の場合と同様, 離散関数の場合にもフーリエ変換と相関との間には以下のような離散相関定理が成立する.
$f(i)$ が実離散関数であるとし, $f(i)$ と $g(j)$ の離散フーリエ変換をそれぞれ $F(i)$ $(i = 0, \cdots, m)$, $G(j)$ $(j = 0, \cdots, m)$ とすると, $f(i)$ と $g(j)$ との離散相関と積 $F(j)^* G(j)$ $(j = 0, \cdots, m)$ はフーリエ変換対をなす. すなわち,

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

が成立する. なお, $\delta_{i,j}$ はクロネッカーのデルタで以下のように定義される.

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

$q(k)$ の有効値は $q(k)$ $(k = 0, \cdots, n_2 - 1)$ および $q(-k) = q(m - k)$ $(k = 1, \cdots, n_1 - 1)$ で与えられるが, このままでは区分化等を行う場合に不便であるので $n_1 - 1$ だけシフトした次式で定義される $\hat{q}(k)$ を計算する方が良い.

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

$\hat{q}(k)$ を計算する場合, $q(k)$ をシフトする代わりに $g(k)$ をシフトする. いま

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

であり,

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

である. したがって, $g(j)$ の方をあらかじめ以下のように定義される $\hat{g}(j)$ となるようにシフトしておけば, 直接 $\hat{q}(k)$ を得ることができる.

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

2.1.2.8 多次元 (連続) 畳み込みと多次元 (連続) 相関

畳み込み積分と相関積分は多次元の場合に拡張することができる. 例えば, 3 次元の場合には $f(x,y,z)$ と $g(x,y,z)$ との畳み込みと相関は 3 重積分として以下のように定義される.
畳み込み:

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

相関:

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

また 1 次元の場合と同様, 次のような畳み込み定理ならびに相関定理が成立する.

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

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

ここで $\mathcal{F}\{f\}$ は f のフーリエ変換を表す. なお, 相関積分において $f(x,y,z)$ が複素関数の場合は, $\mathcal{F}\{f(x,y,z)\}^*$ の代わりに $f(x,y,z)$ の負の周波数のフーリエ変換を用いれば良い.

2.1.2.9 パワー・スペクトル

$f(x)$ フーリエ積分 $F(\xi)$

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

に対して $P(\xi) = |F(\xi)|^2$ で定義される量を $f(x)$ のパワー・スペクトル (密度関数) と呼ぶ. 通常, パワー・スペクトルはパーセバルの定理 (Parseval's Theorem)

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

か成り立つように正規化する. 連続関数 $f(x)$ のパワー・スペクトルに対応する量として n を周期とする離散関数 $c(k)$ $(k = 0, 1, \cdots, n-1)$ の場合には**生のピリオドグラム (raw periodogram)** $p(j)$ を考える. ピリオドグラム $p(j)$ は $c(i)$ の離散フーリエ変換 $C(j)$

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

を用いて

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

と定義される. β は単位系の選び方等で決まる適当な定数である. 本書の関数では通常 $\alpha = 1$; $\beta = \frac{1}{n^2}$ が選ばれる. このときパーセバルの定理による, 時間 (または空間) 領域の対応する全パワーは $\frac{\sum_{k=0}^{n-1}\{c(k)\}^2}{n}$ になる. なお, $c(k)$ が時系列の場合には, 離散フーリエ変換 (フーリエ・スペクトルともいう) の場合と同様**両側パワー・スペクトル (two-sided power spectrum)** と**片側パワー・スペクトル (one-sided power spectrum)** を考えることができる. 両側パワー・スペクトルでは連続フーリエ変換で, 通常, 関数の変域を原点に対して左右対称に取ることに対応して

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

(ただし, $m = \lfloor\frac{n}{2}\rfloor$ であり, $\lfloor x\rfloor$ は x を超えない最大の整数を表す) として半周期分のデータを 1 周期ずらして考える. このとき $p(0)$ がゼロ周波数に対応する要素である. 各ピリオドグラム $\tilde{p}(j)$ に対応する周波数は $\frac{j}{nT}$ $(j =$

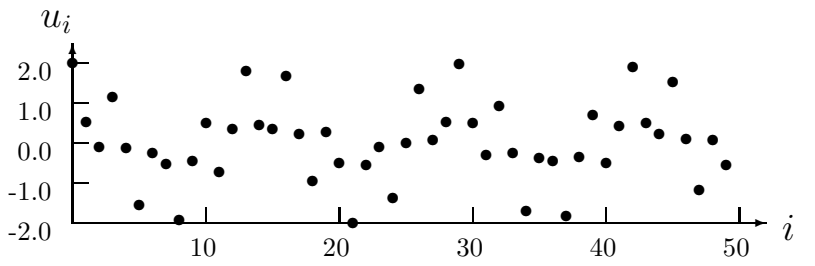
$-(n-m-2), \cdots, -1, 0, 1, \cdots, m)$ である. ただし, T は標本化間隔. 片側パワー・スペクトルでは負の周波数を避けるために,

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

として考える. 各ピリオドグラム $\tilde{p}(j)$ に対応する周波数は $\frac{j}{nT}$ $(j = 0, 1, \cdots, m)$ である. なお, 離散フーリエ変換の周波数領域における標本点の周波数の間隔 $\frac{1}{nT}$ を**分解度 (resolution)** と呼ぶこともある.

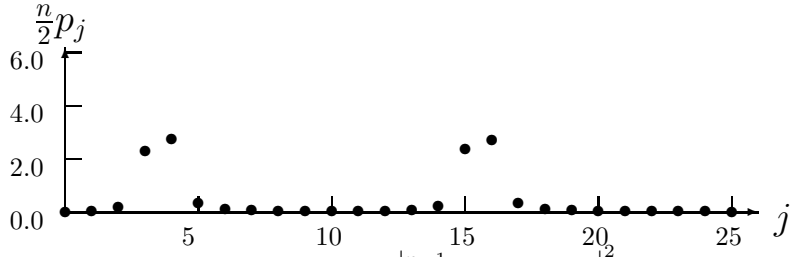
離散フーリエ変換はフーリエ級数に対する方形近似 (台形公式による近似でもある) であるので, 近似精度を上げるためにはデータ数 n を大きくとる必要がある. 一方, 前にも述べたように, 周期関数のフーリエ級数の値と周期関数をその 1 周期で打ち切った関数の連続フーリエ変換とは定数倍を除いて一致するので, ピリオドグラムはデータ数 n が十分大きければこのような連続関数に対するパワー・スペクトルのよい近似を与えると期待できる. ただし, パワー・スペクトルの推定に用いる系列が反映するもとの関数は, 通常, 周期関数ではなく, また周期関数であってもその 1 周期でちょうど打ち切られているという状態にはない. 生のピリオドグラムはその定義から自己相関関数の離散フーリエ変換近似とみなせる.

図 2−4 に離散データ $u_i = \cos(0.62\pi i) + \cos(0.14\pi i)$ $(i = 0, 1, \cdots, n-1;\ n = 50)$ についての生のピリオドグラムと自己相関関数の離散フーリエ変換との計算結果を示す. ここで, 自己相関関数を計算する場合の周期は $2n$ とし, $u_{n+1} = \cdots = u_{2n-1} = 0$ としている. 図には標本化間隔 $T = \Delta t = 0.5[sec]$ とした場合の分解度 Δf の値も参考に示している. この場合, 周波数 $0.14[Hz]$ と周波数 $0.62[Hz]$ に相当する部分にパワーが集中していることが分かり, 期待通りの結果を与える. なお, 自己相関関数の離散フーリエ変換の方がピリオドグラムの場合よりも分解度が小さいのでより望ましい形状をしているが, 対応する計算点数は 2 倍になっていることに注意する必要がある. また, 自己相関関数の離散フーリエ変換は実数となることにも注意する (より一般的な相互相関関数の離散フーリエ変換は通常複素数である).



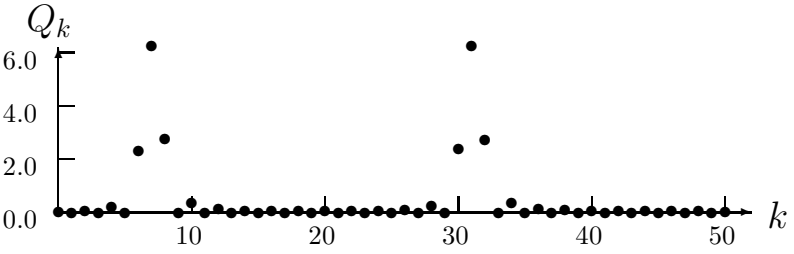
Sample data: $u_i = \cos(0.62\pi i) + \cos(0.14\pi i)$

$(i = 0, 1, \cdots, n-1;\ n = 50;\ \Delta t = 0.5[sec])$



Raw Periodogram : $p_j = \dfrac{\left|\sum_{i=0}^{n-1} u_i e^{-2\pi\sqrt{-1}\frac{ij}{n}}\right|^2}{n^2}$

$(j = 0, 1, \cdots, \lfloor \frac{n}{2} \rfloor;\ n = 50;\ \Delta f = 0.04[Hz])$



Discrete Fourier transform of Correlation :

$$Q_k = \frac{1}{2n} \sum_{j=0}^{2n-1} \{ \sum_{i=0}^{2n-1} u_i u_{j+i} \} e^{-2\pi\sqrt{-1}\frac{jk}{2n}}$$

$(k = 0, 1, \cdots, n;\ u_{n+1} = \cdots = u_{2n-1} = 0;\ n = 50;\ \Delta f = 0.02[Hz])$

図 2–4　ピリオドグラムと自己相関関数のフーリエ変換

有効データ数 n の離散関数の自己相関関数の有効データ長は $2n-1$ であるので, 一般の関数のパワー・スペクトルを生のピリオドグラムで近似することは, 1 つの周期が以下のように与えられる方形打ち切り関数 $w(k)$ で関数を打ち切ったことに相当する.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n-1 \\ 0 & \text{それ以外} \end{cases}$$

方形関数のフーリエ変換は周波数を f とした場合, $\frac{\sin f}{f}$ 型の関数形をしており, 中心周波数の周りに小さくないサイドローブを持っている. したがって, たとえば, 周期関数を 1 周期の整数倍でない幅で単純に打ち切って標本化した場合, 周波数領域では, 生のピリオドグラムはパワー・スペクトルを求めたい周期関数のフーリエ変換と $\frac{\sin f}{f}$ 型関数との畳み込みとなるので, **漏れ (leakage)** と呼ばれる余分な周波数成分が発生する. このような**漏れ**を抑止するためには, 単純な打ち切りを行わずに周波数領域でのサイドローブが小さい打ち切り関数を用いる. これにともない, ピリオドグラムの定義を修正した以下のような**修正ピリオドグラム (modified periodogram)**$\hat{p}$ が通常用いられる.

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

ここで $\hat{C}(j)$ はもとの系列 $c(k)$ に打ち切り関数 $w(k)$ を乗じた値の離散フーリエ変換であり次式で定義される.

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

$w(k)$ は**窓関数 (data window)** とも呼ばれる. 修正ピリオドグラムは, $w(k)$ のフーリエ変換 $W(j)$ を用いて周波数領域の畳み込みとして定義される場合もある. この場合, $W(j)$ はスペクトル・ウィンドウ (spectral window) と呼ばれることもある. これに対応して $w(k)$ はラグ・ウィンドウ (lag window) とも呼ばれる. 窓関数は本来, 上述のように打ち切りによる周波数領域でのサイドローブ抑止のため提案されたが, 数学的には, 窓関数の効果は周波数領域における平滑化式と同じ形式となるので, 窓関数を適当に選べば, パワー・スペクトルの平滑化を行うこともできる. また, 時系列のスペクトル解析を行う場合よく, 平均値 0 の時系列データを考えるが, 平均値はフーリエ変換のゼロ周波数成分に対応するので, 変換後ゼロ周波数成分をカットすれば, 同様な効果が期待できる. ただし, 修正ピリオドグラムを用いる場合には, 窓関数を乗じることにより平均値が変動するので, 前もって平均値を 0 としておく意味合いはない. なお, 打ち切り関数 $w(k)$ を乗じることによってピリオドグラムの定義に定数倍の差が生じる. 本来は, 時間 (または空間) 領域での全パワーを計算し, これが周波数領域の全パワーと一致するように補正すること (パーセバルの定理による) が望ましいが, このような補正を行うための計算コストは小さくない. また, 時間 (または空間) 領域での全パワーの推定が困難な場合もある. 打ち切り関数によるパワーを補正するため次式で定義される $\tilde{p}(j)$ を用いる場合がある.

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

この場合, 用いる打ち切り関数によっては, 2 乗和を解析的に計算できるので, パワーの補正に要するコストはそれほど大きくない. 対応する系列の全パワーは, 方形打ち切りを用いた場合の一般化として $\frac{\sum_{k=0}^{n-1}\{c(k)\}^2}{n}$ とする.
なお, スペクトル・ウィンドウ $W(j)$ のサイドローブを抑止すればする程 $W(j)$ のスペクトル幅が増加するので得られるパワー・スペクトルの推定波形もぼやけたものになる. したがって, パワー・スペクトルを推定する場合, 目的に応じて, すなわち, スペクトル幅を問題としているの中心周波数を問題としているのか等に応じて, 適切な打ち切り関数を選択する必要がある. 代表的な打ち切り関数としては以下のものがある.

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

ただし, $u_j = \frac{j}{n}$. このように選ぶと, $w_0 = 0$ となるので, c_0 に対応する成分が無効となるため, 窓関数を多少変形して定義する場合もある. なお, 窓関数は $|t| \le 1$ でのみ非ゼロとなる時間領域関数としてそれぞれ次のように表される.

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

図 2–5 にこれらの窓関数 (w_i) とその離散フーリエ変換 (W_j) の大きさ $(|W_j|)$ のグラフを示す.

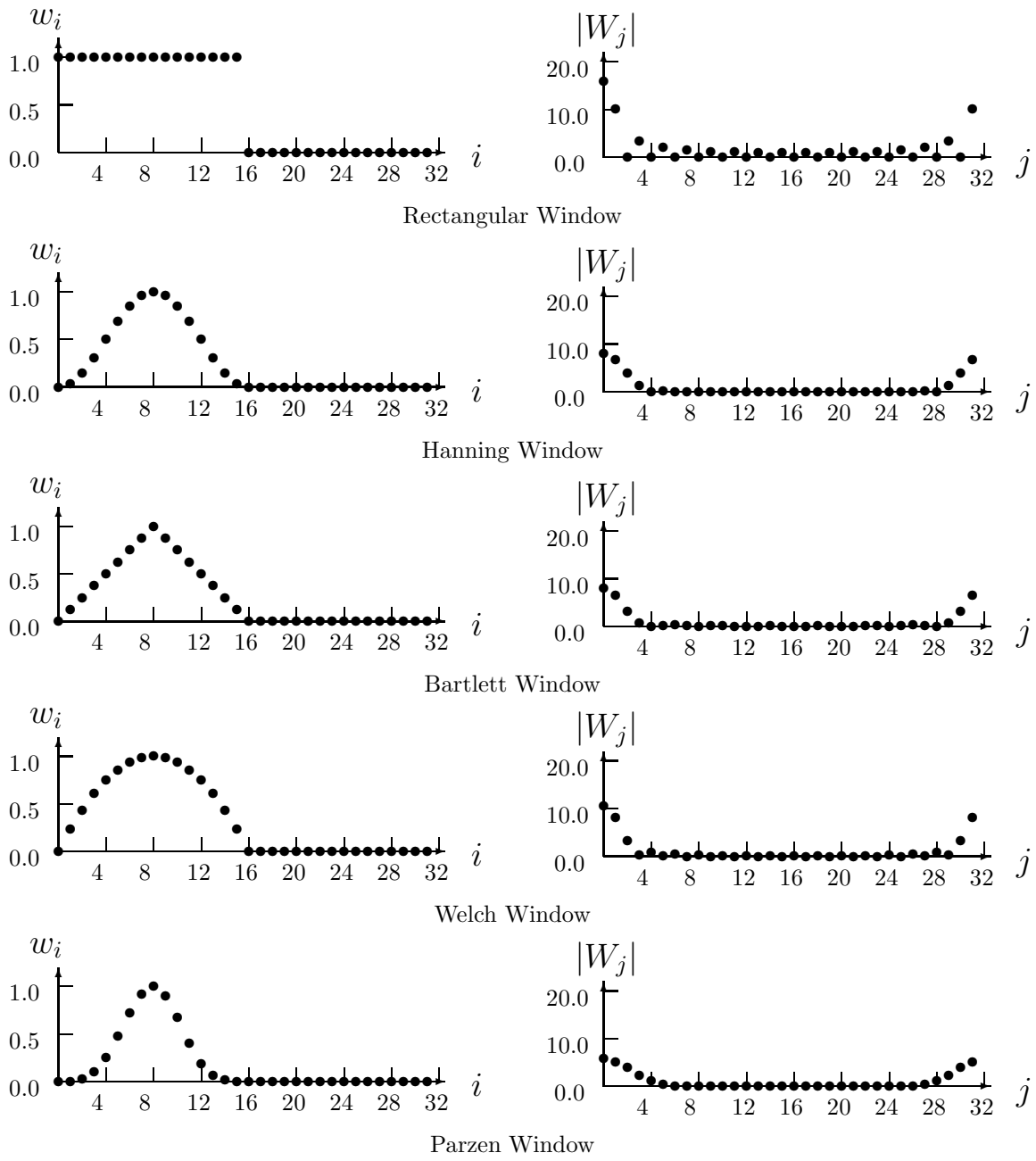


図 2–5　窓関数とその離散フーリエ変換の絶対値

離散フーリエ変換の分解度 $\frac{1}{nT}$ を上げるには標本のデータ数 n を増やすか標本化間隔 T を増やせば良いが, 標本化間隔と分解度を一定に保った状態でパワー・スペクトルの推定値の精度を上げるために, データ数 n の標本を m 組とって m 組それぞれについて修正ピリオドグラムを求めてその平均をとるという手法がよく取られる. この場合, m 組の標本データをもとの系列からオーバーラップして取るというような手法も提案されている. 詳細は参考文献等を参照されたい. また, パワー・スペクトルを求める場合, フーリエ変換の周波数推移に関する性質すなわち時間 (または空間) 領域で $e^{2\pi\sqrt{-1}f_0 t}$ を掛けることは周波数領域では周波数を f_0 だけシフトすることに対応し, 関数の形状は変わらないという性質を利用して, パワー・スペクトルの中心周波数をあらかじめシフトして計算することで計算に必要なデータ点数を削減するという手法も良く用いられる. なお, このような操作は**変調 (modulation)** として知られている.

2.1.2.10 ラプラス変換

(1) **ラプラス変換**

関数 $f(t)$ が与えられたとき, ラプラス変換は複素数 s をパラメータとして

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

で定義される. このとき, 逆変換は **ブロムウィッチ積分**として知られる

$$f(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \tag{2.2}$$

ただし, $\gamma > \gamma_0$, γ_0:収束座標, $i = \sqrt{-1}$

で与えられる. $f(t)$ を**原関数**, $F(s)$ を**像関数**といい,

$$\text{正変換} \quad F(s) = \mathcal{L}\{f(t)\} \tag{2.3}$$

$$\text{逆変換} \quad f(t) = \mathcal{L}^{-1}\{F(s)\} \tag{2.4}$$

と書く.

(a) 像関数 $F(s)$ が以下の条件を満たす場合

(I) $\Re(s) > 0$ で正則

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

(III) $\Re(s) > 0$ で $F(s^*) = F^*(s)$

($\Re(s)$: s の実数部; *:共役複素記号)

(i) **指数関数の近似**

式 (2.2) は指数関数 e^s を $\Re(s) > 0$ にのみ極をもつ有理関数で近似できれば, この極の周りの積分に帰着できる. そこで, 指数関数の近似法として次の関数式を考える.

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

この式は, 次のように 2 通りに書き換えられる.

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

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

式 (2.7) から, $a \gg \Re(s)$ のとき, $E_{ec}(s,a)$ は指数関数のよい近似であることがわかり, 式 (2.6) から, $E_{ec}(s,a)$ の極はすべて 1 位で, $s = a$ という直線上に等間隔でならび, 留数は $\frac{(-1)^n ie^a}{2}$ であることがわかる.

ブロムウィッチ積分中の e^{st} の代わりに, $E_{ec}(st)$ を代入した関数

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

を定義し, 式 (2.6), (2.7) を使って書き直すと

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

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

($\Im(s)$: s の虚数部)

(ii) **オイラー変換**

式 (2.10) のような無限級数の和を数値的に計算する場合は, 適当な項数 N で打ち切らなくてはならない. ここでは, Euler 変換を利用して効率的に計算を行う. Euler 変換は,

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

(ただし, $\Delta a_0 = a_0 + a_1, \Delta^2 a_0 = \Delta a_0 + \Delta a_1, \cdots, \Delta^n a_0 = \Delta^{n-1} a_0 + \Delta^{n-1} a_1$)

により, 左辺の級数を右辺の級数に変換する方法である. $\Delta^n a_k$ は, 第 n 段差であり, 次式で定義されている.

$$\Delta^n a_k = a_k - \left(\begin{array}{c} n \\ 1 \end{array} \right) a_{k+1} + \left(\begin{array}{c} n \\ 2 \end{array} \right) a_{k+2} - \cdots \pm \left(\begin{array}{c} n \\ n \end{array} \right) a_{k+n} \tag{2.12}$$

オイラー変換によって変換された級数は, 次の条件が成り立つ時, 元の級数よりも速く収束することが知られている.

(A) $\sum_{n=0}^{\infty} a_n$ が交代級数で, 数列 $\{|a_n|\}$ が完全に単調に 0 に収束する.

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

オイラー変換の効果は, $|\frac{a_{n+1}}{a_n}|$ が 1 に近いほど大きいので, 式 (2.11) の初めの k 項は普通に計算し, $k+1$ 項以降をオイラー変換すると

$$\begin{aligned} &\sum_{n=0}^{p-1} 2^{-(n+1)} \Delta^n a_0 = 2^{-p} \sum_{q=1}^{p} A_{pq} a_{q-1} \\ &\text{ただし}, A_{pp} = 1, A_{p,q-1} = A_{pq} + \left(\begin{array}{c} p+1 \\ q \end{array} \right) \end{aligned} \tag{2.13}$$

となる. よって $f(t)$ の近似値は次式から計算する.

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

また, 実際の計算では, 式 (2.11) の右辺を p 項で打ち切る. この時の打ち切り誤差は,

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

で与えられるが, さらに次の条件

- $a_n = f(n)$ と書けて, $f(x)$ の p 階微係数 $f^{(n)}(x)$ が x の正値に対して定符号で x の増加とともに単調に減少する.

が成り立つと,

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

となることが証明されている.

[**注意事項**] $F(s)$ が e^{-sx} の形の因数を持つ場合, F_n が条件 (1(a)iiA) を満たさないため, 精度が落ちることがある. このとき像関数は, 次の性質を持っている.

- $t < t_0$ において $f(t) = 0$
- $t = t_0$ のとき $f(t)$ またはその導関数が不連続となる

$t < t_0$ において $f(t) = 0$ とわかっている場合は, t 軸を t_0 だけずらして

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

の像関数

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

を扱うと良い.

(b) $\Re(s) > 0$ に特異点を持つ場合

$F(s)$ の収束座標 α がわかっているとき,

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

とすると, $G(s)$ は $\Re(s) > 0$ で正則である. さらに, $F(t), G(t)$ の原関数をそれぞれ $f(t), g(t)$ とすると

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

の関係がある. したがって, $\Re(s) > 0$ の場合は, 以下の式を使って計算する.

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

[収束座標の判定]

- $F(s)$ が有理関数の場合

  実係数多項式 $Q(s), P(s)$ を用いて

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

  と表せる. 分母多項式 $P(s) = 0$ の根を $s_1, s_2, \cdots, s_n$ とすると収束座標は, 根の実数部の最大値

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

  で与えられる. つまり, $P(s) = 0$ の全根の実数部が負であれば, 収束座標 $\alpha < 0$ となり, $\Re(s) > 0$ で正則である. そこで, 次の定理を使って収束座標の符号を判定する. 多項式 $P(s)$ の全根が負であるための必要十分条件は, 偶数項の和と奇数項の和の比を連分数に展開し,

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

  としたとき, 係数 $h_1, h_2, \cdots,$ がすべて正値となることである. この定理を満たす多項式はHurwitz 多項式と呼ばれている. 連分数の係数 $h_1, h_2, \cdots,$ は, Euclid の互除法を用いて計算する. $F(s)$ が $\Re(s) > 0$ に特異点を持つ場合, 上記の判定法を適当な正数 b に対して多項式 $P(s+b)$ を作り, Hurwitz 多項式か否かと判定する. b を増加させながら繰り返し判定を行って $F(s+b)$ が正則となるような b の収束座標を求める.

- $F(s)$ が一般関数の場合

  $F(s)$ が非有理関数の場合, $\Re(s) > 0$ で $F(s)$ が正則かどうかを判定する有効な方法はないので収束座標は利用者が指定しなければならない.

(c) $F(s^*) \neq F^*(s)$ の場合

x を実数として

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

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

とし, 次に x を s に書き換えて $F_1(s)$ と $F_2(s)$ は条件 (1(a)i)∼(1(a)iii) を満足する. そこで, これを逆変換して $f_1(t)$, $f_2(t)$ を求めると

$$f(t) = f_1(t) - i f_2(t) \tag{2.27}$$

が $\mathcal{L}^{-1} F(s)$ となる.

(d) パラメータの値の決定

(i) 級数和の打ち切り項数 $k+p$

オイラー変換が有効で, 打ち切り誤差の評価式 (2.16) が使えるためには, 少なくとも $|F_n|$ が単調減少しなければならない. 一般に, $|F_n| = |\Im(F[\frac{a}{t} + i\frac{(n+0.5)\pi}{t}])$ は, n とともに複雑に変化する. $F(s)$ の特異点の虚数部の最大値を ω_m とするとき, $\frac{(n+0.5)\pi}{t} > \omega_m$ ならば $|F_n|$ は単調に減少する. したがって, 式 (2.16) での普通和の項数 k は, $k > 0.5\frac{\omega_m}{\pi}t$ のように t に比例して増加させなければならない. 実際には, a の値にも関係するので

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

として, 打ち切り誤差 (2.16) が希望の値となるように k_1, k_2 を決める. $t_1 \leq t \leq t_2$ の範囲で $f(t)$ を求めたい場合, k_1, k_2 を決める一つの方法は, まず, $t = t_1$ として打ち切り誤差が無視できる $k(t_1)$ を決め, 次に, $t = t_2$ として同様に打ち切り誤差が無視できる $k(t_2)$ を決める.

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

より k_1, k_2 を決める.

オイラー変換の次数 p の値は, 計算精度とだいたい比例する傾向にあるので要求する計算精度が 10^{-d} ならば $p = d$ を与えるとよい. (参考文献 (10) 参照)

(ii) 指数関数を $E(s, a)$ で近似するときのパラメータ a

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

をうるので, $f_{ec}(t, a)$ と共に $f_{ec}(3t, a)$, $f_{ec}(5t, a)$ を計算すると

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

と評価できる. 実際には, $f(t)$ と $f(3t)$ を同じオーダーであるとして, e^{-2a} を相対誤差と考えると下記のように相対誤差の指数部はほぼ a の値に等しい.

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 ウェーブレット変換

(1) Haar 関数

ウェーブレット変換対象となる入力データの定義域を $[0, a]$ とする. Haar 関数 $H_{00}(x), H_{01}(x)$ を,

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

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

とする. $H_{01}(x)$ に対して $H_{mn}(x)$ を次のように作る. 区間 $[0, a]$ を 2^m の等しい長さの区間に分割する. 分割された小区間を小さい方から 1 からかぞえてその番号を n とする. この小区間を $[b1, b2]$ とすれば,

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

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

である. 小区間 $[b1, b2]$ において, Haar 関数 を,

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

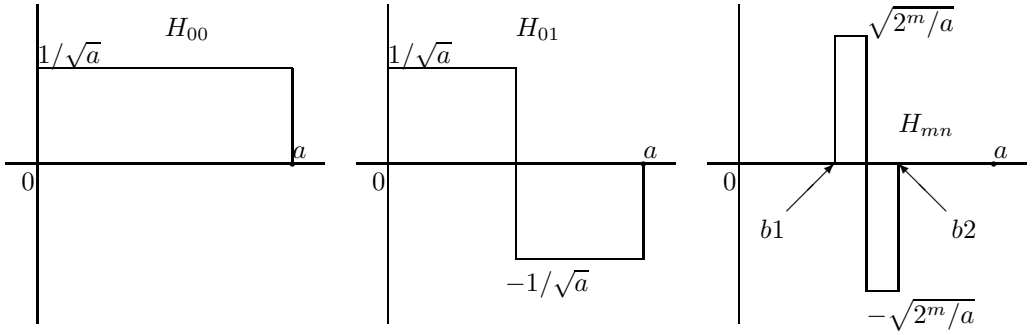
とする. 関数の値は区間 $[0, a]$ で規格化されるように選んでいる. すなわち,

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

また, 異なる m, m', n, n' の間では,

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

になっていることに注意する. すなわち, これらの H_{mn} は正規直交系をなしている. また, 以上の作り方からわかるように, n として指定できる数は m に対して $n = 1, \cdots, 2^m$ である.



(2) Haar 関数によるウェーブレット変換

Haar 関数によってウェーブレット変換を行うということは, 与えられた入力データと各 Haar 関数の積の積分を入力データの存在範囲にわたって行うということである. 入力データを仮に連続関数 $y = f(x)$ とすれば, ウェーブレット変換は

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

である. この計算を遂行するには Haar 関数 H_{mn} の (1) 値 y_H , (2) 立ち上がり点の位置 x_0, (3) 正負が逆転する点の位置 x_1 , (4) 値が 0 に戻る位置 x_2 , の情報が必要である. そのため, インデックス (m, n) を与えると, これらの Haar 関数 $H_{mn}(x)$ の情報を作るために, 本ライブラリでは関数 2.18.1 $\left\{\begin{matrix}\text{ASL\_dfwth1}\\ \text{ASL\_rfwth1}\end{matrix}\right\}$ を用意している. 入力データが連続関数の場合, これらの関数の結果を用いて数値積分でウェーブレット変換が可能である.

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

である. 入力データが離散的な場合には, ウェーブレット変換は次のようにして計算を行う. 離散的な入力データの値の範囲を「隣接するデータの位置との半分の範囲内で一定」と仮定する. すなわち, データ (x_i, y_i) が与えられたとき, このデータに対して $[\frac{1}{2}(x_{i-1} + x_i), \frac{1}{2}(x_i + x_{i+1})]$ において, 入力データは y_i を持つと仮定する. このようにして区分的に連続になった入力データに対して上記の積分を行う. もしもサンプリングが等間隔 dx で行われ, サンプリング数が 2^k (k: 自然数) である場合には, $H_{(k-1)n}$ の [b1, b2] に対して, x=b1+(b2-b1)/4, x=b2-(b2-b1)/4] のどちらかに入力データがあるようにできる. このような入力データに対しては Haar 関数は完全系であり, $H_{mn}, m = 0, 1, \cdots, (k-1), n = 1, 2, \cdots, 2^{k-1}$ の線形結合で入力データを完全に表すことができる. たとえば 2= 2^1 個のデータ (x1, y1), (x2, y2) があるとき,

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

は 2 つのデータの平均値である．ただし, Haar 関数が両端で変換データの存在範囲の外にでるので, 積分範囲を両側で $(b2-b1)/2$ だけ外に広げる．さらに

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

は 2 つのデータの, 平均値からのずれを表す．このずれは正負の差はあるものの, 絶対値は同じで, したがって H_{01} で完全に表現することができる．したがって,

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

で入力データそのものを表現することができる．

サンプリングが等間隔 dx で行われ, サンプリング数が 2^k (k: 自然数) である場合には, 積分の計算で個々の入力データの存在間隔を考慮せずに済むので, 計算が簡単になる．この積分計算を簡単に済ませるために, 入力データ数と同じ数 $na=2^k$ 個の配列に区間 $[0,1]$ で H_{mn} の正, 負, 0 を示す配列 lr を出力するため, 本ライブラリでは, 関数 2.18.4 $\left\{\begin{matrix}\text{ASL\_dfwth2}\\ \text{ASL\_rfwth2}\end{matrix}\right\}$ を用意している．入力データの存在範囲を $[b1,b2]$ としたとき, 上の計算の積分範囲は $[b1-dx/2,b2+dx/2]$ となるが, $a=(b2-b1)+dx$ として,

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

である．

(3) Haar 関数による逆ウェーブレット変換

Haar 関数のウェーブレット変換に対する逆変換は C_{mn} から元のデータを再構築する作業である．元のデータを $f(x)$ とすると, $f(x)$ の再構成は

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

である．連続値またはサンプリングの間隔が等間隔でない場合には $f'(x)$ は $f(x)$ には一致しないことは, この和が Haar 関数が m の最大値のときの, 値を持つ範囲の 1/2 以下の周波数の値を作らないことからわかる．サンプリング間隔が等間隔でサンプリング数が 2^k (k:自然数) の場合には, (m の最大値) $=k$ としてやれば元のデータを再現することができる．

(4) メキシカンハット関数

連続系ウェーブレット変換に用いられるメキシカンハット関数 $\varphi_{MH}(x)$ は以下の式で与えられる．

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

この関数は $[-\infty,+\infty]$ で値を持つ．周波数に対応するパラメータを a とし, b だけ x 軸方向にシフトするとして, ウェーブレット変換の基底を,

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

とする．ただし, C は

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

となるような規格化定数とする.

