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
＜共有メモリ並列機能編＞

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, 共有メモリ並列機能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_openmp)
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_openmp_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. (1.7 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 の共有メモリ並列機能の関数名は, 「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.

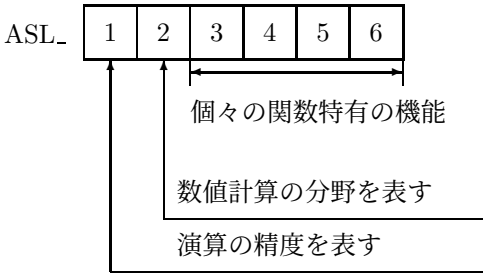


図 1−1　関数名の構成要素

図 1−1 の “1” ：演算の精度を表す.

共有メモリ並列機能編で使用される文字は, 次の 4 種類である.

q　共有メモリ並列機能倍精度実数型演算
p　共有メモリ並列機能単精度実数型演算
h　共有メモリ並列機能倍精度複素数型演算
g　共有メモリ並列機能単精度複素数型演算

また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

文字	計算の分野	分冊
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
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 の共有メモリ並列機能について

1.5.1 共有メモリ並列機能概要

すべての共有メモリ並列機能関数は OpenMP 機能を利用したマルチスレッド化に対応している．すなわち, 共有メモリ並列機能関数内の演算は, 複数のコア間で分散実行される．これにより, 大規模な問題を解析する場合に経過時間 (Elapse time) の短縮が見込まれる．共有メモリ並列処理環境のもとで利用者が ASL を使用するプログラムを並列に実行したい場合, 以下の 2 つの形態が考えられる．

(1) 利用者のプログラムから ASL の共有メモリ並列機能関数を呼び出す．この場合，ASL は内部の処理を複数のコアに分割して割り当て, 割り当てられた処理は並列に行われる．

(2) 利用者プログラム自身マルチスレッドを用いた並列化によって, 利用者関数と ASL 関数, あるいは, 複数の ASL 関数を並列に呼び出す．この場合, 並列に呼び出された関数の処理が複数のコアに割り当てられ, 並列に実行される．

なお, (2) の場合は ASL の非並列処理関数 (基本機能) を使用されたい．

1.5.2 共有メモリ並列機能における性能向上

ASL の共有メモリ並列機能を用いることでプログラムの性能が向上するための一般的な条件を以下に示す．

(1) **実行環境**

並列処理による効果は経過時間の短縮としてのみ現れるため, 複数のコアが空いている環境で実行する必要がある．システムに多数の JOB があるときは, 経過時間の短縮は期待できない．

(2) **問題規模**

共有メモリ並列処理関数では, 並列処理を行わない場合に比べて処理の分割, 同期のオーバヘッドが余分にかかる．したがって, 小さいサイズの問題では並列処理によって短縮される時間をこれらのオーバヘッドが上回ってしまう場合がある．一般に問題のサイズが大きくなると, オーバヘッドの影響は, 全体の処理時間に対して相対的に小さくなり, 並列処理の効果が大きくなる．したがって, 並列処理の効果を得るためには問題の規模は大きい方がよい．

1.5.3 共有メモリ並列機能使用に関する一般的な注意事項

(1) **実行環境**

並列機能を利用するためには OS がマルチプロセッサ対応である必要がある．共有メモリ並列処理 (マルチスレッド) の詳細については対応する OS, コンパイラのマニュアルを参照のこと．

(2) **タスクパラメータ** (nt) **について**

共有メモリ並列機能 関数は引数にタスクパラメータ nt を持つ．これは，ASL 内で共有メモリ並列化を行う際の分割数を指定するパラメータである．

(3) 同時に使用する CPU コア数の指定

環境変数 `OMP_NUM_THREADS` により, 共有メモリ並列処理で同時に使用する CPU コアの個数を指定できる．例えば, 同時に使用する CPU コアを 2 個にする場合, 下記のコマンドを実行する．

- C シェルの場合

  setenv OMP_NUM_THREADS 2 RETURN

- B シェルの場合

  OMP_NUM_THREADS=2 RETURN
  export OMP_NUM_THREADS RETURN

1.6 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 1–8 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.

1.7 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, 1.6 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　基本行列演算

2.1　概 要

本章では行列の積を求める関数について説明する.

本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.

2.1.1　使用上の注意

行列の次元数が小さいと, 演算コストに対して並列処理オーバヘッドの影響が大きいため, 非並列処理関数を用いた場合よりも性能が低下することがある.

2.1.2　使用しているアルゴリズム

2.1.2.1　行列の積

$A = (a_{ik}) \quad (1 \leq i \leq N_i, 1 \leq k \leq N_k)$

$B = (b_{kj}) \quad (1 \leq k \leq N_k, 1 \leq j \leq N_j)$

$C = (c_{ij}) \quad (1 \leq i \leq N_i, 1 \leq j \leq N_j)$

for　$j = 1, 2, \cdots, N_j$ (並列実行)
　for　$i = 1, 2, \cdots, N_i$
　　$c_{ij} = 0.0$

for　$j = 1, 2, \cdots, N_j$ (並列実行)
　for　$k = 1, 2, \cdots, N_k$
　　for　$i = 1, 2, \cdots, N_i$
　　　$c_{ij} = c_{ij} + a_{ik} b_{kj}$

並列化は j のループに関して行う.

2cpu の場合は, 図 2−1 に示すように $C = AB$ の計算を

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

の 2 つに分割し, それぞれを並列実行する.

図 2−1　2cpu の場合の行列積の並列化

$$\underset{\text{行列 } C}{\left[\, C_1 \mid C_2 \,\right]} = \underset{\text{行列 } A}{[A]} \cdot \underset{\text{行列 } B}{\left[\, B_1 \mid B_2 \,\right]}$$

2.2 基本行列演算

2.2.1 ASL_qam1mu, ASL_pam1mu
実行列 (2 次元配列型) の積 ($C = AB$)

(1) **機 能**

2 つの実行列 A, B(2 次元配列型) の積 ($C = AB$) を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lma × nn	入 力	実行列 A(2 次元配列型)
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の列数 (行列 B の行数)
5	b	{D∗ / R∗}	lnb × nl	入 力	実行列 B(2 次元配列型)
6	lnb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
8	c	{D∗ / R∗}	lmc × nl	出 力	行列 A, B の積 ($C = AB$)(2 次元配列型)
9	lmc	I	1	入 力	配列 c の整合寸法
10	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

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

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

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

(5) **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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

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

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

$C = AB$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lma} = 11$, $\mathrm{lnb} = 11$, $\mathrm{lmc} = 11$, $\mathrm{nm} = 4$, $\mathrm{nn} = 4$, $\mathrm{nl} = 4$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

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

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

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

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

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

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

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

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

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

    fclose( fp );


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


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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qam1mu ***

 ** Input **

Matrix a

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

Matrix b

      -3       -1        1       -1
      -3       -1        0        1
```

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

 ** Output **

ierr =       0

Matrix c

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

2.2.2 ASL_qam1mm, ASL_pam1mm 実行列 (2 次元配列型) の積 $(C = C \pm AB)$

(1) **機 能**

実行列の行列積 $(C = [C\pm]AB)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lma×nn	入 力	実行列 A (2 次元配列型)
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の列数 (行列 B の行数)
5	b	{D∗ / R∗}	lnb×nl	入 力	実行列 B (2 次元配列型)
6	lnb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
8	c	{D∗ / R∗}	lmc×nl	入 力	初期実行列 C (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB$ isw= 0 の時: $C = AB$ isw= -1 の時: $C = C - AB$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lma} = 11$, $\mathrm{lnb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 4$, $\mathrm{nn} = 5$, $\mathrm{nl} = 6$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

int main()
{
    double *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double *b;
    int lnb=11;
    int nl=6;
    double *c;
    int lmc=11;
    int isw=0;
```

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


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

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

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

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

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

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


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


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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qam1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 6

isw= 0   nt = 2

Matrix A
       1        1        1        1        1
       2        2        2        2        2
```

```
       3        3        3        3        3
       4        4        4        4        4

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

 ** Output **

ierr =      0

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

2.2.3 ASL_qam1mt, ASL_pam1mt
実行列 (2 次元配列型) の積 $(C = C \pm AB^T)$

(1) **機 能**

実行列の行列積 $(C = [C\pm]AB^T)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lma×nn	入 力	実行列 A (2 次元配列型)
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の列数 (行列 B の列数)
5	b	{D∗ / R∗}	llb×nn	入 力	実行列 B (2 次元配列型)
6	llb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
8	c	{D∗ / R∗}	lmc×nl	入 力	初期実行列 C (isw= ±1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB^T)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB^T$ isw= 0 の時: $C = AB^T$ isw= -1 の時: $C = C - AB^T$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB^T$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lma} = 11$, $\mathrm{llb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 4$, $\mathrm{nn} = 5$, $\mathrm{nl} = 5$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

int main()
{
    double *a;
    int lma=11;
    int nm=4;
    int nn=5;
    double *b;
    int llb=11;
    int nl=5;
    double *c;
    int lmc=11;
    int isw=0;
```

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


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

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

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

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

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

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


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


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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qam1mt ***

 ** Input **

lma=11   llb=11   lmc=11

nm = 4   nn = 5   nl = 5

isw= 0   nt = 2

Matrix A
       1        1        1        1        1
       2        2        2        2        2
```

```
       3        3        3        3        3
       4        4        4        4        4

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

 ** Output **

ierr =      0

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

2.2.4 ASL_qam1tm, ASL_pam1tm
実行列 (2 次元配列型) の積 $(C = C \pm A^T B)$

(1) **機 能**

実行列の行列積 $(C = [C\pm]A^T B)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lna×nm	入 力	実行列 A (2 次元配列型)
2	lna	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の行数 (行列 B の行数)
5	b	{D∗ / R∗}	lnb×nl	入 力	実行列 B (2 次元配列型)
6	lnb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
8	c	{D∗ / R∗}	lmc×nl	入 力	初期実行列 C (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^T B)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^T B$ isw= 0 の時: $C = A^T B$ isw= -1 の時: $C = C - A^T B$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^T B$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lna} = 11$, $\mathrm{lnb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 5$, $\mathrm{nn} = 5$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

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

```
    int lmc=11;
    int isw=0;
    int nt=2;
    int ierr;
    int i,j;


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

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

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

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

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

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


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


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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qam1tm ***

 ** Input **

lna=11   lnb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt = 2

Matrix A(Transposed Storage)
```

```
        1         2         3         4         5
        1         2         3         4         5
        1         2         3         4         5
        1         2         3         4         5
        1         2         3         4         5

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

 ** Output **

ierr =       0

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

2.2.5 ASL_qam1tt, ASL_pam1tt
実行列 (2 次元配列型) の積 $(C = C \pm A^T B^T)$

(1) **機 能**

実行列の行列積 $(C = [C\pm]A^T B^T)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lna×nm	入 力	実行列 A (2 次元配列型)
2	lna	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の行数 (行列 B の列数)
5	b	{D∗ / R∗}	llb×nn	入 力	実行列 B (2 次元配列型)
6	llb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
8	c	{D∗ / R∗}	lmc×nl	入 力	初期実行列 C (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^T B^T)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^T B^T$ isw= 0 の時: $C = A^T B^T$ isw= -1 の時: $C = C - A^T B^T$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lmc}$