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

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

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

から, 規格化定数は

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

とした. 任意の関数 $f(x)$ に対して, この関数でのウェーブレット変換は

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

である.

(5) フレンチハット関数

フレンチハット関数 $\varphi_{FH}(x)$ は

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

で定義される. メキシカンハット関数と同様に, 周波数に対応するパラメータを a とし, b だけ x 軸方向にシフトするとして, ウェーブレット変換の基底を,

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

とする. ただし, C は

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

となるような規格化定数とする.

$$C = 3a$$

となる. 任意の関数 $f(x)$ に対して, この関数でのウェーブレット変換は

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

である.

2.1.3 参考文献

(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) 牧之内三郎, 鳥居達生, “数値解析”, オーム社 (1975).

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

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

(9) Temperton, C. , “Fast Mixed-Radix Real Fourier Transforms”, J. Comp. Phys. , 52, 340 (1983).

(10) 細野敏夫, “BASIC による高速ラプラス変換”, 共立出版 (1984).

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

(12) 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).

(13) 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).

(14) Childers, D. G. (Ed.), “Modern Spectrum Analysis”, IEEE Press (1978).

(15) 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).

(16) Swarztrauber, P. N. , “Vectorizing the FFTs”, Parallel Computations, 51 (1982).

(17) 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).

2.2 1 次元複素フーリエ変換 (実数引数型)

2.2.1 [非推奨]ASL_dfc1fb, ASL_rfc1fb
1 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ順変換 (任意基数) を行う.

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

逆変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	cr	{D∗ / R∗}	ld	入 力	入力データ c_k の実部 (注意事項 (b) 参照)
				出 力	出力データ d_j の実部 (注意事項 (b), (c) 参照)
3	ci	{D∗ / R∗}	ld	入 力	入力データ c_k の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_j の虚部 (注意事項 (b), (c) 参照)
4	ld	I	1	入 力	配列 cr, ci の大きさ
5	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含んだ順変換 isw=−1:初期化を含んだ逆変換
6	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照)
7	trigs	{D∗ / R∗}	2×n	出 力	三角関数テーブル (注意事項 (d) 参照)
8	wk	{D∗ / R∗}	2×n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 0$

(b) $\mathrm{n} \leq \mathrm{ld}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	入力時の内容がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, $\mathrm{n} = 289(=17^2)$ とするよりも $\mathrm{n} = 300(=2^2 \times 3 \times 5^2)$ や $320(=2^6 \times 5)$, $384(=2^7 \times 3)$ などとした方が通常は効率が良い.

(b) 複素数データ c_k $(k = 0, \cdots, n-1)$ の実部と虚部をそれぞれ $\Re\{c_k\}$, $\Im\{c_k\}$ とすると, c_k と配列 cr, ci の各要素は以下の様に対応する.

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

複素数データ $d_j (j = 0, \cdots, n-1)$ についても同様である.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_k (k = 0, \cdots, n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_k (k = 0, \cdots, n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.2.2 $\left\{\begin{array}{l}\mathrm{ASL\_dfc1bf}\\ \mathrm{ASL\_rfc1bf}\end{array}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.2.2 $\left\{\begin{array}{l}\mathrm{ASL\_dfc1bf}\\ \mathrm{ASL\_rfc1bf}\end{array}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 cr, ci に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔

を設定する必要がある．なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる．

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない．

(g) **非推奨** この機能は将来廃止予定である．より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい．

(7) **使用例**

2.2.2 (7) 使用例参照.

2.2.2 [非推奨]ASL_dfc1bf, ASL_rfc1bf
1 次元複素フーリエ変換 (初期化後の変換)

(1) 機 能

順変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ順変換 (任意基数) を行う.

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

逆変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ逆変換 (任意基数) を行う.

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

(2) 使用法

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	cr	{D∗ / R∗}	ld	入 力	入力データ c_k の実部 (注意事項 (b) 参照)
				出 力	出力データ d_j の実部 (注意事項 (b), (c) 参照)
3	ci	{D∗ / R∗}	ld	入 力	入力データ c_k の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_j の虚部 (注意事項 (b), (c) 参照)
4	ld	I	1	入 力	配列 cr, ci の大きさ
5	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
6	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照)
7	trigs	{D∗ / R∗}	2×n	入 力	三角関数テーブル (注意事項 (a) 参照)
8	wk	{D∗ / R∗}	2×n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) n > 0

(b) n ≤ ld

(c) isw ∈ {1, −1}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の内容がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 n の変換を繰り返し行う場合に初期化を含む変換 2.2.1 $\left\{\begin{matrix}\text{ASL\_dfc1fb}\\ \text{ASL\_rfc1fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ c_k $(k = 0, \cdots, n-1)$ の実部と虚部をそれぞれ $\Re\{c_k\}$, $\Im\{c_k\}$ とすると, c_k と配列 cr, ci の各要素は以下の様に対応する.

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

複素数データ $d_j (j = 0, \cdots, n-1)$ についても同様である.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_k (k = 0, \cdots, n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_k (k = 0, \cdots, n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

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

上記の数列を入力データとして, 1 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 cr, ci, n=16, ld=16, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 1 次元複素フーリエ変換 (複素引数型)

2.3.1 [非推奨]ASL_zfc1fb, ASL_cfc1fb
1 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ順変換 (任意基数) を行う.

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

逆変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	c	{Z∗ / C∗}	ld	入 力	入力データ c_k(注意事項 (b) 参照)
				出 力	出力データ d_j(注意事項 (b), (c) 参照)
3	ld	I	1	入 力	配列 c の大きさ
4	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含んだ順変換 isw=−1:初期化を含んだ逆変換
5	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照)
6	trigs	{D∗ / R∗}	2×n	出 力	三角関数テーブル (注意事項 (d) 参照)
7	wk	{Z∗ / C∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 0$

(b) $\mathrm{n} \leq \mathrm{ld}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	入力時の内容がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, $\mathrm{n} = 289(=17^2)$ とするよりも $\mathrm{n} = 300(=2^2 \times 3 \times 5^2)$ や $320(=2^6 \times 5)$, $384(=2^7 \times 3)$ などとした方が通常は効率が良い.

(b) 複素数データ $c_k(k = 0, \cdots, n-1)$ と配列 c の各要素は以下の様に対応する.

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

複素数データ $d_j(j = 0, \cdots, n-1)$ についても同様である.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_k(k = 0, \cdots, n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_k(k = 0, \cdots, n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.3.2 $\left\{\begin{array}{l}\mathrm{ASL\_zfc1bf}\\ \mathrm{ASL\_cfc1bf}\end{array}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.3.2 $\left\{\begin{array}{l}\mathrm{ASL\_zfc1bf}\\ \mathrm{ASL\_cfc1bf}\end{array}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 c に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔

を設定する必要がある．なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる．

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である．より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.3.2 (7) 使用例参照.

2.3.2 [非推奨]ASL_zfc1bf, ASL_cfc1bf 1 次元複素フーリエ変換 (初期化後の変換)

(1) 機 能

順変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ順変換 (任意基数) を行う.

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

逆変換

複素数データ $c_k(k = 0, \cdots, n-1)$ に対して, 複素フーリエ逆変換 (任意基数) を行う.

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

(2) 使用法

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	c	{Z∗ / C∗}	ld	入 力	入力データ c_k(注意事項 (b) 参照)
				出 力	出力データ d_j(注意事項 (b), (c) 参照)
3	ld	I	1	入 力	配列 c の大きさ
4	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
5	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照)
6	trigs	{D∗ / R∗}	2×n	入 力	三角関数テーブル (注意事項 (a) 参照)
7	wk	{Z∗ / C∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 0$

(b) $\mathrm{n} \leq \mathrm{ld}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	入力時の内容がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 n の変換を繰り返し行う場合に初期化を含む変換 2.3.1 $\left\{\begin{matrix}\mathrm{ASL\_zfc1fb}\\ \mathrm{ASL\_cfc1fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ $c_k(k = 0, \cdots, n-1)$ と配列 c の各要素は以下の様に対応する.

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

複素数データ $d_j(j = 0, \cdots, n-1)$ についても同様である.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_k(k = 0, \cdots, n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_k(k = 0, \cdots, n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

$$\begin{aligned}
c\,[0] &= 3.000+\sqrt{-1}\times\ 0.000\\
c\,[1] &= 2.786+\sqrt{-1}\times\ 0.725\\
c\,[2] &= 2.300+\sqrt{-1}\times\ 1.173\\
c\,[3] &= 1.792+\sqrt{-1}\times\ 1.327\\
c\,[4] &= 1.381+\sqrt{-1}\times\ 1.302\\
c\,[5] &= 1.080+\sqrt{-1}\times\ 1.197\\
c\,[6] &= 0.865+\sqrt{-1}\times\ 1.065\\
c\,[7] &= 0.711+\sqrt{-1}\times\ 0.930\\
c\,[8] &= 0.600+\sqrt{-1}\times\ 0.800\\
c\,[9] &= 0.519+\sqrt{-1}\times\ 0.679\\
c\,[10] &= 0.459+\sqrt{-1}\times\ 0.566\\
c\,[11] &= 0.415+\sqrt{-1}\times\ 0.461\\
c\,[12] &= 0.383+\sqrt{-1}\times\ 0.361\\
c\,[13] &= 0.360+\sqrt{-1}\times\ 0.267\\
c\,[14] &= 0.345+\sqrt{-1}\times\ 0.176\\
c\,[15] &= 0.336+\sqrt{-1}\times\ 0.087
\end{aligned}$$

上記の数列を入力データとして, 1 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 c, n=16, ld=16, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 1 次元実フーリエ変換

2.4.1 [非推奨]ASL_dfr1fb, ASL_rfr1fb
1 次元実フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

実数データ $r_k(k=0,\cdots,n-1)$ に対して, フーリエ順変換 (任意基数) の半周期分を求める.

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

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

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n-j} = c_j$ を満たす n 個の複素数データ $c_j(j=0,\cdots,n-1)$ についてその半周期分 $c_j(j=0,\cdots,\lfloor\frac{n}{2}\rfloor)$ を与えて以下のように定義されるフーリエ逆変換 (任意基数) を求める.

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

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ複素数 z の実部と虚部を表す. また, n が奇数のとき $\hat{c}_{\frac{n}{2}}=0$, n が偶数のとき $\hat{c}_{\frac{n}{2}} = c_{\frac{n}{2}}$ である.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	r	{D∗ / R∗}	ld	入 力	入力データ r_k(順変換), または c_j(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_j(順変換), または r_k(逆変換) (注意事項 (b), (c) 参照)
3	ld	I	1	入 力	配列 r の大きさ
4	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
5	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照)
6	trigs	{D∗ / R∗}	n	出 力	三角関数テーブル (注意事項 (d) 参照)
7	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 n が奇数の時, 大きさ n+1 n が偶数の時, 大きさ n+2
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $n > 0$

(b) n が奇数の時：
$n + 1 \leq ld$
n が偶数の時：
$n + 2 \leq ld$

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n = 289(=17^2) とするよりも n = 300(=$2^2\times3\times5^2$) や 320(=$2^6\times5$), 384(=$2^7\times3$) などとした方が通常は効率が良い.

(b) 実数データ $r_k(k=0,\cdots,n-1)$ と配列 r の各要素は以下の様に対応する.

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

なお, 逆変換を行った場合, n(=n) が奇数のとき r[n] = 0, n が偶数のとき r[n] = r[n + 1] = 0 となる. また, 実数データ $r_k(k=0,\cdots,n-1)$ を配列 r に入力する場合, r [n] 以降に対応する 0 を特に格納する必要はない.

複素数データ c_j $(j=0,\cdots,\lfloor\frac{n}{2}\rfloor)$ の実部と虚部をそれぞれ $\Re\{c_j\}$, $\Im\{c_j\}$ とすると, c_j と配列 r の各要素は以下の様に対応する. ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す.

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

ただし, n が奇数のとき, m=n+1, n が偶数のとき m=n+2 とする. 実フーリエ変換の性質より, n が奇数のとき $\Im\{c_0\}=0$, n が偶数のとき $\Im\{c_0\}=\Im\{c_{\frac{n}{2}}\}=0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, c_j $(j=\lfloor\frac{n}{2}\rfloor+1,\cdots,n-1)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_k(k=0,\cdots,n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_k(k=0,\cdots,n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.4.2 $\left\{\begin{array}{l}\mathrm{ASL\_dfr1bf}\\ \mathrm{ASL\_rfr1bf}\end{array}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.4.2 $\left\{\begin{array}{l}\mathrm{ASL\_dfr1bf}\\ \mathrm{ASL\_rfr1bf}\end{array}\right\}$ の入力としなければならない.

なお, isw=0 として初期化だけを行う場合には, 配列 r に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.4.2 (7) 使用例参照.

2.4.2 [非推奨]ASL_dfr1bf, ASL_rfr1bf 1 次元実フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

実数データ $r_k(k = 0, \cdots, n-1)$ に対して, フーリエ順変換 (任意基数) の半周期分を求める.

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

ここで $\lfloor x \rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

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

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n-j} = c_j$ を満たす n 個の複素数データ $c_j(j = 0, \cdots, n-1)$ についてその半周期分 $c_j(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor)$ を与えて以下のように定義されるフーリエ逆変換 (任意基数) を求める.

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

ここで $\lceil x \rceil$ は x 以上の最小の整数を, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ複素数 z の実部と虚部を表す. また, n が奇数のとき $\hat{c}_{\frac{n}{2}} = 0$, n が偶数のとき $\hat{c}_{\frac{n}{2}} = c_{\frac{n}{2}}$ である.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	入力データ数 n (注意事項 (a) 参照)
2	r	{ D∗ / R∗ }	ld	入 力	入力データ r_k(順変換), または c_j(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_j(順変換), または r_k(逆変換) (注意事項 (b), (c) 参照)
3	ld	I	1	入 力	配列 r の大きさ
4	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
5	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照)
6	trigs	{ D∗ / R∗ }	n	入 力	三角関数テーブル (注意事項 (a) 参照)
7	wk	{ D∗ / R∗ }	内容参照	ワーク	作業領域 n が奇数の時大きさ, n+1 n が偶数の時大きさ, n+2
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $n > 0$

(b) n が奇数の時：
$n + 1 \leq ld$
n が偶数の時：
$n + 2 \leq ld$

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$n = 1$ であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 n の変換を繰り返し行う場合に初期化を含む変換 2.4.1 $\left\{\begin{matrix}\text{ASL\_dfr1fb}\\ \text{ASL\_rfr1fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 実数データ $r_k(k=0,\cdots,n-1)$ と配列 r の各要素は以下の様に対応する.

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

なお, 逆変換を行った場合, n(=n) が奇数のとき r[n] = 0, n が偶数のとき r[n] = r[n + 1] = 0 となる. また, 実数データ $r_k(k=0,\cdots,n-1)$ を配列 r に入力する場合, r [n] 以降に対応する 0 を特に格納する必要はない.

複素数データ c_j $(j=0,\cdots,\lfloor\frac{n}{2}\rfloor)$ の実部と虚部をそれぞれ $\Re\{c_j\}$, $\Im\{c_j\}$ とすると, c_j と配列 r の各要素は以下の様に対応する. ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す.

$$\begin{aligned} \Re\{c_0\} &\leftrightarrow \text{r[0]}\\ \Im\{c_0\} &\leftrightarrow \text{r[1]}\\ \Re\{c_1\} &\leftrightarrow \text{r[2]}\\ \Im\{c_1\} &\leftrightarrow \text{r[3]}\\ \cdots &\cdots \cdots\\ \Re\{c_{\lfloor\frac{n}{2}\rfloor}\} &\leftrightarrow \text{r}[m-2]\\ \Im\{c_{\lfloor\frac{n}{2}\rfloor}\} &\leftrightarrow \text{r}[m-1] \quad (m = \text{n+1[n:奇数] または n+2[n:偶数]}) \end{aligned}$$

ただし, n が奇数のとき, m=n+1, n が偶数のとき m=n+2 とする. 実フーリエ変換の性質より, n が奇数のとき $\Im\{c_0\}=0$, n が偶数のとき $\Im\{c_0\}=\Im\{c_{\frac{n}{2}}\}=0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, c_j $(j=\lfloor\frac{n}{2}\rfloor+1,\cdots,n-1)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_k(k=0,\cdots,n-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_k(k=0,\cdots,n-1)$ とすると

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

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T=\frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

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

上記の数列を入力データとして, 1 次元実フーリエ順・逆変換を行う.

(b) 入力データ

配列 r, n=16, ld=18, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 多重 1 次元複素フーリエ変換 (実数引数型)

2.5.1 [非推奨]ASL_dfcmfb, ASL_rfcmfb
多重 1 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ順変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

逆変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ逆変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	cr	{D∗ / R∗}	内容参照	入 力	入力データ $c_{k,l}$ の実部 (注意事項 (b) 参照) **大きさ**: $\mathrm{incn} \times (\mathrm{n}-1) + \mathrm{incm} \times (\mathrm{m}-1) + 1$
				出 力	出力データ $d_{j,l}$ の実部 (注意事項 (b), (c) 参照)
4	ci	{D∗ / R∗}	内容参照	入 力	入力データ $c_{k,l}$ の虚部 (注意事項 (b) 参照) **大きさ**: $\mathrm{incn} \times (\mathrm{n}-1) + \mathrm{incm} \times (\mathrm{m}-1) + 1$
				出 力	出力データ $d_{j,l}$ の虚部 (注意事項 (b), (c) 参照)
5	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
6	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
7	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw=0: 初期化のみ isw=1: 初期化を含む順変換 isw=−1: 初期化を含む逆変換
8	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照).
9	trigs	{D∗ / R∗}	$2 \times \mathrm{n}$	出 力	三角関数テーブル (注意事項 (d) 参照)
10	wk	{D∗ / R∗}	$2 \times \mathrm{m} \times \mathrm{n}$	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $\mathrm{n} > 0$
$\mathrm{m} > 0$

(b) $\mathrm{incn} > 0$
$\mathrm{incm} > 0$

(c) $\mathrm{incn} \geq \mathrm{m} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$ または
$\mathrm{incm} \geq \mathrm{n} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$
(ただし, $\mathrm{gcm}(i, j)$ は i, j の最大公約数を表す.)

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 変換データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n = 289(=17^2) とするよりも n = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 複素数データ $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ の実部と虚部をそれぞれ $\Re\{c_{k,l}\}$, $\Im\{c_{k,l}\}$ とすると, $c_{k,l}$ と配列 cr, ci の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. 複素数データ $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ についても同様である. なお, 配列 cr と ci のデータを格納しない領域の値はこの関数の呼びだしで変更されない.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ とすると

$$\hat{c}_{k,l} = n c_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じ変換データ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.5.2 $\left\{\begin{matrix}\mathrm{ASL\_dfcmbf}\\ \mathrm{ASL\_rfcmbf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.5.2 $\left\{\begin{matrix}\mathrm{ASL\_dfcmbf}\\ \mathrm{ASL\_rfcmbf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 cr, ci に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔

を設定する必要がある．なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる．

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である．より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.5.2 (7) 使用例参照.

2.5.2 [非推奨]ASL_dfcmbf, ASL_rfcmbf
多重 1 次元複素フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ順変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

逆変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ逆変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	cr	{D∗ / R∗}	内容参照	入 力	入力データ $c_{k,l}$ の実部 (注意事項 (b) 参照) **大きさ**: $\mathrm{incn} \times (\mathrm{n}-1) + \mathrm{incm} \times (\mathrm{m}-1) + 1$
				出 力	出力データ $d_{j,l}$ の実部 (注意事項 (b), (c) 参照)
4	ci	{D∗ / R∗}	内容参照	入 力	入力データ $c_{k,l}$ の虚部 (注意事項 (b) 参照) **大きさ**: $\mathrm{incn} \times (\mathrm{n}-1) + \mathrm{incm} \times (\mathrm{m}-1) + 1$
				出 力	出力データ $d_{j,l}$ の虚部 (注意事項 (b), (c) 参照)
5	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
6	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
7	isw	I	1	入 力	処理スイッチ isw=1: 初期化後の順変換 isw=−1: 初期化後の逆変換
8	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照).
9	trigs	{D∗ / R∗}	$2 \times \mathrm{n}$	入 力	三角関数テーブル (注意事項 (a) 参照)
10	wk	{D∗ / R∗}	$2 \times \mathrm{m} \times \mathrm{n}$	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $\mathrm{n} > 0$
$\mathrm{m} > 0$

(b) $\mathrm{incn} > 0$
$\mathrm{incm} > 0$

(c) $\mathrm{incn} \geq \mathrm{m} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$ または
$\mathrm{incm} \geq \mathrm{n} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$
(ただし, $\mathrm{gcm}(i, j)$ は i, j の最大公約数を表す.)

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じ変換データ数 n の変換を繰り返し行う場合に初期化を含む変換 2.5.1 $\left\{\begin{array}{l}\text{ASL\_dfcmfb}\\ \text{ASL\_rfcmfb}\end{array}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ の実部と虚部をそれぞれ $\Re\{c_{k,l}\}$, $\Im\{c_{k,l}\}$ とすると, $c_{k,l}$ と配列 cr, ci の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. 複素数データ $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ についても同様である. なお, 配列 cr と ci のデータを格納しない領域の値はこの関数の呼びだしで変更されない.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ とすると

$$\hat{c}_{k,l} = nc_{k,l}\quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

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

上記の数列を入力データとして, 多重 1 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 cr, ci, n=8, m=4, incn=1, incm=9, isw=1 (順変換) および isw=−1 (逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 多重 1 次元複素フーリエ変換 (複素引数型)

2.6.1 [非推奨]ASL_zfcmfb, ASL_cfcmfb
多重 1 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ順変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

逆変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ逆変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	c	Z∗ / C∗	内容参照	入 力	入力データ $c_{k,l}$ (注意事項 (b) 参照) **大きさ**: $\mathrm{incn} \times (\mathrm{n}-1) + \mathrm{incm} \times (\mathrm{m}-1) + 1$
				出 力	出力データ $d_{j,l}$ (注意事項 (b), (c) 参照)
4	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
5	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw=0: 初期化のみ isw=1: 初期化を含む順変換 isw=−1: 初期化を含む逆変換
7	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照).
8	trigs	D∗ / R∗	2×n	出 力	三角関数テーブル (注意事項 (d) 参照)
9	wk	Z∗ / C∗	m × n	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $\mathrm{n} > 0$
　　$\mathrm{m} > 0$

(b) $\mathrm{incn} > 0$
　　$\mathrm{incm} > 0$

(c) $\mathrm{incn} \geq \mathrm{m} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$ または
　　$\mathrm{incm} \geq \mathrm{n} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$
　　(ただし, $\mathrm{gcm}(i, j)$ は i, j の最大公約数を表す.)

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) 変換データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n = 289(=17^2) とするよりも n = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 複素数データ $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ と配列 c の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. 複素数データ $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ についても同様である. なお, 配列 c のデータを格納しない領域の値はこの関数の呼びだしで変更されない.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ とすると

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じ変換データ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.6.2 $\left\{\begin{matrix}\mathrm{ASL\_zfcmbf}\\ \mathrm{ASL\_cfcmbf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.6.2 $\left\{\begin{matrix}\mathrm{ASL\_zfcmbf}\\ \mathrm{ASL\_cfcmbf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 c に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.6.2 (7) 使用例参照.

2.6.2 [非推奨]ASL_zfcmbf, ASL_cfcmbf
多重 1 次元複素フーリエ変換 (初期化後の変換)

(1) 機 能

順変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ順変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{-2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

逆変換

複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して, m 重 1 次元複素フーリエ逆変換 (任意基数) を行う.

$$d_{j,l} = \sum_{k=0}^{n-1} c_{k,l} e^{2\pi\sqrt{-1}\frac{jk}{n}} \quad (j = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

(2) 使用法

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	c	{Z∗ / C∗}	内容参照	入 力	入力データ $c_{k,l}$ (注意事項 (b) 参照) **大きさ**: incn × (n − 1) + incm × (m − 1) + 1
				出 力	出力データ $d_{j,l}$ (注意事項 (b), (c) 参照)
4	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
5	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ isw=1: 初期化後の順変換 isw=−1: 初期化後の逆変換
7	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照).
8	trigs	{D∗ / R∗}	2 × n	入 力	三角関数テーブル (注意事項 (a) 参照)
9	wk	{Z∗ / C∗}	m × n	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 0$
$\mathrm{m} > 0$

(b) $\mathrm{incn} > 0$
$\mathrm{incm} > 0$

(c) $\mathrm{incn} \geq \mathrm{m} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$ または
$\mathrm{incm} \geq \mathrm{n} \times \mathrm{gcm}(\mathrm{incn}, \mathrm{incm})$
(ただし, $\mathrm{gcm}(i, j)$ は i, j の最大公約数を表す.)

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じ変換データ数 n の変換を繰り返し行う場合に初期化を含む変換 2.6.1 $\left\{\begin{matrix}\mathrm{ASL\_zfcmfb}\\ \mathrm{ASL\_cfcmfb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ と配列 c の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. 複素数データ $d_{j,l}(j = 0, \cdots, n-1;\ l = 1, \cdots, m)$ についても同様である. なお, 配列 c のデータを格納しない領域の値はこの関数の呼びだしで変更されない.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ とすると

$$\hat{c}_{k,l} = nc_{k,l} \quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

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)

上記の数列を入力データとして, 多重 1 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 c, n=8, m=4, incn=1, incm=9, isw=1 (順変換) および isw=−1 (逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 多重 1 次元実フーリエ変換

2.7.1 [非推奨]ASL_dfrmfb, ASL_rfrmfb 多重 1 次元実フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

実数データ $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ に対して, m 重 1 次元フーリエ順変換 (任意基数) の半周期分を求める.

$$c_{j,l}=\sum_{k=0}^{n-1} r_{k,l}e^{-2\pi\sqrt{-1}\frac{jk}{n}}\quad (j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$$

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

$$c^*_{n-j,l}=c_{j,l}$$

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n-j,l}=c_{j,l}$ を満たす n 個の複素数データの組み $c_{j,l}(j=0,\cdots,n-1;\ l=1,\cdots,m)$ についてその半周期分 $c_{j,l}(j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$ を与えて以下のように定義される m 重 1 次元フーリエ逆変換 (任意基数) を求める.

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

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ複素数 z の実部と虚部を表す. また, n が奇数のとき $\hat{c}_{\frac{n}{2},l}=0$, n が偶数のとき $\hat{c}_{\frac{n}{2},l}=c_{\frac{n}{2},l}$ である.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	r	{D∗ / R∗}	内容参照	入 力	入力データ $r_{k,l}$(順変換), または $c_{j,l}$(逆変換) (注意事項 (b) 参照) **大きさ**: n が奇数の時, incn × (n) + incm × (m − 1) + 1 n が偶数の時, incn × (n + 1) + incm × (m − 1) + 1
				出 力	出力データ $c_{j,l}$(順変換), または $r_{k,l}$(逆変換) (注意事項 (b)(c) 参照)
4	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
5	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
7	ifax	I*	20	出 力	基数分け情報 (注意事項 (d) 参照)
8	trigs	{D∗ / R∗}	n	出 力	三角関数テーブル (注意事項 (d) 参照)
9	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: n が奇数の時, 大きさ (n + 1) × m n が偶数の時, 大きさ (n + 2) × m
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) n > 0
　　m > 0

(b) incn > 0
　　incm > 0

(c) incn ≥ m × gcm(incn, incm) または
　　n が奇数の時：
　　incm ≥ (n + 1) × gcm(incn, incm)
　　n が偶数の時：
　　incm ≥ (n + 2) × gcm(incn, incm)
　　(ただし, gcm(i, j) は i, j の最大公約数を表す.)

(d) isw ∈ {0, 1, −1}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) データ数 n の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n = 289(=17^2) とするよりも n = 300(=$2^2\times3\times5^2$) や 320(=$2^6\times5$), 384(=$2^7\times3$) などとした方が通常は効率が良い.

(b) 実数データ $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ と配列 r の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. なお, 逆変換を行った場合, n(=n) が奇数のとき r[incn∗n+incm∗(l−1)] = 0, n が偶数のとき r[incn∗n+incm∗(l−1)] = r[incn∗(n+1)+incm∗(l−1)] = 0 となる. 複素数データ $c_{j,l}$ $(j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$ の実部と虚部をそれぞれ $\Re\{c_{j,l}\}$, $\Im\{c_{j,l}\}$ とすると, $c_{j,l}$ と配列 r の各要素は以下の様に対応する. ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, n が奇数のとき $\Im\{c_{0,l}\}=0$, n が偶数のとき $\Im\{c_{0,l}\}=\Im\{c_{\frac{n}{2},l}\}=0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, $c_{j,l}\quad(j=\lfloor\frac{n}{2}\rfloor+1,\cdots,n-1;\ l=1,\cdots,m)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

$$c_{n-j,l}=c^*_{j,l}$$

ただし, z^* は複素数 z の共役複素数を表す.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ とすると

$$\hat{r}_{k,l}=nr_{k,l}\quad(k=0,\cdots,n-1;\ l=1,\cdots,m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じ変換データ数 n の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.7.2 $\left\{\begin{matrix}\text{ASL\_dfrmbf}\\ \text{ASL\_rfrmbf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.7.2 $\left\{\begin{matrix}\text{ASL_dfrmbf}\\ \text{ASL_rfrmbf}\end{matrix}\right\}$の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 r に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.7.2 (7) 使用例参照.

2.7.2 [非推奨]ASL_dfrmbf, ASL_rfrmbf 多重 1 次元実フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

実数データ $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ に対して, m 重 1 次元フーリエ順変換 (任意基数) の半周期分を求める.

$$c_{j,l}=\sum_{k=0}^{n-1} r_{k,l}e^{-2\pi\sqrt{-1}\frac{jk}{n}}\quad (j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$$

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

$$c^*_{n-j,l}=c_{j,l}$$

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n-j,l}=c_{j,l}$ を満たす n 個の複素数データの組み $c_{j,l}(j=0,\cdots,n-1;\ l=1,\cdots,m)$ についてその半周期分 $c_{j,l}(j=0,\cdots,\lfloor\frac{n}{2}\rfloor;\ l=1,\cdots,m)$ を与えて以下のように定義される m 重 1 次元フーリエ逆変換 (任意基数) を求める.

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

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ複素数 z の実部と虚部を表す. また, n が奇数のとき $\hat{c}_{\frac{n}{2},l}=0$, n が偶数のとき $\hat{c}_{\frac{n}{2},l}=c_{\frac{n}{2},l}$ である.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	変換データ数 n (注意事項 (a) 参照)
2	m	I	1	入 力	多重度 m
3	r	{ D∗ / R∗ }	内容参照	入 力	入力データ $r_{k,l}$(順変換), または $c_{j,l}$(逆変換) (注意事項 (b) 参照). **大きさ**: n が奇数の時, incn × (n) + incm × (m − 1) + 1 n が偶数の時, incn × (n + 1) + incm × (m − 1) + 1
				出 力	出力データ $c_{j,l}$(順変換), または $r_{k,l}$(逆変換) (注意事項 (b)(c) 参照)
4	incn	I	1	入 力	変換データの格納間隔 (注意事項 (b) 参照)
5	incm	I	1	入 力	変換データ間の格納間隔 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
7	ifax	I*	20	入 力	基数分け情報 (注意事項 (a) 参照)
8	trigs	{ D∗ / R∗ }	n	入 力	三角関数テーブル (注意事項 (a) 参照)
9	wk	{ D∗ / R∗ }	内容参照	ワーク	作業領域 **大きさ**: n が奇数の時, 大きさ (n + 1) × m n が偶数の時, 大きさ (n + 2) × m
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) n > 0
　　m > 0

(b) incn > 0
　　incm > 0

(c) incn ≥ m × gcm(incn, incm) または
　　n が奇数の時 :
　　incm ≥ (n + 1) × gcm(incn, incm)
　　n が偶数の時 :
　　incm ≥ (n + 2) × gcm(incn, incm)
　　(ただし, gcm(i, j) は i, j の最大公約数を表す.)

(d) isw ∈ {1, −1}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	入力時の情報がそのまま出力される.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じ変換データ数 n の変換を繰り返し行う場合に初期化を含む変換 2.7.1 $\left\{\begin{array}{l}\text{ASL\_dfrmfb}\\ \text{ASL\_rfrmfb}\end{array}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 実数データ $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ と配列 r の各要素は以下の様に対応する.

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

例えば, incn=1, incm=n とすると,

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

となり, 添え字 k について連続に詰めて格納することになり incn=m, incm=1 とすると,

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

となり, 添え字 l について連続に詰めて格納することになる. なお, 逆変換を行った場合, n(=n) が奇数のとき r[incn∗n+incm∗(l−1)] = 0, n が偶数のとき r[incn∗n+incm∗(l−1)] = r[incn∗(n+1)+incm∗(l−1)] = 0 となる. 複素数データ $c_{j,l}$ $(j = 0, \cdots, \lfloor \frac{n}{2} \rfloor;\ l = 1, \cdots, m)$ の実部と虚部をそれぞれ $\Re\{c_{j,l}\}$, $\Im\{c_{j,l}\}$ とすると, $c_{j,l}$ と配列 r の各要素は以下の様に対応する. ここで $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, n が奇数のとき $\Im\{c_{0,l}\} = 0$, n が偶数のとき $\Im\{c_{0,l}\} = \Im\{c_{\frac{n}{2},l}\} = 0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, $c_{j,l}\quad (j = \lfloor \frac{n}{2} \rfloor + 1, \cdots, n-1;\ l = 1, \cdots, m)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

$$c_{n-j,l} = c^*_{j,l}$$

ただし, z^* は複素数 z の共役複素数を表す.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k,l}(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ とすると

$$\hat{r}_{k,l} = n r_{k,l}\quad (k = 0, \cdots, n-1;\ l = 1, \cdots, m)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

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

上記の数列を入力データとして, 多重 1 次元実フーリエ順・逆変換を行う.

(b) 入力データ

配列 r, n=8, m=4, incn=1, incm=12, isw=1 (順変換) および isw=−1 (逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 2 次元複素フーリエ変換 (実数引数型)

2.8.1 [非推奨]ASL_dfc2fb, ASL_rfc2fb
2 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_dfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	cr	{ D∗ / R∗ }	lx×ly	入 力	入力データ c_{k_x,k_y} の実部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} の実部 (注意事項 (b), (c) 参照)
4	ci	{ D∗ / R∗ }	lx×ly	入 力	入力データ c_{k_x,k_y} の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} の虚部 (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 cr, ci の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 cr, ci の第 2 寸法 (注意事項 (b) 参照)
7	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
8	ifax	I*	40	出 力	基数分け情報 (注意事項 (d) 参照)
9	trigs	{ D∗ / R∗ }	2 × (nx + ny)	出 力	三角関数テーブル (注意事項 (d) 参照)
10	wk	{ D∗ / R∗ }	2 × lx × ly	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $nx > 1$
　　$ny > 1$

(b) $nx \leq lx$
　　$ny \leq ly$

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx や ny の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 複素数データ c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ の実部と虚部をそれぞれ $\Re\{c_{k_x,k_y}\}$, $\Im\{c_{k_x,k_y}\}$ とすると, c_{k_x,k_y} と配列 cr, ci の各要素は以下の様に対応する.

$$\begin{aligned} \Re\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{cr}[k_x + \mathrm{lx} * k_y] \\ \Im\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{ci}[k_x + \mathrm{lx} * k_y] \end{aligned}$$

複素数データ $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ についても同様である.
なお, 主記憶のバンク競合を避けるために配列 cr, ci の整合寸法 lx, ly は奇数に設定するのが望ましい. 通常, たとえば nx が偶数の時は lx=nx+1 とする.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.8.2 $\left\{\begin{matrix}\mathrm{ASL\_dfc2bf}\\ \mathrm{ASL\_rfc2bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.8.2 $\left\{\begin{matrix}\mathrm{ASL\_dfc2bf}\\ \mathrm{ASL\_rfc2bf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 cr, ci に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.8.2 (7) 使用例参照.

2.8.2 [非推奨]ASL_dfc2bf, ASL_rfc2bf
2 次元複素フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_dfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	cr	{D∗ / R∗}	lx×ly	入 力	入力データ c_{k_x,k_y} の実部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} の実部 (注意事項 (b), (c) 参照)
4	ci	{D∗ / R∗}	lx×ly	入 力	入力データ c_{k_x,k_y} の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} の虚部 (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 cr, ci の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 cr, ci の第 2 寸法 (注意事項 (b) 参照)
7	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
8	ifax	I*	40	入 力	基数分け情報 (注意事項 (a) 参照)
9	trigs	{D∗ / R∗}	2 × (nx + ny)	入 力	三角関数テーブル (注意事項 (a) 参照)
10	wk	{D∗ / R∗}	2 × lx × ly	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $nx > 1$
　　$ny > 1$

(b) $nx \leq lx$
　　$ny \leq ly$

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny) の変換を繰り返し行う場合に初期化を含む変換 2.8.1 $\left\{\begin{matrix}\text{ASL\_dfc2fb}\\ \text{ASL\_rfc2fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ の実部と虚部をそれぞれ $\Re\{c_{k_x,k_y}\}$, $\Im\{c_{k_x,k_y}\}$ とすると, c_{k_x,k_y} と配列 cr, ci の各要素は以下の様に対応する.

$$\begin{aligned}\Re\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{cr}[k_x + \mathrm{lx} * k_y]\\ \Im\{c_{k_x,k_y}\} &\leftrightarrow \mathrm{ci}[k_x + \mathrm{lx} * k_y]\end{aligned}$$

複素数データ $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ についても同様である.
なお, 主記憶のバンク競合を避けるために配列 cr, ci の整合寸法 lx, ly は奇数に設定するのが望ましい. 通常, たとえば nx が偶数の時は lx=nx+1 とする.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y}\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

$$\begin{aligned}c_{k_x,k_y} &= (k_x + 1) + (k_y + 1) + \sqrt{-1}\frac{(k_x + 1)(k_y + 1)}{n_x n_y}\\ &(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)\end{aligned}$$

を入力データとして, 2 次元複素フーリエ順・逆変換を行う.

(b) 入力データ
配列 cr, ci, nx=5, ny=4, lx=5, ly=5, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 2 次元複素フーリエ変換 (複素引数型)

2.9.1 [非推奨]ASL_zfc2fb, ASL_cfc2fb
2 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_zfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

単精度関数:

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

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	c	{Z∗ / C∗}	lx×ly	入 力	入力データ c_{k_x,k_y} (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} (注意事項 (b), (c) 参照)
4	lx	I	1	入 力	配列 c の整合寸法 (注意事項 (b) 参照)
5	ly	I	1	入 力	配列 c の第 2 寸法 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
7	ifax	I*	40	出 力	基数分け情報 (注意事項 (d) 参照)
8	trigs	{D∗ / R∗}	2 × (nx + ny)	出 力	三角関数テーブル (注意事項 (d) 参照)
9	wk	{Z∗ / C∗}	lx × ly	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nx} > 1$
$\mathrm{ny} > 1$

(b) $\mathrm{nx} \leq \mathrm{lx}$
$\mathrm{ny} \leq \mathrm{ly}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx や ny の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, $\mathrm{nx} = 289(=17^2)$ とするよりも $\mathrm{nx} = 300(=2^2 \times 3 \times 5^2)$ や $320(=2^6 \times 5)$, $384(=2^7 \times 3)$ などとした方が通常は効率が良い.

(b) 複素数データ c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ と配列 c の各要素は以下の様に対応する.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{c}[k_x + \mathrm{lx} * k_y]$$

複素数データ $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ についても同様である.
なお, 主記憶のバンク競合を避けるために配列 c の整合寸法 lx, ly は奇数に設定するのが望ましい. 通常, たとえば nx が偶数の時は lx=nx+1 とする.

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.9.2 $\left\{\begin{matrix}\mathrm{ASL\_zfc2bf}\\ \mathrm{ASL\_cfc2bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.9.2 $\left\{\begin{matrix}\mathrm{ASL\_zfc2bf}\\ \mathrm{ASL\_cfc2bf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 c に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や

標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.9.2 (7) 使用例参照.

2.9.2 [非推奨]ASL_zfc2bf, ASL_cfc2bf 2 次元複素フーリエ変換 (初期化後の変換)

(1) 機 能

順変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

2 次元複素データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して, 2 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) 使用法

倍精度関数:

ierr = ASL_zfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	c	{Z∗ / C∗}	lx×ly	入 力	入力データ c_{k_x,k_y} (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y} (注意事項 (b), (c) 参照)
4	lx	I	1	入 力	配列 c の整合寸法 (注意事項 (b) 参照)
5	ly	I	1	入 力	配列 c の第 2 寸法 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
7	ifax	I*	40	入 力	基数分け情報 (注意事項 (a) 参照)
8	trigs	{D∗ / R∗}	2 × (nx + ny)	入 力	三角関数テーブル (注意事項 (a) 参照)
9	wk	{Z∗ / C∗}	lx × ly	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nx >1
ny >1

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

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny) の変換を繰り返し行う場合に初期化を含む変換 2.9.1 $\left\{\begin{matrix}\text{ASL\_zfc2fb}\\ \text{ASL\_cfc2fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ c_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ と配列 c の各要素は以下の様に対応する.

$$c_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{c}[k_x + \mathrm{lx} * k_y]$$

複素数データ $d_{j_x,j_y}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ についても同様である.
なお, 主記憶のバンク競合を避けるために配列 c **の整合寸法** lx, ly **は奇数に設定するのが望ましい. 通常, たとえば** nx **が偶数の時は** lx=nx+1 **とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{c}_{k_x,k_y} = n_x n_y c_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

$$c_{k_x,k_y} = (k_x+1)+(k_y+1)+\sqrt{-1}\frac{(k_x+1)(k_y+1)}{n_x n_y}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$$

を入力データとして, 2 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 c, nx=5, ny=4, lx=5, ly=5, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 2 次元実フーリエ変換

2.10.1 [非推奨]ASL_dfr2fb, ASL_rfr2fb
2 次元実フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

2 次元実数データ $r_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$ に対して, 2 次元フーリエ順変換 (任意基数) の j_x についての半周期分を求める.

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$, $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$ を満たす $n_x n_y$ 個の複素数データ $c_{j_x,j_y}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$ について j_x についての半周期分 $c_{j_x,j_y}(j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1)$ を与えて以下のように定義される 2 次元フーリエ逆変換 (任意基数) を求める.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1}\{c_{0,j_y} + (-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y}\}e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})}\} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1) \end{aligned}$$

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ は複素数 z の実部を表す. また, n_x が奇数のとき $\hat{c}_{\frac{n_x}{2},j_y} = 0$, n_x が偶数のとき $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$ である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfr2fb  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　R:単精度実数型　C:単精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	r	{D∗ / R∗}	lx×ly	入 力	入力データ r_{k_x,k_y}(順変換), または c_{j_x,j_y}(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_{j_x,j_y}(順変換), または r_{k_x,k_y}(逆変換) (注意事項 (b), (c) 参照)
4	lx	I	1	入 力	配列 r の整合寸法 (注意事項 (b) 参照)
5	ly	I	1	入 力	配列 r の第 2 寸法 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
7	ifax	I*	40	出 力	基数分け情報 (注意事項 (d) 参照)
8	trigs	{D∗ / R∗}	nx+2×ny	出 力	三角関数テーブル (注意事項 (d) 参照)
9	wk	{D∗ / R∗}	lx × ly	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $\mathrm{nx} > 1$
$\mathrm{ny} > 1$

(b) nx が奇数の時：
$\mathrm{nx} + 1 \leq \mathrm{lx}$
nx が偶数の時：
$\mathrm{nx} + 2 \leq \mathrm{lx}$

(c) $\mathrm{ny} \leq \mathrm{ly}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) または (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx や ny の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 実数データ r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ と配列 r の各要素は以下の様に対応する.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{r}[k_x + \mathrm{lx} * k_y]$$

なお, 逆変換を行った場合, nx(=n_x) が奇数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * k_y] = 0$, nx が偶数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * k_y] = \mathrm{r}[\mathrm{nx} + 1 + \mathrm{lx} * k_y] = 0$ となる. また, 実数データ $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ を配列 r に入力する場合, 上述の対応する 0 を特に格納する必要はない.
複素数データ $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ の実部と虚部をそれぞれ $\Re\{c_{j_x,j_y}\}$, $\Im\{c_{j_x,j_y}\}$ とすると, c_{j_x,j_y} と配列 r の各要素は以下の様に対応する. ここで $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, $\Im\{c_{0,0}\} = 0$ であり, nx と ny が共に偶数であれば, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly が奇数になるように設定するのが望ましい. 通常, たとえば nx が (4 の倍数)+2 の時は lx=nx+4 とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.10.2 $\left\{\begin{matrix}\mathrm{ASL\_dfr2bf} \\ \mathrm{ASL\_rfr2bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのま

ま 2.10.2 $\left\{\begin{matrix}\text{ASL_dfr2bf}\\ \text{ASL_rfr2bf}\end{matrix}\right\}$の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 r に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.10.2 (7) 使用例参照.

2.10.2 [非推奨]ASL_dfr2bf, ASL_rfr2bf 2 次元実フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

2 次元実数データ $r_{k_x,k_y}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1)$ に対して, 2 次元フーリエ順変換 (任意基数) の j_x についての半周期分を求める.

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下の関係から得られる.

$$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$$
$$c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$$

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n_x-j_x,n_y-j_y} = c_{j_x,j_y}$, $c^*_{n_x-j_x,j_y} = c_{j_x,n_y-j_y}$ を満たす $n_x n_y$ 個の複素数データ $c_{j_x,j_y}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1)$ について j_x についての半周期分 $c_{j_x,j_y}(j_x = 0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y = 0,\cdots,n_y-1)$ を与えて以下のように定義される 2 次元フーリエ逆変換 (任意基数) を求める.

$$\begin{aligned} r_{k_x,k_y} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1} c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})} \\ &= \sum_{j_y=0}^{n_y-1}\{c_{0,j_y} + (-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y}\} e^{2\pi\sqrt{-1}\frac{j_y k_y}{n_y}} + 2\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y} e^{2\pi\sqrt{-1}(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y})}\} \\ &\quad (k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1) \end{aligned}$$

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ は複素数 z の実部を表す. また, n_x が奇数のとき $\hat{c}_{\frac{n_x}{2},j_y} = 0$, n_x が偶数のとき $\hat{c}_{\frac{n_x}{2},j_y} = c_{\frac{n_x}{2},j_y}$ である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfr2bf  (nx, ny, r, lx, ly, isw, ifax, trigs, wk);
```

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	r	{ D∗ / R∗ }	lx×ly	入 力	入力データ r_{k_x,k_y}(順変換), または c_{j_x,j_y}(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_{j_x,j_y}(順変換), または r_{k_x,k_y}(逆変換) (注意事項 (b), (c) 参照)
4	lx	I	1	入 力	配列 r の整合寸法 (注意事項 (b) 参照)
5	ly	I	1	入 力	配列 r の第 2 寸法 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
7	ifax	I*	40	入 力	基数分け情報 (注意事項 (a) 参照)
8	trigs	{ D∗ / R∗ }	nx+2×ny	入 力	三角関数テーブル (注意事項 (a) 参照)
9	wk	{ D∗ / R∗ }	lx × ly	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) nx > 1
　　ny > 1

(b) nx が奇数の時 :
　　nx $+ 1 \leq$ lx
　　nx が偶数の時 :
　　nx $+ 2 \leq$ lx

(c) ny $\leq$ ly

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) または (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny) の変換を繰り返し行う場合に初期化を含む変換
2.10.1 $\left\{\begin{matrix}\text{ASL\_dfr2fb}\\ \text{ASL\_rfr2fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 実数データ r_{k_x,k_y} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ と配列 r の各要素は以下の様に対応する.

$$r_{k_x,k_y} \quad \leftrightarrow \quad \mathrm{r}[k_x + \mathrm{lx} * k_y]$$

なお, 逆変換を行った場合, nx(=n_x) が奇数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * k_y] = 0$, nx が偶数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * k_y] = \mathrm{r}[\mathrm{nx} + 1 + \mathrm{lx} * k_y] = 0$ となる. また, 実数データ $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ を配列 r に入力する場合, 上述の対応する 0 を特に格納する必要はない.
複素数データ $c_{j_x,j_y}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1)$ の実部と虚部をそれぞれ $\Re\{c_{j_x,j_y}\}$, $\Im\{c_{j_x,j_y}\}$ とすると, c_{j_x,j_y} と配列 r の各要素は以下の様に対応する. ここで $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, $\Im\{c_{0,0}\} = 0$ であり, nx と ny が共に偶数であれば, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2}}\} = 0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, c_{j_x,j_y} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ の各要素は実フーリエ変換の対称性から以下の関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly が奇数になるように設定するのが望ましい. 通常, たとえば nx が (4 の倍数)+2 の時は lx=nx+4 とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k_x,k_y}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$ とすると

$$\hat{r}_{k_x,k_y} = n_x n_y r_{k_x,k_y} \quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ とれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

$$r_{k_x,k_y} = \frac{n_x + n_y}{(k_x+1)+(k_y+1)}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1)$$

を入力データとして, 2 次元実フーリエ順・逆変換を行う.

(b) 入力データ

配列 r, nx=6, ny=4, lx=10, ly=5, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 3 次元複素フーリエ変換 (実数引数型)

2.11.1 [非推奨]ASL_dfc3fb, ASL_rfc3fb
3 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_dfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	cr	{D∗ / R∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} の実部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} の実部 (注意事項 (b), (c) 参照)
5	ci	{D∗ / R∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} の虚部 (注意事項 (b), (c) 参照)
6	lx	I	1	入 力	配列 cr, ci の整合寸法 (注意事項 (b) 参照)
7	ly	I	1	入 力	配列 cr, ci の第 2 寸法 (注意事項 (b) 参照)
8	lz	I	1	入 力	配列 cr, ci の第 3 寸法 (注意事項 (b) 参照)
9	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
10	ifax	I*	60	出 力	基数分け情報 (注意事項 (d) 参照)
11	trigs	{D∗ / R∗}	2 × (nx + ny + nz)	出 力	三角関数テーブル (注意事項 (d) 参照)
12	wk	{D∗ / R∗}	2×lx×ly×lz	ワーク	作業領域
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx, ny や nz の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 複素数データ c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ の実部と虚部をそれぞれ $\Re\{c_{k_x,k_y,k_z}\}$, $\Im\{c_{k_x,k_y,k_z}\}$ とすると, c_{k_x,k_y,k_z} と配列 cr, ci の各要素は以下の様に対応する.

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

複素数データ $d_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ についても同様である. **なお, 主記憶のバンク競合を避けるために配列** cr, ci **の整合寸法** lx, ly, lz **は奇数に設定するのが望ましい. また, 高速化のために配列** cr, ci **内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば** nx **が偶数の時は** lx=nx+1 **とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ とすると

$$\begin{aligned} &\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z} \\ &\quad (k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1) \end{aligned}$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.11.2 $\left\{\begin{matrix}\mathrm{ASL\_dfc3bf}\\ \mathrm{ASL\_rfc3bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.11.2 $\left\{\begin{matrix}\mathrm{ASL\_dfc3bf}\\ \mathrm{ASL\_rfc3bf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 cr, ci に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.11.2 (7) 使用例参照.

2.11.2 [非推奨]ASL_dfc3bf, ASL_rfc3bf 3 次元複素フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_dfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	cr	{D∗ / R∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} の実部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} の実部 (注意事項 (b), (c) 参照)
5	ci	{D∗ / R∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} の虚部 (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} の虚部 (注意事項 (b), (c) 参照)
6	lx	I	1	入 力	配列 cr, ci の整合寸法 (注意事項 (b) 参照)
7	ly	I	1	入 力	配列 cr, ci の第 2 寸法 (注意事項 (b) 参照)
8	lz	I	1	入 力	配列 cr, ci の第 3 寸法 (注意事項 (b) 参照)
9	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
10	ifax	I*	60	入 力	基数分け情報 (注意事項 (d) 参照)
11	trigs	{D∗ / R∗}	2 × (nx + ny + nz)	入 力	三角関数テーブル (注意事項 (d) 参照)
12	wk	{D∗ / R∗}	2×lx×ly×lz	ワーク	作業領域
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合に初期化を含む変換
2.11.1 $\left\{\begin{matrix}\text{ASL\_dfc3fb}\\ \text{ASL\_rfc3fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ c_{k_x,k_y,k_z} $(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ の実部と虚部をそれぞれ $\Re\{c_{k_x,k_y,k_z}\}$, $\Im\{c_{k_x,k_y,k_z}\}$ とすると, c_{k_x,k_y,k_z} と配列 cr, ci の各要素は以下の様に対応する.

$$\begin{aligned}\Re\{c_{k_x,k_y,k_z}\} &\leftrightarrow \text{cr}[k_x+\text{lx}*(k_y+\text{ly}*k_z)]\\ \Im\{c_{k_x,k_y,k_z}\} &\leftrightarrow \text{ci}[k_x+\text{lx}*(k_y+\text{ly}*k_z)]\end{aligned}$$

複素数データ $d_{j_x,j_y,j_z}(j_x=0,\cdots,n_x-1;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ についても同様である. **なお, 主記憶のバンク競合を避けるために配列** cr, ci **の整合寸法** lx, ly, lz **は奇数に設定するのが望ましい. また, 高速化のために配列** cr, ci **内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば** nx **が偶数の時は** lx=nx+1 **とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ とすると

$$\begin{aligned}&\hat{c}_{k_x,k_y,k_z}=n_xn_yn_zc_{k_x,k_y,k_z}\\ &(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)\end{aligned}$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T=\frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

(a) 問 題

$$\begin{aligned}&c_{k_x,k_y,k_z}=\frac{n_x+n_y+n_z}{(k_x+1)+(k_y+1)+(k_z+1)}+\sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_xn_yn_z}\\ &(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)\end{aligned}$$

を入力データとして, 3 次元複素フーリエ順・逆変換を行う.

(b) 入力データ
配列 cr, ci, nx=5, ny=4, nz=3, lx=5, ly=5, lz=3, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 3 次元複素フーリエ変換 (複素引数型)

2.12.1 [非推奨]ASL_zfc3fb, ASL_cfc3fb
3 次元複素フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_zfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

単精度関数:

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

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	c	{Z∗ / C∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 c の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 c の第 2 寸法 (注意事項 (b) 参照)
7	lz	I	1	入 力	配列 c の第 3 寸法 (注意事項 (b) 参照)
8	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
9	ifax	I*	60	出 力	基数分け情報 (注意事項 (d) 参照)
10	trigs	{D∗ / R∗}	2 × (nx + ny + nz)	出 力	三角関数テーブル (注意事項 (d) 参照)
11	wk	{Z∗ / C∗}	lx × ly × lz	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx, ny や nz の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(b) 複素数データ c_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ と配列 c の各要素は以下の様に対応する.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{c}[k_x + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)]$$

複素数データ $d_{j_x,j_y,j_z}(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ についても同様である. **なお, 主記憶のバンク競合を避けるために配列 c の整合寸法 lx, ly, lz は奇数に設定するのが望ましい. また, 高速化のために配列 c 内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば nx が偶数の時は lx=nx+1 とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ とすると

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.12.2 $\left\{\begin{matrix}\mathrm{ASL\_zfc3bf}\\ \mathrm{ASL\_cfc3bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.12.2 $\left\{\begin{matrix}\mathrm{ASL\_zfc3bf}\\ \mathrm{ASL\_cfc3bf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 c に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.12.2 (7) 使用例参照.

2.12.2 [非推奨]ASL_zfc3bf, ASL_cfc3bf 3 次元複素フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ順変換 (任意基数) を行う.

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

逆変換

3 次元複素データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して, 3 次元複素フーリエ逆変換 (任意基数) を行う.

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

(2) **使用法**

倍精度関数:

ierr = ASL_zfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

単精度関数:

ierr = ASL_cfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	c	{Z∗ / C∗}	lx×ly×lz	入 力	入力データ c_{k_x,k_y,k_z} (注意事項 (b) 参照)
				出 力	出力データ d_{j_x,j_y,j_z} (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 c の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 c の第 2 寸法 (注意事項 (b) 参照)
7	lz	I	1	入 力	配列 c の第 3 寸法 (注意事項 (b) 参照)
8	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
9	ifax	I*	60	入 力	基数分け情報 (注意事項 (a) 参照)
10	trigs	{D∗ / R∗}	2 × (nx + ny + nz)	入 力	三角関数テーブル (注意事項 (a) 参照)
11	wk	{Z∗ / C∗}	lx × ly × lz	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合に初期化を含む変換 2.12.1 $\left\{\begin{matrix}\text{ASL\_zfc3fb}\\ \text{ASL\_cfc3fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 複素数データ c_{k_x,k_y,k_z} $(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ と配列 c の各要素は以下の様に対応する.

$$c_{k_x,k_y,k_z} \leftrightarrow \mathrm{c}[k_x + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)]$$

複素数データ $d_{j_x,j_y,j_z}(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ についても同様である. **なお, 主記憶のバンク競合を避けるために配列** c **の整合寸法** lx, ly, lz **は奇数に設定するのが望ましい. また, 高速化のために配列** c **内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば** nx **が偶数の時は** lx=nx+1 **とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 複素数データ $c_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ に対して順変換を行い引き続き逆変換を行ったデータを $\hat{c}_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ とすると

$$\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

$$c_{k_x,k_y,k_z} = \frac{n_x+n_y+n_z}{(k_x+1)+(k_y+1)+(k_z+1)} + \sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

を入力データとして, 3 次元複素フーリエ順・逆変換を行う.

(b) 入力データ

配列 c, nx=5, ny=4, nz=3, lx=5, ly=5, lz=3, isw=1 (順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 3 次元実フーリエ変換

2.13.1 [非推奨]ASL_dfr3fb, ASL_rfr3fb 3 次元実フーリエ変換 (初期化を含む変換)

(1) **機 能**

順変換

3 次元実数データ $r_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ に対して, 3 次元フーリエ順変換 (任意基数) の j_x についての半周期分を求める.

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下のような関係から得られる.

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

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n_x-j_x,n_y-j_y,n_z-j_z}=c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z}=c_{j_x,n_y-j_y,n_z-j_z}$, $c^*_{n_x-j_x,n_y-j_y,j_z}=c_{j_x,j_y,n_z-j_z}$ 等の関係を満たす 3 次元実フーリエ変換後の $n_xn_yn_z$ 個の複素数データ $c_{j_x,j_y,j_z}(j_x=0,\cdots,n_x-1;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ について j_x についての半周期分 $c_{j_x,j_y,j_z}(j_x=0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ を与えて以下のように定義される 3 次元フーリエ逆変換 (任意基数) を求める.

$$
\begin{aligned}
r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})} \\
&= \sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\{c_{0,j_y,j_z}+(-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y,j_z}\}e^{2\pi\sqrt{-1}(\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})} \\
&\quad + 2\sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y,j_z}e^{2\pi\sqrt{-1}(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})}\} \\
&\quad (k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)
\end{aligned}
$$

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ は複素数 z の実部を表す. また, n_x が奇数のとき $\hat{c}_{\frac{n_x}{2},j_y,j_z}=0$, n_x が偶数のとき $\hat{c}_{\frac{n_x}{2},j_y,j_z}=c_{\frac{n_x}{2},j_y,j_z}$ である.

(2) **使用法**

倍精度関数:

ierr = ASL_dfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);

単精度関数:

ierr = ASL_rfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	r	D∗ / R∗	lx×ly×lz	入 力	入力データ r_{k_x,k_y,k_z}(順変換), または c_{j_x,j_y,j_z}(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_{j_x,j_y,j_z}(順変換), または r_{k_x,k_y,k_z}(逆変換) (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 r の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 r の第 2 寸法 (注意事項 (b) 参照)
7	lz	I	1	入 力	配列 r の第 3 寸法 (注意事項 (b) 参照)
8	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:初期化のみ isw= 1:初期化を含む順変換 isw=−1:初期化を含む逆変換
9	ifax	I*	60	出 力	基数分け情報 (注意事項 (d) 参照)
10	trigs	D∗ / R∗	nx + 2 × (ny + nz)	出 力	三角関数テーブル (注意事項 (d) 参照)
11	wk	D∗ / R∗	lx × ly × lz	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(b) nx が奇数の時：
$nx + 1 \leq lx$
nx が偶数の時：
$nx + 2 \leq lx$

(c) $ny \leq ly$

(d) $nz \leq lz$

(e) lx は偶数である.

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b), (c) または (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	
3030	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) データ数 nx, ny や nz の値を調整できる場合には, 混合基数 FFT アルゴリズムが有効に働く数 (2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx $= 289(=17^2)$ とするよりも nx $= 300(=2^2 \times 3 \times 5^2)$ や $320(=2^6 \times 5)$, $384(=2^7 \times 3)$ などとした方が通常は効率が良い.

(b) 実数データ r_{k_x,k_y,k_z} $(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ と配列 r の各要素は以下の様に対応する.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{r}[k_x + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)]$$

なお, 逆変換を行った場合, nx$(=n_x)$ が奇数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)] = 0$, nx が偶数のとき $\mathrm{r}[\mathrm{nx} + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)] = \mathrm{r}[\mathrm{nx} + 1 + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)] = 0$ となる. また, 実数データ $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ を配列 r に入力する場合, 上述の対応する 0 を特に格納する必要はない.
複素数データ $c_{j_x,j_y,j_z}(j_x = 0, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ の実部と虚部をそれぞれ $\Re\{c_{j_x,j_y,j_z}\}$, $\Im\{c_{j_x,j_y,j_z}\}$ とすると, c_{j_x,j_y,j_z} と配列 r の各要素は以下の様に対応する. ここで $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, $\Im\{c_{0,0,0}\} = 0$ であり, nx, ny と nz が共に偶数であれば, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\} = 0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, c_{j_x,j_y,j_z} $(j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1;\ j_z = 0, \cdots, n_z - 1)$ の各要素は実フーリエ変換の対称性から以下のような関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly, lz が奇数になるように設定するのが望ましい. また, 高速化のために配列 r 内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば nx が (4 の倍数)+2 の時は lx=nx+4 とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k_x,k_y,k_z}(k_x = 0, \cdots, n_x - 1;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$ に対

して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ とすると

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合, 一度この関数を呼びその後は初期化後の変換 2.13.2 $\left\{\begin{matrix}\text{ASL\_dfr3bf}\\ \text{ASL\_rfr3bf}\end{matrix}\right\}$ を利用すれば良い. このようにすれば, 初期化 (基数分けや三角関数テーブルの作成) が一度だけしか行われないため, 効率のよい処理ができる. ただしこの場合は配列 ifax, trigs の内容をそのまま 2.13.2 $\left\{\begin{matrix}\text{ASL\_dfr3bf}\\ \text{ASL\_rfr3bf}\end{matrix}\right\}$ の入力としなければならない.
なお, isw=0 として初期化だけを行う場合には, 配列 r に入力データを設定する必要がない.

(e) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(f) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(g) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) **使用例**

2.13.2 (7) 使用例参照.

2.13.2 [非推奨]ASL_dfr3bf, ASL_rfr3bf 3 次元実フーリエ変換 (初期化後の変換)

(1) **機 能**

順変換

3 次元実数データ $r_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ に対して, 3 次元フーリエ順変換 (任意基数) の j_x についての半周期分を求める.

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

ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す. なお, 残りの半周期分は以下のような関係から得られる.

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

ただし, z^* は複素数 z の共役複素数を表す.

逆変換

$c^*_{n_x-j_x,n_y-j_y,n_z-j_z}=c_{j_x,j_y,j_z}$, $c^*_{n_x-j_x,j_y,j_z}=c_{j_x,n_y-j_y,n_z-j_z}$, $c^*_{n_x-j_x,n_y-j_y,j_z}=c_{j_x,j_y,n_z-j_z}$ 等の関係を満たす 3 次元実フーリエ変換後の $n_xn_yn_z$ 個の複素数データ $c_{j_x,j_y,j_z}(j_x=0,\cdots,n_x-1;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ について j_x についての半周期分 $c_{j_x,j_y,j_z}(j_x=0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ を与えて以下のように定義される 3 次元フーリエ逆変換 (任意基数) を求める.

$$\begin{aligned} r_{k_x,k_y,k_z} &= \sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} c_{j_x,j_y,j_z} e^{2\pi\sqrt{-1}(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})} \\ &= \sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\{c_{0,j_y,j_z}+(-1)^{k_x}\hat{c}_{\frac{n_x}{2},j_y,j_z}\}e^{2\pi\sqrt{-1}(\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})} \\ &\quad + 2\sum_{j_z=0}^{n_z-1}\sum_{j_y=0}^{n_y-1}\sum_{j_x=1}^{\lceil\frac{n_x}{2}\rceil-1} \Re\{c_{j_x,j_y,j_z}e^{2\pi\sqrt{-1}(\frac{j_xk_x}{n_x}+\frac{j_yk_y}{n_y}+\frac{j_zk_z}{n_z})}\} \\ &\quad (k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1) \end{aligned}$$

ここで $\lceil x\rceil$ は x 以上の最小の整数を, $\Re\{z\}$ は複素数 z の実部を表す. また, n_x が奇数のとき $\hat{c}_{\frac{n_x}{2},j_y,j_z}=0$, n_x が偶数のとき $\hat{c}_{\frac{n_x}{2},j_y,j_z}=c_{\frac{n_x}{2},j_y,j_z}$ である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

単精度関数:

```
ierr = ASL_rfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk);
```

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	1 次元目のデータ数 n_x (注意事項 (a) 参照)
2	ny	I	1	入 力	2 次元目のデータ数 n_y (注意事項 (a) 参照)
3	nz	I	1	入 力	3 次元目のデータ数 n_z (注意事項 (a) 参照)
4	r	{D∗ / R∗}	lx×ly×lz	入 力	入力データ r_{k_x,k_y,k_z}(順変換), または c_{j_x,j_y,j_z}(逆変換) (注意事項 (b) 参照)
				出 力	出力データ c_{j_x,j_y,j_z}(順変換), または r_{k_x,k_y,k_z}(逆変換) (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 r の整合寸法 (注意事項 (b) 参照)
6	ly	I	1	入 力	配列 r の第 2 寸法 (注意事項 (b) 参照)
7	lz	I	1	入 力	配列 r の第 3 寸法 (注意事項 (b) 参照)
8	isw	I	1	入 力	処理スイッチ isw= 1:初期化後の順変換 isw=−1:初期化後の逆変換
9	ifax	I*	60	入 力	基数分け情報 (注意事項 (a) 参照)
10	trigs	{D∗ / R∗}	nx + 2 × (ny + nz)	入 力	三角関数テーブル (注意事項 (a) 参照)
11	wk	{D∗ / R∗}	lx × ly × lz	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(b) nx が奇数の時：
$\mathrm{nx}+1 \le \mathrm{lx}$
nx が偶数の時：
$\mathrm{nx}+2 \le \mathrm{lx}$

(c) ny $\le$ ly

(d) nz $\le$ lz

(e) lx は偶数である.

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b), (c) または (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	
3030	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) この関数は, 同じデータ数 (nx, ny, nz) の変換を繰り返し行う場合に初期化を含む変換
2.13.1 $\left\{\begin{matrix}\text{ASL\_dfr3fb}\\ \text{ASL\_rfr3fb}\end{matrix}\right\}$ を行った後で利用する. なお, この場合は配列 ifax, trigs の内容はそのままこの関数の入力とする必要がある.

(b) 実数データ r_{k_x,k_y,k_z} $(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ と配列 r の各要素は以下の様に対応する.

$$r_{k_x,k_y,k_z} \quad \leftrightarrow \quad \mathrm{r}[k_x+\mathrm{lx}*(k_y+\mathrm{ly}*k_z)]$$

なお, 逆変換を行った場合, nx(=n_x) が奇数のとき $\mathrm{r}[\mathrm{nx}+\mathrm{lx}*(k_y+\mathrm{ly}*k_z)]=0$, nx が偶数のとき $\mathrm{r}[\mathrm{nx}+\mathrm{lx}*(k_y+\mathrm{ly}*k_z)]=\mathrm{r}[\mathrm{nx}+1+\mathrm{lx}*(k_y+\mathrm{ly}*k_z)]=0$ となる. また, 実数データ $r_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ を配列 r に入力する場合, 上述の対応する 0 を特に格納する必要はない.
複素数データ $c_{j_x,j_y,j_z}(j_x=0,\cdots,\lfloor\frac{n_x}{2}\rfloor;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ の実部と虚部をそれぞれ $\Re\{c_{j_x,j_y,j_z}\}$, $\Im\{c_{j_x,j_y,j_z}\}$ とすると, c_{j_x,j_y,j_z} と配列 r の各要素は以下の様に対応する. ここで $\lfloor x\rfloor$ は x を超えない最大の整数を表す.

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

実フーリエ変換の性質より, $\Im\{c_{0,0,0}\}=0$ であり, nx, ny と nz が共に偶数であれば, $\Im\{c_{\frac{n_x}{2},\frac{n_y}{2},\frac{n_z}{2}}\}=0$ である. したがって, 配列 r の対応する要素に 0 以外の値が設定されていても 0 とみなして処理を行う. なお, $c_{j_x,j_y,j_z}\quad(j_x=\lfloor\frac{n_x}{2}\rfloor+1,\cdots,n_x-1;\ j_y=0,\cdots,n_y-1;\ j_z=0,\cdots,n_z-1)$ の各要素は実フーリエ変換の対称性から以下のような関係より得られるので逆変換の場合, 入力として与える必要は無く, また順変換の場合, 出力は行わない.

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

ただし, z^* は複素数 z の共役複素数を表す. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly, lz が奇数になるように設定するのが望ましい. また, 高速化のために配列 r 内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば nx が (4 の倍数)+2 の時は lx=nx+4 とする.**

(c) この関数を使用して順変換に引き続き逆変換を行った場合, 得られるデータは, 元のデータをデータ数倍した値になる. 例えば, 実数データ $r_{k_x,k_y,k_z}(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)$ に対

して順変換を行い引き続き逆変換を行ったデータを $\hat{r}_{k_x,k_y,k_z}(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$ とすると

$$\hat{r}_{k_x,k_y,k_z} = n_x n_y n_z r_{k_x,k_y,k_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

となる. したがって, 順変換または逆変換の結果のどちらかに対して正規化を行う必要がある. なお, 文献によっては, 順変換と逆変換の定義を本書と逆にしている場合や正規化を行った結果を定義としている場合があるので注意されたい.

(d) 離散フーリエ変換は変換前後のデータ列がデータ数 (n_x または n_y または n_z) を周期とする周期関数となっていることを前提としているので, 連続フーリエ変換を標本化して近似する場合にはこのことに注意して標本数や標本化間隔を設定する必要がある. なお, **標本化定理**によれば, 周波数 f_c で帯域制限された時間関数 $h(t)$ の場合, 標本化間隔を $T = \frac{1}{2f_c}$ ととれば, 以下の様に標本値列 $\{h(iT)\}$ だけの知識から $h(t)$ を復元できる.

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

(e) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(f) **非推奨** この機能は将来廃止予定である. より高速な実装が **ASL 統合インタフェース**で提供されているので, そちらを使用されたい.

(7) 使用例

(a) 問 題

$$r_{k_x,k_y,k_z} = \frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}$$
$$(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)$$

を入力データとして, 3 次元実フーリエ順・逆変換を行う.

(b) 入力データ

配列 r, nx=6, ny=4, nz=3, lx=10, ly=5, lz=3, isw=1(順変換) および isw=−1(逆変換)

(c) 主プログラム