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

(c) $0 < \mathrm{nl} \leq \mathrm{lnb}$

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^T B^T$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lna} = 11$, $\mathrm{llb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 5$, $\mathrm{nn} = 5$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

int main()
{
    double *a;
    int lna=11;
    int nm=5;
    int nn=5;
    double *b;
    int llb=11;
    int nl=4;
    double *c;
    int lmc=11;
    int isw=0;
```

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


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

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

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

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

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

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


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


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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qam1tt ***

 ** Input **

lna=11   llb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt = 2

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

```
        1         2         3         4         5
        1         2         3         4         5
        1         2         3         4         5

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

 ** Output **

ierr =       0

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

2.2.6 ASL_ham1mm, ASL_gam1mm 複素行列 (2 次元配列型) (実数引数型) の積 ($C = C \pm AB$)

(1) **機 能**

複素行列 (2 次元配列型) の行列積 ($C = [C\pm]AB$) を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	ar	{D∗ / R∗}	lma×nn	入 力	複素行列 A の実部 (2 次元配列型)
2	ai	{D∗ / R∗}	lma×nn	入 力	複素行列 A の虚部 (2 次元配列型)
3	lma	I	1	入 力	配列 ar と ai の整合寸法
4	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
5	nn	I	1	入 力	行列 A の列数 (行列 B の行数)
6	br	{D∗ / R∗}	lnb×nl	入 力	複素行列 B の実部 (2 次元配列型)
7	bi	{D∗ / R∗}	lnb×nl	入 力	複素行列 B の虚部 (2 次元配列型)
8	lnb	I	1	入 力	配列 br と bi の整合寸法
9	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
10	cr	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の実部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB)$ の実部
11	ci	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の虚部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB)$ の虚部
12	lmc	I	1	入 力	配列 cr と ci の整合寸法
13	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB$ isw= 0 の時: $C = AB$ isw= -1 の時: $C = C - AB$
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(d) isw=0, 1 または -1

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

(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	nn = 1 であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB$ を求める.

(b) 入力データ

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

(c) 主プログラム

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

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

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


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

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

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

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

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

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

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

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

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

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


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


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

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

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

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

        return 0;
    }
```

(d) 出力結果

```
*** ASL_ham1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt= 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.7 ASL_ham1mh, ASL_gam1mh 複素行列 (2 次元配列型) (実数引数型) の積 $(C = C \pm AB^*)$

(1) **機 能**

複素行列 (2 次元配列型) の行列積 $(C = [C\pm]AB^*)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	ar	{D∗ / R∗}	lma×nn	入 力	複素行列 A の実部 (2 次元配列型)
2	ai	{D∗ / R∗}	lma×nn	入 力	複素行列 A の虚部 (2 次元配列型)
3	lma	I	1	入 力	配列 ar と ai の整合寸法
4	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
5	nn	I	1	入 力	行列 A の列数 (行列 B の列数)
6	br	{D∗ / R∗}	llb×nn	入 力	複素行列 B の実部 (2 次元配列型)
7	bi	{D∗ / R∗}	llb×nn	入 力	複素行列 B の虚部 (2 次元配列型)
8	llb	I	1	入 力	配列 br と bi の整合寸法
9	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
10	cr	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の実部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB^*)$ の実部
11	ci	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の虚部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]AB^*)$ の虚部
12	lmc	I	1	入 力	配列 cr と ci の整合寸法
13	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB^*$ isw= 0 の時: $C = AB^*$ isw= -1 の時: $C = C - AB^*$
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(d) isw=0, 1 または -1

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

(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	nn = 1 であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB^*$ を求める.

(b) 入力データ

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

(c) 主プログラム

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

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

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


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

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

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

```
}

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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_ham1mh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt= 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.8 ASL_ham1hm, ASL_gam1hm
複素行列 (2 次元配列型) (実数引数型) の積 ($C = C \pm A^*B$)

(1) **機 能**

複素行列の行列積 ($C = [C\pm]A^*B$) を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	ar	{D∗ / R∗}	lna×nm	入 力	複素行列 A の実部 (2 次元配列型)
2	ai	{D∗ / R∗}	lna×nm	入 力	複素行列 A の虚部 (2 次元配列型)
3	lna	I	1	入 力	配列 ar と ai の整合寸法
4	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
5	nn	I	1	入 力	行列 A の行数 (行列 B の行数)
6	br	{D∗ / R∗}	lnb×nl	入 力	複素行列 B の実部 (2 次元配列型)
7	bi	{D∗ / R∗}	lnb×nl	入 力	複素行列 B の虚部 (2 次元配列型)
8	lnb	I	1	入 力	配列 br と bi の整合寸法
9	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
10	cr	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の実部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B)$ の実部
11	ci	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の虚部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B)$ の虚部
12	lmc	I	1	入 力	配列 cr と ci の整合寸法
13	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^*B$ isw= 0 の時: $C = A^*B$ isw= -1 の時: $C = C - A^*B$
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

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

(d) isw=0, 1 または -1

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

(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	nn = 1 であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^*B$ を求める.

(b) 入力データ

行列 A, 行列 B, lna = 11, lnb = 11, lnc = 11, nm = 5, nn = 4, nl = 4, isw = 0.

(c) 主プログラム

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

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

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


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

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

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

```
}

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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_ham1hm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 5   nn = 4   nl = 4

isw= 0   nt= 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.9 ASL_ham1hh, ASL_gam1hh
複素行列 (2 次元配列型) (実数引数型) の積 $(C = C \pm A^* B^*)$

(1) **機 能**

複素行列の行列積 $(C = [C\pm]A^* B^*)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	ar	{D∗ / R∗}	lna×nm	入 力	複素行列 A の実部 (2 次元配列型)
2	ai	{D∗ / R∗}	lna×nm	入 力	複素行列 A の虚部 (2 次元配列型)
3	lna	I	1	入 力	配列 ar と ai の整合寸法
4	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
5	nn	I	1	入 力	行列 A の行数 (行列 B の列数)
6	br	{D∗ / R∗}	llb×nn	入 力	複素行列 B の実部 (2 次元配列型)
7	bi	{D∗ / R∗}	llb×nn	入 力	複素行列 B の虚部 (2 次元配列型)
8	llb	I	1	入 力	配列 br と bi の整合寸法
9	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
10	cr	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の実部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B^*)$ の実部
11	ci	{D∗ / R∗}	lmc×nl	入 力	初期複素行列 C の虚部 (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B^*)$ の虚部
12	lmc	I	1	入 力	配列 cr と ci の整合寸法
13	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^*B^*$ isw= 0 の時: $C = A^*B^*$ isw= -1 の時: $C = C - A^*B^*$
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

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

(c) $0 < \mathrm{nl} \leq \mathrm{lnb}$

(d) isw=0, 1 または -1

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

(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	nn = 1 であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^*B^*$ を求める.

(b) 入力データ

行列 A, 行列 B, lna = 11, llb = 11, lnc = 11, nm = 5, nn = 5, nl = 4, isw = 0.

(c) 主プログラム

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

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

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


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

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

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

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

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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_ham1hh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt= 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.10 ASL_han1mm, ASL_gan1mm 複素行列 (2 次元配列型) (複素引数型) の積 ($C = C \pm AB$)

(1) **機 能**

複素行列 (2 次元配列型) の行列積 ($C = [C\pm]AB$) を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	{Z∗ / C∗}	lma×nn	入 力	複素行列 A (2 次元配列型)
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の列数 (行列 B の行数)
5	b	{Z∗ / C∗}	lnb×nl	入 力	複素行列 B (2 次元配列型)
6	lnb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
8	c	{Z∗ / C∗}	lmc×nl	入 力	初期複素行列 C (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 ($C = [C\pm]AB$)
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB$ isw= 0 の時: $C = AB$ isw= -1 の時: $C = C - AB$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lma} = 11$, $\mathrm{lnb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 4$, $\mathrm{nn} = 5$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

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

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


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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_han1mm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt = 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.11 ASL_han1mh, ASL_gan1mh
複素行列 (2 次元配列型) (複素引数型) の積 ($C = C \pm AB^*$)

(1) **機 能**

複素行列 (2 次元配列型) の行列積 ($C = [C\pm]AB^*$) を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	{Z∗ / C∗}	lma×nn	入 力	複素行列 A (2 次元配列型)
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の行数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の列数 (行列 B の列数)
5	b	{Z∗ / C∗}	llb×nn	入 力	複素行列 B (2 次元配列型)
6	llb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
8	c	{Z∗ / C∗}	lmc×nl	入 力	初期複素行列 C (isw= ± 1 の時)(2 次元配列型)
				出 力	行列積 ($C = [C\pm]AB$)
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + AB^*$ isw= 0 の時: $C = AB^*$ isw= -1 の時: $C = C - AB^*$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lma}, \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = AB^*$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lma} = 11$, $\mathrm{llb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 4$, $\mathrm{nn} = 5$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$, $\mathrm{nt} = 2$.

(c) 主プログラム

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

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

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

```
int ierr;
int i,j;


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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_han1mh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 4   nn = 5   nl = 4

isw= 0   nt = 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.12 ASL_han1hm, ASL_gan1hm
複素行列 (2 次元配列型) (複素引数型) の積 $(C = C \pm A^*B)$

(1) **機 能**

複素行列の行列積 $(C = [C\pm]A^*B)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	{Z∗ / C∗}	lna×nm	入 力	複素行列 A (2 次元配列型)
2	lna	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の行数 (行列 B の行数)
5	b	{Z∗ / C∗}	lnb×nl	入 力	複素行列 B (2 次元配列型)
6	lnb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の列数 (行列 C の列数)
8	c	{Z∗ / C∗}	lmc×nl	入 力	初期複素行列 C (isw= $\pm$1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^*B$ isw= 0 の時: $C = A^*B$ isw= -1 の時: $C = C - A^*B$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lmc}$

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

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

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^*B$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lna} = 11$, $\mathrm{lnb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 5$, $\mathrm{nn} = 4$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$.

(c) 主プログラム

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

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

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

```
int ierr;
int i,j;


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

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

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

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

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

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


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


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

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

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

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

        return 0;
    }
```

(d) 出力結果

```
*** ASL_han1hm ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 5   nn = 4   nl = 4

isw= 0   nt = 2

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

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

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

 ** Output **

ierr =      0

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

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

2.2.13 ASL_han1hh, ASL_gan1hh
複素行列 (2 次元配列型) (複素引数型) の積 $(C = C \pm A^*B^*)$

(1) **機 能**

実行列の行列積 $(C = [C\pm]A^*B^*)$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	{Z∗ / C∗}	lna×nm	入 力	行列 A (2 次元配列型)
2	lna	I	1	入 力	配列 a の整合寸法
3	nm	I	1	入 力	行列 A の列数 (行列 C の行数)
4	nn	I	1	入 力	行列 A の行数 (行列 B の列数)
5	b	{Z∗ / C∗}	llb×nn	入 力	行列 B (2 次元配列型)
6	llb	I	1	入 力	配列 b の整合寸法
7	nl	I	1	入 力	行列 B の行数 (行列 C の列数)
8	c	{Z∗ / C∗}	lmc×nl	入 力	初期複素行列 C (isw= $\pm$1 の時)(2 次元配列型)
				出 力	行列積 $(C = [C\pm]A^*B^*)$
9	lmc	I	1	入 力	配列 c の整合寸法
10	isw	I	1	入 力	処理スイッチ isw= 1 の時: $C = C + A^*B^*$ isw= 0 の時: $C = A^*B^*$ isw= -1 の時: $C = C - A^*B^*$
11	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{nm} \leq \mathrm{lmc}$

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

(c) $0 < \mathrm{nl} \leq \mathrm{lnb}$

(d) isw=0, 1 または -1

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

(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	$\mathrm{nn} = 1$ であった.	処理を続ける.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
3010	制限条件 (d) を満足しなかった.	
3020	制限条件 (e) を満足しなかった.	