```
/*      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) 出力結果

```
    *** 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 畳み込み

2.14.1 ASL_dfcn1d, ASL_rfcn1d
1 次元畳み込み

(1) **機 能**

任意の整数 k に対して

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

を満たす 2 組の周期 m の離散関数 $f(i)$, $g(j)$ ただし,

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

が与えられているとき次式で定義される離散畳み込み $p(k)$ $(k = 0, \cdots, m-1)$ を計算する.

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

ただし, $m = \min(n_1 + n_2 - 1, M)$ であり, M は $M \geq \max(n_1, n_2)$ を満たす任意の整数である. なお, $p(k)$ の実フーリエ変換を求めることもできる.

(2) **使用法**

倍精度関数:

ierr = ASL_dfcn1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

単精度関数:

ierr = ASL_rfcn1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n1	I	1	入 力	離散関数 $f(i)$ の有効データ数 n_1
2	n2	I	1	入 力	離散関数 $g(j)$ の有効データ数 n_2
3	r1	{D∗ / R∗}	ld1	入 力	離散関数 $f(i)$ の値 (注意事項 (a), (b) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i)$ の実フーリエ変換結果 (周期 M)
4	ld1	I	1	入 力	配列 r1 の大きさ
5	r2	{D∗ / R∗}	ld2	入 力	離散関数 $g(j)$ の値 (注意事項 (a), (b) 参照)
				出 力	離散関数 $p(k)$ の値またはその実フーリエ変換 (注意事項 (a), (c) 参照)
6	ld2	I	1	入 力	配列 r2 の大きさ
7	m	I	1	入 力	離散関数 $f(i)$, $g(j)$, $p(k)$ の周期 m に対応するパラメータ M (注意事項 (d) 参照)
8	isw	I	1	入 力	処理スイッチ (注意事項 (a), (e) 参照) isw= 0:定義により畳み込みを計算する isw= 1:FFT 法により畳み込みを計算する isw= 2:畳み込みの実フーリエ変換を計算する isw= 3:区分化 FFT 法により畳み込みを計算する
9	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 20 (isw ≥ 1 のとき)
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: n2 (isw= 0 のとき) 2 × m + 1 (isw= 1 または 2, m が奇数のとき) 2 × m + 2 (isw= 1 または 2, m が偶数のとき) 2 × m + n1 (isw= 3, m が奇数のとき) 2 × m + n1 + 1 (isw= 3, m が偶数のとき)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{isw} \in \{0, 1, 2, 3\}$

(b) $\mathrm{n1} > 1$

(c) $\mathrm{n2} > 1$

(d) $\mathrm{m} \geq \max(\mathrm{n1}, \mathrm{n2})$

(e) $\mathrm{isw} = 0$ の時 :
$\mathrm{ld1} \geq \mathrm{n1}$
$\mathrm{isw} > 0$ で m が奇数の時 :
$\mathrm{ld1} \geq \mathrm{m} + 1$
$\mathrm{isw} > 0$ で m が偶数の時 :
$\mathrm{ld1} \geq \mathrm{m} + 2$

(f) $\mathrm{isw} = 0$ の時 :
$\mathrm{ld2} \geq \mathrm{m}$
$\mathrm{isw} > 0$ で m が奇数の時 :
$\mathrm{ld2} \geq \mathrm{m} + 1$
$\mathrm{isw} > 0$ で m が偶数の時 :
$\mathrm{ld2} \geq \mathrm{m} + 2$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{m} < \mathrm{n1} + \mathrm{n2} - 1$ であった.	畳み込みの計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 畳み込みを計算する関数値の一方の有効データ数が他方に比較して非常に大きい場合, 区分化によりデータ数の多い方を (必要であれば最後に 0 を付加して) 等分し, 本関数を繰り返し適用して離散畳み込みを計算した方が効率が良く, 必要な記憶容量も小さくて済む.
たとえば, 2 つの系列 $\{u_1, u_2, \cdots, u_k\}$(有効データ数 k), $\{v_1, \cdots, v_{pq}\}$(有効データ数 pq $(pq >> k)$) の離散畳み込みを計算したい場合, まず isw=1, n1=k, n2=q, m $\geq$ n1 + n2 $-$ 1, r1=$\{u_1, u_2, \cdots, u_k\}$ r2=$\{v_1, v_2, \cdots, v_q\}$ として本関数を適用する. この結果, 計算したい畳み込みの最初の q 個の値が配列 r2 の先頭から q 個の要素として得られる.
次に, isw=3, r2=$\{v_{q+1}, \cdots, v_{2q}\}$ と変更し, ほかの引数の内容はそのままとして本関数を適用する. この結果, 計算したい畳み込みの次の q 個の値が配列 r2 の先頭から q 個の要素として得られる. 以下同様にして r2 に設定する値を順次ずらして計算する. なお, 最後の繰り返しすなわち r2=$\{v_{(p-1)q+1}, \cdots, v_{pq}\}$ と設定して計算した畳み込みは計算したい畳み込みの最後の $2q-1$ 個の要素を与える (ただし, 系列 $\{v_j\}$ が有限波形でない場合は最後から $q-1$ 個は不定).

(b) 配列 r1, r2 にはそれぞれ離散関数 $f(i)$ と離散関数 $g(j)$ の値を次のように格納する. ただし, isw = 3 とする場合には, r2 にのみ値を格納し, r1 の内容はそのまま利用する (注意事項 (a) 参照).

$$
\begin{array}{lcl}
f(0) & \rightarrow & \mathrm{r1}[0] \\
f(1) & \rightarrow & \mathrm{r1}[1] \\
\cdots & \cdots & \cdots \\
f(n_1-1) & \rightarrow & \mathrm{r1}[\mathrm{n1}-1]
\end{array}
$$

$$
\begin{array}{lcl}
g(0) & \rightarrow & \mathrm{r2}[0] \\
g(1) & \rightarrow & \mathrm{r2}[1] \\
\cdots & \cdots & \cdots \\
g(n_2-1) & \rightarrow & \mathrm{r2}[\mathrm{n2}-1]
\end{array}
$$

なお, 配列 r1, r2 の r1 [n1] 並びに r2 [n2] 以降の要素には値を入力する必要が無い. また, 特に, isw = 3 とする場合には, r2 [n2] 以降の要素は計算に利用するので変更してはならない.

(c) 離散畳み込み $p(k)$ の値は配列 r2 に以下のように得られる.

$$
\begin{array}{lcl}
p(0) & \rightarrow & \mathrm{r2}[0] \\
p(1) & \rightarrow & \mathrm{r2}[1] \\
\cdots & \cdots & \cdots \\
p(M-1) & \rightarrow & \mathrm{r2}[\mathrm{m}-1]
\end{array}
$$

なお, m が奇数のとき r2 [m] が m が偶数のとき r2 [m] と r2 [m+1] がそれぞれ 0.0 となる. また, 区分化を行う場合には, 通常, 最初の n2 個のデータが畳み込みとして意味を持つ (注意事項 (a) 参照).
isw=2 として, 離散畳み込み $p(k)$ の実フーリエ変換 $P(j)$:

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

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{array}{lcl}
\Re\{P(0)\} & \leftrightarrow & \mathrm{r2}[0] \\
\Im\{P(0)\} & \leftrightarrow & \mathrm{r2}[1] \\
\Re\{P(1)\} & \leftrightarrow & \mathrm{r2}[2] \\
\Im\{P(1)\} & \leftrightarrow & \mathrm{r2}[3] \\
\cdots & \cdots & \cdots \\
\Re\{P(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \mathrm{r2}[l-2] \\
\Im\{P(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \mathrm{r2}[l-1] \quad (l = \mathrm{m}{+}1[\text{m:奇数}] \text{ または } \mathrm{m}{+}2[\text{m:偶数}])
\end{array}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$P(M-j) = P(j)^*$$

(ただし, z^* は複素数 z の共役複素数) から得られる.

(d) $\mathrm{m} \geq \mathrm{n1} + \mathrm{n2} - 1$ とすれば, 次の周期の畳み込みとの重なりを起こさずに畳み込みを計算できる. $\mathrm{m} > \mathrm{n1} + \mathrm{n2} - 1$ の場合 n1 + n2 以降の要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, $\mathrm{m} = \mathrm{n1} + \mathrm{n2} - 1$ とするのがよい. $\mathrm{isw} \geq 1$ とする場合, m の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n1=n2=145 の場合, isw=0 のときは, $\mathrm{m} = 289(=17^2)$ とした方が良いが, $\mathrm{isw} \geq 1$ の場合には $\mathrm{m} = 300(=2^2 \times 3 \times 5^2)$ や $320(=2^6 \times 5)$, $384(=2^7 \times 3)$ などとした方が通常は効率が良い.

(e) **通常は isw=1 と設定して FFT 畳み込みを計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ m の選び方に制限がある場合などは isw=0 として計算する.

(f) 非ゼロ部分の開始位置が原点から離れている離散関数の畳み込みを計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i)$, $g(j)$ の非ゼロ部分がそれぞれ区間 $[i_0, i_0+n_1-1]$, $[j_0, j_0+n_2-1]$ のとき

$$\hat{f}(i) = f(i-i_0), \quad \hat{g}(j) = g(j-j_0)$$

として $\hat{f}(i)$, $\hat{g}(j)$ についてこの関数を適用し, 得られた結果を $\hat{p}(k)$ とすれば, もとの $f(i)$, $g(j)$ の畳み込み $p(k)$ は

$$p(k) = \hat{p}(k + (i_0 + j_0))$$

となる. すなわち, 離散畳み込みを計算する前に $f(i)$, $g(j)$ をそれぞれ負の方向に i_0, j_0 だけシフトしたとすれば, この関数の適用後畳み込みの計算値を正の方向に $i_0 + j_0$ だけシフトすれば所望の結果が得られる.

(g) この関数で計算する離散畳み込みに標本化間隔を乗じたものは帯域制限された関数の連続畳み込み積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続畳み込みと対応をとる場合には, $p(n_1+n_2-1) = 0$ として $p(k)$ $(k = 0, 1, \cdots, n_1+n_2-1)$ の n_1+n_2 個のデータを考えた方が, 対応をとりやすい. このとき通常は座標 0 の要素は $p(0)$ に対応させる. ただし,
isw=0 の場合, ld1=n1, ld2=m, nwk=n2
isw=1 または 2 の場合, m が奇数ならば ld1=ld2=m+1, $\mathrm{nwk} = 2 \times \mathrm{m} + 1$
m が偶数ならば ld1=ld2=m+2, $\mathrm{nwk} = 2 \times \mathrm{m} + 2$
isw=3 の場合, m が奇数ならば ld1=ld2=m+1, $\mathrm{nwk} = 2 \times \mathrm{m} + \mathrm{n1}$
m が偶数ならば ld1=ld2=m+2, $\mathrm{nwk} = 2 \times \mathrm{m} + \mathrm{n1} + 1$
である.

(h) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題

次式で定義される 2 つの有限波形を標本化間隔 Δx で離散化し, 離散畳み込みを計算する.

$$f(x) = \begin{cases} x & 0 \le x \le a \\ 0 & \text{それ以外} \end{cases}$$

$$g(x) = \begin{cases} b - x & 0 \le x \le b \\ 0 & \text{それ以外} \end{cases}$$

備考

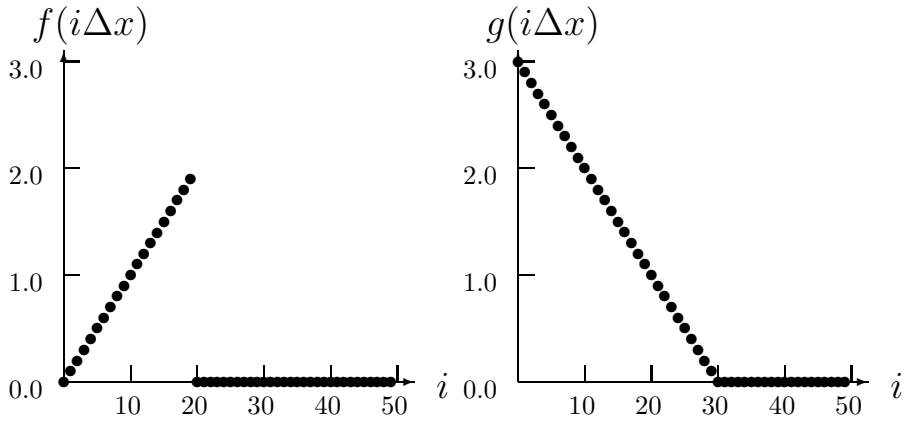
$f(x)$ と $g(x)$ の連続畳み込み $p(x) = (f \times g)(x)$ は

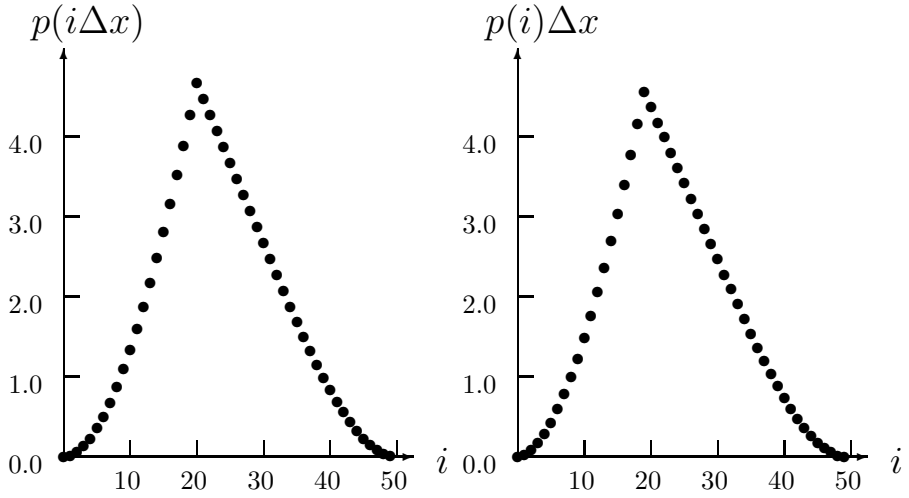
$$p(x) = \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi = \begin{cases} G(0, x, x) & 0 \le x \le a \\ G(0, a, x) & a \le x \le b \\ G(x - b, a, x) & b \le x \le a + b \\ 0 & \text{それ以外} \end{cases}$$

である. ここで, $G(\alpha, \beta, x)$ は

$$\begin{aligned} G(\alpha, \beta, x) &= \left[\frac{\xi^2}{6} (3(b - x) + 2\xi) \right]_{\alpha}^{\beta} \\ &= \left. \frac{\xi^2}{6} (3(b - x) + 2\xi) \right|_{\xi=\beta} - \left. \frac{\xi^2}{6} (3(b - x) + 2\xi) \right|_{\xi=\alpha} \end{aligned}$$

$a = 2$, $b = 3$ とした場合, $f(x)$, $g(x)$, $p(x) = (f \times g)(x)$ を $\Delta x = 0.1$ として標本化した値 $f(i\Delta x)$, $g(i\Delta x)$, $p(i\Delta x)$ はそれぞれ以下のようなグラフとなる. 参考のため本関数で計算した離散畳み込みに Δx を乗じた値 $p(i)\Delta x$ も示してある. 標本数が少ないわりには連続畳み込みの値と良く一致する.





なお, 以下の使用例では参考までに連続畳み込みの値も計算している.

(b) 入力データ

標本化データ

$\mathrm{r1[i-1]} = \mathrm{f((i-1)\Delta x)}\ (\mathrm{i} = 1, 2, \cdots, \mathrm{n1})$

$\mathrm{r2[j-1]} = \mathrm{g((j-1)\Delta x)}\ (\mathrm{j} = 1, 2, \cdots, \mathrm{n2})$

ただし, $\Delta x = 0.1$

$\mathrm{n1} = \dfrac{a}{\Delta x}$, $\mathrm{n2} = \dfrac{b}{\Delta x}$, m, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
2 次元畳み込み

(1) **機 能**

任意の整数 L_x, L_y に対して

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & \quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

を満たす 2 組の周期 (m_x, m_y) の多重周期離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$ についてこれらがそれぞれ基本周期内では $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$, $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$ でのみ非ゼロ値をとるとする. ここで, $[0, a] \times [0, b]$ は平面座標 (i, j) がはる平面の直積領域 (点 (0, 0) と点 (a, b) を対角頂点とする長方形で囲まれる領域) とする. このとき, 次式で定義される離散畳み込み $p(k_x, k_y)$ を計算する.

$$\begin{aligned} p(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x - i_x, k_y - i_y) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} g(j_x, j_y) f(k_x - j_x, k_y - j_y) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

ただし, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ であり, M_x, M_y はそれぞれ $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$ を満たす任意の整数である. なお, $p(k_x, k_y)$ の 2 次元実フーリエ変換を求めることもできる.

(2) **使用法**

倍精度関数:

ierr = ASL_dfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);

単精度関数:

ierr = ASL_rfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx1	I	1	入 力	離散関数 $f(i_x, i_y)$ の i_x 方向の有効データ数 $n_x^{(f)}$
2	ny1	I	1	入 力	離散関数 $f(i_x, i_y)$ の i_y 方向の有効データ数 $n_y^{(f)}$
3	nx2	I	1	入 力	離散関数 $g(j_x, j_y)$ の j_x 方向の有効データ数 $n_x^{(g)}$
4	ny2	I	1	入 力	離散関数 $g(j_x, j_y)$ の j_y 方向の有効データ数 $n_y^{(g)}$
5	r1	{D∗ / R∗}	lx1×ly1	入 力	離散関数 $f(i_x, i_y)$ の値 (注意事項 (a) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i_x, i_y)$ の 2 次元実フーリエ変換結果 (周期 (M_x, M_y))
6	lx1	I	1	入 力	配列 r1 の整合寸法
7	ly1	I	1	入 力	配列 r1 の第 2 寸法
8	r2	{D∗ / R∗}	lx2×ly2	入 力	離散関数 $g(j_x, j_y)$ の値 (注意事項 (a) 参照)
				出 力	離散関数 $p(k_x, k_y)$ の値またはその 2 次元実フーリエ変換 (注意事項 (b) 参照)
9	lx2	I	1	入 力	配列 r2 の整合寸法
10	ly2	I	1	入 力	配列 r2 の第 2 寸法
11	mx	I	1	入 力	離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$, $p(k_x, k_y)$ の周期 (m_x, m_y) に対応するパラメータ M_x (注意事項 (c) 参照)
12	my	I	1	入 力	離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$, $p(k_x, k_y)$ の周期 (m_x, m_y) に対応するパラメータ M_y (注意事項 (c) 参照)
13	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:定義により畳み込みを計算する isw= 1:FFT 法により畳み込みを計算する isw= 2:畳み込みの実フーリエ変換を計算する
14	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 40 (isw ≥ 1 のとき)
15	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: nx2 × ny2 (isw= 0, nx2:奇数のとき) (nx2 + 1) × ny2 (isw= 0, nx2:偶数のとき) mx + 2 × my + max(lx1 × ly1, lx2 × ly2) (isw ≥ 1 のとき)
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{isw} \in \{0, 1, 2\}$

(b) $\mathrm{nx1} > 1$
$\mathrm{ny1} > 1$

(c) $\mathrm{nx2} > 1$
$\mathrm{ny2} > 1$

(d) $\mathrm{mx} \geq \max(\mathrm{nx1}, \mathrm{nx2})$
$\mathrm{my} \geq \max(\mathrm{ny1}, \mathrm{ny2})$

(e) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx1} \geq \mathrm{nx1}$
$\mathrm{ly1} \geq \mathrm{ny1}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 1$
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 2$
$\mathrm{ly1} \geq \mathrm{my}$

(f) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx2} \geq \mathrm{mx}$
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 1$
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 2$
$\mathrm{ly2} \geq \mathrm{my}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{mx} < \mathrm{nx1} + \mathrm{nx2} - 1$ または $\mathrm{my} < \mathrm{ny1} + \mathrm{ny2} - 1$ であった.	畳み込みの計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 配列 r1, r2 の各要素と離散関数 $f(i_x, i_y)$ と離散関数 $g(j_x, j_y)$ の値は以下の様に対応する.

$$\begin{aligned} f(i_x, i_y) &\leftrightarrow \mathrm{r1}[i_x + \mathrm{lx1} * i_y] \\ g(j_x, j_y) &\leftrightarrow \mathrm{r2}[j_x + \mathrm{lx2} * j_y] \end{aligned}$$

ただし, $i_x = 0, \cdots, n_x^{(f)} - 1;\ i_y = 0, \cdots, n_y^{(f)} - 1$, $j_x = 0, \cdots, n_x^{(g)} - 1;\ j_y = 0, \cdots, n_y^{(g)} - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列** r1, r2 **の整合寸法について** lx1**/2,** ly1, lx2**/2,** ly2 **が奇数になるように設定するのが望ましい. 通常, たとえば** mx **が 4 の倍数のときは** lx1=mx**+3 とする.**

(b) 離散畳み込み $p(k_x, k_y)$ の値は配列 r2 の各要素と以下の様に対応する.

$$p(k_x, k_y) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

ただし, $k_x = 0, \cdots, M_x - 1;\ k_y = 0, \cdots, M_y - 1$ である. isw=2 として, 離散畳み込み $p(k_x, k_y)$ の 2 次元実フーリエ変換 $P(j_x, j_y)$:

$$\begin{aligned} P(j_x, j_y) &= \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} p(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})} \\ &\qquad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor) \end{aligned}$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{aligned} \Re\{P(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + \mathrm{lx2} * j_y] \\ \Im\{P(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * j_y] \end{aligned}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$\begin{aligned} P(M_x - j_x, M_y - j_y)^* &= P(j_x, j_y) \\ P(M_x - j_x, j_y)^* &= P(j_x, M_y - j_y) \end{aligned}$$

(ただし, z^* は複素数 z の共役複素数) から得られる.

(c) $\mathrm{mx} \geq \mathrm{nx1} + \mathrm{nx2} - 1$ かつ $\mathrm{my} \geq \mathrm{ny1} + \mathrm{ny2} - 1$ とすれば, 次の周期の畳み込みとの重なりを起こさずに畳み込みを計算できる. $\mathrm{mx} > \mathrm{nx1} + \mathrm{nx2} - 1$ または $\mathrm{my} > \mathrm{ny1} + \mathrm{ny2} - 1$ の場合

$$p(k_x, k_y) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

$k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1$ に対応する要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, $\mathrm{mx} = \mathrm{nx1} + \mathrm{nx2} - 1$, $\mathrm{my} = \mathrm{ny1} + \mathrm{ny2} - 1$ とするのがよい. $\mathrm{isw} \geq 1$ とする場合, mx, my の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx1=nx2=145 の場合, isw=0 のときは, $\mathrm{mx} = 289 (= 17^2)$ とした方が良いが, $\mathrm{isw} \geq 1$ の場合には $\mathrm{mx} = 300 (= 2^2 \times 3 \times 5^2)$ や $320 (= 2^6 \times 5)$, $384 (= 2^7 \times 3)$ などとした方が通常は効率が良い.

(d) **通常は** isw=**1 と設定して FFT 畳み込みを計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ mx や my の選び方に制限がある場合などは isw=0 として計算する.

(e) 非ゼロ部分の開始位置が原点から離れている離散関数の畳み込みを計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$ の非ゼロ部分が i_x, j_x についてそれぞれ区間 $[i_0, i_0 + n_x^{(f)} - 1]$, $[j_0, j_0 + n_x^{(g)} - 1]$ のとき

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

として $\hat{f}(i_x, i_y)$, $\hat{g}(j_x, j_y)$ についてこの関数を適用し, 得られた結果を $\hat{p}(k_x, k_y)$ とすれば, もとの $f(i_x, i_y)$, $g(j_x, j_y)$ の畳み込み $p(k_x, k_y)$ は

$$p(k_x, k_y) = \hat{p}(k_x + (i_0 + j_0), k_y)$$

となる. すなわち, 離散畳み込みを計算する前に $f(i_x, i_y)$, $g(j_x, j_y)$ をそれぞれ i_x, j_x の負の方向に i_0, j_0 だけシフトしたとすれば, この関数の適用後畳み込みの計算値を k_x の正の方向に $i_0 + j_0$ だけシフトすれば所望の結果が得られる. i_y, j_y, k_y についても同様である.

(f) この関数で計算する離散畳み込みに標本化間隔の 2 乗を乗じたものは帯域制限された関数の連続畳み込み積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続畳み込みと対応をとる場合には, $p(n_x^{(f)} + n_x^{(g)} - 1, k_y) = 0$, $p(k_x, n_y^{(f)} + n_y^{(g)} - 1) = 0$ として $p(k_x, k_y)$ $(k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1;\ k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1)$ の $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ 個のデータを考えた方が, 対応をとりやすい. このとき通常は座標 (0, 0) の要素は $p(0, 0)$ に対応させる. ただし,
isw=0 の場合,
$\mathrm{lx1} = \mathrm{nx1}, \mathrm{ly1} = \mathrm{ny1}, \mathrm{lx2} = \mathrm{mx}, \mathrm{ly2} = \mathrm{my}$,
$\mathrm{nwk} = \mathrm{nx2} \times \mathrm{ny2}$(nx2:奇数のとき) または
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times \mathrm{ny2}$(nx2:偶数のとき)
$\mathrm{isw} \geq 1$ の場合,
lx1=lx2=mx+1(mx が奇数のとき) または
lx1=lx2=mx+2(mx が偶数のとき),
ly1=ly2=my, $\mathrm{nwk} = \mathrm{mx} + (\mathrm{lx1} + 2) \times \mathrm{my}$
である.

(g) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題
次式で定義される 2 つの有限波形を標本化間隔 Δ で離散化し, 離散畳み込みを計算する.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & (\text{それ以外}) \end{cases}$$

$$g(x, y) = \begin{cases} x_g - x & ((x, y) \in [0, x_g] \times [0, y_g]) \\ 0 & (\text{それ以外}) \end{cases}$$

(b) 入力データ
標本化データ
$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * \mathrm{i_y}] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1)$
$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * \mathrm{j_y}] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1)$
ただし, $\Delta = 0.5$
nx1, ny1, nx2, ny2, mx, my, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
3 次元畳み込み

(1) **機 能**

任意の整数 L_x, L_y, L_z に対して

$$\begin{aligned} f(i_x, i_y, i_z) &= f(i_x + L_x m_x, i_y + L_y m_y, i_z + L_z m_z), \\ g(j_x, j_y, j_z) &= g(j_x + L_x m_x, j_y + L_y m_y, j_z + L_z m_z), \\ & \quad (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1;\ i_z, j_z = 0, \cdots, m_z - 1) \end{aligned}$$

を満たす 2 組の周期 (m_x, m_y, m_z) の多重周期離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ についてこれらがそれぞれ基本周期内では $(i_x, i_y, i_z) \in [0, n_x^{(f)}-1] \times [0, n_y^{(f)}-1] \times [0, n_z^{(f)}-1]$, $(j_x, j_y, j_z) \in [0, n_x^{(g)}-1] \times [0, n_y^{(g)}-1] \times [0, n_z^{(g)}-1]$ でのみ非ゼロ値をとるとする．ここで, $[0, a] \times [0, b] \times [0, c]$ は空間座標 (i, j, k) がはる空間の直積領域 (点 (0, 0, 0) と点 (a, b, c) を対角頂点とする直方体で囲まれる領域) とする．このとき, 次式で定義される離散畳み込み $p(k_x, k_y, k_z)$ を計算する．

$$\begin{aligned} p(k_x, k_y, k_z) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} \sum_{i_z=0}^{m_z-1} f(i_x, i_y, i_z) g(k_x - i_x, k_y - i_y, k_z - i_z) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} \sum_{j_z=0}^{m_z-1} g(j_x, j_y, j_z) f(k_x - j_x, k_y - j_y, k_z - j_z) \\ & \quad (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1;\ k_z = 0, \cdots, m_z - 1) \end{aligned}$$

ただし, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$, $m_z = \min(n_z^{(f)} + n_z^{(g)} - 1, M_z)$ であり, M_x, M_y, M_z はそれぞれ $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$, $M_z \geq \max(n_z^{(f)}, n_z^{(g)})$ を満たす任意の整数である．なお, $p(k_x, k_y, k_z)$ の 3 次元実フーリエ変換を求めることもできる．

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfcn3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rfcn3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx1	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$ の i_x 方向の有効データ数 $n_x^{(f)}$
2	ny1	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$ の i_y 方向の有効データ数 $n_y^{(f)}$
3	nz1	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$ の i_z 方向の有効データ数 $n_z^{(f)}$
4	nx2	I	1	入 力	離散関数 $g(j_x, j_y, j_z)$ の j_x 方向の有効データ数 $n_x^{(g)}$
5	ny2	I	1	入 力	離散関数 $g(j_x, j_y, j_z)$ の j_y 方向の有効データ数 $n_y^{(g)}$
6	nz2	I	1	入 力	離散関数 $g(j_x, j_y, j_z)$ の j_z 方向の有効データ数 $n_z^{(g)}$
7	r1	{ D∗ / R∗ }	lx1×ly1×lz1	入 力	離散関数 $f(i_x, i_y, i_z)$ の値 (注意事項 (a) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i_x, i_y, i_z)$ の 3 次元実フーリエ変換結果 (周期 (M_x, M_y, M_z))
8	lx1	I	1	入 力	配列 r1 の整合寸法
9	ly1	I	1	入 力	配列 r1 の第 2 寸法
10	lz1	I	1	入 力	配列 r1 の第 3 寸法
11	r2	{ D∗ / R∗ }	lx2×ly2×lz2	入 力	離散関数 $g(j_x, j_y, j_z)$ の値 (注意事項 (a) 参照)
				出 力	離散関数 $p(k_x, k_y, k_z)$ の値またはその 3 次元実フーリエ変換 (注意事項 (b) 参照)
12	lx2	I	1	入 力	配列 r2 の整合寸法
13	ly2	I	1	入 力	配列 r2 の第 2 寸法
14	lz2	I	1	入 力	配列 r2 の第 3 寸法
15	mx	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, $p(k_x, k_y, k_z)$ の周期 (m_x, m_y, m_z) に対応するパラメータ M_x (注意事項 (c) 参照)
16	my	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, $p(k_x, k_y, k_z)$ の周期 (m_x, m_y, m_z) に対応するパラメータ M_y (注意事項 (c) 参照)

項番	引数と戻り値	型	大きさ	入出力	内　　容
17	mz	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, $p(k_x, k_y, k_z)$ の周期 (m_x, m_y, m_z) に対応するパラメータ M_z (注意事項 (c) 参照)
18	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:定義により畳み込みを計算する isw= 1:FFT 法により畳み込みを計算する isw= 2:畳み込みの実フーリエ変換を計算する
19	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 60 (isw $\geq$ 1 のとき)
20	wk	$\left\{\begin{matrix}\text{D}*\\ \text{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: (nx2+1)×(ny2+1)×nz2 (isw= 0, nx2:偶数, ny2:偶数のとき) nx2 × (ny2 + 1) × nz2 (isw= 0, nx2:奇数, ny2:偶数のとき) (nx2 + 1) × ny2 × nz2 (isw= 0, nx2:偶数, ny2:奇数のとき) nx2 × ny2 × nz2 (isw= 0, nx2:奇数, ny2:奇数のとき) mx + 2 × (my + mz) + max(lx1 × ly1 × lz1, lx2 × ly2 × lz2) (isw $\geq$ 1 のとき)
21	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{isw} \in \{0, 1, 2\}$

(b) $\mathrm{nx1} > 1$
$\mathrm{ny1} > 1$
$\mathrm{nz1} > 1$

(c) $\mathrm{nx2} > 1$
$\mathrm{ny2} > 1$
$\mathrm{nz2} > 1$

(d) $\mathrm{mx} \geq \max(\mathrm{nx1}, \mathrm{nx2})$
$\mathrm{my} \geq \max(\mathrm{ny1}, \mathrm{ny2})$
$\mathrm{mz} \geq \max(\mathrm{nz1}, \mathrm{nz2})$

(e) $\mathrm{isw} = 0$ の時：
$\mathrm{lx1} \geq \mathrm{nx1}$
$\mathrm{ly1} \geq \mathrm{ny1}$
$\mathrm{lz1} \geq \mathrm{nz1}$
$\mathrm{isw} > 0$ で mx が奇数の時：
$\mathrm{lx1} \geq \mathrm{mx} + 1$, lx1 は偶数
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{lz1} \geq \mathrm{mz}$
$\mathrm{isw} > 0$ で mx が偶数の時：
$\mathrm{lx1} \geq \mathrm{mx} + 2$, lx1 は偶数
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{lz1} \geq \mathrm{mz}$

(f) $\mathrm{isw} = 0$ の時：
$\mathrm{lx2} \geq \mathrm{mx}$
$\mathrm{ly2} \geq \mathrm{ny2}$
$\mathrm{lz2} \geq \mathrm{nz2}$
$\mathrm{isw} > 0$ で mx が奇数の時：
$\mathrm{lx2} \geq \mathrm{mx} + 1$, lx2 は偶数
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{lz2} \geq \mathrm{mz}$
$\mathrm{isw} > 0$ で mx が偶数の時：
$\mathrm{lx2} \geq \mathrm{mx} + 2$, lx2 は偶数
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{lz2} \geq \mathrm{mz}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	mx $<$ nx1 + nx2 − 1, my $<$ ny1 + ny2 − 1 または mz $<$ nz1 + nz2 − 1 であった.	畳み込みの計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 配列 r1, r2 の各要素と離散関数 $f(i_x, i_y, i_z)$ と離散関数 $g(j_x, j_y, j_z)$ の値は以下の様に対応する.

$$\begin{aligned} f(i_x, i_y, i_z) &\leftrightarrow \mathrm{r1}[i_x + \mathrm{lx1} * (i_y + \mathrm{ly1} * i_z)] \\ g(j_x, j_y, j_z) &\leftrightarrow \mathrm{r2}[j_x + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \end{aligned}$$

ただし, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$; $i_z = 0, \cdots, n_z^{(f)} - 1$, $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$; $j_z = 0, \cdots, n_z^{(g)} - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列** r1, r2 **の整合寸法について** lx1**/2,** ly1**,** lz1**,** lx2**/2,** ly2**,** lz2 **が奇数になるように設定するのが望ましい. また, 高速化のために配列** r1, r2 **内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば** mx **が (4 の倍数)+2 のときは** lx1=mx+4 **とする.**

(b) 離散畳み込み $p(k_x, k_y, k_z)$ の値は配列 r2 の各要素と以下の様に対応する.

$$p(k_x, k_y, k_z) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * (k_y + \mathrm{ly2} * k_z)]$$

ただし, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$; $k_z = 0, \cdots, M_z - 1$ である. isw=2 として, 離散畳み込み $p(k_x, k_y, k_z)$ の 3 次元実フーリエ変換 $P(j_x, j_y, j_z)$:

$$\begin{aligned} P(j_x, j_y, j_z) &= \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} p(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor) \end{aligned}$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{aligned} \Re\{P(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{r2}[2 * j_x + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \\ \Im\{P(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \end{aligned}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$\begin{aligned} P(M_x - j_x, M_y - j_y, M_z - j_z)^* &= P(j_x, j_y, j_z) \\ P(M_x - j_x, j_y, j_z)^* &= P(j_x, M_y - j_y, M_z - j_z) \\ P(M_x - j_x, M_y - j_y, j_z)^* &= P(j_x, j_y, M_z - j_z) \end{aligned}$$

(ただし, z^* は複素数 z の共役複素数) から得られる.

(c) $\mathrm{mx} \geq \mathrm{nx1} + \mathrm{nx2} - 1$ かつ $\mathrm{my} \geq \mathrm{ny1} + \mathrm{ny2} - 1$ かつ $\mathrm{mz} \geq \mathrm{nz1} + \mathrm{nz2} - 1$ とすれば, 次の周期の畳み込みとの重なりを起こさずに畳み込みを計算できる. $\mathrm{mx} > \mathrm{nx1} + \mathrm{nx2} - 1$ または $\mathrm{my} > \mathrm{ny1} + \mathrm{ny2} - 1$ または $\mathrm{mz} > \mathrm{nz1} + \mathrm{nz2} - 1$ の場合

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r2}[k_x + \mathrm{lx2} * (k_y + \mathrm{ly2} * k_z)]$$

$k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = \mathrm{nz1} + \mathrm{nz2} - 1, \cdots, \mathrm{mz} - 1$ に対応する要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, $\mathrm{mx} = \mathrm{nx1} + \mathrm{nx2} - 1$, $\mathrm{my} = \mathrm{ny1} + \mathrm{ny2} - 1$, $\mathrm{mz} = \mathrm{nz1} + \mathrm{nz2} - 1$ とするのがよい. $\mathrm{isw} \geq 1$ とする場合, mx, my, mz の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx1=nx2=145 の場合, isw=0 のときは, $\mathrm{mx} = 289 (= 17^2)$ とした方が良いが, $\mathrm{isw} \geq 1$ の場合には $\mathrm{mx} = 300 (= 2^2 \times 3 \times 5^2)$ や $320 (= 2^6 \times 5)$, $384 (= 2^7 \times 3)$ などとした方が通常は効率が良い.

(d) **通常は isw=1 と設定して FFT 畳み込みを計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ mx や my, mz の選び方に制限がある場合などは isw=0 として計算する.

(e) 非ゼロ部分の開始位置が原点から離れている離散関数の畳み込みを計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ の非ゼロ部分が i_x, j_x についてそれぞれ区間 $[i_0, i_0 + n_x^{(f)} - 1]$, $[j_0, j_0 + n_x^{(g)} - 1]$ のとき

$$\hat{f}(i_x, i_y, i_z) = f(i_x - i_0, i_y, i_z), \quad \hat{g}(j_x, j_y, j_z) = g(j_x - j_0, j_y, j_z)$$

として $\hat{f}(i_x, i_y, i_z)$, $\hat{g}(j_x, j_y, j_z)$ についてこの関数を適用し, 得られた結果を $\hat{p}(k_x, k_y, k_z)$ とすれば, もとの $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ の畳み込み $p(k_x, k_y, k_z)$ は

$$p(k_x, k_y, k_z) = \hat{p}(k_x + (i_0 + j_0), k_y, k_z)$$

となる. すなわち, 離散畳み込みを計算する前に $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ をそれぞれ i_x, j_x の負の方向に i_0, j_0 だけシフトしたとすれば, この関数の適用後畳み込みの計算値を k_x の正の方向に $i_0 + j_0$ だけシフトすれば所望の結果が得られる. i_y, j_y, k_y; i_z, j_z, k_z についても同様である.

(f) この関数で計算する離散畳み込みに標本化間隔の 3 乗を乗じたものは帯域制限された関数の連続畳み込み積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続畳み込みと対応をとる場合には, $p(n_x^{(f)} + n_x^{(g)} - 1, k_y, k_z) = 0$, $p(k_x, n_y^{(f)} + n_y^{(g)} - 1, k_z) = 0$, $p(k_x, k_y, n_z^{(f)} + n_z^{(g)} - 1) = 0$, として $p(k_x, k_y, k_z)$ $(k_x = 0, 1, \cdots, n_x^{(f)} + n_x^{(g)} - 1;\ k_y = 0, 1, \cdots, n_y^{(f)} + n_y^{(g)} - 1;\ k_z = 0, 1, \cdots, n_z^{(f)} + n_z^{(g)} - 1)$ の $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})(n_z^{(f)} + n_z^{(g)})$ 個のデータを考えた方が, 対応をとりやすい. このとき通常は座標 (0, 0, 0) の要素は $p(0, 0, 0)$ に対応させる. ただし,
isw=0 の場合,
$\mathrm{lx1} = \mathrm{nx1}, \mathrm{ly1} = \mathrm{ny1}, \mathrm{lz1} = \mathrm{nz1}, \mathrm{lx2} = \mathrm{mx}, \mathrm{ly2} = \mathrm{my}, \mathrm{lz2} = \mathrm{mz}$,
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times (\mathrm{ny2} + 1) \times \mathrm{nz2}$ (nx2:偶数, ny2:偶数のとき) または
$\mathrm{nwk} = \mathrm{nx2} \times (\mathrm{ny2} + 1) \times \mathrm{nz2}$ (nx2:奇数, ny2:偶数のとき) または
$\mathrm{nwk} = (\mathrm{nx2} + 1) \times \mathrm{ny2} \times \mathrm{nz2}$ (nx2:偶数, ny2:奇数のとき) または
$\mathrm{nwk} = \mathrm{nx2} \times \mathrm{ny2} \times \mathrm{nz2}$ (nx2:奇数, ny2:奇数のとき)
$\mathrm{isw} \geq 1$ の場合,
lx1=lx2=mx+1(mx が奇数のとき) または
lx1=lx2=mx+2(mx が偶数のとき),
ly1=ly2=my, lz1=lz2=mz, $\mathrm{nwk} = \mathrm{mx} + 2 \times (\mathrm{my} + \mathrm{mz}) + \mathrm{lx1} \times \mathrm{my} \times \mathrm{mz}$ である.

(g) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) **使用例**

(a) 問 題

次式で定義される 2 つの有限波形を標本化間隔 Δ で離散化し, 離散畳み込みを計算する.

$$f(x,y,z) = \begin{cases} x & ((x,y,z) \in [0,x_f] \times [0,y_f] \times [0,z_f]) \\ 0 & (\text{それ以外}) \end{cases}$$

$$g(x,y,z) = \begin{cases} x_g - x & ((x,y,z) \in [0,x_g] \times [0,y_g] \times [0,z_g]) \\ 0 & (\text{それ以外}) \end{cases}$$

(b) 入力データ

標本化データ

$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * (\mathrm{i_y} + \mathrm{ly1} * \mathrm{i_z})] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta, \mathrm{i_z}\Delta)$

$(i_x = 0, 1, \cdots, \mathrm{nx1} - 1;\ i_y = 0, 1, \cdots, \mathrm{ny1} - 1;\ i_z = 0, 1, \cdots, \mathrm{nz1} - 1)$

$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$

$(j_x = 0, 1, \cdots, \mathrm{nx2} - 1;\ j_y = 0, 1, \cdots, \mathrm{ny2} - 1;\ j_z = 0, 1, \cdots, \mathrm{nz2} - 1)$

ただし, $\Delta = 0.5$

nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 相関

2.15.1 ASL_dfcr1d, ASL_rfcr1d
1 次元相関

(1) **機 能**

任意の整数 k に対して

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

を満たす 2 組の周期 m の離散関数 $f(i)$, $g(j)$ ただし,

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

が与えられているとき次式で定義される離散相関 $q(k)$ $(k = 0, \cdots, m-1)$:

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

を正の方向に $n_1 - 1$ だけシフトした量 $\tilde{q}(k)$ $(k = 0, \cdots, m-1)$

$$\tilde{q}(k) = q(k - (n_1 - 1))\quad (k = 0, \cdots, m-1)$$

を計算する. ただし, $m = \min(n_1 + n_2 - 1, M)$ であり, M は $M \geq \max(n_1, n_2)$ を満たす任意の整数である. なお, 離散相関 $q(k)$ の実フーリエ変換を求めることもできる.

(2) **使用法**

倍精度関数:

ierr = ASL_dfcr1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

単精度関数:

ierr = ASL_rfcr1d (n1, n2, r1, ld1, r2, ld2, m, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n1	I	1	入 力	離散関数 $f(i)$ の有効データ数 n_1
2	n2	I	1	入 力	離散関数 $g(j)$ の有効データ数 n_2
3	r1	{D∗ / R∗}	ld1	入 力	離散関数 $f(i)$ の値 (注意事項 (a), (b) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i)$ の実フーリエ変換結果 (周期 M)
4	ld1	I	1	入 力	配列 r1 の大きさ
5	r2	{D∗ / R∗}	ld2	入 力	離散関数 $g(j)$ の値 (注意事項 (a), (b) 参照)
				出 力	離散関数 $\tilde{q}(k)$ の値またはその実フーリエ変換 (注意事項 (a), (c) 参照)
6	ld2	I	1	入 力	配列 r2 の大きさ
7	m	I	1	入 力	離散関数 $f(i)$, $g(j)$, $\tilde{q}(k)$ の周期 m に対応するパラメータ M (注意事項 (d) 参照)
8	isw	I	1	入 力	処理スイッチ (注意事項 (a), (e) 参照) isw= 0:定義により相関を計算する isw= 1:FFT 法により相関を計算する isw= 2:相関の実フーリエ変換を計算する isw= 3:区分化 FFT 法により相関を計算する
9	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 20 (isw ≥ 1 のとき)
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: n2 (isw= 0 のとき) 2 × m + 1 (isw= 1 または 2, m が奇数のとき) 2 × m + 2 (isw= 1 または 2, m が偶数のとき) 2 × m + n1 (isw= 3, m が奇数のとき) 2 × m + n1 + 1 (isw= 3, m が偶数のとき)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) isw $\in \{0, 1, 2, 3\}$

(b) n1 > 1

(c) n2 > 1

(d) m $\geq \max$(n1, n2)

(e) isw $= 0$ の時 :
ld1 $\geq$ n1
isw > 0 で m が奇数の時 :
ld1 $\geq$ m $+ 1$
isw > 0 で m が偶数の時 :
ld1 $\geq$ m $+ 2$

(f) isw $= 0$ の時 :
ld2 $\geq$ m
isw > 0 で m が奇数の時 :
ld2 $\geq$ m $+ 1$
isw > 0 で m が偶数の時 :
ld2 $\geq$ m $+ 2$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	m $<$ n1 $+$ n2 $- 1$ であった.	相関の計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 相関を計算する関数値の一方の有効データ数が他方に比較して非常に大きい場合, 区分化によりデータ数の多い方を (必要であれば最後に 0 を付加して) 等分し, 本関数を繰り返し適用して離散相関を計算した方が効率が良く, 必要な記憶容量も小さくて済む.
たとえば, 2 つの系列 $\{u_1, u_2, \cdots, u_k\}$(有効データ数 k), $\{v_1, \cdots, v_{pq}\}$(有効データ数 pq $(pq >> k)$) の離散相関を計算したい場合, まず isw=1, n1=k, n2=q, m $\geq$ n1+n2-1, r1=$\{u_1, u_2, \cdots, u_k\}$ r2=$\{v_1, v_2, \cdots, v_q\}$ として本関数を適用する. この結果, 計算したい相関の最初の q 個の値が配列 r2 の先頭から q 個の要素として得られる.
次に, isw=3, r2=$\{v_{q+1}, \cdots, v_{2q}\}$ と変更し, ほかの引数の内容はそのままとして本関数を適用する. この結果, 計算したい相関の次の q 個の値が配列 r2 の先頭から q 個の要素として得られる. 以下同様にして r2 に設定する値を順次ずらして計算する. なお, 最後の繰り返しすなわち r2=$\{v_{(p-1)q+1}, \cdots, v_{pq}\}$ と設定して計算した相関は計算したい相関の最後の $2q - 1$ 個の要素を与える (ただし, 系列 $\{v_j\}$ が有限波形でない場合は最後から $q - 1$ 個は不定).

(b) 配列 r1, r2 にはそれぞれ離散関数 $f(i)$ と離散関数 $g(j)$ の値を次のように格納する. ただし, isw = 3 とする場合には, r2 にのみ値を格納し, r1 の内容はそのまま利用する (注意事項 (a) 参照).

$$\begin{array}{lcl} f(0) & \rightarrow & \mathrm{r1}[0] \\ f(1) & \rightarrow & \mathrm{r1}[1] \\ \cdots & \cdots & \cdots \\ f(n_1-1) & \rightarrow & \mathrm{r1}[\mathrm{n1}-1] \end{array}$$

$$\begin{array}{lcl} g(0) & \rightarrow & \mathrm{r2}[0] \\ g(1) & \rightarrow & \mathrm{r2}[1] \\ \cdots & \cdots & \cdots \\ g(n_2-1) & \rightarrow & \mathrm{r2}[\mathrm{n2}-1] \end{array}$$

なお, 配列 r1, r2 の r1 [n1] 並びに r2 [n2] 以降の要素には値を入力する必要が無い. また, 特に, isw = 3 とする場合には, r2 [n2] 以降の要素は計算に利用するので変更してはならない.

(c) 離散相関 $\tilde{q}(k)$ の値は配列 r2 に以下のように得られる.

$$\begin{array}{lcl} \tilde{q}(0) & \rightarrow & \mathrm{r2}[0] \\ \tilde{q}(1) & \rightarrow & \mathrm{r2}[1] \\ \cdots & \cdots & \cdots \\ \tilde{q}(M-1) & \rightarrow & \mathrm{r2}[\mathrm{m}-1] \end{array}$$

なお, m が奇数のとき r2 [m] が m が偶数のとき r2 [m] と r2 [m+1] がそれぞれ 0.0 となる. また, 区分化を行う場合には, 通常, 最初の n2 個のデータが畳み込みとして意味を持つ (注意事項 (a) 参照).
isw=2 として, 離散相関 $q(k)$ の実フーリエ変換 $Q(j)$:

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

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{array}{lcl} \Re\{Q(0)\} & \leftrightarrow & \mathrm{r2}[0] \\ \Im\{Q(0)\} & \leftrightarrow & \mathrm{r2}[1] \\ \Re\{Q(1)\} & \leftrightarrow & \mathrm{r2}[2] \\ \Im\{Q(1)\} & \leftrightarrow & \mathrm{r2}[3] \\ \cdots & \cdots & \cdots \\ \Re\{Q(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \mathrm{r2}[l-2] \\ \Im\{Q(\lfloor \frac{M}{2} \rfloor)\} & \leftrightarrow & \mathrm{r2}[l-1] \quad (l = \mathrm{m}+1[\text{m:奇数}] \text{ または } \mathrm{m}+2[\text{m:偶数}]) \end{array}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$Q(M-j) = Q(j)^*$$

(ただし, z^* は複素数 z の共役複素数) から得られる. なお, $Q(j)$ は相関を計算するもとの 2 つの関数のクロス・スペクトルの近似量と考えることができる. この場合, $M = n_1 + n_2$ として考えた方が良い. 特に, 相関を計算するもとの 2 つの関数が同じ関数であれば, $Q(j)$ は生のフーリエ・ピリオドグラム (パワー・スペクトルの近似量) に対応し, $Q(j)$ は実数となる.

(d) $\mathrm{m} \geq \mathrm{n1}+\mathrm{n2}-1$ とすれば, 次の周期の相関との重なりを起こさずに相関を計算できる. $\mathrm{m} > \mathrm{n1}+\mathrm{n2}-1$ の場合 n1+n2 以降の要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, $\mathrm{m} = \mathrm{n1}+\mathrm{n2}-1$ とするのがよい. isw ≥ 1 とする場合, m の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n1=n2=145 の

場合, isw=0 のときは, m = 289(=17^2) とした方が良いが, isw ≥ 1 の場合には m = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(e) **通常は isw=1 と設定して FFT 相関を計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ m の選び方に制限がある場合などは isw=0 として計算する.

(f) 非ゼロ部分の開始位置が原点から離れている離散関数の相関を計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i)$, $g(j)$ の非ゼロ部分がそれぞれ区間 $[i_0, i_0+n_1-1]$, $[j_0, j_0+n_2-1]$ のとき

$$\hat{f}(i) = f(i-i_0), \quad \hat{g}(j) = g(j-j_0)$$

として $\hat{f}(i)$, $\hat{g}(j)$ についてこの関数を適用し, 得られた結果を $\tilde{q}(k)$ とすれば, もとの $f(i)$, $g(j)$ の相関 $q(k)$ は

$$q(k) = \tilde{q}(k-(j_0-i_0)+(n_1-1))$$

となる. したがって, $i_0 = j_0 = 0$ の場合でも通常の定義と整合する相関 $q(k)$ を考える場合にはこの関数の適用後, 負の方向に n_1-1 だけシフトして考える必要があり, また, 離散相関を計算する前に $f(i)$, $g(j)$ をそれぞれ負の方向に i_0, j_0 だけシフトしたとすれば, 計算結果をさらに j_0-i_0 だけ正の方向にシフトする必要がある.

(g) この関数で計算する離散相関に標本化間隔を乗じたものは帯域制限された関数の連続相関積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続相関と対応をとる場合には, $q(-n_1) = \tilde{q}(-1) = 0$ として $q(k)$ $(k = -n_1, \cdots, -1, 0, 1, \cdots, n_2-1)$ の n_1+n_2 個のデータを考えた方が, 対応をとりやすい. もちろん, $q(n_1+n_2) = \tilde{q}(n_2) = 0$ として $q(k)$ $(k = -(n_1-1), \cdots, -1, 0, 1, \cdots, n_2)$ を考えても同じである. このとき通常は座標 0 の要素は $q(0)$ に対応させる. ただし,

isw=0 の場合,
ld1=n1, ld2=m, nwk=n2

isw=1 または 2 の場合,
m が奇数ならば ld1=ld2=m+1, nwk = 2 × m + 1
m が偶数ならば ld1=ld2=m+2, nwk = 2 × m + 2

isw=3 の場合,
m が奇数ならば ld1=ld2=m+1, nwk = 2 × m + n1
m が偶数ならば ld1=ld2=m+2, nwk = 2 × m + n1 + 1

である.

(h) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題

次式で定義される 2 つの有限波形を標本化間隔 Δx で離散化し, 離散相関を計算する.

$$f(x) = \begin{cases} x & 0 \le x \le a \\ 0 & \text{それ以外} \end{cases}$$

$$g(x) = \begin{cases} b-x & 0 \le x \le b \\ 0 & \text{それ以外} \end{cases}$$

備考

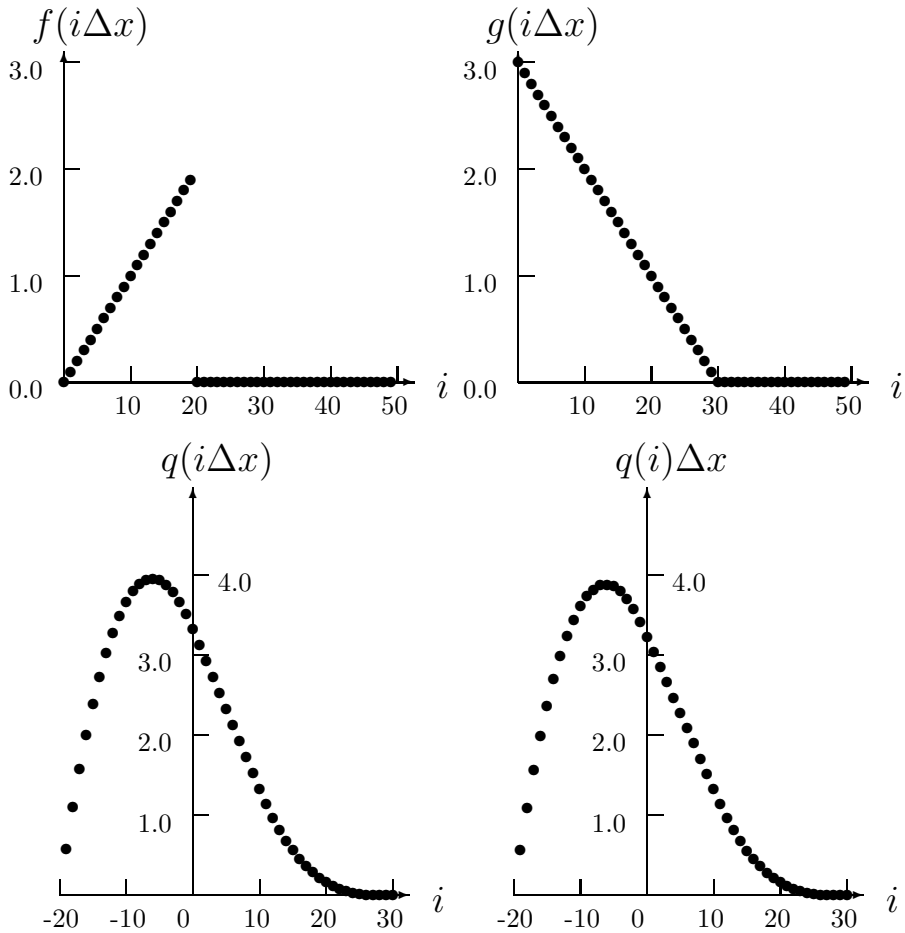
$f(x)$ と $g(x)$ の連続相関 $q(x)$ は

$$q(x) = \int_{-\infty}^{\infty} f(\xi)g(x+\xi)d\xi = \begin{cases} G(-x,a,x) & -a \le x \le 0 \\ G(0,a,x) & 0 \le x \le b-a \\ G(0,b-x,x) & b-a \le x \le b \\ 0 & \text{それ以外} \end{cases}$$

である. ここで, $G(\alpha,\beta,x)$ は

$$\begin{aligned} G(\alpha,\beta,x) &= \left[\frac{\xi^2}{6}(3(b-x)-2\xi)\right]_{\alpha}^{\beta} \\ &= \frac{\xi^2}{6}(3(b-x)-2\xi)\Big|_{\xi=\beta} - \frac{\xi^2}{6}(3(b-x)-2\xi)\Big|_{\xi=\alpha} \end{aligned}$$

$a=2$, $b=3$ とした場合, $f(x)$, $g(x)$, $q(x)$ を $\Delta x = 0.1$ として標本化した値 $f(i\Delta x)$, $g(i\Delta x)$, $q(i\Delta x)$ はそれぞれ以下のようなグラフとなる. 参考のため本関数で計算した離散相関に Δx を乗じた値 $q(i)\Delta x$ も示してある. 標本数が少ないわりには連続相関の値と良く一致する.



なお, 以下の使用例では参考までに連続相関の値も計算している.

(b) 入力データ

標本化データ

r1[i − 1] = f((i − 1)Δx) (i = 1, 2, ⋯, n1)

r2[j − 1] = g((j − 1)Δx) (j = 1, 2, ⋯, n2)

ただし, $\Delta x = 0.1$

n1 $= \dfrac{a}{\Delta x}$, n2 $= \dfrac{b}{\Delta x}$, m, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
2 次元相関

(1) **機 能**

任意の整数 L_x, L_y に対して

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

を満たす 2 組の周期 (m_x, m_y) の多重周期離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$ についてこれらがそれぞれ基本周期内では $(i_x, i_y) \in [0, n_x^{(f)} - 1] \times [0, n_y^{(f)} - 1]$, $(j_x, j_y) \in [0, n_x^{(g)} - 1] \times [0, n_y^{(g)} - 1]$ でのみ非ゼロ値をとるとする. ここで, $[0, a] \times [0, b]$ は平面座標 (i, j) がはる平面の直積領域 (点 (0, 0) と点 (a, b) を対角頂点とする長方形で囲まれる領域) とする. このとき, 次式で定義される離散相関 $q(k_x, k_y)$:

$$\begin{aligned} q(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x + i_x, k_y + i_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

を (k_x, k_y) についてそれぞれ正の方向に $(n_x^{(f)} - 1, n_y^{(f)} - 1)$ だけシフトした量 $\tilde{q}(k_x, k_y)$:

$$\begin{aligned} \tilde{q}(k_x, k_y) &= q(k_x - (n_x^{(f)} - 1), k_y - (n_y^{(f)} - 1)) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

を計算する. ただし, $m_x = \min(n_x^{(f)} + n_x^{(g)} - 1, M_x)$, $m_y = \min(n_y^{(f)} + n_y^{(g)} - 1, M_y)$ であり, M_x, M_y はそれぞれ $M_x \geq \max(n_x^{(f)}, n_x^{(g)})$, $M_y \geq \max(n_y^{(f)}, n_y^{(g)})$ を満たす任意の整数である. なお, 離散相関 $q(k_x, k_y)$ の 2 次元実フーリエ変換を求めることもできる.

(2) **使用法**

倍精度関数:

ierr = ASL_dfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);

単精度関数:

ierr = ASL_rfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx1	I	1	入 力	離散関数 $f(i_x, i_y)$ の i_x 方向の有効データ数 $n_x^{(f)}$
2	ny1	I	1	入 力	離散関数 $f(i_x, i_y)$ の i_y 方向の有効データ数 $n_y^{(f)}$
3	nx2	I	1	入 力	離散関数 $g(j_x, j_y)$ の j_x 方向の有効データ数 $n_x^{(g)}$
4	ny2	I	1	入 力	離散関数 $g(j_x, j_y)$ の j_y 方向の有効データ数 $n_y^{(g)}$
5	r1	{D∗ / R∗}	lx1×ly1	入 力	離散関数 $f(i_x, i_y)$ の値 (注意事項 (a) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i_x, i_y)$ の 2 次元実フーリエ変換結果 (周期 (M_x, M_y))
6	lx1	I	1	入 力	配列 r1 の整合寸法
7	ly1	I	1	入 力	配列 r1 の第 2 寸法
8	r2	{D∗ / R∗}	lx2×ly2	入 力	離散関数 $g(j_x, j_y)$ の値 (注意事項 (a) 参照)
				出 力	離散関数 $\tilde{q}(k_x, k_y)$ の値または $q(k_x, k_y)$ の 2 次元実フーリエ変換 (注意事項 (b) 参照)
9	lx2	I	1	入 力	配列 r2 の整合寸法
10	ly2	I	1	入 力	配列 r2 の第 2 寸法
11	mx	I	1	入 力	離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$, $\tilde{q}(k_x, k_y)$ の周期 (m_x, m_y) に対応するパラメータ M_x (注意事項 (c) 参照)
12	my	I	1	入 力	離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$, $\tilde{q}(k_x, k_y)$ の周期 (m_x, m_y) に対応するパラメータ M_y (注意事項 (c) 参照)
13	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:定義により相関を計算する isw= 1:FFT 法により相関を計算する isw= 2:相関の実フーリエ変換を計算する
14	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 40 (isw ≥ 1 のとき)
15	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: nx2 × ny2 (isw= 0, nx2:奇数のとき) (nx2 + 1) × ny2 (isw= 0, nx2:偶数のとき) mx + 2 × my + max(lx1 × ly1, lx2 × ly2) (isw ≥ 1 のとき)
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{isw} \in \{0, 1, 2\}$

(b) $\mathrm{nx1} > 1$
$\mathrm{ny1} > 1$

(c) $\mathrm{nx2} > 1$
$\mathrm{ny2} > 1$

(d) $\mathrm{mx} \geq \max(\mathrm{nx1}, \mathrm{nx2})$
$\mathrm{my} \geq \max(\mathrm{ny1}, \mathrm{ny2})$

(e) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx1} \geq \mathrm{nx1}$
$\mathrm{ly1} \geq \mathrm{ny1}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 1$
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 2$
$\mathrm{ly1} \geq \mathrm{my}$

(f) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx2} \geq \mathrm{mx}$
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 1$
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 2$
$\mathrm{ly2} \geq \mathrm{my}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{mx} < \mathrm{nx1} + \mathrm{nx2} - 1$ または $\mathrm{my} < \mathrm{ny1} + \mathrm{ny2} - 1$ であった.	相関の計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 配列 r1, r2 の各要素と離散関数 $f(i_x, i_y)$ と離散関数 $g(j_x, j_y)$ の値は以下の様に対応する.

$$\begin{aligned} f(i_x, i_y) &\leftrightarrow \mathrm{r1}[i_x + \mathrm{lx1} * i_y] \\ g(j_x, j_y) &\leftrightarrow \mathrm{r2}[j_x + \mathrm{lx2} * j_y] \end{aligned}$$

ただし, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$, $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列** r1**,** r2 **の整合寸法について** lx1**/2,** ly1**,** lx2**/2,** ly2 **が奇数になるように設定するのが望ましい. 通常, たとえば** mx **が** 4 **の倍数のときは** lx1=mx+**3 とする.**

(b) 離散相関 $\tilde{q}(k_x, k_y)$ の値は配列 r2 の各要素と以下の様に対応する.

$$\tilde{q}(k_x, k_y) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

ただし, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$ である. isw=2 として, 離散相関 $q(k_x, k_y)$ の 2 次元実フーリエ変換 $Q(j_x, j_y)$:

$$\begin{aligned} Q(j_x, j_y) &= \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} q(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor) \end{aligned}$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{aligned} \Re\{Q(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + \mathrm{lx2} * j_y] \\ \Im\{Q(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * j_y] \end{aligned}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$\begin{aligned} Q(M_x - j_x, M_y - j_y)^* &= Q(j_x, j_y) \\ Q(M_x - j_x, j_y)^* &= Q(j_x, M_y - j_y) \end{aligned}$$

(ただし, z^* は複素数 z の共役複素数) から得られる. なお, $Q(j_x, j_y)$ は相関を計算するもとの 2 つの関数のクロス・スペクトルの近似量と考えることができる. この場合, $M_x = n_x^{(f)} + n_x^{(g)}$, $M_y = n_y^{(f)} + n_y^{(g)}$ として考えた方が良い. 特に, 相関を計算するもとの 2 つの関数が同じ関数であれば, $Q(j_x, j_y)$ は生のフーリエ・ピリオドグラム (パワー・スペクトルの近似量) に対応し, $Q(j_x, j_y)$ は実数となる.

(c) $\mathrm{mx} \geq \mathrm{nx1} + \mathrm{nx2} - 1$ かつ $\mathrm{my} \geq \mathrm{ny1} + \mathrm{ny2} - 1$ とすれば, 次の周期の相関との重なりを起こさずに相関を計算できる. $\mathrm{mx} > \mathrm{nx1} + \mathrm{nx2} - 1$ または $\mathrm{my} > \mathrm{ny1} + \mathrm{ny2} - 1$ の場合

$$\tilde{q}(k_x, k_y) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

$k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1$; $k_y = 0, \cdots, \mathrm{my} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1$; $k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1$ に対応する要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, $\mathrm{mx} = \mathrm{nx1} + \mathrm{nx2} - 1$, $\mathrm{my} = \mathrm{ny1} + \mathrm{ny2} - 1$ とするのがよい. $\mathrm{isw} \geq 1$ とする場合, mx, my の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx1=nx2=145 の場合, isw=0 のときは, $\mathrm{mx} = 289 (= 17^2)$ とした方が良いが, $\mathrm{isw} \geq 1$ の場合には $\mathrm{mx} = 300 (= 2^2 \times 3 \times 5^2)$ や $320 (= 2^6 \times 5)$, $384 (= 2^7 \times 3)$ などとした方が通常は効率が良い.

(d) **通常は** isw=**1 と設定して FFT 相関を計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ mx や my の選び方に制限がある場合などは isw=0 として計算する.

(e) 非ゼロ部分の開始位置が原点から離れている離散関数の相関を計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i_x, i_y)$, $g(j_x, j_y)$ の非ゼロ部分が i_x, j_x についてそれぞれ区間 $[i_0, i_0 + n_x^{(f)} - 1]$, $[j_0, j_0 + n_x^{(g)} - 1]$ のとき

$$\hat{f}(i_x, i_y) = f(i_x - i_0, i_y), \quad \hat{g}(j_x, j_y) = g(j_x - j_0, j_y)$$

として $\hat{f}(i_x, i_y)$, $\hat{g}(j_x, j_y)$ についてこの関数を適用し, 得られた結果を $\tilde{q}(k_x, k_y)$ とすれば, もとの $f(i_x, i_y)$, $g(j_x, j_y)$ の相関 $q(k_x, k_y)$ は

$$q(k_x, k_y) = \tilde{q}(k_x - (j_0 - i_0) + (n_x^{(f)} - 1), k_y)$$

となる. したがって, $i_0 = j_0 = 0$ の場合でも通常の定義と整合する相関 $q(k_x, k_y)$ を考える場合にはこの関数の適用後, k_x の負の方向に $n_x^{(f)} - 1$ だけシフトして考える必要があり, また, 離散相関を計算する前に $f(i_x, i_y)$, $g(j_x, j_y)$ をそれぞれ i_x, j_x の負の方向に i_0, j_0 だけシフトしたとすれば, 計算結果をさらに $j_0 - i_0$ だけ k_x の正の方向にシフトする必要がある. i_y, j_y, k_y についても同様である.

(f) この関数で計算する離散相関に標本化間隔の 2 乗を乗じたものは帯域制限された関数の連続相関積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続相関と対応をとる場合には, $q(-n_x^{(f)}, k_y) = \tilde{q}(-1, k_y) = 0$, $q(k_x, -n_y^{(f)}) = \tilde{q}(k_x, -1) = 0$ として $q(k_x, k_y)$ $(k_x = -n_x^{(f)}, \cdots, -1, 0, 1, \cdots, n_x^{(g)} - 1;\ k_y = -n_y^{(f)}, \cdots, -1, 0, 1, \cdots, n_y^{(g)} - 1)$ の $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})$ 個のデータを考えた方が, 対応をとりやすい. もちろん, $q(n_x^{(f)} + n_x^{(g)}, k_y) = \tilde{q}(n_x^{(g)}, k_y) = 0$, $q(k_x, n_y^{(f)} + n_y^{(g)}) = \tilde{q}(k_x, n_y^{(g)}) = 0$ として $q(k_x, k_y)$ $(k_x = -(n_x^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_x^{(g)};\ k_y = -(n_y^{(f)} - 1), \cdots, -1, 0, 1, \cdots, n_y^{(g)})$ を考えても同じである. このとき通常は座標 (0, 0) の要素は $q(0, 0)$ に対応させる. ただし,

isw=0 の場合,
lx1 = nx1, ly1 = ny1, lx2 = mx, ly2 = my,
nwk = nx2 × ny2(nx2:奇数のとき) または
nwk = (nx2 + 1) × ny2(nx2:偶数のとき)

isw ≥ 1 の場合,
lx1=lx2=mx+1(mx が奇数のとき) または
lx1=lx2=mx+2(mx が偶数のとき),
ly1=ly2=my, nwk = mx + (lx1 + 2) × my

である.

(g) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題
次式で定義される 2 つの有限波形を標本化間隔 Δ で離散化し, 離散相関を計算する.

$$f(x, y) = \begin{cases} x & ((x, y) \in [0, x_f] \times [0, y_f]) \\ 0 & (\text{それ以外}) \end{cases}$$

$$g(x, y) = \begin{cases} x_g - x & ((x, y) \in [0, x_g] \times [0, y_g]) \\ 0 & (\text{それ以外}) \end{cases}$$

(b) 入力データ
標本化データ

$r1[i_x + lx1 * i_y] = f(i_x\Delta, i_y\Delta)$ $(i_x = 0, 1, \cdots, nx1 - 1;\ i_y = 0, 1, \cdots, ny1 - 1)$

$r2[j_x + lx2 * j_y] = g(j_x\Delta, j_y\Delta)$ $(j_x = 0, 1, \cdots, nx2 - 1;\ j_y = 0, 1, \cdots, ny2 - 1)$

ただし, $\Delta = 0.5$

nx1, ny1, nx2, ny2, mx, my, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
3 次元相関

(1) **機 能**

任意の整数 L_x, L_y, L_z に対して

$$\begin{aligned} f(i_x,i_y,i_z) &= f(i_x+L_xm_x, i_y+L_ym_y, i_z+L_zm_z), \\ g(j_x,j_y,j_z) &= g(j_x+L_xm_x, j_y+L_ym_y, j_z+L_zm_z), \\ & \quad (i_x,j_x=0,\cdots,m_x-1;\ i_y,j_y=0,\cdots,m_y-1;\ i_z,j_z=0,\cdots,m_z-1) \end{aligned}$$

を満たす 2 組の周期 (m_x,m_y,m_z) の多重周期離散関数 $f(i_x,i_y,i_z)$, $g(j_x,j_y,j_z)$ についてこれらがそれぞれ基本周期内では $(i_x,i_y,i_z)\in[0,n_x^{(f)}-1]\times[0,n_y^{(f)}-1]\times[0,n_z^{(f)}-1]$, $(j_x,j_y,j_z)\in[0,n_x^{(g)}-1]\times[0,n_y^{(g)}-1]\times[0,n_z^{(g)}-1]$ でのみ非ゼロ値をとるとする．ここで, $[0,a]\times[0,b]\times[0,c]$ は空間座標 (i,j,k) がはる空間の直積領域 (点 (0, 0, 0) と点 (a,b,c) を対角頂点とする直方体で囲まれる領域) とする．このとき, 次式で定義される離散相関 $q(k_x,k_y,k_z)$:

$$\begin{aligned} q(k_x,k_y,k_z) &= \sum_{i_x=0}^{m_x-1}\sum_{i_y=0}^{m_y-1}\sum_{i_z=0}^{m_z-1} f(i_x,i_y,i_z)g(k_x+i_x,k_y+i_y,k_z+i_z) \\ & \quad (k_x=0,\cdots,m_x-1;\ k_y=0,\cdots,m_y-1;\ k_z=0,\cdots,m_z-1) \end{aligned}$$

を (k_x,k_y,k_z) についてそれぞれ正の方向に $(n_x^{(f)}-1, n_y^{(f)}-1, n_z^{(f)}-1)$ だけシフトした量 $\tilde{q}(k_x,k_y,k_z)$:

$$\begin{aligned} \tilde{q}(k_x,k_y,k_z) &= q(k_x-(n_x^{(f)}-1), k_y-(n_y^{(f)}-1), k_z-(n_z^{(f)}-1)) \\ & \quad (k_x=0,\cdots,m_x-1;\ k_y=0,\cdots,m_y-1;\ k_z=0,\cdots,m_z-1) \end{aligned}$$

を計算する．ただし, $m_x=\min(n_x^{(f)}+n_x^{(g)}-1, M_x)$, $m_y=\min(n_y^{(f)}+n_y^{(g)}-1, M_y)$, $m_z=\min(n_z^{(f)}+n_z^{(g)}-1, M_z)$ であり, M_x, M_y, M_z はそれぞれ $M_x\ge\max(n_x^{(f)}, n_x^{(g)})$, $M_y\ge\max(n_y^{(f)}, n_y^{(g)})$, $M_z\ge\max(n_z^{(f)}, n_z^{(g)})$ を満たす任意の整数である．なお, 離散相関 $q(k_x,k_y,k_z)$ の 3 次元実フーリエ変換を求めることもできる．

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk);
```

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　R:単精度実数型　C:単精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx1	I	1	入 力	離散関数 $f(i_x,i_y,i_z)$ の i_x 方向の有効データ数 $n_x^{(f)}$
2	ny1	I	1	入 力	離散関数 $f(i_x,i_y,i_z)$ の i_y 方向の有効データ数 $n_y^{(f)}$
3	nz1	I	1	入 力	離散関数 $f(i_x,i_y,i_z)$ の i_z 方向の有効データ数 $n_z^{(f)}$
4	nx2	I	1	入 力	離散関数 $g(j_x,j_y,j_z)$ の j_x 方向の有効データ数 $n_x^{(g)}$
5	ny2	I	1	入 力	離散関数 $g(j_x,j_y,j_z)$ の j_y 方向の有効データ数 $n_y^{(g)}$
6	nz2	I	1	入 力	離散関数 $g(j_x,j_y,j_z)$ の j_z 方向の有効データ数 $n_z^{(g)}$
7	r1	{D∗ / R∗}	lx1×ly1×lz1	入 力	離散関数 $f(i_x,i_y,i_z)$ の値 (注意事項 (a) 参照)
				出 力	isw ≥ 1 のとき離散関数 $f(i_x,i_y,i_z)$ の 3 次元実フーリエ変換結果 (周期 (M_x,M_y,M_z))
8	lx1	I	1	入 力	配列 r1 の整合寸法
9	ly1	I	1	入 力	配列 r1 の第 2 寸法
10	lz1	I	1	入 力	配列 r1 の第 3 寸法
11	r2	{D∗ / R∗}	lx2×ly2×lz2	入 力	離散関数 $g(j_x,j_y,j_z)$ の値 (注意事項 (a) 参照)
				出 力	離散関数 $\tilde{q}(k_x,k_y,k_z)$ の値 または $q(k_x,k_y,k_z)$ の 3 次元実フーリエ変換 (注意事項 (b) 参照)
12	lx2	I	1	入 力	配列 r2 の整合寸法
13	ly2	I	1	入 力	配列 r2 の第 2 寸法
14	lz2	I	1	入 力	配列 r2 の第 3 寸法
15	mx	I	1	入 力	離散関数 $f(i_x,i_y,i_z)$, $g(j_x,j_y,j_z)$, $\tilde{q}(k_x,k_y,k_z)$ の周期 (m_x,m_y,m_z) に対応するパラメータ M_x (注意事項 (c) 参照)
16	my	I	1	入 力	離散関数 $f(i_x,i_y,i_z)$, $g(j_x,j_y,j_z)$, $\tilde{q}(k_x,k_y,k_z)$ の周期 (m_x,m_y,m_z) に対応するパラメータ M_y (注意事項 (c) 参照)

項番	引数と戻り値	型	大きさ	入出力	内　　容
17	mz	I	1	入 力	離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$, $\tilde{q}(k_x, k_y, k_z)$ の周期 (m_x, m_y, m_z) に対応するパラメータ M_z (注意事項 (c) 参照)
18	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw= 0:定義により相関を計算する isw= 1:FFT 法により相関を計算する isw= 2:相関の実フーリエ変換を計算する
19	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: 0 (isw= 0 のとき) 60 (isw $\geq$ 1 のとき)
20	wk	$\left\{\begin{matrix}\text{D}* \\ \text{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: (nx2+1)×(ny2+1)×nz2 (isw= 0, nx2:偶数, ny2:偶数のとき) nx2 × (ny2 + 1) × nz2 (isw= 0, nx2:奇数, ny2:偶数のとき) (nx2 + 1) × ny2 × nz2 (isw= 0, nx2:偶数, ny2:奇数のとき) nx2 × ny2 × nz2 (isw= 0, nx2:奇数, ny2:奇数のとき) mx + 2 × (my + mz) + max(lx1 × ly1 × lz1, lx2 × ly2 × lz2) (isw $\geq$ 1 のとき)
21	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{isw} \in \{0, 1, 2\}$

(b) $\mathrm{nx1} > 1$
$\mathrm{ny1} > 1$
$\mathrm{nz1} > 1$

(c) $\mathrm{nx2} > 1$
$\mathrm{ny2} > 1$
$\mathrm{nz2} > 1$

(d) $\mathrm{mx} \geq \max(\mathrm{nx1}, \mathrm{nx2})$
$\mathrm{my} \geq \max(\mathrm{ny1}, \mathrm{ny2})$
$\mathrm{mz} \geq \max(\mathrm{nz1}, \mathrm{nz2})$

(e) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx1} \geq \mathrm{nx1}$
$\mathrm{ly1} \geq \mathrm{ny1}$
$\mathrm{lz1} \geq \mathrm{nz1}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 1$, lx1 は偶数
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{lz1} \geq \mathrm{mz}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx1} \geq \mathrm{mx} + 2$, lx1 は偶数
$\mathrm{ly1} \geq \mathrm{my}$
$\mathrm{lz1} \geq \mathrm{mz}$

(f) $\mathrm{isw} = 0$ の時 :
$\mathrm{lx2} \geq \mathrm{mx}$
$\mathrm{ly2} \geq \mathrm{ny2}$
$\mathrm{lz2} \geq \mathrm{nz2}$
$\mathrm{isw} > 0$ で mx が奇数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 1$, lx2 は偶数
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{lz2} \geq \mathrm{mz}$
$\mathrm{isw} > 0$ で mx が偶数の時 :
$\mathrm{lx2} \geq \mathrm{mx} + 2$, lx2 は偶数
$\mathrm{ly2} \geq \mathrm{my}$
$\mathrm{lz2} \geq \mathrm{mz}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	mx $<$ nx1 + nx2 − 1, my $<$ ny1 + ny2 − 1 または mz $<$ nz1 + nz2 − 1 であった.	相関の計算で重なりが発生する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	
3040	制限条件 (e) を満足しなかった.	
3050	制限条件 (f) を満足しなかった.	

(6) **注意事項**

(a) 配列 r1, r2 の各要素と離散関数 $f(i_x, i_y, i_z)$ と離散関数 $g(j_x, j_y, j_z)$ の値は以下の様に対応する.

$$\begin{aligned} f(i_x, i_y, i_z) &\leftrightarrow \mathrm{r1}[i_x + \mathrm{lx1} * (i_y + \mathrm{ly1} * i_z)] \\ g(j_x, j_y, j_z) &\leftrightarrow \mathrm{r2}[j_x + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \end{aligned}$$

ただし, $i_x = 0, \cdots, n_x^{(f)} - 1$; $i_y = 0, \cdots, n_y^{(f)} - 1$; $i_z = 0, \cdots, n_z^{(f)} - 1$, $j_x = 0, \cdots, n_x^{(g)} - 1$; $j_y = 0, \cdots, n_y^{(g)} - 1$; $j_z = 0, \cdots, n_z^{(g)} - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列** r1, r2 **の整合寸法について** lx1**/2,** ly1**,** lz1**,** lx2**/2,** ly2**,** lz2 **が奇数になるように設定するのが望ましい. また, 高速化のために配列** r1, r2 **内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば** mx **が (4 の倍数)+2 のときは** lx1=mx+4 **とする.**

(b) 離散相関 $\tilde{q}(k_x, k_y, k_z)$ の値は配列 r2 の各要素と以下の様に対応する.

$$\tilde{q}(k_x, k_y, k_z) \leftrightarrow \mathrm{r2}[k_x + \mathrm{lx2} * (k_y + \mathrm{ly2} * k_z)]$$

ただし, $k_x = 0, \cdots, M_x - 1$; $k_y = 0, \cdots, M_y - 1$; $k_z = 0, \cdots, M_z - 1$ である. isw=2 として, 離散相関 $q(k_x, k_y, k_z)$ の 3 次元実フーリエ変換 $Q(j_x, j_y, j_z)$:

$$\begin{aligned} Q(j_x, j_y, j_z) &= \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} q(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z})} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor) \end{aligned}$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{aligned} \Re\{Q(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{r2}[2 * j_x + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \\ \Im\{Q(j_x, j_y, j_z)\} &\leftrightarrow \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * (j_y + \mathrm{ly2} * j_z)] \end{aligned}$$

と対応する. なお, この場合, 得られるフーリエ変換は正規化されていることに注意する必要がある. フーリエ変換の残りの半周期分は実フーリエ変換の対称性

$$\begin{aligned} Q(M_x - j_x, M_y - j_y, M_z - j_z)^* &= Q(j_x, j_y, j_z) \\ Q(M_x - j_x, j_y, j_z)^* &= Q(j_x, M_y - j_y, M_z - j_z) \\ Q(M_x - j_x, M_y - j_y, j_z)^* &= Q(j_x, j_y, M_z - j_z) \end{aligned}$$

(ただし, z^* は複素数 z の共役複素数) から得られる. なお, $Q(j_x, j_y, j_z)$ は相関を計算するもとの 2 つの関数のクロス・スペクトルの近似量と考えることができる. この場合, $M_x = n_x^{(f)} + n_x^{(g)}$, $M_y = n_y^{(f)} + n_y^{(g)}$,

$M_z = n_z^{(f)} + n_z^{(g)}$ として考えた方が良い. 特に, 相関を計算するもとの 2 つの関数が同じ関数であれば, $Q(j_x, j_y, j_z)$ は生のフーリエ・ピリオドグラム (パワー・スペクトルの近似量) に対応し, $Q(j_x, j_y, j_z)$ は実数となる.

(c) mx $\geq$ nx1 + nx2 − 1 かつ my $\geq$ ny1 + ny2 − 1 かつ mz $\geq$ nz1 + nz2 − 1 とすれば, 次の周期の相関との重なりを起こさずに相関を計算できる. mx $>$ nx1 + nx2 − 1 または my $>$ ny1 + ny2 − 1 または mz $>$ nz1 + nz2 − 1 の場合

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2}[k_x + \mathrm{lx2} * (k_y + \mathrm{ly2} * k_z)]$$

$k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = \mathrm{ny1} + \mathrm{ny2} - 1, \cdots, \mathrm{my} - 1;\ k_z = 0, \cdots, \mathrm{mz} - 1$ または $k_x = 0, \cdots, \mathrm{mx} - 1;\ k_y = 0, \cdots, \mathrm{my} - 1;\ k_z = \mathrm{nz1} + \mathrm{nz2} - 1, \cdots, \mathrm{mz} - 1$ に対応する要素には誤差の範囲で 0.0 と一致する値が格納される. isw=0 のときは, mx = nx1 + nx2 − 1, my = ny1 + ny2 − 1, mz = nz1 + nz2 − 1 とするのがよい. isw ≥ 1 とする場合, mx, my, mz の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx1=nx2=145 の場合, isw=0 のときは, mx = 289(=17^2) とした方が良いが, isw ≥ 1 の場合には mx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が通常は効率が良い.

(d) **通常は** isw**=1 と設定して FFT 相関を計算した方が効率良く計算を行える.** ただし, 作業領域を節約したい場合やパラメータ mx や my, mz の選び方に制限がある場合などは isw=0 として計算する.

(e) 非ゼロ部分の開始位置が原点から離れている離散関数の相関を計算したい場合には, まず開始位置が原点に来るようにシフトして計算した後, 計算結果を再度シフトして最終結果を得た方が効率が良い. 例えば, 離散関数 $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ の非ゼロ部分が i_x, j_x についてそれぞれ区間 $[i_0, i_0 + n_x^{(f)} - 1]$, $[j_0, j_0 + n_x^{(g)} - 1]$ のとき

$$\hat{f}(i_x, i_y, i_z) = f(i_x - i_0, i_y, i_z), \quad \hat{g}(j_x, j_y, j_z) = g(j_x - j_0, j_y, j_z)$$

として $\hat{f}(i_x, i_y, i_z)$, $\hat{g}(j_x, j_y, j_z)$ についてこの関数を適用し, 得られた結果を $\tilde{q}(k_x, k_y, k_z)$ とすれば, もとの $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ の相関 $q(k_x, k_y, k_z)$ は

$$q(k_x, k_y, k_z) = \tilde{q}(k_x - (j_0 - i_0) + (n_x^{(f)} - 1), k_y, k_z)$$

となる. したがって, $i_0 = j_0 = 0$ の場合でも通常の定義と整合する相関 $q(k_x, k_y, k_z)$ を考える場合にはこの関数の適用後, k_x の負の方向に $n_x^{(f)} - 1$ だけシフトして考える必要があり, また, 離散相関を計算する前に $f(i_x, i_y, i_z)$, $g(j_x, j_y, j_z)$ をそれぞれ i_x, j_x の負の方向に i_0, j_0 だけシフトしたとすれば, 計算結果をさらに $j_0 - i_0$ だけ k_x の正の方向にシフトする必要がある. i_y, j_y, k_y; i_z, j_z, k_z についても同様である.

(f) この関数で計算する離散相関に標本化間隔の 3 乗を乗じたものは帯域制限された関数の連続相関積分を方形近似 (台形公式による近似でもある) したものになる. したがって, 近似精度を上げるためには, 標本化間隔を小さくとり, 標本データ数を大きくとる必要がある. なお, 連続相関と対応をとる場合には,

$$q(-n_x^{(f)}, k_y, k_z) = \tilde{q}(-1, k_y, k_z) = 0,$$
$$q(k_x, -n_y^{(f)}, k_z) = \tilde{q}(k_x, -1, k_z) = 0,$$
$$q(k_x, k_y, -n_z^{(f)}) = \tilde{q}(k_x, k_y, -1) = 0$$

として $q(k_x, k_y, k_z)$ ($k_x = -n_x^{(f)}, \cdots, -1, 0, 1, \cdots, n_x^{(g)} - 1;\ k_y = -n_y^{(f)}, \cdots, -1, 0, 1, \cdots, n_y^{(g)} - 1;\ k_z = -n_z^{(f)}, \cdots, -1, 0, 1, \cdots, n_z^{(g)} - 1$) の $(n_x^{(f)} + n_x^{(g)})(n_y^{(f)} + n_y^{(g)})(n_z^{(f)} + n_z^{(g)})$ 個のデータを考えた方が, 対応をとりやすい. もちろん,

$$q(n_x^{(f)} + n_x^{(g)}, k_y, k_z) = \tilde{q}(n_x^{(g)}, k_y, k_z) = 0,$$
$$q(k_x, n_y^{(f)} + n_y^{(g)}, k_z) = \tilde{q}(k_x, n_y^{(g)}, k_z) = 0,$$
$$q(k_x, k_y, n_z^{(f)} + n_z^{(g)}) = \tilde{q}(k_x, k_y, n_y^{(g)}) = 0,$$

として $q(k_x, k_y, k_z)$ ($k_x = -(n_x^{(f)}-1), \cdots, -1, 0, 1, \cdots, n_x^{(g)}$; $k_y = -(n_y^{(f)}-1), \cdots, -1, 0, 1, \cdots, n_y^{(g)}$; $k_z = -(n_z^{(f)}-1), \cdots, -1, 0, 1, \cdots, n_z^{(g)}$) を考えても同じである. このとき通常は座標 (0, 0, 0) の要素は $q(0,0,0)$ に対応させる. ただし,

isw=0 の場合,

lx1 = nx1, ly1 = ny1, lz1 = nz1, lx2 = mx, ly2 = my, lz2 = mz,

nwk = (nx2 + 1) × (ny2 + 1) × nz2 (nx2:偶数, ny2:偶数のとき) または

nwk = nx2 × (ny2 + 1) × nz2 (nx2:奇数, ny2:偶数のとき) または

nwk = (nx2 + 1) × ny2 × nz2 (nx2:偶数, ny2:奇数のとき) または

nwk = nx2 × ny2 × nz2 (nx2:奇数, ny2:奇数のとき)

isw ≥ 1 の場合,

lx1=lx2=mx+1(mx が奇数のとき) または

lx1=lx2=mx+2(mx が偶数のとき),

ly1=ly2=my, lz1=lz2=mz, nwk = mx + 2 × (my + mz) + lx1 × my × mz

である.

(g) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題

次式で定義される 2 つの有限波形を標本化間隔 Δ で離散化し, 離散相関を計算する.

$$f(x,y,z) = \begin{cases} x & ((x,y,z) \in [0,x_f] \times [0,y_f] \times [0,z_f]) \\ 0 & (\text{それ以外}) \end{cases}$$

$$g(x,y,z) = \begin{cases} x_g - x & ((x,y,z) \in [0,x_g] \times [0,y_g] \times [0,z_g]) \\ 0 & (\text{それ以外}) \end{cases}$$

(b) 入力データ

標本化データ

r1[i_x + lx1 ∗ (i_y + ly1 ∗ i_z)] = f(i_xΔ, i_yΔ, i_zΔ) (i_x = 0, 1, ⋯, nx1 − 1; i_y = 0, 1, ⋯, ny1 − 1; i_z = 0, 1, ⋯, nz1 − 1)

r2[j_x + lx2 ∗ (j_y + ly2 ∗ j_z)] = g(j_xΔ, j_yΔ, j_zΔ) (j_x = 0, 1, ⋯, nx2 − 1; j_y = 0, 1, ⋯, ny2 − 1; j_z = 0, 1, ⋯, nz2 − 1)

ただし, Δ = 0.5

nx1, ny1, nz1, nx2, ny2, nz2, mx, my, mz, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 パワー・スペクトル解析

2.16.1 ASL_dfps1d, ASL_rfps1d
1 次元フーリエ・ピリオドグラム

(1) **機 能**

系列 u_j $(j = 0, \cdots, n-1)$ の (修正) フーリエ・ピリオドグラムを求める. フーリエ・ピリオドグラム p_k は次式で定義される.

$$p_k = \frac{\left| \sum_{j=0}^{n-1} w_j u_j e^{-2\pi\sqrt{-1}\frac{jk}{n}} \right|^2}{n\beta} \quad (k = 0, 1, \cdots, \lfloor \frac{n}{2} \rfloor)$$

ただし, $\lfloor x \rfloor$ は x を超えない最大の整数を表す. w_j は打ち切り関数 (窓関数) であり, 生のフーリエ・ピリオドグラムの場合には, $w_j = 1$ $(j = 0, \cdots, n-1)$, $\beta = n$ とし, 修正ピリオドグラムの場合には

$$\beta = \begin{cases} \sum_{j=0}^{n-1} {w_j}^2 & \text{(窓関数によるパワー補正式を用いる場合)} \\ n & \text{(それ以外)} \end{cases}$$

とする. なお, ピリオドグラム p_k は両側パワー・スペクトルの半周期分 (周期 n) に相当し, 残りは $p_{-k} = p_k$ の関係より得られる. また, 対応する系列の全パワーは,

$$\frac{\sum_{j=0}^{n-1} \{u_j\}^2}{n}$$

である.

(2) **使用法**

倍精度関数:

ierr = ASL_dfps1d (n, r, ld, isw, iwk, wk);

単精度関数:

ierr = ASL_rfps1d (n, r, ld, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	系列 u_j の長さ n (注意事項 (d) 参照)
2	r	{D∗ / R∗}	ld	入 力	系列 u_j の値 (注意事項 (a) 参照)
				出 力	系列 u_j のフーリエ・ピリオドグラム p_k の値 (注意事項 (b), (c) 参照)
3	ld	I	1	入 力	配列 r の大きさ
4	isw	I	1	入 力	処理スイッチ (注意事項 (e) 参照) isw= 0:生のフーリエ・ピリオドグラムを計算する isw=±1:ユーザ定義窓関数を利用して計算する isw=±2:Hanning 窓関数を利用して計算する isw=±3:Bartlett 窓関数を利用して計算する isw=±4:Welch 窓関数を利用して計算する isw=±5:Parzen 窓関数を利用して計算する なお, 窓関数によるパワー補正式を用いる場合は isw >0, それ以外の場合は isw <0 とする.
5	iwk	I*	20	ワーク	作業領域
6	wk	{D∗ / R∗}	n+ld	ワーク	作業領域 isw=±1 の場合には, ユーザ定義窓関数の値を入力する. (注意事項 (e) 参照)
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) n > 1

(c) n が奇数の時：
ld $\geq$ n $+ 1$
n が偶数の時：
ld $\geq$ n $+ 2$

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3030	制限条件 (c) を満足しなかった.	
4000	isw $= 1$ の時に, ユーザ定義窓関数が $w_j = 0$ $(j = 0, \cdots, n-1)$ であった.	

(6) **注意事項**

(a) 配列 r にはそれぞれ系列 u_j の値を次のように格納する.

$$\begin{array}{lcl} u_0 & \rightarrow & \mathrm{r}[0] \\ u_1 & \rightarrow & \mathrm{r}[1] \\ \cdots & \cdots & \cdots \\ u_{n-1} & \rightarrow & \mathrm{r}[\mathrm{n}-1] \end{array}$$

なお, 配列 r の r [n] 以降の要素には値を入力する必要が無い.