(6) **注意事項**

なし.

(7) **使用例**

(a) 問 題

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

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

$C = A^*B^*$ を求める.

(b) 入力データ

行列 A, 行列 B, $\mathrm{lna} = 11$, $\mathrm{llb} = 11$, $\mathrm{lnc} = 11$, $\mathrm{nm} = 5$, $\mathrm{nn} = 5$, $\mathrm{nl} = 4$, $\mathrm{isw} = 0$.

(c) 主プログラム

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

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

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

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


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

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

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

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

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

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


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


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

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_han1hh ***

 ** Input **

lma=11   lnb=11   lmc=11

nm = 5   nn = 5   nl = 4

isw= 0   nt = 2

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

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

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

 ** Output **

ierr =      0

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

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

第 3 章　連立 1 次方程式 (直接法)

3.1　概 要

本章では, 連立 1 次方程式の解および行列の行列式の値と逆行列を求める関数について説明する.

本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.

本ライブラリでは, 個々の行列の性質および格納形式ごとに以下の機能をもつ関数が用意されている.

(1) 三角分解を行い, 連立 1 次方程式を解く.

(2) 係数行列の三角分解を行う.

(3) 係数行列の三角分解を行い, 条件数を求める.

(4) 三角分解後の連立 1 次方程式の解を求める.

(5) 行列式の値, 逆行列を求める.

利用者は, (1) ～ (5) の各関数を目的とする処理に合わせて自由に組み合わせることができる. これにより, 演算回数の無駄などのない効率のよい処理を行うことができる.

なお, (4), (5) については < 基本機能第 2 分冊 > 第 2 章を参照のこと.

3.1.1 使用方法

実行列 (2 次元配列型) の場合を例に挙げて説明する.

(1) 連立 1 次方程式

(a) ASL_qbgmsl を利用する方法

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

係数行列 A の三角分解を行い, $A\boldsymbol{x} = \boldsymbol{b}$ の解を求める.

(b) ASL_qbgmlu と $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ を利用する方法

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

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

ASL_qbgmlu で係数行列 A の三角分解を行い, $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ (< 基本機能第 2 分冊 > 2.2.5 参照) で $A\boldsymbol{x} = \boldsymbol{b}$ の解を求める.

(c) 条件数も求める方法

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

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

ASL_qbgmlc で係数行列 A の三角分解と条件数の算出を行い, $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ (< 基本機能第 2 分冊 > 2.2.5 参照) で $A\boldsymbol{x} = \boldsymbol{b}$ の解を求める.

(2) 行列式, 逆行列

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

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

ASL_qbgmlu で行列 A の三角分解を行い, $\left\{\begin{matrix}\text{ASL\_dbgmdi}\\ \text{ASL\_rbgmdi}\end{matrix}\right\}$ (< 基本機能第 2 分冊 > 2.2.7 参照) で行列式と逆行列を求める.

(3) 解の改良

(a) ASL_qbgmsl を利用する方法

$A_2 \leftarrow A$

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

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

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

ASL_qbgmsl で求められた解を改良する.

(b) ASL_qbgmlu と $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ を利用する方法

$A_2 \leftarrow A$

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

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

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

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

ASL_qbgmlu で A を三角分解し, $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ (< 基本機能第 2 分冊 > 2.2.5 参照) でその解を求め, $\left\{\begin{matrix}\text{ASL\_dbgmlx}\\ \text{ASL\_rbgmlx}\end{matrix}\right\}$ (< 基本機能第 2 分冊 > 2.2.8 参照) で解を改良する.

3.1.2 使用上の注意

(1) 係数行列の次元が小さいと, 演算コストに対して並列処理オーバヘッドの影響が大きいため非並列処理関数を用いた場合よりも性能が低下することがある.

(2) 連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く場合, 数式上は $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ であるが, 逆行列 A^{-1} を求めて, それを定数ベクトルに掛けるのはあまり得策ではない. たとえば, 実行列 (2 次元配列型) の場合, 係数行列の三角分解を行ってから解を求める場合と比べると, 変数が n 個の場合では前者が約 n^3 回, 後者が約 $n^3/3$ 回の乗算を必要とし, 明らかに後者の方が有利である. したがって, 逆行列 A^{-1} はそれ自体を必要とするときにのみ求めるべきである.

(3) 定数ベクトルのみが異なる複数の連立 1 次方程式を解く場合など, 同一の行列に対して何度も演算を行う場合には, 最初に一度だけ三角分解を行い, 以降はその結果を繰り返し利用すると効率がよい.
例

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

を解く場合,

$\mathrm{ierr} = \mathrm{ASL\_qbgmsl}\ (A, \cdots, \boldsymbol{b}_1, \cdots);$

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

とするか, または

$\mathrm{ierr} = \mathrm{ASL\_qbgmlu}\ (A, \cdots);$

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

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

のようにするとよい. 係数行列 A は ASL_qbgmsl または ASL_qbgmlu によって三角分解され, 以降は内容を変えることなく参照のみが行われる.

(4) 三角分解を行う関数としては, 条件数を求めるものと求めないものの 2 通りが用意されているが, 前者の方が条件数を求める演算の分だけ演算回数が多くなっている. n 次元の行列の場合, 前者は後者より約 n^2 回乗算が多い. 従って, 条件数を特に必要としない限り, 条件数を求めずに三角分解のみを行った方が実行時間を節約できる.

3.1.3 使用しているアルゴリズム

3.1.3.1 連立 1 次方程式の解法

連立 1 次方程式

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

を解くことを考えよう. 係数行列を $A = (a_{ij})$ $(i, j = 1, 2, \cdots, n)$, 右辺ベクトルを $\boldsymbol{b} = (b_i)$ $(i = 1, 2, \cdots, n)$, 解ベクトルを $\boldsymbol{x} = (x_i)$ $(i = 1, 2, \cdots, n)$ と定義すると, 上の連立 1 次方程式は

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

と行列形式で表せる. いま, 係数行列 $A = (a_{ij})$ $(i, j = 1, 2, \cdots, n)$ を下三角行列 $L = (\ell_{ij})$ $(i, j = 1, 2, \cdots, n)$ と上三角行列 $U = (u_{ij})$ $(i, j = 1, 2, \cdots, n)$ の積として $PA = LU$ と分解 (LU 分解) すれば, 解ベクトル $\boldsymbol{x} = (x_i)$ $(i = 1, 2, \cdots, n)$ は次の行列方程式を順次解けば得られる.

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

ここで, 行列 P は n 行 n 列の n 回の行交換に対応する行交換行列であり, ベクトル $\boldsymbol{y} = (y_i)$ $(i = 1, 2, \cdots, n)$ は計算の中間結果を与える作業ベクトルである. この行列方程式は係数行列が三角行列であるので前進代入または後退代入を用いて容易に解くことができる.

3.1.3.2 LU 分解 (ガウス法)

正則な行列 $A = (a_{ij})$ $(i, j = 1, 2, \cdots, n)$ を下三角行列 $L = (\ell_{ij})$ $(i, j = 1, 2, \cdots, n)$ と上三角行列 $U = (u_{ij})$ $(i, j = 1, 2, \cdots, n)$ の積として $PA = LU$ と分解することを考えよう. 正則な行列の場合にはこのような分解をいつでも行える. 簡単のため $A' = PA = (a'_{ij})$ $(i, j = 1, 2, \cdots, n)$ と置く. 行列積の定義より,

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

が成り立つ. なお, 分解を一意に決定するために $\ell_{ii} = 1$ $(i = 1, 2, \cdots, n)$ と置いている. この式を変形することによって, LU 分解を計算する各種アルゴリズムが得られる.
ここでは, n 行 n 列の行列 $M = (m_{ij})$ $(i, j = 1, 2, \cdots, n)$ の初期値として行列 A' を与え, 計算後, その行列の各要素に下三角行列 $L = (\ell_{ij})$ $(i, j = 1, 2, \cdots, n)$ と上三角行列 $U = (u_{ij})$ $(i, j = 1, 2, \cdots, n)$ の各要素がそれぞれ

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

と与えられるようなアルゴリズムを考える. アルゴリズム (kji-SAXPY 形式ガウス法などと呼ばれる) は次のように表せる.

do $k = 1, \cdots, n-1$
  do $i = k+1, \cdots, n$
    $m_{ik} = -\dfrac{m_{ik}}{m_{kk}}$
  do $j = k+1, \cdots, n$
    do $i = k+1, \cdots, n$
      $m_{ij} = m_{ij} + m_{ik} m_{kj}$

なお, 実現にあたっては上のアルゴリズムで $m_{kk} \simeq 0$ となる要素による除算を避けるために,
$\mid m_{pk} \mid = \max_{k \leq i \leq n} \mid m_{ik} \mid$ が成立する要素 m_{pk} を選択し, 除算の前に p 行と k 行を交換する部分軸選択を採用している. この行交換操作の蓄積を行列で表したものが, 行交換行列 P である.

なお, 本ライブラリでは k に対するループを r 個ずつまとめ, 次のようなアルゴリズム変換 (ブロック化) を繰り返し適用して, 代入の数を減らすことによって高速化を実現している.

do $k = 1, \cdots, r$
  do $j = k+1, \cdots, n$
    do $i = k+1, \cdots, n$
      $m_{ij} = m_{ij} + m_{ik} m_{kj}$

⇓

前処理 1 :

do $k = 1, \cdots, r$
  do $j = k+1, \cdots, r$
    do $i = k+1, \cdots, n$
      $m_{ij} = m_{ij} + m_{ik} m_{kj}$

前処理 2 :

do $j = r+1, \cdots, n$
  do $k = 1, \cdots, r$
    do $i = k+1, \cdots, r$
      $m_{ij} = m_{ij} + m_{ik} m_{kj}$

ブロック更新 :

do $j = r+1, \cdots, n$
  do $i = r+1, \cdots, n$
    $m_{ij} = m_{ij} + m_{i1}\ m_{1j} + m_{i2}\ m_{2j} + \cdots + m_{ir} m_{rj}$

また, r を変更することによって前処理 1 に対しても再帰的にこのような変換を実行して, 高速化を計っている.

次に, このアルゴリズムの並列化について述べる.

上に並べたアルゴリズムは前処理 2 とブロック更新をまとめてこの部分の j に対するループを並列処理タスク数で分割して並列実行することにより, 並列化できる. なお, 本ライブラリでは, 次のブロック更新のための前処理 1 も考慮したタスク分割を採用することによって効率化を図っている (参考文献 (1) 参照).

3.1.4 参考文献

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

3.2 実行列 (2 次元配列型)

3.2.1 ASL_qbgmsm
多重右辺連立 1 次方程式 (実行列)

(1) **機 能**

実行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を, ガウス法を用いて解く. すなわち, $n \times m$ 行列 B を $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ と定義した時, $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	ab	D*	lna × (n + m)	入 力	係数行列 A と右辺ベクトル $\boldsymbol{b_i}$ からなる行列 (実行列, 2 次元配列型) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	係数行列 A の分解行列 A' と解ベクトル $\boldsymbol{x_i}$ からなる行列 (実行列, 2 次元配列型) $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (注意事項 (a), (b) 参照)
2	lna	I	1	入 力	配列 ab の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	右辺ベクトルの数 m
5	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 (注意事項 (c) 参照)
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

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

(5) エラーインディケータ (戻り値)

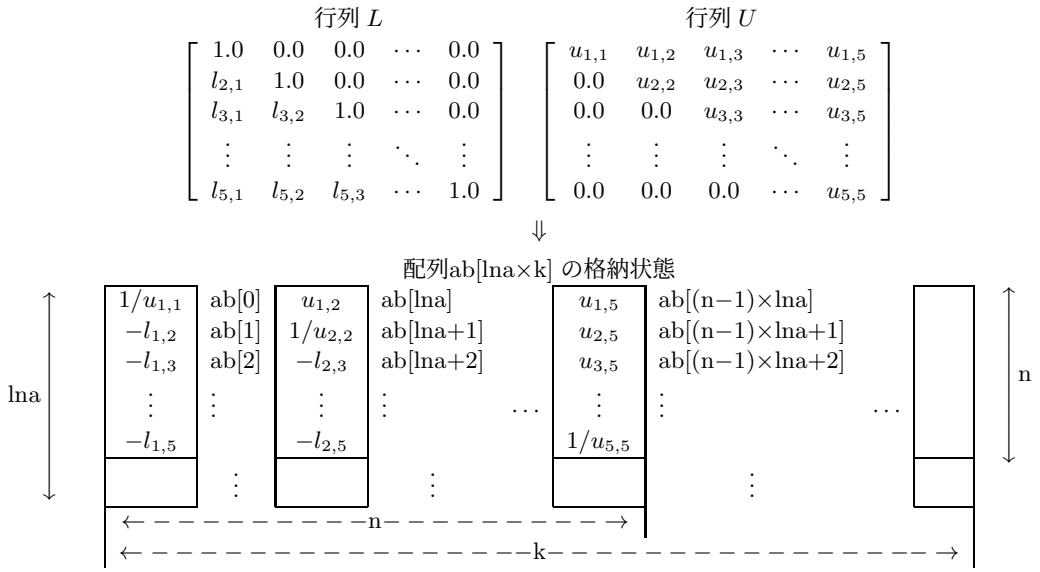
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	ab[lna∗(n+i−1)] ← ab[lna∗(n+i−1)]/ab[0] ($i = 1, 2, \cdots,$ m) とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000+i	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) 注意事項

(a) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(b) 配列 ab の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角成分は常に 1.0 であるので, 配列 ab には格納されない. また, U の対角成分はその逆数が格納される.

図 3−1　行列 L と行列 U の格納状態



備　考

a.　lna ≥ n, n+m ≤ k を満たさなければならない.

(c) 単精度の共有メモリ並列機能はサポートしていない.

(7) 使用例

(a) 問 題

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

を解く.

(b) 入力データ

係数行列 A と定数ベクトル$\boldsymbol{b_1}$, $\boldsymbol{b_2}$を格納した配列 ab, lna=11, n=4, m=2, nt=2

(c) 主プログラム

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

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

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

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

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

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

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

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

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

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

    fclose( fp );
```

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

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


    free( ab );
    free( ipvt );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qbgmsm ***

 ** Input **


Coefficient Matrix

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

Constant Vectors

      36        11
      15         0
      22         7
      -6         4

 ** Output **

ierr =      0

Solution

       1         1
       2         1
       4         1
       5         1
```

3.2.2 ASL_qbgmsl
連立 1 次方程式 (実行列)

(1) **機 能**

実行列 A(2 次元配列型) を係数行列とする連立方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を, ガウス法を用いて解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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*	lna×n	入 力	係数行列 A(実行列, 2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U, および下三角行列 L(注意事項 (b), (c) 参照)
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
4	b	D*	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解ベクトル$\boldsymbol{x}$
5	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 (注意事項 (b) 参照)
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	b[0] ← b[0]/a[0] とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000 + i	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. a は特異である.	

(6) **注意事項**

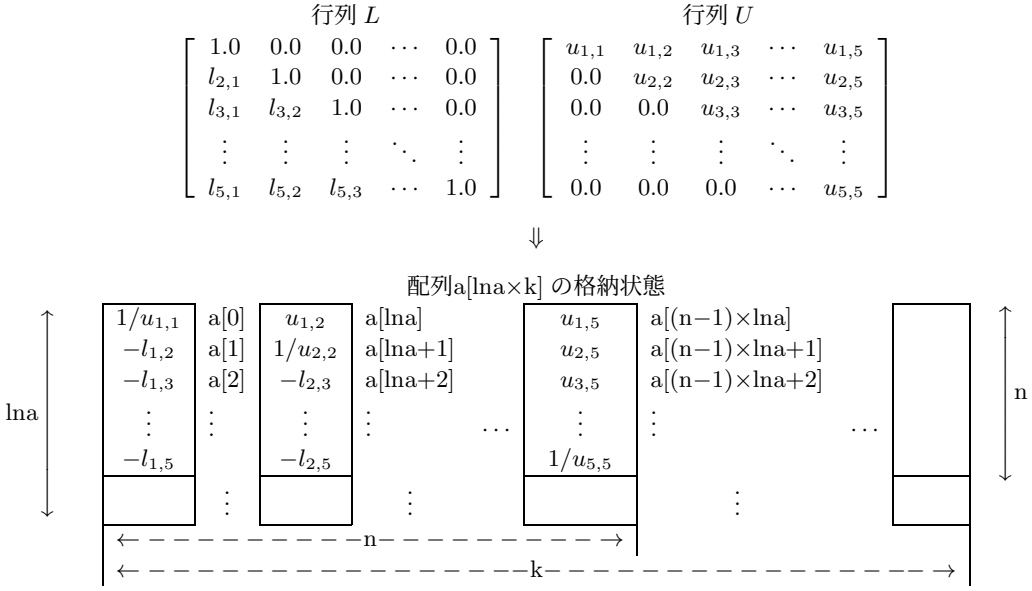
(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, 直接関数 3.2.1 ASL_qbgmsm を用いて計算する方が効率よく解が求まる. ただし, 右辺ベクトル***b***のすべてが前もって分からない場合など, 3.2.1 ASL_qbgmsm を利用できない場合には, この関数を一度使用した後, 続けて関数 < 基本機能第 2 分冊 > 2.2.5 $\left\{\begin{array}{l}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{array}\right\}$ を配列 b の内容のみを変えて使用すればよい. このようにすれば, 行列 A の LU 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 配列 a の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角部分は常に 1.0 であるので, 配列 a には格納されない. また, U の対角成分はその逆数が格納される (図 3−2 参照).

(d) 単精度の共有メモリ並列機能はサポートしていない.

図 3−2　行列 L と行列 U の格納状態



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

(7) 使用例

(a) 問 題

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

を解く.

(b) 入力データ

係数行列 A, lna $= 11$, n $= 4$, 定数ベクトル$\boldsymbol{b}$, nt $= 2$

(c) 主プログラム

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

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

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

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

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

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

    a = ( double * )malloc((size_t)( sizeof(double) * (lna*n) ));
    if( a == NULL )
    {
```

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

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

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

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

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

    fclose( fp );


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


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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_qbgmsl ***

 ** Input **

 n  =      4
 nt =      2

Coefficient Matrix

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

Constant Vector

      36
      15
      22
      -6

 ** Output **

ierr =      0

Solution

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

3.2.3 ASL_qbgmlu
実行列の LU 分解

(1) **機 能**

実行列 A(2 次元配列型) をガウス法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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*	lna×n	入 力	実行列 A(2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L(注意事項 (a), (b) 参照)
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
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 (注意事項 (b) 参照)
5	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000 + i	i 段目の処理において, ピボットが 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) 配列 a には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されない. また U の対角成分は, その逆数が格納される (3.2.2 図 3−2 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 単精度の共有メモリ並列機能はサポートしていない.

3.2.4 ASL_qbgmlc
実行列の LU 分解と条件数

(1) **機 能**

実行列 A (2 次元配列型) をガウス法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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*	lna×n	入 力	実行列 A(2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L(注意事項 (a), (b) 参照)
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
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 (注意事項 (b) 参照)
5	cond	D*	1	出 力	条件数の逆数
6	w1	D*	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない. cond ← 1.0 とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000 + i	i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	処理を打ち切る. 条件数は求められない.

(6) **注意事項**

(a) 配列 a には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 U の対角成分は常に 1.0 であるので, 配列 a には格納されない. また, U の対角成分はその逆数が格納される (3.2.2 図 3−2 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 条件数は $\|A\| \cdot \|A^{-1}\|$ で定義されるが, この関数で求められるのはその概算値である.

(d) 単精度の共有メモリ並列機能はサポートしていない.

3.3 複素行列 (2 次元配列型)(実数引数型)

3.3.1 ASL_hbgmsm
多重右辺連立 1 次方程式 (複素行列)

(1) **機 能**

複素行列 A=(ar, ai) (2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を, ガウス法を用いて解く. すなわち, $n \times m$ 行列 B を $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ と定義した時, $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	abr	D*	lna × (n+m)	入 力	係数行列 A と右辺ベクトル $\boldsymbol{b_i}$ からなる行列の実部 (複素行列, 2 次元配列型) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	係数行列 A の分解行列 A' と解ベクトル $\boldsymbol{x_i}$ からなる行列の実部 (複素行列, 2 次元配列型) $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (注意事項 (a), (b) 参照)
2	abi	D*	lna × (n+m)	入 力	係数行列 A と右辺ベクトル $\boldsymbol{b_i}$ からなる行列の虚部 (複素行列, 2 次元配列型) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	係数行列 A の分解行列 A' と解ベクトル $\boldsymbol{x_i}$ からなる行列の虚部 (複素行列, 2 次元配列型) $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (注意事項 (a), (b) 参照)
3	lna	I	1	入 力	配列 abr, abi の整合寸法
4	n	I	1	入 力	行列 A の次数
5	m	I	1	入 力	右辺ベクトルの数 m
6	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 (注意事項 (a) 参照)
7	w1	D*	N	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

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

(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.	abr[lna∗(n+i-1)] ← (abr[lna∗(n+i−1)] × abr[0] + abi[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2) , abi[lna∗(n+i−1)] ← (abi[lna∗(n+i−1)] × abr[0] − abr[lna∗(n+i−1)] × abi[0]) / (abr[0]^2 + abi[0]^2) (i=1, 2, ···, m) とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000+i	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(b) 配列 abr, abi の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角成分は常に 1.0 であるので, 配列 abr, abi には格納されない. また, U の対角成分はその逆数が格納される. 図 3–3 において, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ, 複素数 z の実部と虚数部を表す.

(c) 単精度の共有メモリ並列機能はサポートしていない.

図 3−3　行列 L と行列 U の格納状態

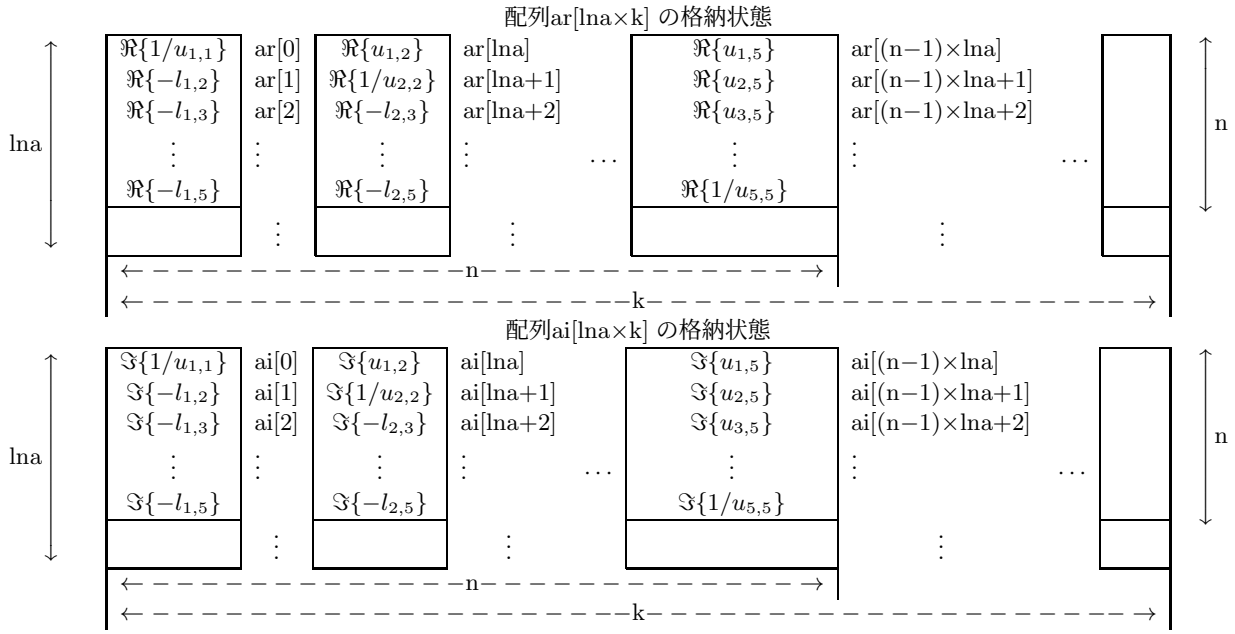
行列 L

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

行列 U

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

$\Downarrow$



備　考

a.　lna ≥ n, n+m ≤ k を満たさなければならない.

(7) 使用例

(a) 問 題

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

を解く.

(b) 入力データ

係数行列 A と定数ベクトル$\boldsymbol{b_1}$, $\cdots$,$\boldsymbol{b_4}$ からなる行列の実部 abr および虚部 abi, lna=11, n=4, m=4, nt=2

(c) 主プログラム

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

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

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