(b) フーリエ・ピリオドグラム p_k の値は配列 r に以下のように得られる.

$$\begin{array}{lcl} p_0 & \rightarrow & \mathrm{r}[0] \\ p_1 & \rightarrow & \mathrm{r}[1] \\ \cdots & \cdots & \cdots \\ p_{\lfloor \frac{n+1}{2} \rfloor -1} & \rightarrow & \mathrm{r}[\lfloor \frac{n+1}{2} \rfloor - 1] \end{array}$$

なお, $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

(c) 得られるフーリエ･ピリオドグラム p_k は両側パワー･スペクトル (負の周波数を考える場合に相当) の半周期に対応し, 対応する周波数 ξ_k は $\xi_k = \frac{k}{n\Delta x}$ (Δx:標本化間隔) で与えられる. このとき $-\xi_k$ に対応する成分は p_k となる. 片側スペクトルに対応するフーリエ･ピリオドグラム $\hat{p}_k$ は $\hat{p}_0 = p_0$; $\hat{p}_k = 2p_k$ $(k = 1, 2, \cdots, m-1)$ とすることによって得られる. ただし, n が偶数の場合には $m = \frac{n}{2}$, $\hat{p}_m = p_m$ とし, n が奇数の場合は $m = \frac{n+1}{2}$ とする.

(d) 系列 u_j の長さ n の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, n = 289(=17^2) とするよりも n = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が効率が良い. なお, データ数を大きくできない場合には, データの最後に 0 を必要なだけ補って n を調整して計算を行う.

(e) 処理スイッチ isw の値によって, 以下の様に打ち切り関数 (窓関数) を変更することができる.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{isw= } \pm 2 \text{ (Hanning 窓)} \\ 1 - |2v_j - 1| & \text{isw= } \pm 3 \text{ (Bartlett 窓)} \\ 1 - (2v_j - 1)^2 & \text{isw= } \pm 4 \text{ (Welch 窓)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & \text{isw= } \pm 5 \text{ (Parzen 窓)} \end{cases}$$

ただし, $v_j = \frac{j}{n}$. したがって, 上述のような窓関数を用いる場合には系列 u_j の最初の要素 u_0 は修正ピリオドグラムの計算に影響しない. これを避けたい場合には, 実際に計算したい系列の長さよりも 1 大きい数を n に指定し, u_1 以降に有効なデータを設定すれば良い. なお, 窓関数は $|x| \le 1$ でのみ非ゼロとなる時間 (または空間) 領域関数としてそれぞれ次の様に表される.

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

また, ユーザ独自の窓関数値 w_j を利用したい場合には isw = ±1 として作業配列 wk に

$$\mathrm{wk}[j] = w_j \quad (j = 0, \cdots, n-1)$$

と設定してこの関数を呼び出す.

(f) 生のピリオドグラムはその定義から自己相関関数の離散フーリエ変換近似とみなせる. 有効データ数 n の離散関数の自己相関関数の有効データ長は $2n-1$ であるので, 一般の関数のパワー・スペクトルを生のピリオドグラムで近似することは, 1 つの周期が以下のように与えられる方形打ち切り関数 $w(k)$ で関数を打ち切ったことに相当する.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n-1 \\ 0 & \text{それ以外} \end{cases}$$

方形関数のフーリエ変換は周波数を f とした場合, $\frac{\sin f}{f}$ 型の関数形をしており, 中心周波数の周りに小さくないサイドローブを持っている. したがって, たとえば, 周期関数を 1 周期の整数倍でない幅で単純に打ち切って標本化した場合, 周波数領域では, 生のピリオドグラムはパワー・スペクトルを求めたい周期関数のフーリエ変換と $\frac{\sin f}{f}$ 型関数との畳み込みとなるので, **漏れ (leakage)** と呼ばれる余分な周波数成分が発生する. このような**漏れ**を抑止するためには, 単純な打ち切りを行わずに Hanning 窓関数のような周波数領域でのサイドローブが小さい打ち切り関数を用いる. ただし, 一般に漏れを抑圧すればする程離散フーリエ変換の結果は拡がりぼやけたものとなる. したがって, パワー・スペクトルを推定する場合, 目的に応じて, すなわち, スペクトル幅を問題としているの中心周波数を問題としているのか等に応じて, 適切な打ち切り関数を選択する必要がある.

(g) 離散フーリエ変換の分解度 (周波数領域での標本間隔)$\frac{1}{nT}$ を上げるには標本数 n を増やすか標本間隔 T を増やせば良いが, 標本間隔と分解度を一定に保った状態でパワー・スペクトルの推定値の精度を上げるために, 標本数 n の標本を m 組とって m 組それぞれについて修正ピリオドグラムを求めてその平均をとるという手法がよく取られる. この場合, 系列から m 組の標本データをオーバーラップして取るというような手法も提案されている. 詳細は参考文献等を参照されたい.

(h) パワー・スペクトルを求める場合, フーリエ変換の周波数推移に関する性質すなわち時間 (または空間) 領域で $e^{2\pi\sqrt{-1}f_0 t}$ を掛けることは周波数領域では周波数を f_0 だけシフトすることに対応し, 関数の形状は変わらないという性質を利用して, パワー・スペクトルの中心周波数をあらかじめシフトして計算することで計算に必要なデータ点数を削減するという手法も良く用いられる. なお, このような操作は**変調 (modulation)** として知られている. ただし, n が奇数ならば ld=n+1, n が偶数ならば ld=n+2 である.

(i) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

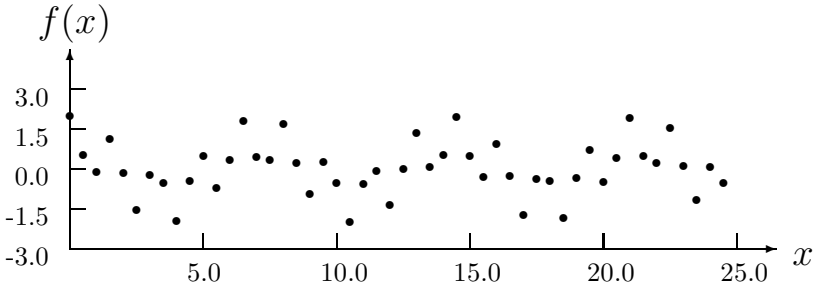
(a) 問 題

次式で定義される波形を標本化間隔 Δx で離散化し, フーリエ・ピリオドグラムを計算し, パワー・スペクトルを推定する.

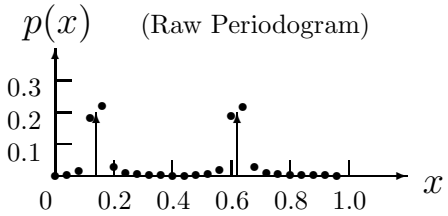
$$f(x) = \cos 2\pi f_1 x + \cos 2\pi f_2 x$$

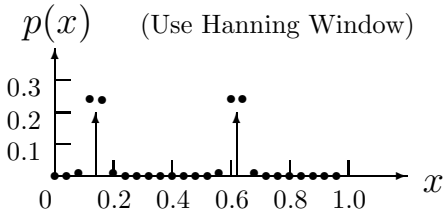
備考

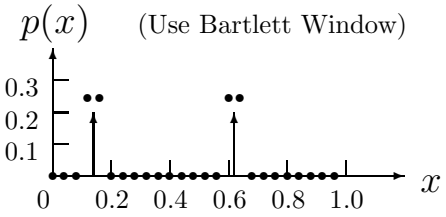
$f_1 = 0.62$, $f_2 = 0.14$ とした場合, $f(x)$ を区間 $[0, 25)$, $\Delta x = 0.5$ で標本化すると, 以下のようなグラフとなる. 本来は, 標本化定理を参考に目的に応じてより細かく標本化すべきであるが, この程度の標本化でも窓関数の選択による違いについての傾向はわかる.

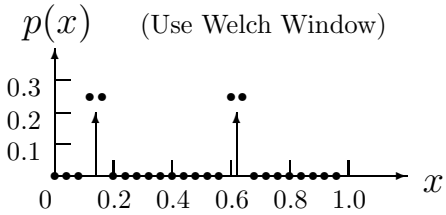


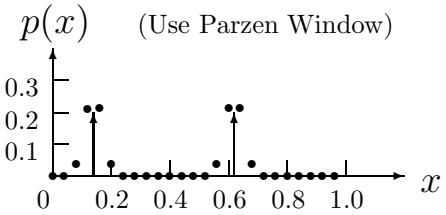
また, 対応するフーリエ・ピリオドグラムは以下の様なグラフとなる (上向き矢印は信号周波数). 信号周波数としてわざと打ち切りによる不連続が大きくなる周波数を用いているので生のフーリエ・ピリオドグラムでは漏れが大きくなっている.











(b) 入力データ

標本化データ

$\mathrm{r[j-1]} = \mathrm{f((j-1)\Delta x)}$ $(\mathrm{j} = 1, 2, \cdots, \mathrm{n})$

ただし, $\Delta x = 0.5$

n, isw

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
2 次元フーリエ・ピリオドグラム

(1) **機 能**

系列 u_{j_x,j_y} $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ の (修正) フーリエ・ピリオドグラムを求める. フーリエ・ピリオドグラム p_{k_x,k_y} は次式で定義される.

$$p_{k_x,k_y} = \frac{\left| \sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} w_{j_x}^{(x)} w_{j_y}^{(y)} u_{j_x,j_y} e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{n_x} + \frac{j_y k_y}{n_y})} \right|^2}{n_x n_y \beta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1)$$

ただし, $\lfloor x \rfloor$ は x を超えない最大の整数を表す. $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ は打ち切り関数 (窓関数) であり, 生のフーリエ・ピリオドグラムの場合には, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = 1$ $(j_x = 0, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$, $\beta = n_x n_y$ とし, 修正ピリオドグラムの場合には

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1} (w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1} (w_{j_y}^{(y)})^2) & \text{(窓関数によるパワー補正式を用いる場合)} \\ n_x n_y & \text{(それ以外)} \end{cases}$$

とする. なお, ピリオドグラム p_{k_x,k_y} は k_x についての半周期分 (周期 (n_x, n_y)) に相当し, 残りの半周期分は以下の関係から得られる.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y} &= p_{k_x,k_y} \\ p_{n_x-k_x,k_y} &= p_{k_x,n_y-k_y} \end{aligned}$$

また, 対応する系列の全パワーは,

$$\frac{\sum_{j_x=0}^{n_x-1} \sum_{j_y=0}^{n_y-1} \{u_{j_x,j_y}\}^2}{n_x n_y}$$

である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfps2d  (nx, ny, r, lx, ly, isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rfps2d  (nx, ny, r, lx, ly, isw, iwk, wk);
```

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	系列 u_{j_x,j_y} の j_x 方向の長さ n_x (注意事項 (d) 参照)
2	ny	I	1	入 力	系列 u_{j_x,j_y} の j_y 方向の長さ n_y (注意事項 (d) 参照)
3	r	{D∗ / R∗}	lx×ly	入 力	系列 u_{j_x,j_y} の値 (注意事項 (a) 参照)
				出 力	系列 u_{j_x,j_y} のフーリエ・ピリオドグラム p_{k_x,k_y} の値 (注意事項 (b), (c) 参照)
4	lx	I	1	入 力	配列 r の整合寸法
5	ly	I	1	入 力	配列 r の第 2 寸法
6	isw	I	1	入 力	処理スイッチ (注意事項 (e) 参照) isw= 0:生のフーリエ・ピリオドグラムを計算する isw=±1:ユーザ定義窓関数を利用して計算する isw=±2:Hanning 窓関数を利用して計算する isw=±3:Bartlett 窓関数を利用して計算する isw=±4:Welch 窓関数を利用して計算する isw=±5:Parzen 窓関数を利用して計算する なお, 窓関数によるパワー補正式を用いる場合は isw >0, それ以外の場合は isw <0 とする.
7	iwk	I*	40	ワーク	作業領域
8	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 isw=±1 の場合には, ユーザ定義窓関数の値を入力する (注意事項 (e) 参照). **大きさ**: nx $+ 2 \times$ ny $+$ lx $\times$ ly
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx > 1
ny > 1

(c) nx が奇数の時:
lx $\geq$ nx $+ 1$
ly $\geq$ ny
nx が偶数の時:
lx $\geq$ nx $+ 2$
ly $\geq$ ny

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3030	制限条件 (c) を満足しなかった.	
4000	isw = 1 の場合にユーザ定義窓関数が $w_{j_x}^{(x)} = 0 \quad (j_x = 0, \cdots, n_x - 1)$ であった.	
4010	isw = 1 の場合にユーザ定義窓関数が $w_{j_y}^{(y)} = 0 \quad (j_y = 0, \cdots, n_y - 1)$ であった.	

(6) **注意事項**

(a) 配列 r の各要素と系列 u_{j_x,j_y} の値は以下の様に対応する.

$$u_{j_x,j_y} \quad \leftrightarrow \quad \mathrm{r}[j_x + \mathrm{lx} * j_y]$$

ただし, $j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly が奇数になるように設定するのが望ましい. 通常, たとえば nx が (4 の倍数)+2 のときは lx=nx+4 とする.**

(b) フーリエ・ピリオドグラム p_{k_x,k_y} の値は配列 r の各要素と以下の様に対応する.

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r}[k_x + \mathrm{lx} * k_y] \quad (k_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, \cdots, n_y - 1)$$

なお, $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

(c) 得られるフーリエ・ピリオドグラム $p_{k_x,k_y} \quad (k_x = 0, 1, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1)$ に対応する周波数 (ξ_{k_x}, η_{k_y}) は

$$\xi_{k_x} = \frac{k_x}{n_x \Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y \Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y \Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

(Δ:標本化間隔) で与えられる.

(d) 系列 u_{j_x,j_y} の長さ nx, ny の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が効率が良い. なお, データ数を大きくできない場合には, データの最後に 0 を必要なだけ補って nx を調整して計算を行う.

(e) 処理スイッチ isw の値によって, 以下の様に打ち切り関数 (窓関数) を変更することができる.

$$w_j = \begin{cases} \sin^2(\pi v_j) & \text{isw= } \pm 2 \text{ (Hanning 窓)} \\ 1 - |2v_j - 1| & \text{isw= } \pm 3 \text{ (Bartlett 窓)} \\ 1 - (2v_j - 1)^2 & \text{isw= } \pm 4 \text{ (Welch 窓)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & \text{isw= } \pm 5 \text{ (Parzen 窓)} \end{cases}$$

ただし, $v_j = \frac{j}{n}$ で, $w_{j_x}^{(x)}$ については $j = j_x, n = n_x$ とし, $w_{j_y}^{(y)}$ については $j = j_y, n = n_y$ とする. したがって, 上述のような窓関数を用いる場合には系列 u_{j_x,j_y} の要素 u_{0,j_y}, $u_{j_x,0}$ は修正ピリオドグラムの計算に影響しない. これを避けたい場合には, 実際に計算したい系列の長さよりも 1 大きい数を nx, ny に指定し, j_x, j_y について 1 以降の対応する要素に有効なデータを設定すれば良い. なお, 窓関数は $|x| \leq 1$ でのみ非ゼロとなる時間 (または空間) 領域関数としてそれぞれ次の様に表される.

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

また, ユーザ独自の窓関数値 $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$ を利用したい場合には isw $= \pm 1$ として作業配列 wk に

$$\mathrm{wk}[j_x] = w_{j_x}^{(x)} \ \ (j_x = 0, \cdots, n_x - 1), \mathrm{wk}[n_x + j_y] = w_{j_y}^{(y)} \ \ (j_y = 0, \cdots, n_y - 1)$$

と設定してこの関数を呼び出す.

(f) 生のピリオドグラムはその定義から自己相関関数の離散フーリエ変換近似とみなせる. 有効データ数 n の離散関数の自己相関関数の有効データ長は $2n-1$ であるので, 一般の関数のパワー・スペクトルを生のピリオドグラムで近似することは, 1 つの周期が以下のように与えられる方形打ち切り関数 $w(k)$ で関数を打ち切ったことに相当する.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n-1 \\ 0 & \text{それ以外} \end{cases}$$

方形関数のフーリエ変換は周波数を f とした場合, $\frac{\sin f}{f}$ 型の関数形をしており, 中心周波数の周りに小さくないサイドローブを持っている. したがって, たとえば, 周期関数を 1 周期の整数倍でない幅で単純に打ち切って標本化した場合, 周波数領域では, 生のピリオドグラムはパワー・スペクトルを求めたい周期関数のフーリエ変換と $\frac{\sin f}{f}$ 型関数との畳み込みとなるので, **漏れ (leakage)** と呼ばれる余分な周波数成分が発生する. このような**漏れ**を抑止するためには, 単純な打ち切りを行わずに Hanning 窓関数のような周波数領域でのサイドローブが小さい打ち切り関数を用いる. ただし, 一般に漏れを抑圧すればする程離散フーリエ変換の結果は拡がりぼやけたものとなる. したがって, パワー・スペクトルを推定する場合, 目的に応じて, すなわち, スペクトル幅を問題としているの中心周波数を問題としているのか等に応じて, 適切な打ち切り関数を選択する必要がある.

(g) 離散フーリエ変換の分解度 (周波数領域での標本間隔) $\frac{1}{nT}$ を上げるには標本数 n を増やすか標本間隔 T を増やせば良いが, 標本間隔と分解度を一定に保った状態でパワー・スペクトルの推定値の精度を上げるために, 標本数 n の標本を m 組とって m 組それぞれについて修正ピリオドグラムを求めてその平均をとるという手法がよく取られる. この場合, 系列から m 組の標本データをオーバーラップして取るというような手法も提案されている. 詳細は参考文献等を参照されたい.

(h) パワー・スペクトルを求める場合, フーリエ変換の周波数推移に関する性質すなわち時間 (または空間) 領域で $e^{2\pi\sqrt{-1}f_0 t}$ を掛けることは周波数領域では周波数を f_0 だけシフトすることに対応し, 関数の形状は変わらないという性質を利用して, パワー・スペクトルの中心周波数をあらかじめシフトして計算することで計算に必要なデータ点数を削減するという手法も良く用いられる. なお, このような操作は**変調 (modulation)** として知られている. ただし, lx=nx+1(nx が奇数のとき) または
lx=nx+2(nx が偶数のとき)
ly=ny である.

(i) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題

次式で定義される波形を標本化間隔 Δ で離散化し, フーリエ・ピリオドグラムを計算し, パワー・スペクトルを推定する.

$$f(x,y) = \cos 2\pi f_1 x + \cos 2\pi f_2 y$$

(b) 入力データ

標本化データ

$\mathrm{r}[\mathrm{j_x} + \mathrm{lx} * \mathrm{j_y}] = \mathrm{f}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny} - 1)$

ただし, $\Delta = 0.5$

nx, ny, isw

(c) 主プログラム

```
/*      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) 出力結果

```
  *** 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
3 次元フーリエ・ピリオドグラム

(1) **機 能**

系列 u_{j_x,j_y,j_z} $(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$ の (修正) フーリエ・ピリオドグラムを求める. フーリエ・ピリオドグラム p_{k_x,k_y,k_z} は次式で定義される.

$$p_{k_x,k_y,k_z} = \frac{\left|\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1} w_{j_x}^{(x)} w_{j_y}^{(y)} w_{j_z}^{(z)} u_{j_x,j_y,j_z} e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{n_x}+\frac{j_y k_y}{n_y}+\frac{j_z k_z}{n_z}\right)}\right|^2}{n_x n_y n_z \beta}$$
$$(k_x = 0,1,\cdots,\lfloor\tfrac{n_x}{2}\rfloor;\ k_y = 0,1,\cdots,n_y-1; k_z = 0,1,\cdots,n_z-1)$$

ただし, $\lfloor x\rfloor$ は x を超えない最大の整数を表す. $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$, $w_{j_z}^{(z)}$ は打ち切り関数 (窓関数) であり, 生のフーリエ・ピリオドグラムの場合には, $w_{j_x}^{(x)} = w_{j_y}^{(y)} = w_{j_z}^{(z)} = 1$ $(j_x = 0,\cdots,n_x-1;\ j_y = 0,\cdots,n_y-1;\ j_z = 0,\cdots,n_z-1)$, $\beta = n_x n_y n_z$ とし, 修正ピリオドグラムの場合には

$$\beta = \begin{cases} (\sum_{j_x=0}^{n_x-1}(w_{j_x}^{(x)})^2)(\sum_{j_y=0}^{n_y-1}(w_{j_y}^{(y)})^2)(\sum_{j_z=0}^{n_z-1}(w_{j_z}^{(z)})^2) & \text{(窓関数によるパワー補正式を用いる場合)} \\ n_x n_y n_z & \text{(それ以外)} \end{cases}$$

とする. なお, ピリオドグラム p_{k_x,k_y,k_z} は k_x についての半周期分 (周期 (n_x,n_y,n_z)) に相当し, 残りの半周期分は以下の関係から得られる.

$$\begin{aligned} p_{n_x-k_x,n_y-k_y,n_z-k_z} &= p_{k_x,k_y,k_z} \\ p_{n_x-k_x,k_y,k_z} &= p_{k_x,n_y-k_y,n_z-k_z} \\ p_{n_x-k_x,n_y-k_y,k_z} &= p_{k_x,k_y,n_z-k_z} \end{aligned}$$

また, 対応する系列の全パワーは,

$$\frac{\sum_{j_x=0}^{n_x-1}\sum_{j_y=0}^{n_y-1}\sum_{j_z=0}^{n_z-1}\{u_{j_x,j_y,j_z}\}^2}{n_x n_y n_z}$$

である.

(2) **使用法**

倍精度関数:

ierr = ASL_dfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk);

単精度関数:

ierr = ASL_rfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	nx	I	1	入 力	系列 u_{j_x,j_y,j_z} の j_x 方向の長さ n_x (注意事項 (d) 参照)
2	ny	I	1	入 力	系列 u_{j_x,j_y,j_z} の j_y 方向の長さ n_y (注意事項 (d) 参照)
3	nz	I	1	入 力	系列 u_{j_x,j_y,j_z} の j_z 方向の長さ n_z (注意事項 (d) 参照)
4	r	{ D∗ / R∗ }	lx×ly× lz	入 力	系列 u_{j_x,j_y,j_z} の値 (注意事項 (a) 参照)
				出 力	系列 u_{j_x,j_y,j_z} のフーリエ・ピリオドグラム p_{k_x,k_y,k_z} の値 (注意事項 (b), (c) 参照)
5	lx	I	1	入 力	配列 r の整合寸法
6	ly	I	1	入 力	配列 r の第 2 寸法
7	lz	I	1	入 力	配列 r の第 3 寸法
8	isw	I	1	入 力	処理スイッチ (注意事項 (e) 参照) isw= 0:生のフーリエ・ピリオドグラムを計算する isw=±1:ユーザ定義窓関数を利用して計算する isw=±2:Hanning 窓関数を利用して計算する isw=±3:Bartlett 窓関数を利用して計算する isw=±4:Welch 窓関数を利用して計算する isw=±5:Parzen 窓関数を利用して計算する なお, 窓関数によるパワー補正式を用いる場合は isw >0, それ以外の場合は isw <0 とする.
9	iwk	I*	60	ワーク	作業領域
10	wk	{ D∗ / R∗ }	内容参照	ワーク	作業領域 isw=±1 の場合には, ユーザ定義窓関数の値を入力する (注意事項 (e) 参照). **大きさ**: nx + 2 × (ny + nz) + lx × ly × lz
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx > 1
ny > 1
nz > 1

(c) nx が奇数の時 :
lx $\geq$ nx $+ 1$, lx は偶数
ly $\geq$ ny
lz $\geq$ nz
nx が偶数の時 :
lx $\geq$ nx $+ 2$, lx は偶数
ly $\geq$ ny
lz $\geq$ nz

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3030	制限条件 (c) を満足しなかった.	
4000	isw $= 1$ の場合にユーザ定義窓関数が $w_{j_x}^{(x)} = 0 \quad (j_x = 0, \cdots, n_x - 1)$ であった.	
4010	isw=1 の場合にユーザ定義窓関数が $w_{j_y}^{(y)} = 0$ $(j_y = 0, \cdots, n_y - 1)$ であった.	
4020	isw=1 の場合にユーザ定義窓関数が $w_{j_z}^{(z)} = 0$ $(j_z = 0, \cdots, n_z - 1)$ であった.	

(6) **注意事項**

(a) 配列 r の各要素と系列 u_{j_x,j_y,j_z} の値は以下の様に対応する.

$$u_{j_x,j_y,j_z} \quad \leftrightarrow \quad \mathrm{r}[j_x + \mathrm{lx} * (j_y + \mathrm{ly} * j_z)]$$

ただし, $j_x = 0, \cdots, n_x - 1$; $j_y = 0, \cdots, n_y - 1$; $j_z = 0, \cdots, n_z - 1$ であり, それ以外の要素には値を入力する必要が無い. **なお, 主記憶のバンク競合を避けるために配列 r の整合寸法について lx/2, ly, lz が奇数になるように設定するのが望ましい. また, 高速化のために配列 r 内のデータ設定領域以外の要素に対しても演算を実行する. 通常, たとえば nx が (4 の倍数)+2 のときは lx=nx+4 とする.**

(b) フーリエ・ピリオドグラム p_{k_x,k_y,k_z} の値は配列 r の各要素と以下の様に対応する.

$$p(k_x, k_y, k_z) \quad \leftrightarrow \quad \mathrm{r}[k_x + \mathrm{lx} * (k_y + \mathrm{ly} * k_z)]$$
$$(k_x = 0, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, \cdots, n_y - 1;\ k_z = 0, \cdots, n_z - 1)$$

なお, $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

(c) 得られるフーリエ・ピリオドグラム p_{k_x,k_y,k_z} $(k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1\ k_z = 0, \cdots, n_z - 1)$ に対応する周波数 $(\xi_{k_x}, \eta_{k_y}, \zeta_{k_z})$ は

$$\xi_{k_x} = \frac{k_x}{n_x \Delta} \quad (k_x = 0, 1, \cdots, \lfloor \frac{n_x}{2} \rfloor)$$

$$\eta_{k_y} = \begin{cases} \dfrac{k_y}{n_y \Delta} & (k_y = 0, 1, \cdots, \lfloor \dfrac{n_y}{2} \rfloor) \\ \dfrac{k_y - n_y}{n_y \Delta} & (k_y = \lfloor \dfrac{n_y}{2} \rfloor + 1, \cdots, n_y - 1) \end{cases}$$

$$\zeta_{k_z} = \begin{cases} \dfrac{k_z}{n_z \Delta} & (k_z = 0, 1, \cdots, \lfloor \dfrac{n_z}{2} \rfloor) \\ \dfrac{k_z - n_z}{n_z \Delta} & (k_z = \lfloor \dfrac{n_z}{2} \rfloor + 1, \cdots, n_z - 1) \end{cases}$$

(Δ:標本化間隔) で与えられる.

(d) 系列 u_{j_x,j_y,j_z} の長さ nx, ny, nz の値は混合基数 FFT アルゴリズムが有効に働く数 (FFT の混合基数である 2, 3, 5 等の倍数) となるように設定した方が効率良い計算を行える. たとえば, nx = 289(=17^2) とするよりも nx = 300(=$2^2 \times 3 \times 5^2$) や 320(=$2^6 \times 5$), 384(=$2^7 \times 3$) などとした方が効率が良い. なお, データ数を大きくできない場合には, データの最後に 0 を必要なだけ補って nx を調整して計算を行う.

(e) 処理スイッチ isw の値によって, 以下の様に打ち切り関数 (窓関数) を変更することができる.

$$w_j = \begin{cases} \sin^2(\pi v_j) & & \text{isw= } \pm 2 \text{ (Hanning 窓)} \\ 1 - |2v_j - 1| & & \text{isw= } \pm 3 \text{ (Bartlett 窓)} \\ 1 - (2v_j - 1)^2 & & \text{isw= } \pm 4 \text{ (Welch 窓)} \\ \left\{ \begin{array}{ll} 16{v_j}^3 & 0 \le v_j < \frac{1}{4} \\ 1 - 6v_j(v_j - 1)^2 & \frac{1}{4} \le v_j \le \frac{1}{2} \\ 1 - 6v_j(v_{n-j+1} - 1)^2 & \frac{1}{2} \le v_j \le \frac{3}{4} \\ 16{v_{n-j+1}}^3 & \frac{3}{4} \le v_j < 1 \end{array} \right\} & & \text{isw= } \pm 5 \text{ (Parzen 窓)} \end{cases}$$

ただし, $v_j = \frac{j}{n}$ で, $w_{j_x}^{(x)}$ については $j = j_x, n = n_x$ とし, $w_{j_y}^{(y)}$ については $j = j_y, n = n_y$, $w_{j_z}^{(z)}$ については $j = j_z, n = n_z$ とする. したがって, 上述のような窓関数を用いる場合には系列 u_{j_x,j_y,j_z} の要素 u_{0,j_y,j_z}, $u_{j_x,0,j_z}$, $u_{j_x,j_y,0}$ は修正ピリオドグラムの計算に影響しない. これを避けたい場合には, 実際に計算したい系列の長さよりも 1 大きい数を nx, ny, nz に指定し, j_x, j_y, j_z について 1 以降の対応する要素に有効なデータを設定すれば良い. なお, 窓関数は $|x| \le 1$ でのみ非ゼロとなる時間 (または空間) 領域関数としてそれぞれ次の様に表される.

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

また, ユーザ独自の窓関数値 $w_{j_x}^{(x)}$, $w_{j_y}^{(y)}$, $w_{j_z}^{(z)}$ を利用したい場合には isw = ±1 として作業配列 wk に

$$\text{wk}[j_x] = w_{j_x}^{(x)} \quad (j_x = 0, \cdots, n_x - 1),$$
$$\text{wk}[n_x + j_y] = w_{j_y}^{(y)} \quad (j_y = 0, \cdots, n_y - 1),$$
$$\text{wk}[n_x + n_y + j_z] = w_{j_z}^{(z)} \quad (j_z = 0, \cdots, n_z - 1)$$

と設定してこの関数を呼び出す.

(f) 生のピリオドグラムはその定義から自己相関関数の離散フーリエ変換近似とみなせる. 有効データ数 n の離散関数の自己相関関数の有効データ長は $2n - 1$ であるので, 一般の関数のパワー・スペクトルを生のピ

リオドグラムで近似することは, 1 つの周期が以下のように与えられる方形打ち切り関数 $w(k)$ で関数を打ち切ったことに相当する.

$$w(k) = \begin{cases} 1 & k = 0, 1, \cdots, n-1 \\ 0 & \text{それ以外} \end{cases}$$

方形関数のフーリエ変換は周波数を f とした場合, $\frac{\sin f}{f}$ 型の関数形をしており, 中心周波数の周りに小さくないサイドローブを持っている. したがって, たとえば, 周期関数を 1 周期の整数倍でない幅で単純に打ち切って標本化した場合, 周波数領域では, 生のピリオドグラムはパワー・スペクトルを求めたい周期関数のフーリエ変換と $\frac{\sin f}{f}$ 型関数との畳み込みとなるので, **漏れ (leakage)** と呼ばれる余分な周波数成分が発生する. このような**漏れ**を抑止するためには, 単純な打ち切りを行わずに Hanning 窓関数のような周波数領域でのサイドローブが小さい打ち切り関数を用いる. ただし, 一般に漏れを抑圧すればする程離散フーリエ変換の結果は拡がりぼやけたものとなる. したがって, パワー・スペクトルを推定する場合, 目的に応じて, すなわち, スペクトル幅を問題としているの中心周波数を問題としているのか等に応じて, 適切な打ち切り関数を選択する必要がある.

(g) 離散フーリエ変換の分解度 (周波数領域での標本間隔) $\frac{1}{nT}$ を上げるには標本数 n を増やすか標本間隔 T を増やせば良いが, 標本間隔と分解度を一定に保った状態でパワー・スペクトルの推定値の精度を上げるために, 標本数 n の標本を m 組とって m 組それぞれについて修正ピリオドグラムを求めてその平均をとるという手法がよく取られる. この場合, 系列から m 組の標本データをオーバーラップして取るというような手法も提案されている. 詳細は参考文献等を参照されたい.

(h) パワー・スペクトルを求める場合, フーリエ変換の周波数推移に関する性質すなわち時間 (または空間) 領域で $e^{2\pi\sqrt{-1}f_0 t}$ を掛けることは周波数領域では周波数を f_0 だけシフトすることに対応し, 関数の形状は変わらないという性質を利用して, パワー・スペクトルの中心周波数をあらかじめシフトして計算することで計算に必要なデータ点数を削減するという手法も良く用いられる. なお, このような操作は**変調 (modulation)** として知られている. ただし,

lx=nx+1(nx が奇数のとき) または

lx=nx+2(nx が偶数のとき)

ly=ny, lz=nz

である.

(i) この機能は逐次版および OpenMP 不使用の MPI 版ライブラリにおいてスレッドセーフではない.

(7) 使用例

(a) 問 題

次式で定義される波形を標本化間隔 Δ で離散化し, フーリエ・ピリオドグラムを計算し, パワー・スペクトルを推定する.

$$f(x, y, z) = \cos 2\pi f_1 x + \cos 2\pi f_2 y + \cos 2\pi f_3 z$$

(b) 入力データ

標本化データ

$\mathrm{r[j_x + lx * (j_y + ly * j_z)]} = \mathrm{f(j_x\Delta, j_y\Delta, j_z\Delta)}$ ($\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx}-1$; $\mathrm{j_y} = 0, 1, \cdots, \mathrm{ny}-1$; $\mathrm{j_z} = 0, 1, \cdots, \mathrm{nz}-1$)

ただし, $\Delta = 0.5$

nx, ny, nz, isw

(c) 主プログラム