```
int nt=2;
FILE *fp;

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

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

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

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

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

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

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

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

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

fclose( fp );


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


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

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

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_hbgmsm ***

 ** Input **

 n =      4 m =      4

Coefficient Matrix

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

Constant Vectors

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

 ** Output **

ierr =      0

Solution

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

3.3.2 ASL_hbgmsl
連立 1 次方程式 (複素行列)

(1) **機 能**

複素行列 A=(ar, ai)(2 次元配列型) を係数行列とする連立方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を, ガウス法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_hbgmsl (ar, ai, lna, n, br, bi, ipvt, w, nt);

単精度関数:

なし

(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	ar	D*	lna×n	入 力	係数行列 A の実部 (複素行列, 2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U, および下三角行列 L(注意事項 (b), (c) 参照).
2	ai	D*	lna×n	入 力	係数行列 A の虚部 (複素行列, 2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U, および下三角行列 L(注意事項 (b), (c) 参照).
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	br	D*	n	入 力	定数ベクトル$\boldsymbol{b}$の実部
				出 力	解ベクトル$\boldsymbol{x}$の実部
6	bi	D*	n	入 力	定数ベクトル $\boldsymbol{b}$の虚部
				出 力	解ベクトル$\boldsymbol{x}$の虚部
7	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 (注意事項 (b) 参照)
8	w	D*	n	ワーク	作業領域
9	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	$\mathrm{br}[0] \leftarrow \{\mathrm{br}[0] \times \mathrm{ar}[0] + \mathrm{bi}[0] \times \mathrm{ai}[0]\}/\{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$ $\mathrm{bi}[1] \leftarrow \{\mathrm{bi}[0] \times \mathrm{ar}[0] - \mathrm{br}[0] \times \mathrm{ai}[0]\}/\{\mathrm{ar}[0]^2 + \mathrm{ai}[0]^2\}$ とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
$4000 + \mathrm{i}$	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) 定数ベクトル$\boldsymbol{b}$のみが異なる複数の連立 1 次方程式を解く場合には, 直接関数 3.3.1 ASL_hbgmsm を用いて計算する方が効率よく解が求まる. ただし, 右辺ベクトル$\boldsymbol{b}$のすべてが前もって分からない場合など, 3.3.1 ASL_hbgmsm を利用できない場合には, この関数を一度使用した後, 続けて関数 < 基本機能第 2 分冊 > 2.3.5 $\left\{\begin{matrix}\mathrm{ASL\_zbgmls}\\ \mathrm{ASL\_cbgmls}\end{matrix}\right\}$ を配列 b の内容のみを変えて使用すればよい. このようにすれば, 行列 A の LU 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 ($\mathrm{i} \leq \mathrm{j}$) となった場合, $\mathrm{ipvt}[\mathrm{i}-1]$ に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 配列 ar, ai の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角部分は常に 1.0 であるので, 配列 ar, ai には格納されない. また, U の対角成分はその逆数が格納される. 図 3−4 において, $\Re\{z\}$ と $\Im\{z\}$ はそれぞれ, 複素数 z の実部と虚数部を表す.

(d) 単精度の共有メモリ並列機能はサポートしていない.

図 3−4　行列 L と行列 U の格納状態

行列 L

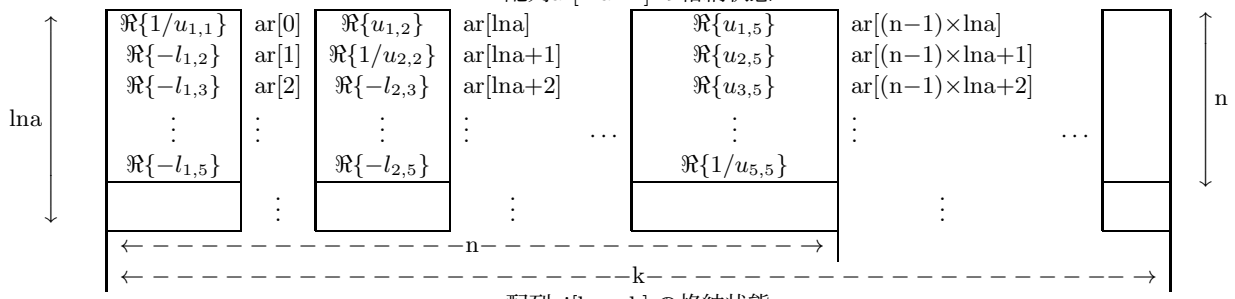
$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0 \end{bmatrix}$$

行列 U

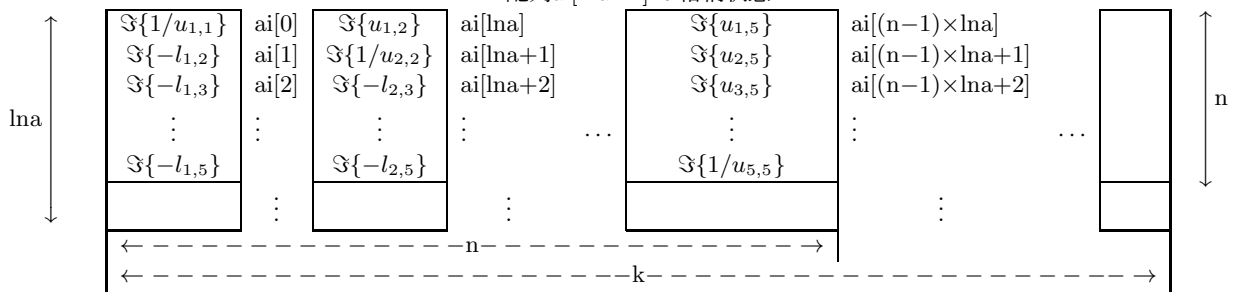
$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,5} \\ 0.0 & u_{2,2} & u_{2,3} & \cdots & u_{2,5} \\ 0.0 & 0.0 & u_{3,3} & \cdots & u_{3,5} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & u_{5,5} \end{bmatrix}$$

$\Downarrow$

配列ar[lna×k] の格納状態



配列ai[lna×k] の格納状態



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

(7) **使用例**

(a) 問 題

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

を解き, 条件数を求める.

(b) 入力データ

係数行列の実部 ar および虚部 ai, lna=11, n=4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

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

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

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    int nt = 2;
    double *br;
    double *bi;
```

```
int *kpvt;
double *w;
int ierr;
int i,j;
FILE *fp;

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

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

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

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

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

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

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

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

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

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

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

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

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

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


    fclose( fp );


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


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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_hbgmsl ***

 ** Input **

 n  =      4
 nt =      2

Coefficient Matrix (Real, Imaginary)

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

Constant Vector (Real, Imaginary)

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

 ** Output **

ierr =      0

Solution (Real, Imaginary)

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

3.3.3 ASL_hbgmlu
複素行列の LU 分解

(1) **機 能**

複素行列 A(2 次元配列型) をガウス法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	ar	D*	lna×n	入 力	複素行列 A の実部 (2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L の実部 (注意事項 (a), (b) 参照)
2	ai	D*	lna×n	入 力	複素行列 A の虚部 (2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L の虚部 (注意事項 (a), (b) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	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 (注意事項 (b) 参照)
6	w	D*	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 ar, ai の内容は変更されない.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000 + i	i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) 配列 ar, ai には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 ar, ai には格納されない. また U の対角成分は, その逆数が格納される (3.3.2 図 3−4 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 ($i \leq j$) となった場合, ipvt[i − 1] に j が格納される. また, このとき行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 単精度の共有メモリ並列機能はサポートしていない.

3.3.4 ASL_hbgmlc
複素行列の LU 分解と条件数

(1) **機 能**

複素行列 A=(ar, ai)(2 次元配列型) をガウス法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	ar	D*	lna×n	入 力	複素行列 A の実部 (2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L の実部 (注意事項 (a), (b) 参照)
2	ai	D*	lna×n	入 力	複素行列 A の虚部 (2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L の虚部 (注意事項 (a), (b) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	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 (注意事項 (b) 参照)
6	cond	D*	1	出 力	条件数の逆数
7	w	D*	2 × n	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 ar, ai の内容は変更されない. cond ← 1.0 とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000+i	i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	処理を打ち切る. 条件数は求められない.

(6) **注意事項**

(a) 配列 ar, ai には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 U の対角成分は常に 1.0 であるので, 配列 ar, ai には格納されない. また, U の対角成分はその逆数が格納される (3.3.2 図 3−4 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 ($i \leq j$) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 条件数は $\|A\| \cdot \|A^{-1}\|$ で定義されるが, この関数で求められるのはその概算値である.

(d) 単精度の共有メモリ並列機能はサポートしていない.

3.4 複素行列 (2 次元配列型)(複素引数型)

3.4.1 ASL_hbgnsm
多重右辺連立 1 次方程式 (複素行列)

(1) **機 能**

複素行列 A (2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を, ガウス法を用いて解く. すなわち, $n \times m$ 行列 B を $B = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ と定義した時, $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}] = A^{-1}B$ を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	ab	Z*	lna × (n+m)	入 力	係数行列 A と右辺ベクトル $\boldsymbol{b_i}$ からなる行列 (複素行列, 2 次元配列型) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	係数行列 A の分解行列 A' と解ベクトル $\boldsymbol{x_i}$ からなる行列 (複素行列, 2 次元配列型) $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$ (注意事項 (a), (b) 参照)
2	lna	I	1	入 力	配列 ab の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	右辺ベクトルの数 m
5	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 (注意事項 (a) 参照)
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

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

(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.	ab[lna∗(n+i−1)] ← ab[lna∗(n+i−1)]/ab[0] $(i = 1, 2, \cdots, \mathrm{m})$ とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
3020	制限条件 (c) を満足しなかった.	
4000+i	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) 注意事項

(a) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(b) 配列 ab の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角成分は常に 1.0 であるので, 配列 a には格納されない. また, U の対角成分はその逆数が格納される (3.2.1 図 3−1 参照).

(c) 単精度の共有メモリ並列機能はサポートしていない.

(7) 使用例

(a) 問 題

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

を解く.

(b) 入力データ
係数行列 A と定数ベクトル$\boldsymbol{b_1}, \cdots, \boldsymbol{b_4}$ からなる行列 ab, lna=11, n=4, m=4, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_hbgnsm */

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

int main()
{
    double _Complex *ab;
    int lna=11, lma=5;
    int n;
    int m;
    int *ipvt;
    int nt = 2;
```

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

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

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

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

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

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

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

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

    fclose( fp );

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

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

    free( ab );
    free( ipvt );

    return 0;
}
```

(d) 出力結果

```
*** ASL_hbgnsm ***

 ** Input **

 n =      4 m =      4
```

```
Coefficient Matrix

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

Constant Vectors

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

 ** Output **

ierr =      0

Solution

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

3.4.2 ASL_hbgnsl
連立 1 次方程式 (複素行列)

(1) **機 能**

複素行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ をガウス法を用いて解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	Z*	lna×n	入 力	係数行列 A(複素行列, 2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U, および下三角行列 L (注意事項 (b), (c) 参照)
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
4	b	Z*	n	入 力	定数ベクトル $\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
5	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 (注意事項 (b) 参照)
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$ かつ $\mathrm{nt} \geq 1$

(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.	b[0] ← b[0]/a[0] とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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.
4000+i	係数行列 A の LU 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, 直接関数 3.4.1 ASL_hbgnsm を用いて計算する方が効率よく解が求まる. ただし, 右辺ベクトル***b***のすべてが前もって分からない場合など, 3.4.1 ASL_hbgnsm を利用できない場合には, この関数を一度使用した後, 続けて関数 < 基本機能第 2 分冊 > 2.4.5 $\left\{\begin{array}{l}\text{ASL\_zbgnls}\\ \text{ASL\_cbgnls}\end{array}\right\}$ を配列 b の内容のみを変えて使用すればよい. このようにすれば, 行列 A の LU 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) この関数では, 係数行列 A の LU 分解時に, 部分軸選択 (partial pivoting) が行われている. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 配列 a の下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, L の対角成分は常に 1.0 であるので, 配列 a には格納されない. また, U の対角成分はその逆数が格納される (3.2.2 図 3−2 参照).

(d) 単精度の共有メモリ並列機能はサポートしていない.

(7) **使用例**

(a) 問 題

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

(b) 入力データ
係数行列 A, lna = 11, n = 4, 定数ベクトル***b***, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_hbgnsl */

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

int main()
{
    double _Complex *a;
    int lna;
    int n;
```

```
double _Complex *b;
int *ipvt;
int nt = 2;
int ierr;
int i,j;
FILE *fp;

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

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

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

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

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

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

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

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

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

fclose( fp );

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

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

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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_hbgnsl ***

 ** Input **

 n =      4

Coefficient Matrix   (Real, Imaginary)

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

Constant Vector  (Real, Imaginary)

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

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

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

3.4.3 ASL_hbgnlu
複素行列の LU 分解

(1) **機 能**

複素行列 A(2 次元配列型) をガウス法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hbgnlu  (a, lna, n, ipvt, nt);
```

単精度関数:

なし

(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	Z*	lna×n	入 力	複素行列 A(2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L(注意事項 (a), (b) 参照)
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
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 (注意事項 (b) 参照)
5	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$ かつ $\mathrm{nt} \geq 1$

(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.	配列 a の内容は変更されない.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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.
4000+i	i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) 配列 a には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されない. また U の対角成分は, その逆数が格納される (3.2.2 図 3−2 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 単精度の共有メモリ並列機能はサポートしていない.

3.4.4 ASL_hbgnlc
複素行列の LU 分解と条件数

(1) **機 能**

複素行列 A(2 次元配列型) をガウス法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

なし

(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	Z*	lna×n	入 力	複素行列 A(2 次元配列型)
				出 力	$A = LU$ と分解したときの上三角行列 U および下三角行列 L(注意事項 (a), (b) 参照)
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
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 (注意事項 (b) 参照)
5	cond	D*	1	出 力	条件数の逆数
6	w1	Z*	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$ かつ $\mathrm{nt} \geq 1$

(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.	配列 a の内容は変更されない. cond ← 1.0 とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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.
4000+i	i 段目の処理において, 対角要素が 0.0 となった. 行列 A は特異である.	

(6) **注意事項**

(a) 配列 a には, 下三角部分に単位下三角行列 L が符号をかえて, 上三角部分に上三角行列 U が格納される. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されない. また, U の対角成分はその逆数が格納される (3.2.2 図 3−2 参照).

(b) この関数においては, 部分軸選択 (partial pivoting) が行われている. このときの情報は後続の関数で使用されるため, 配列 ipvt に格納される. 第 i 段目のピボット行が第 j 行 ($i \leq j$) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行と第 j 行の対応する列要素のうち, 第 1 列から第 n 列までの要素が実際に交換される.

(c) 条件数は $\|A\| \cdot \|A^{-1}\|$ で定義されるが, この関数で求められる値はその概算値である.

(d) 単精度の共有メモリ並列機能はサポートしていない.

3.5 実対称行列 (2 次元配列型)(上三角型)

3.5.1 ASL_qbspsl, ASL_pbspsl
連立 1 次方程式 (実対称行列)

(1) **機 能**

実対称行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗}	lna×n	入 力	係数行列 A(実対称行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^T$ と分解した時の上三角行列 L^T (注意事項 (b) 参照)
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
4	b	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
5	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 (列)i と交換した行 (列) の番号 (注意事項 (c) 参照)
6	wk	{D∗ R∗}	N	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	b[0] ← b[0]/a[0] とする.
2100	係数行列 A の $\mathrm{LDL^T}$ 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の $\mathrm{LDL^T}$ 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて < 基本機能第 2 分冊 > 2.6.4 $\left\{\begin{matrix}\text{ASL_dbspls}\\ \text{ASL_rbspls}\end{matrix}\right\}$を配列 b の内容のみを変えて使用すればよい. このようにすれば行列 A の $\mathrm{LDL^T}$ 分解が一度だけしか行われないため, 演算効率よく解が求まる.

(b) 配列 a には, 上三角行列 L^T のみが格納される. 対角行列 D, および下三角行列 L は L^T より算出されるので, 配列 a には格納されない. 行列 L は行列 L^T の転置行列であり, 行列 D は行列 L^T の対角要素の逆数を成分とする対角行列である. この関数は配列 a の上三角部分のみを使用する.

図 3−5　行列 L^T の格納状態と行列 D の内容

行列 L^T

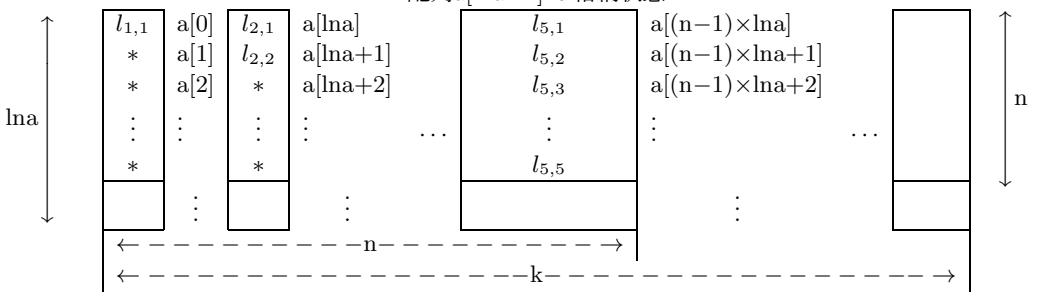
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

行列 D

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

⇓

配列a[lna×k] の格納状態



備　考
a.　lna ≥ n, n ≤ k を満たさなければならない.
b.　＊に対応する入力時の値は保証されない.

(c) この関数では, 係数行列 A の $\mathrm{LDL^T}$ 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)(i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 4 \\ -4 \end{bmatrix}$$ を解く.

(b) 入力データ

係数行列 A, lna=11, n = 4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_qbspsl */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int *ipvt;
    double *wk;
    int nt = 2;
    int ierr;

    int i,j;

    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

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

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

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

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

    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
```

```
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }

    fclose( fp );


    ierr = ASL_qbspsl(a, na, n, b, ipvt, wk, nt);


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

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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_qbspsl ***

 ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

 ** Output **

ierr =      0

Solution

  x[     0] =         1
  x[     1] =        -1
  x[     2] =         2
  x[     3] =        -2
```

3.5.2 ASL_qbspud, ASL_pbspud 実対称行列の $\mathrm{LDL^T}$ 分解

(1) **機 能**

実対称行列 A(2 次元配列型)(上三角型) を修正コレスキー法を用いて $\mathrm{LDL^T}$ 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qbspud  (a, lna, n, ipvt, wk, nt);
```

単精度関数:

```
ierr = ASL_pbspud  (a, lna, n, ipvt, wk, nt);
```

(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∗}	lna×n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	$A = LDL^T$ と分解した時の上三角行列 L^T (注意事項 (a) 参照)
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
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 (列)i と交換した行 (列) の番号 (注意事項 (b) 参照)
5	wk	{D∗ / R∗}	N	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない.
2100	係数行列 A の $\mathrm{LDL^T}$ 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の $\mathrm{LDL^T}$ 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

(6) **注意事項**

(a) 配列 a には, 上三角行列 L^T のみが格納される. 対角行列 D, および下三角行列 L は L^T より算出されるので, 配列 a には格納されない (3.5.1 図 3−5 参照).