```
/*      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) 出力結果

```
    *** 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 ラプラス変換

2.17.1 ASL_dflara, ASL_rflara
ラプラス逆変換 (有理関数)

(1) **機 能**

有理関数 $F(s) = \dfrac{Q(s)}{P(s)} = \dfrac{q_1 s^{nq} + q_2 s^{nq-1} + \cdots + q_{nq} s + q_{nq+1}}{p_1 s^{np} + p_2 s^{np-1} + \cdots + p_{np} s + p_{np+1}}$ $(np \leq nq; p_1, \cdots, p_{np+q}, q_1, \cdots, q_{nq+1}$:実数)
のラプラス逆変換

$$f(t) = \frac{1}{2\pi i} \int_{\gamma - i\infty}^{\gamma + i\infty} F(s) e^{st} ds \qquad (0 < t < \infty)$$

を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dflara (p, np, q, nq, t, n, a, ip, k1, k2, & r, f, er, isw, w1);

単精度関数:

ierr = ASL_rflara (p, np, q, nq, t, n, a, ip, k1, k2, & r, f, er, isw, w1);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	p	{D∗ / R∗}	np+1	入 力	分母多項式 $P(s)$ の係数
2	np	I	1	入 力	分母多項式 $P(s)$ の次数
3	q	{D∗ / R∗}	nq+1	入 力	分子多項式 $Q(s)$ の係数
4	nq	I	1	入 力	分母多項式 $Q(s)$ の次数
5	t	{D∗ / R∗}	n	入 力	原関数 $f(t)$ の計算点
6	n	I	1	入 力	配列 t の寸法
7	a	{D / R}	1	入 力	近似誤差をきめる a の値 (2.1.2(1)(d) 参照)
8	ip	I	1	入 力	オイラー変換の次数 p (2.1.2(1)(d) 参照)
9	k1	{D / R}	1	入 力	打ち切り項数決定のためのパラメータ k_1 (2.1.2 (1)(d) 参照)
10	k2	{D / R}	1	入 力	打ち切り項数決定のためのパラメータ k_2 (2.1.2 (1)(d) 参照)
11	r	{D∗ / R∗}	1	入 力 出 力	isw=1:r=0.0 を出力する isw=2:収束座標を入力する isw=3:収束座標の計算結果を出力する (注意事項 (b) 参照)
12	f	{D∗ / R∗}	n	出 力	各 t [i] における原関数 $f(t)$ の値
13	er	{D∗ / R∗}	n	出 力	各 f [i] (i=0, ⋯, n−1) 計算時の打ち切り誤差
14	isw	I	1	入 力	isw=1: $\Re(s) > 0$ で正則な場合 isw=2: r に収束座標の値を入力する ($\Re(s) > 0$ で非正則で収束座標の値が既知の場合) isw=3: r に収束座標の値を出力する (収束座標の値が未知の場合)
15	w1	{D∗ / R∗}	2 × (ip + np + 1)	ワーク	作業領域
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) n > 0

(b) t[i] $> 0.0 \quad (i = 0, \cdots, \mathrm{n}-1)$

(c) p[0] > 0.0

(d) $0 <$ nq $\leq$ np

(e) a > 0.0

(f) ip > 0

(g) k1 > 0.0
k2 ≥ 0.0

(h) r ≥ 0.0

(i) isw $\in \{1, 2, 3\}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	t[i] $= 0.0$ だった. $(i = 0, \cdots, \mathrm{n}-1)$	注意事項 (a), (b) 参照.
1100	t[i] < 0.0 だった. $(i = 0, \cdots, \mathrm{n}-1)$	t[i] の処理を打ち切り, t[$i+1$] 以降の処理を行う.
2000	r < 0.0 だった (isw $= 2$ の時).	r $= 0.0$ として処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	
3030	制限条件 (e) を満足しなかった.	
3040	制限条件 (f) を満足しなかった.	
3050	制限条件 (g) を満足しなかった.	
3060	制限条件 (i) を満足しなかった.	

(6) **注意事項**

(a) 利用者は, a, ip, k1, k2 の値を変えることにより誤差の制御が簡単にできる.

(b) isw=1 の場合, r=0.0 として処理を行う.
isw=2 の場合, $F(s)$ が $\Re(s) > \alpha$ で正則であるとき, $\gamma \geq \alpha$ となる γ を r に入力する. r ≤ 0.0 の時は r=0.0 として処理を行う.
isw=3 の場合, 収束座標 γ_0 を求め, r=γ_0 として処理を行う.

(c) np=nq のとき

$$F(s) = \frac{Q(s)}{P(s)} = \frac{q_1}{p_1} + G(s)$$

$$\text{ただし}, G(s) = \frac{o_2 s^{np-1} + o_3 s^{np-2} + \cdots + o_{np-1}}{p_1 s^{np} + p_2 s^{np-1} + \cdots + p_{np+1}}$$

と分解できる. $G(s)$ の逆変換を $g(t)$ とすると

$$f(t) = \frac{q_1}{p_1}\delta(t) + g(t)$$

となる. ここで $\delta(t)$ は, Dirac の δ-関数である.

よって

$$f(t) = \begin{cases} \text{正の最大値} & (t = 0) \\ g(t) & (t > 0) \end{cases}$$

(d) t [i] =0 を与えたとき, $f(0)$ の値は $f(0) = [sF(s)]_{s=\infty}$ という公式から計算する.

$$f(0) = \begin{cases} \text{正の最大値} & (np = nq) \\ \frac{q_1}{p_1} & (np = nq + 1) \\ 0 & (np > nq + 1) \end{cases}$$

(7) **使用例**

(a) 問 題

$\Re(s) > 0$ で正則な有理関数

$$F(s) = \frac{1}{s+1}$$

のラプラス逆変換 $f(t)$ を t =1.0, 2.0, 3.0, 4.0, 5.0 について求める.

(b) 入力データ

$\mathrm{p} = \{1, 1\}, \mathrm{np} = 1, \mathrm{q} = 1, \mathrm{nq} = 0, \mathrm{n} = 5, \mathrm{t} = \{1.0, 2.0, 3.0, 4.0, 5.0\}, \mathrm{a} = 10, \mathrm{ip} = 10, \mathrm{k1} = 10.0, \mathrm{k2} = 0.0, \mathrm{isw} = 1$

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
ラプラス逆変換 (一般関数)

(1) **機 能**

一般関数 $F(s)$ のラプラス逆変換

$$f(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} F(s)e^{st}ds \qquad (0 < t < \infty)$$

を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dflage (fi, t, n, a, ip, k1, k2, r, f, er, w1);

単精度関数:

ierr = ASL_rflage (fi, t, n, a, ip, k1, k2, r, f, er, w1);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	fi	{D / R}	−	入 力	像関数 $F(s)$ の虚数部を求める関数名
2	t	{D∗ / R∗}	n	入 力	原関数 $f(t)$ の計算点
3	n	I	1	入 力	配列 t の寸法
4	a	{D / R}	1	入 力	近似誤差をきめる a の値 (2.1.2(1) (d) 参照)
5	ip	I	1	入 力	オイラー変換の次数 p (2.1.2(1) (d) 参照)
6	k1	{D / R}	1	入 力	打ち切り項数決定のためのパラメータ k_1 (2.1.2 (1) (d) 参照)
7	k2	{D / R}	1	入 力	打ち切り項数決定のためのパラメータ k_2 (2.1.2 (1) (d) 参照)
8	r	{D / R}	1	入 力	収束座標
9	f	{D∗ / R∗}	n	出 力	各 t [i] における原関数 $f(t)$ の値
10	er	{D∗ / R∗}	n	出 力	各 f [i] (i=0, ···, n−1) 計算時の打ち切り誤差
11	w1	{D∗ / R∗}	2×(ip+2)	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 0$

(b) $\mathrm{t}[i] > 0.0 \quad (i = 0, \cdots, \mathrm{n}-1)$

(c) $\mathrm{a} > 0.0$

(d) $\mathrm{ip} > 0$

(e) $\mathrm{k1} > 0.0$
$\mathrm{k2} \geq 0.0$

(f) $\mathrm{r} \geq 0.0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{t}[i] \leq 0.0$ だった. $(i = 0, \cdots, \mathrm{n}-1)$	$\mathrm{t}[i]$ の処理を打ち切り, $\mathrm{t}[i+1]$ 以降の処理を行う.
2000	$\mathrm{r} < 0.0$ だった.	$\mathrm{r} = 0.0$ として処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (c) を満足しなかった.	
3020	制限条件 (d) を満足しなかった.	
3030	制限条件 (e) を満足しなかった.	

(6) **注意事項**

(a) 関数 fi の作り方は以下のようにする.

```
double FORTRAN fi( Complex_d *s )
{
        return (fi(*s));
}
```

なお, 本関数は, 関数 $F(s)$ の虚数部の値を $\Re(s) > 0.0$, $\Im(s) > 0.0$ の点で評価する. $F(s)$ が多価関数である場合には, 関数の中で正しい枝を計算するように注意が必要である.

(b) 利用者は, a, ip, k1, k2 の値を変えることにより誤差の制御が簡単にできる.

(c) $F(s)$ が $\Re(s) > \alpha$ で正則であるとき, $\gamma \geq \alpha$ となる γ を r に入力する. r≤0.0 の時は r=0.0 として処理を行う.

(7) 使用例

(a) 問 題

$\Re(s) > 0$ で正則な関数

$$F(s) = e^{-\sqrt{s}}$$

のラプラス逆変換 $f(t)$ を t =1.0, 2.0, 3.0, 4.0, 5.0 について求める.

(b) 入力データ

t={1.0, 2.0, 3.0, 4.0, 5.0}, n=5, a=10, ip=10, k1=10.0, k2=0.0, r=0.0

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 ウェーブレット変換

2.18.1 ASL_dfwth1, ASL_rfwth1
Haar 関数の生成

(1) **機 能**

1 次元ウェーブレット変換に必要な Haar 関数

$$H_{mn}(x) = \begin{cases} \sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1) \le x \le \frac{a}{2^m}(n-1/2) \\ -\sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1/2) < x \le \frac{a}{2^m}n \\ 0 & \text{その他} \end{cases}$$

を区間 $[0, a](a \ge 0.0)$ で生成する.

(2) **使用法**

倍精度関数:

ierr = ASL_dfwth1 (a, m, n, & c, & bl, & bm, & br);

単精度関数:

ierr = ASL_rfwth1 (a, m, n, & c, & bl, & bm, & br);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D / R}	1	入 力	Haar 関数 $H_{mn}(x)$ を生成する区間 $[0, a]$ の上限 a.
2	m	I	1	入 力	Haar 関数 $H_{mn}(x)$ の世代 m.
3	n	I	1	入 力	Haar 関数 $H_{mn}(x)$ のインデックス n. (注意事項 (a) 参照)
4	c	{D∗ / R∗}	1	出 力	Haar 関数 $H_{mn}(x)$ の区間 $[\frac{a}{2^m}(n-1), \frac{a}{2^m}n]$ での絶対値 $\sqrt{\frac{2^m}{a}}$.
5	bl	{D∗ / R∗}	1	出 力	Haar 関数 $H_{mn}(x)$ の小さい方の 立ち上がり位置 $\frac{a}{2^m}(n-1)$.
6	bm	{D∗ / R∗}	1	出 力	Haar 関数 $H_{mn}(x)$ の値が正から負に変化する 位置 $\frac{a}{2^m}(n-1/2)$
7	br	{D∗ / R∗}	1	出 力	Haar 関数 $H_{mn}(x)$ の大きい方の 立ち上がり位置 $\frac{a}{2^m}n$.
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $a > 0$

(b) $m \geq 0$

(c) $n \leq 2^m$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数では定数項を与えない. 基底としての Haar 関数には, この関数で与えられるものの他に, 区間 $[0, a]$ で一定値 $1/\sqrt{a}$ が必要である. 入力データが連続, あるいはサンプリングされた間隔が不規則な場合, ウェーブレット変換において本関数を用いる. 変換されたデータのサンプルリングされた間隔が一定で, かつ 2^k 個 (k は自然数) である場合, より簡単な 2.18.4 $\left\{\begin{matrix}\text{ASL\_dfwth2}\\ \text{ASL\_rfwth2}\end{matrix}\right\}$ を用いてウェーブレット変換のための Haar 関数を生成することができる.

(7) **使用例**

(a) 問 題

$a = 2$ の場合に Haar 関数 $H_{34}(x)$ の値を計算する.

(b) 入力データ

a=2, m=3, n=4

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
Haar 関数によるウェーブレット変換

(1) **機 能**

入力データ $\{(x_i, f(x_i))\}$ のサンプリングされた間隔が等間隔でない場合や, サンプリングされたデータ数が 2^k(k は自然数) 個でない場合に Haar 関数によるウェーブレット変換

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

を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dfwthr (xd, yd, nd, mr, nr, dr, imr, inr);

単精度関数:

ierr = ASL_rfwthr (xd, yd, nd, mr, nr, dr, imr, inr);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	xd	{D∗ / R∗}	nd	入 力	入力データの x 座標の組 $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	入 力	入力データの関数値の組 $\{f(x_i)\}$.
3	nd	I	1	入 力	入力データの個数 N.
4	mr	I	1	入 力	ウェーブレット変換に用いる Haar 関数 $H_{mn}(x)$ のインデックス m の最大値.
5	nr	I	1	入 力	ウェーブレット変換に用いる Haar 関数 $H_{mn}(x)$ のインデックス n の最大値. m 世代でインデックス n までの Haar 関数をウェーブレット変換に用いる.
6	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	出 力	ウェーブレット変換 C_{mn}. (注意事項 (a) および (b) 参照)
7	imr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	出 力	dr に格納されているウェーブレット変換 C_{mn} のインデックス m の情報 (注意事項 (b) 参照)
8	inr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	出 力	dr に格納されているウェーブレット変換 C_{mn} のインデックス n の情報 (注意事項 (b) 参照)
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{mr} \geq 0$

(b) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(c) $\mathrm{nd} \leq 100000$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) dr [0] には入力データ $\{f(x_i)\}$ の平均値が格納される.

(b) dr $[i]$ $(i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$ には, ウェーブレット変換 C_{mn} の値が格納されている. 対応するインデックス m は imr $[i-1]$ に, インデックス n は inr $[i-1]$ に, それぞれ格納されている.

(7) **使用例**

(a) 問 題

関数

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}$$

を区間 $[0, 1]$ で等間隔にサンプリングして得たデータを用いて, $m = 3, n = 8$ までのウェーブレット変換を行う.

(b) 入力データ

$\{(x_i, f(x_i))\}$, nd=10, mr=4, nr=2

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
Haar 関数による逆ウェーブレット変換

(1) **機 能**

入力データ $\{(x_i, f(x_i))\}$ のサンプリングされた間隔が等間隔でない場合や, サンプリングされたデータ数が 2^k(k は自然数) 個でない場合の Haar 関数によるウェーブレット変換 C_{mn} に対して逆ウェーブレット変換

$$\sum_{m,n} C_{mn} H_{mn}(x)$$

により, $f(x)$ の近似値を計算する

(2) **使用法**

倍精度関数:

ierr = ASL_dfwths (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr);

単精度関数:

ierr = ASL_rfwths (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型　R:単精度実数型　C:単精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D / R}	1	入 力	入力データの x 座標の最大値と最小値の差. データは区間 $[0, a]$ で再生される.
2	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	入 力	ウェーブレット変換 C_{mn}. (注意事項 (a) 参照)
3	mr	I	1	入 力	ウェーブレット変換時に使った Haar 関数 $H_{mn}(x)$ の m の最大値.
4	nr	I	1	入 力	ウェーブレット変換時に使った Haar 関数 $H_{mn}(x)$ の n の最大値.
5	mr2	I	1	入 力	逆ウェーブレット変換に用いる Haar 関数 $H_{mn}(x)$ のインデックス m の最大値.
6	nr2	I	1	入 力	逆ウェーブレット変換に用いる Haar 関数 $H_{mn}(x)$ のインデックス n の最大値.
7	imr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	入 力	ウェーブレット変換 C_{mn} のインデックス m の情報. (注意事項 (a) 参照)
8	inr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	入 力	ウェーブレット変換 C_{mn} のインデックス n の情報. (注意事項 (a) 参照)
9	fr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	出 力	ウェーブレット変換 C_{mn} から再構成された $f(x)$ の値. (注意事項 (b) 参照)
10	xr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	出 力	ウェーブレット変換 C_{mn} から再構成した $f(x)$ の x 座標の値. (注意事項 (b) 参照)
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(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) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) dr $[i]$ $(i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$ には, ウェーブレット変換 C_{mn} の値が格納されている. 対応するインデックス m は imr $[i-1]$ に, インデックス n は inr $[i-1]$ に, それぞれ格納されている.

(b) fr に格納されるのは元のデータの近似値であり, xr に格納されているその x 座標値は等間隔である.

(7) **使用例**

(a) 問 題

2.18.2 $\left\{\begin{matrix}\mathrm{ASL_dfwthr}\\ \mathrm{ASL_rfwthr}\end{matrix}\right\}$の例題を実行して得られたウェーブレット変換を入力データとして $m = 3, n = 8$ までの逆ウェーブレット変換を行う.

(b) 入力データ

ウェーブレット変換 $\{C_{mn}\}$, a=1, mr=3, nr=8, mr2=3, nr2=8

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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 関数の生成 (等間隔サンプリングデータ)

(1) **機 能**

ウェーブレット変換する対象のデータのサンプルリングされた間隔が一定で, かつ個数が $N = 2^k$ 個 (k は自然数) である場合, 1 次元ウェーブレット変換に必要な Haar 関数

$$H_{mn}(x) = \begin{cases} \sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1) \le x \le \frac{a}{2^m}(n-1/2) \\ -\sqrt{\frac{2^m}{a}} & \frac{a}{2^m}(n-1/2) < x \le \frac{a}{2^m}n \\ 0 & \text{その他} \end{cases}$$

を区間 $[0, 1]$ で生成する.

(2) **使用法**

倍精度関数:

ierr = ASL_dfwth2 (na, m, n, & c, lr);

単精度関数:

ierr = ASL_rfwth2 (na, m, n, & c, lr);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	na	I	1	入 力	入力データの個数 N.
2	m	I	1	入 力	Haar 関数 $H_{mn}(x)$ の世代 m.
3	n	I	1	入 力	Haar 関数 $H_{mn}(x)$ のインデックス n. (注意事項 (a) 参照)
4	c	{ D∗ / R∗ }	1	出 力	Haar 関数 $H_{mn}(x)$ の値.
5	lr	I*	na	出 力	Haar 関数 $H_{mn}(x)$ の値の符号. Haar 関数が正のときには 1 が, 負のときには 1 が零のときには -1 が, 値として格納される.
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{na} = 2^k$　(k は 20 以下の自然数.)

(b) $0 \le \mathrm{m} \le k$

(c) $\mathrm{n} \le 2^{\mathrm{m}}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数では定数項を与えない. 基底としての Haar 関数には, この関数で与えられるものの他に, $[0,1]$ 区間で一定値 1 が必要である.

(b) 変換されるべきデータが連続, あるいはサンプリングされた間隔が不規則な場合, Haar 関数の生成には 2.18.1 $\left\{\begin{matrix}\text{ASL\_dfwth1}\\ \text{ASL\_rfwth1}\end{matrix}\right\}$ を用いなければならない.

(7) **使用例**

(a) 問 題

区間 $[0,1]$ での等間隔にならんだ 16 個の x 座標について Haar 関数 $H_{34}(x)$ を計算する.

(b) 入力データ

na=16, m=3, n=4

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
Haar 関数によるウェーブレット変換 (等間隔サンプリングデータ)

(1) **機 能**

入力データ $\{(x_i, f(x_i))\}$ のサンプリングされた間隔が等間隔でかつその回数が 2^k(k は自然数) 個である場合に Haar 関数によるウェーブレット変換

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

を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dfwtht (xd, yd, nd, mr, nr, & a, dr, imr, inr, iwk);

単精度関数:

ierr = ASL_rfwtht (xd, yd, nd, mr, nr, & a, dr, imr, inr, iwk);

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	xd	{D∗ / R∗}	nd	入 力	入力データの x 座標の組 $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	入 力	入力データの関数値の組 $\{f(x_i)\}$.
3	nd	I	1	入 力	入力データの個数 N.
4	mr	I	1	入 力	ウェーブレット変換に用いる Haar 関数 H_{mn} のインデックス m の最大値.
5	nr	I	1	入 力	ウェーブレット変換に用いる Haar 関数 H_{mn} のインデックス n の最大値.
6	a	{D∗ / R∗}	1	出 力	ウェーブレット変換に際しての積分の範囲 $[0, a]$ の上限 a.
7	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	出 力	ウェーブレット変換 C_{mn}. (注意事項 (a) および (b) 参照)
8	imr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	出 力	dr に格納されているウェーブレット変換 C_{mn} のインデックス m の情報 (注意事項 (b) 参照)
9	inr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	出 力	dr に格納されているウェーブレット変換 C_{mn} のインデックス n の情報 (注意事項 (b) 参照)
10	iwk	I*	nd	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nd} = 2^k$　(k は 20 以下の自然数.)

(b) $0 \leq \mathrm{mr} \leq k - 1$

(c) $\mathrm{nr} \leq 2^{\mathrm{mr}}$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) dr [0] には入力データ $\{f(x_i)\}$ の平均値が格納される.

(b) dr $[i]$ $(i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$ には, ウェーブレット変換 C_{mn} の値が格納されている. 対応するインデックス m は imr $[i-1]$ に, インデックス n は inr $[i-1]$ に, それぞれ格納されている.

(7) **使用例**

(a) 問 題

関数

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}$$

を区間 $[0, 1]$ で等間隔にサンプリングして得たデータを用いて, $m = 3, n = 8$ までのウェーブレット変換を行う.

(b) 入力データ

$\{(x_i, f(x_i))\}$, nd=16, mr=3, nr=8

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
Haar 関数による逆ウェーブレット変換 (等間隔サンプリングデータ)

(1) **機 能**

入力データ $\{(x_i, f(x_i))\}$ のサンプリングされた間隔が等間隔でかつその回数が 2^k(k は自然数) 個である場合の Haar 関数によるウェーブレット変換 C_{mn} に対して逆ウェーブレット変換

$$\sum_{m,n} C_{mn} H_{mn}(x)$$

により, $f(x)$ を計算する

(2) **使用法**

倍精度関数:

ierr = ASL_dfwthi (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr, iwk);

単精度関数:

ierr = ASL_rfwthi (a, dr, mr, nr, mr2, nr2, imr, inr, fr, xr, iwk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D / R}	1	入 力	入力データの x 座標の最大値と最小値の差にデータのサンプリング間隔を足した値. データは $[0, \mathrm{a}]$ の間で再生される.
2	dr	{D∗ / R∗}	$2^{\mathrm{mr}} + \mathrm{nr}$	入 力	ウェーブレット変換 C_{mn}. (注意事項 (a) 参照)
3	mr	I	1	入 力	ウェーブレット変換時に使った Haar 関数 H_{mn} の m の最大値.
4	nr	I	1	入 力	ウェーブレット変換時に使った Haar 関数 H_{mn} の n の最大値.
5	mr2	I	1	入 力	逆ウェーブレット変換に用いる Haar 関数 H_{mn} のインデックス n の最大値.
6	nr2	I	1	入 力	逆ウェーブレット変換に用いる Haar 関数 H_{mn} のインデックス n の最大値.
7	imr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	入 力	ウェーブレット変換 C_{mn} のインデックス m の情報. (注意事項 (a) 参照)
8	inr	I*	$2^{\mathrm{mr}} + \mathrm{nr} - 1$	入 力	ウェーブレット変換 C_{mn} のインデックス n の情報. (注意事項 (a) 参照)
9	fr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	出 力	ウェーブレット変換 C_{mn} から再構成された $f(x)$ の値. (注意事項 (b) 参照)
10	xr	{D∗ / R∗}	$2^{\mathrm{mr2}+1}$	出 力	ウェーブレット変換 C_{mn} から再構成した $f(x)$ の x 座標の値. (注意事項 (b) 参照)
11	iwk	I*	$2^{\mathrm{mr2}+1}$	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(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) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
3030	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) dr $[i]$ $(i = 1, 2, \cdots, 2^{\mathrm{mr}} + \mathrm{nr} - 1)$ には, ウェーブレット変換 C_{mn} の値が格納されている. 対応するインデックス m は imr $[i-1]$ に, インデックス n は inr $[i-1]$ に, それぞれ格納されている.

(b) fr に格納されるのは元のデータの近似値であり, xr に格納されているその x 座標値は等間隔である.

(7) **使用例**

(a) 問 題

2.18.5 $\left\{\begin{matrix}\mathrm{ASL_dfwtht}\\ \mathrm{ASL_rfwtht}\end{matrix}\right\}$の例題を実行して得られたウェーブレット変換を入力データとして $m = 3, n = 8$ までの逆ウェーブレット変換を行う.

(b) 入力データ

ウェーブレット変換 $\{C_{mn}\}$, a=1, mr=3, nr=8, mr2=3, nr2=8

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
メキシカンハット関数の計算

(1) **機 能**

メキシカンハット関数

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

によるウェーブレット変換の基底

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

を計算する. ここで

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

は規格化の定数である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfwtmf  (a, b, x, & v);
```

単精度関数:

```
ierr = ASL_rfwtmf  (a, b, x, & v);
```

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D / R}	1	入 力	メキシカンハット関数によるウェーブレット変換の基底 $\phi_{MH}(x; a, b)$ の周波数パラメータ a.
2	b	{D / R}	1	入 力	メキシカンハット関数によるウェーブレット変換の基底 $\phi_{MH}(x; a, b)$ のシフトパラメータ b.
3	x	{D / R}	1	入 力	変数値 x.
4	v	{D∗ / R∗}	1	出 力	メキシカンハット関数によるウェーブレット変換の基底 $\phi_{MH}(x; a, b)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a > 0

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) 注意事項

なし.

(7) 使用例

(a) 問 題

メキシカンハット関数によるウェーブレット変換の基底 $\phi_{MH}(4;2,3)$ を計算する.

(b) 入力データ

$\mathrm{a} = 4.0, \mathrm{b} = 2.0, \mathrm{x} = 3.0$

(c) 主プログラム

```
/*      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) 出力結果

```
*** ASL_dfwtmf ***

 ** Input **

a =        4 b =        2 c =        3

 ** Output **

ierr =      0

   v =     28.6
```

2.18.8 ASL_dfwtmt, ASL_rfwtmt
メキシカンハット関数によるウェーブレット変換

(1) 機 能

入力データとして与えられた n 個の x 座標と関数値 $f(x)$ の組 $\{(x_i, f(x_i))\}$ $(i = 1, 2, \cdots, n)$ について, メキシカンハット関数によるウェーブレット変換

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

を計算する.

(2) 使用法

倍精度関数:

ierr = ASL_dfwtmt (xd, yd, nd, a, b, & c);

単精度関数:

ierr = ASL_rfwtmt (xd, yd, nd, a, b, & c);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型
R:単精度実数型　C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	xd	{D∗ / R∗}	nd	入 力	入力データの x 座標の組 $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	入 力	入力データの関数値の組 $\{f(x_i)\}$.
3	nd	{D / R}	1	入 力	入力データの個数 n.
4	a	{D / R}	1	入 力	ウェーブレット変換の周波数パラメータ a
5	b	{D / R}	1	入 力	ウェーブレット変換のシフトパラメータ b
6	c	{D∗ / R∗}	1	出 力	ウェーブレット変換の値 $(W_{\phi_{MH}} f)(b, a)$.
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) a > 0

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) 注意事項

なし

(7) 使用例

(a) 問 題

関数

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}$$

を区間 $[0,1]$ で等間隔にサンプリングして得たデータを用いて, メキシカンハット関数によるウェーブレット変換 $(W_{\phi_{MH}} f)(2,3)$ を計算する.

(b) 入力データ

$\{(x_i, f(x_i))\}$,a=3, b=2

(c) 主プログラム

```
/*      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) 出力結果

```
*** 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
フレンチハット関数の計算

(1) **機 能**

フレンチハット関数

$$\varphi_{FH}(x) = \begin{cases} 1 & -1 \le x < 1 \\ -\frac{1}{2} & -3 \le x < -1 \text{ または } 1 \le x < 3 \\ 0 & \text{それ以外} \end{cases}$$

によるウェーブレット変換の基底

$$\phi_{FH}(x; a, b) = \frac{1}{\sqrt{3a}} \varphi\left(\frac{x-b}{a}\right)$$

を計算する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dfwtff  (a, b, x, & v);
```

単精度関数:

```
ierr = ASL_rfwtff  (a, b, x, & v);
```

(3) **引数と戻り値**

D:倍精度実数型　Z:倍精度複素数型　I: { 32 ビット整数版では int / 64 ビット整数版では long }
R:単精度実数型　C:単精度複素数型

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D / R}	1	入 力	フレンチハット関数によるウェーブレット変換の基底 $\phi_{FH}(x; a, b)$ の周波数パラメータ a.
2	b	{D / R}	1	入 力	フレンチハット関数によるウェーブレット変換の基底 $\phi_{FH}(x; a, b)$ のシフトパラメータ b
3	x	{D / R}	1	入 力	変数値 x
4	v	{D∗ / R∗}	1	出 力	フレンチハット関数によるウェーブレット変換の基底 $\phi_{FH}(x; a, b)$ の値.
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $a > 0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

フレンチハット関数によるウェーブレット変換の基底 $\phi_{FH}(1.5; 2, 1)$ を計算する.

(b) 入力データ

x=1.5,a=2, b=1

(c) 主プログラム

```
/*      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) 出力結果

```
*** ASL_dfwtff ***

 ** Input **

a =        2 b =        1 x =      1.5

 ** Output **

ierr =      0

   v =        0
```

2.18.10 ASL_dfwtft, ASL_rfwtft
フレンチハット関数によるウェーブレット変換

(1) **機 能**

入力データとして与えられた n 個の x 座標と関数値 $f(x)$ の組 $\{(x_i, f(x_i))\}$ $(i = 1, 2, \cdots, n)$ について, フレンチハット関数によるウェーブレット変換

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

を計算する.

(2) **使用法**

倍精度関数:

ierr = ASL_dfwtft (xd, yd, nd, a, b, & c);

単精度関数:

ierr = ASL_rfwtft (xd, yd, nd, a, b, & c);

(3) **引数と戻り値**

D:倍精度実数型 Z:倍精度複素数型
R:単精度実数型 C:単精度複素数型
I: { 32 ビット整数版では int / 64 ビット整数版では long }

項番	引数と戻り値	型	大きさ	入出力	内 容
1	xd	{D∗ / R∗}	nd	入 力	入力データの x 座標の組 $\{x_i\}$.
2	yd	{D∗ / R∗}	nd	入 力	入力データの関数値の組 $\{f(x_i)\}$.
3	nd	{D / R}	1	入 力	入力データの個数 n
4	a	{D / R}	1	入 力	ウェーブレット変換の周波数パラメータ a
5	b	{D / R}	1	入 力	ウェーブレット変換のシフトパラメータ b
6	c	{D∗ / R∗}	1	出 力	ウェーブレット変換の値 $(W_{\phi_{FH}} f)(b, a)$.
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a > 0

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

関数

$$f(x) = \sin(2\pi x) + \frac{1}{5}\sin(6\pi x) + \frac{1}{2}$$

を区間 $[0,1]$ で等間隔にサンプリングして得たデータを用いて, フレンチハット関数によるウェーブレット変換 $(W_{\phi_{FH}} f)(1,2)$ を計算する.

(b) 入力データ

$\{(x_i, f(x_i))\}$, a=2, b=1

(c) 主プログラム

```
/*      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) 出力結果

```
    *** 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
```

付 録 A　　ASL で使用している計算機依存定数

A.1　誤差判定のための単位

ASL では, 浮動小数点演算における誤差判定のための単位として次の値を設定している. 誤差判定のための単位は, 浮動小数点データの内部表現によって決まる数値であり, ASL C 言語インタフェース ではこの単位を収束判定, 零判定などに用いることがある.

表 A−1　誤差判定のための単位

単精度演算	倍精度演算
$2^{-23}(\simeq 1.19\times 10^{-7})$	$2^{-52}(\simeq 2.22\times 10^{-16})$

備考 誤差判定の単位 ε はマシン ε と呼ばれることもあり, 通常, 対応する浮動小数点形式で $1+\varepsilon$ の計算結果が 1 と異なるような最小の正の定数として定義される. したがって, 誤差判定の単位を見れば, その浮動小数点形式での (仮数部の) 演算の最大有効桁数がわかる.

A.2　浮動小数点データの値の最大値・最小値

ASL の内部で定義している浮動小数点データの値の最大値, 最小値を以下に示す.
なお, 以下の最大値, 最小値はハードウェアが実際に採用している浮動小数点形式のそれとは異なる場合があるので注意されたい.

表 A−2　浮動小数点データの値の最大値・最小値

	単精度演算	倍精度演算
最大値	$2^{127}(2-2^{-23})$ $(\simeq 3.40\times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80\times 10^{308})$
正の最小値	2^{-126} $(\simeq 1.17\times 10^{-38})$	2^{-1022} $(\simeq 2.23\times 10^{-308})$
負の最大値	-2^{-126} $(\simeq -1.17\times 10^{-38})$	-2^{-1022} $(\simeq -2.23\times 10^{-308})$
最小値	$-2^{127}(2-2^{-23})$ $(\simeq -3.40\times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80\times 10^{308})$

索 引

アプリケーションシステム

科学技術計算ライブラリ
ASL C 言語インタフェース
ユーザーズガイド

〈 基本機能編 第 3 分冊 〉

2023 年 3 月 ASL (1.1)
付属説明書 3.0.0-230301

日 本 電 気 株 式 会 社