(b) この関数では, 係数行列 A の $\mathrm{LDL^T}$ 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)($\mathrm{i} \leq \mathrm{j}$) となった場合, $\mathrm{ipvt[i-1]}$ に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

3.6 実対称行列 (2 次元配列型) (下三角型) (軸選択なし)

3.6.1 ASL_qbsnsl, ASL_pbsnsl
連立 1 次方程式 (実対称行列) (軸選択なし)

(1) **機 能**

実対称行列 A(2 次元配列型)(下三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_qbsnsl (a, lna, n, b, nt);

単精度関数:

ierr = ASL_pbsnsl (a, lna, n, b, nt);

(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∗}	lna×n	入 力	係数行列 A(実対称行列, 2 次元配列型, 下三角型)
				出 力	$A = U^T DU$ と分解した時の下三角行列 U^T (注意事項 (b) 参照)
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
4	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
5	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	b[0] ← b[0]/a[0] とする.
2100	係数行列 A の $\mathrm{U^TDU}$ 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の $\mathrm{U^TDU}$ 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて < 基本機能第 2 分冊 > 2.8.3 $\left\{\begin{matrix}\text{ASL_dbsnls}\\ \text{ASL_rbsnls}\end{matrix}\right\}$を配列 b の内容のみを変えて使用すればよい. このようにすれば行列 A の $\mathrm{U^TDU}$ 分解が一度だけしか行われないため, 演算効率よく解が求まる.

(b) 配列 a には, 下三角行列 U^T のみが格納される. U^T の対角成分はその逆数が符号をかえて格納される. 対角行列 D, および上三角行列 U は U^T より算出されるので, 配列 a には格納されない. 行列 U は行列 U^T の転置行列であり, 行列 D は行列 U^T の対角要素の逆数を成分とする対角行列である. この関数は配列 a の下三角部分のみを使用する.

図 3–6　行列 U^T の格納状態と行列 D の内容

行列 U^T

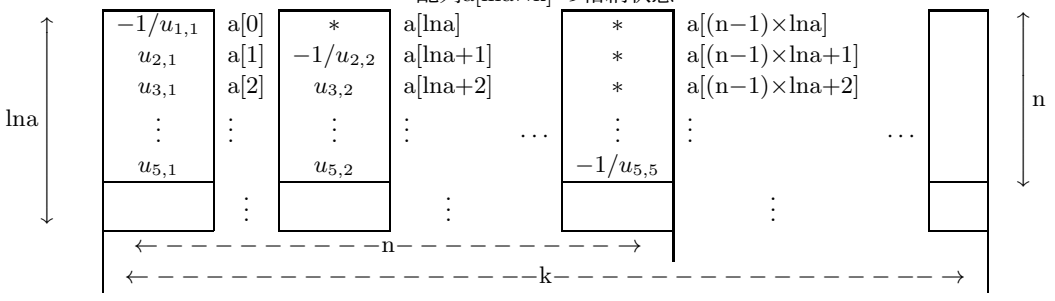
$$\begin{bmatrix} u_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ u_{2,1} & u_{2,2} & 0.0 & \cdots & 0.0 \\ u_{3,1} & u_{3,2} & u_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{5,1} & u_{5,2} & u_{5,3} & \cdots & u_{5,5} \end{bmatrix}$$

行列 D

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

$\Downarrow$

配列a[lna×k] の格納状態



備　考

a. lna ≥ n, n ≤ k を満たさなければならない.

b. ∗ に対応する入力時の値は保証されない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 4 \\ -4 \end{bmatrix}$$

を解く.

(b) 入力データ

係数行列 A, lna=11, n = 4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*    C interface example for ASL_qbsnsl */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int ierr;

    int i,j;
    int nt = 2;

    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

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

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

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


    fclose( fp );


    ierr = ASL_qbsnsl(a, na, n, b, nt);
```

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


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

    free( a );
    free( b );


    return 0;
}
```

(d) 出力結果

```
   *** ASL_qbsnsl ***

  ** Input **

 n =      4

Coefficient Matrix

       5         4         1         1
       4         5         1         1
       1         1         4         2
       1         1         2         4

Constant Vector

       1
      -1
       4
      -4

  ** Output **

ierr =      0

Solution

  x[     0] =         1
  x[     1] =        -1
  x[     2] =         2
  x[     3] =        -2
```

3.6.2 ASL_qbsnud, ASL_pbsnud
実対称行列の U^TDU 分解 (軸選択なし)

(1) **機 能**

実対称行列 A(2 次元配列型)(下三角型) を修正コレスキー法を用いて U^TDU 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qbsnud  (a, lna, n, nt);
```

単精度関数:

```
ierr = ASL_pbsnud  (a, lna, n, nt);
```

(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∗ }	lna×n	入 力	実対称行列 A(2 次元配列型)(下三角型)
				出 力	$A = U^TDU$ と分解した時の下三角行列 U^T (注意事項 (a) 参照)
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
4	nt	I	1	入 力	生成するタスク数
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) $0 < n \leq lna$

(b) $nt \geq 1$

(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.	配列 a の内容は変更されない.
2100	係数行列 A の U^TDU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の U^TDU 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

(6) **注意事項**

(a) 配列 a には, 下三角行列 U^T のみが格納される. U^T の対角成分はその逆数が符号をかえて格納される. 対角行列 D, および上三角行列 U は U^T より算出されるので, 配列 a には格納されない (3.6.1 図 3−6 参照).

3.7 エルミート行列 (2 次元配列型) (上三角型) (実数引数型)

3.7.1 ASL_hbhpsl, ASL_gbhpsl
連立 1 次方程式 (エルミート行列)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_hbhpsl (ar, ai, lna, n, br, bi, ipvt, w1, nt);

単精度関数:

ierr = ASL_gbhpsl (ar, ai, lna, n, br, bi, ipvt, w1, nt);

(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	ar	{D∗ / R∗}	lna×n	入 力	係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解した時の, 上三角行列 L^* の実部 (注意事項 (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解した時の, 上三角行列 L^* の虚部 (注意事項 (b) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	br	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の実部
				出 力	解 $\boldsymbol{x}$ の実部
6	bi	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の虚部
				出 力	解 $\boldsymbol{x}$ の虚部
7	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 (列)i と交換した行 (列) の番号 (注意事項 (c) 参照)
8	w1	{D∗ / R∗}	n	ワーク	作業領域
9	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(b) nt ≥ 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.	配列 ar, ai の内容は変更されない. b[0] $\leftarrow$ b[0]/ar[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

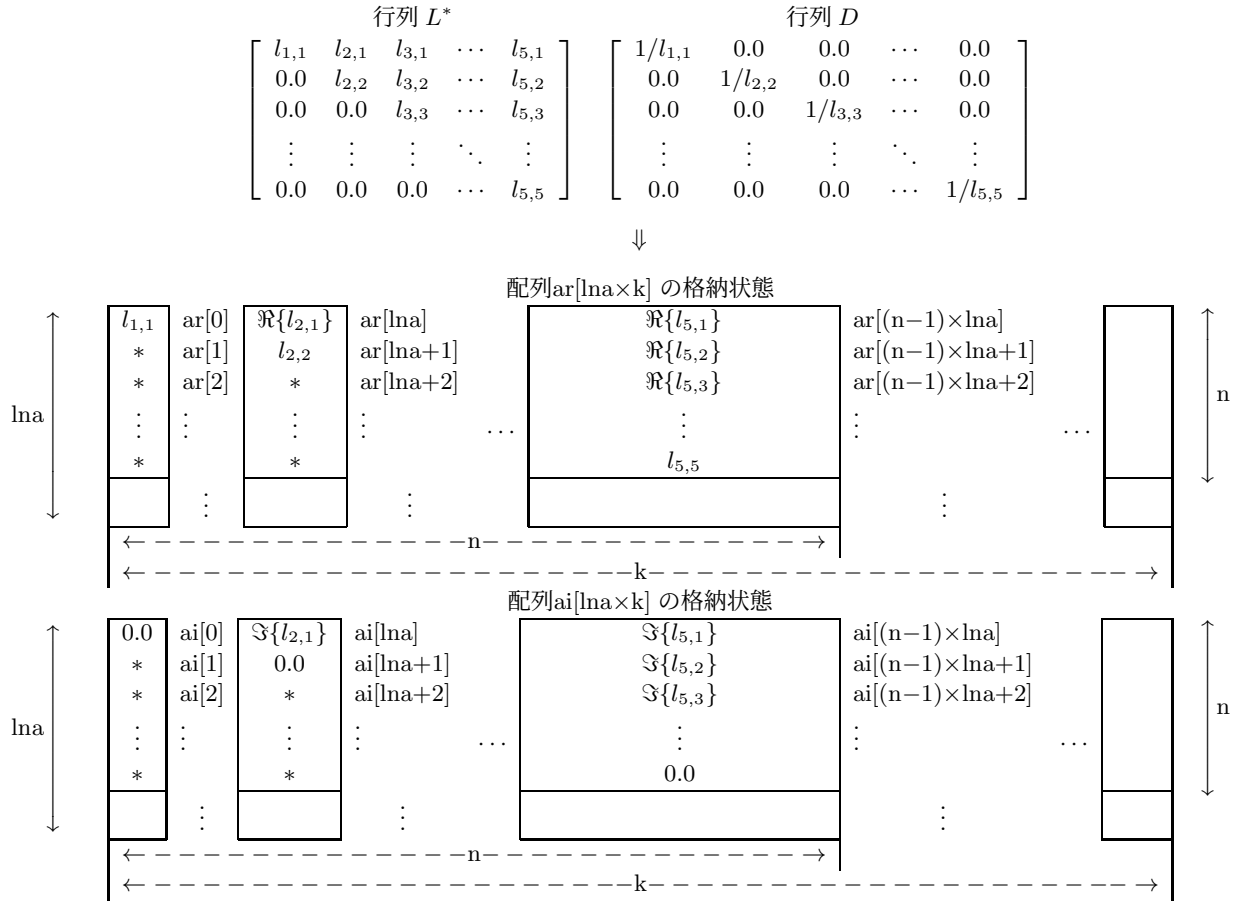
(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて関数 < 基本機能第 2 分冊 > 2.9.4 $\left\{\begin{matrix}\text{ASL\_zbhpls}\\ \text{ASL\_cbhpls}\end{matrix}\right\}$ を配列 br, bi の内容のみを変えて使用すればよい. このようにすれば行列 A の LDL* 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) 配列 ar, ai の上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されない. 行列 L は行列 L^* の随伴行列であり, 行列 D は行列 L^* の対角要素の逆数を成分とする対角行列である. この関数は配列 ar, ai の上三角部分のみを使用する. (図 3−7 参照)

(c) この関数では, 係数行列 A の LDL* 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)(i $\leq$ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

図 3–7　行列 L^* の格納状態と行列 D の内容



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

b.　∗ に対応する入力時の値は保証されない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

を解く.ただし, $i = \sqrt{-1}$.

(b) 入力データ

係数行列の実部 ar および虚部 ai, lna = 11, n = 4, 定数ベクトル***b***

(c) 主プログラム

```
/*      C interface example for ASL_hbhpsl */

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

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    int *ipvt;
    double *w1;
    int nt = 2;
```

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

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

printf( "    *** ASL_hbhpsl ***\n" );
printf( "\n     ** Input **\n\n" );
fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );

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

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

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

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

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

w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

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

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

```
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
    }

    fclose( fp );


    ierr = ASL_hbhpsl(ar, ai, na, n, br, bi, ipvt, w1, nt);


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

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

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( ipvt );
    free( w1 );


    return 0;
}
```

(d) 出力結果

```
*** ASL_hbhpsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector  (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 ,        0)
  x[     2] = (-6.63e-17 ,        1)
  x[     3] = (2.09e-17 ,        1)
```

3.7.2 ASL_hbhpud, ASL_gbhpud
エルミート行列の LDL* 分解

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を修正コレスキー法を用いて LDL* 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hbhpud  (ar, ai, lna, n, ipvt, w1, nt);
```

単精度関数:

```
ierr = ASL_gbhpud  (ar, ai, lna, n, ipvt, w1, nt);
```

(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	ar	{D*, R*}	lna×n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
				出 力	$A = LDL^*$ と分解した時の上三角行列 L^* の実部 (注意事項 (a) 参照)
2	ai	{D*, R*}	lna×n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
				出 力	$A = LDL^*$ と分解した時の上三角行列 L^* の虚部 (注意事項 (a) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	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 (列)i と交換した行 (列) の番号 (注意事項 (b) 参照)
6	w1	{D*, R*}	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 ar, ai の内容は変更されない.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0 になった. 行列 A は, 特異である.	

(6) **注意事項**

(a) 配列 ar, ai には, 上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されない. この関数は配列 ar, ai の上三角部分のみを使用する (3.7.1 図 3−7 参照).

(b) この関数では, 係数行列 A の LDL* 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)(i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

3.8 エルミート行列 (2 次元配列型) (上三角型) (実数引数型) (軸選択なし)

3.8.1 ASL_hbhrsl, ASL_gbhrsl
連立 1 次方程式 (エルミート行列) (軸選択なし)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_hbhrsl (ar, ai, lna, n, br, bi, w1, nt);

単精度関数:

ierr = ASL_gbhrsl (ar, ai, lna, n, br, bi, w1, nt);

(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	ar	{D∗ / R∗}	lna×n	入 力	係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解した時の, 上三角行列 L^* の実部 (注意事項 (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解した時の, 上三角行列 L^* の虚部 (注意事項 (b) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	br	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の実部
				出 力	解 $\boldsymbol{x}$ の実部
6	bi	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の虚部
				出 力	解 $\boldsymbol{x}$ の虚部
7	w1	{D∗ / R∗}	n	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 ar, ai の内容は変更されない. b[0] ← b[0]/ar[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて関数 < 基本機能第 2 分冊 > 2.10.4 $\left\{\begin{matrix}\mathrm{ASL\_zbhrls}\\ \mathrm{ASL\_cbhrls}\end{matrix}\right\}$ を配列 br, bi の内容のみを変えて使用すればよい. このようにすれば行列 A の LDL* 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) 配列 ar, ai の上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されない. 行列 L は行列 L^* の随伴行列であり, 行列 D は行列 L^* の対角要素の逆数を成分とする対角行列である. この関数は配列 ar, ai の上三角部分のみを使用する. (図 3−8 参照)

図 3−8　行列 L^* の格納状態と行列 D の内容

行列 L^*

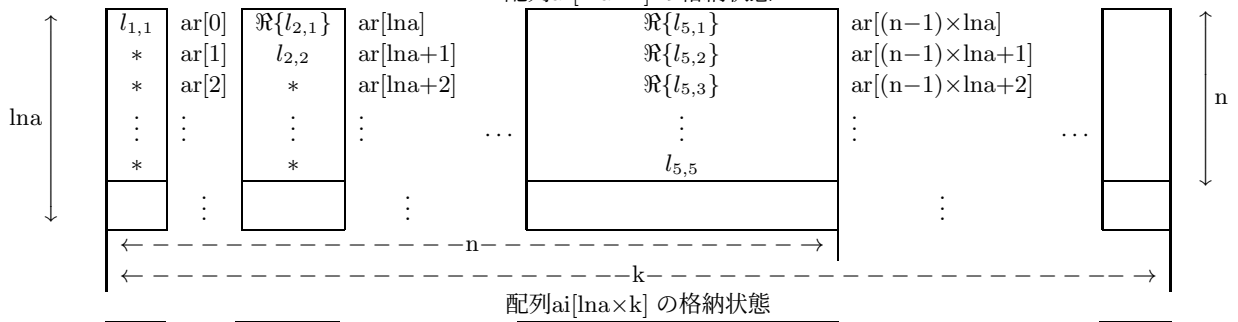
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

行列 D

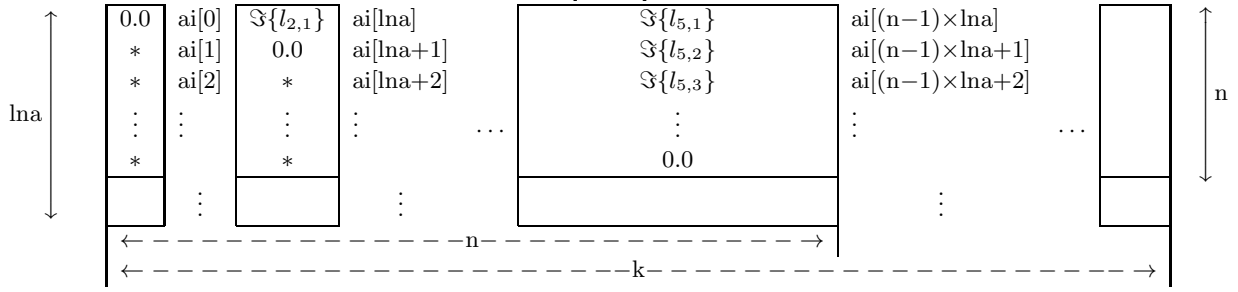
$$\begin{bmatrix} 1/l_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ 0.0 & 1/l_{2,2} & 0.0 & \cdots & 0.0 \\ 0.0 & 0.0 & 1/l_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 1/l_{5,5} \end{bmatrix}$$

$\Downarrow$

配列ar[lna×k] の格納状態



配列ai[lna×k] の格納状態



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

b.　∗ に対応する入力時の値は保証されない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$

を解く.ただし, $i = \sqrt{-1}$.

(b) 入力データ

係数行列の実部 ar および虚部 ai, lna = 11, n = 4, 定数ベクトル***b***

(c) 主プログラム

```
/*      C interface example for ASL_hbhrsl */

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

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    double *w1;
    int nt = 2;
    int ierr;
    int i,j;
    FILE *fp;
```

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

printf( "    *** ASL_hbhrsl ***\n" );
printf( "\n     ** Input **\n\n" );
fscanf( fp, "%d", &na );
fscanf( fp, "%d", &n );

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

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

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

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

w1 = ( double * )malloc((size_t)( sizeof(double) * (n*2) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

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

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

for( i=0 ; i<n ; i++ )
{
    printf( "\t(%8.3g , %8.3g) \n", br[i],bi[i] );
}

fclose( fp );
```

```
    ierr = ASL_hbhrsl(ar, ai, na, n, br, bi, w1, nt);


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

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( w1 );


    return 0;
}
```

(d) 出力結果

```
*** ASL_hbhrsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector  (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution   (Real, Imaginary)

  x[     0] = (       1 , 1.48e-16)
  x[     1] = (       1 , -3.9e-17)
  x[     2] = (-1.02e-16 ,        1)
  x[     3] = (8.34e-17 ,        1)
```

3.8.2 ASL_hbhrud, ASL_gbhrud
エルミート行列の LDL* 分解 (軸選択なし)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を修正コレスキー法を用いて LDL* 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hbhrud  (ar, ai, lna, n, w1, nt);
```

単精度関数:

```
ierr = ASL_gbhrud  (ar, ai, lna, n, w1, nt);
```

(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	ar	{D*, R*}	lna×n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
				出 力	$A = LDL^*$ と分解した時の上三角行列 L^* の実部 (注意事項 (a) 参照)
2	ai	{D*, R*}	lna×n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
				出 力	$A = LDL^*$ と分解した時の上三角行列 L^* の虚部 (注意事項 (a) 参照)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	w1	{D*, R*}	n	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 ar, ai の内容は変更されない.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 配列 ar, ai には, 上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されない. この関数は配列 ar, ai の上三角部分のみを使用する (3.8.1 図 3−8 参照).

3.9 エルミート行列 (2 次元配列型) (上三角型) (複素引数型)

3.9.1 ASL_hbhfsl, ASL_gbhfsl
連立 1 次方程式 (エルミート行列)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_hbhfsl (a, lna, n, b, ipvt, w1, nt);

単精度関数:

ierr = ASL_gbhfsl (a, lna, n, b, ipvt, w1, nt);

(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	{Z∗ / C∗}	lna × n	入 力	係数行列 A(エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解したときの, 上三角行列 L^* (注意事項 (b) 参照)
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
4	b	{Z∗ / C∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
5	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 (列)i と交換した行 (列) の番号 (注意事項 (c) 参照)
6	w1	{D∗ / R∗}	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない. b[0] ← b[0]/a[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

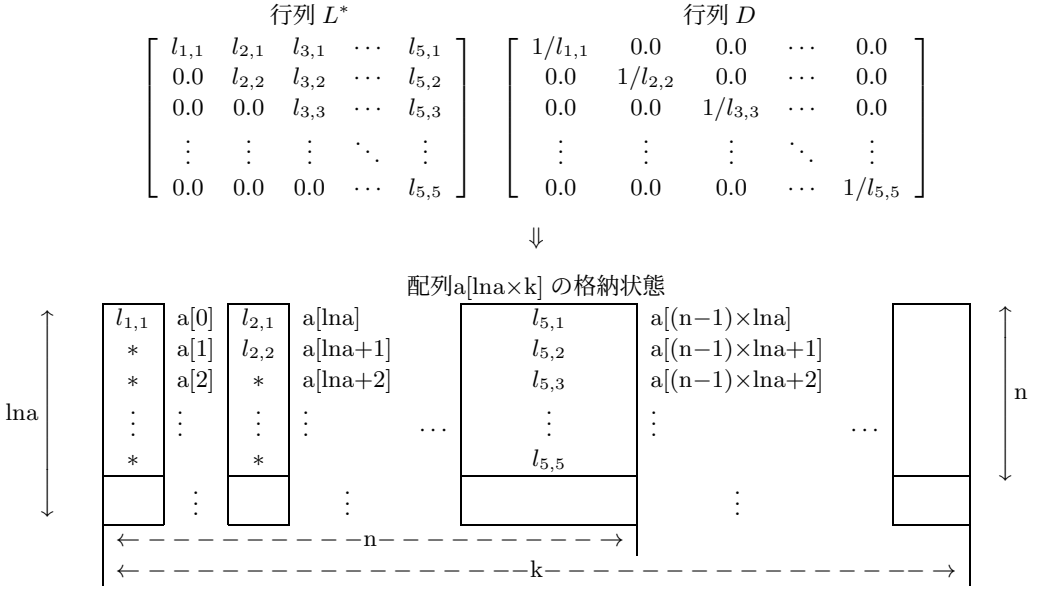
(6) **注意事項**

(a) 定数ベクトル$\boldsymbol{b}$のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて < 基本機能第 2 分冊 > 2.11.4 $\left\{\begin{matrix}\text{ASL_zbhfls}\\ \text{ASL_cbhfls}\end{matrix}\right\}$を配列 b の内容のみを変えて使用すればよい. このようにすれば行列 A の LDL* 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) 配列 a の上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 a には格納されない. 行列 L は行列 L^* の随伴行列であり, 行列 D は行列 L^* の対角要素の逆数を成分とする対角行列である (図 3−9 参照).

(c) この関数では, 係数行列 A の U*DU 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)(i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

図 3−9　行列 L^* の格納状態と行列 D の内容



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

b.　∗ に対応する入力時の値は保証されない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$ を解く.

(b) 入力データ

係数行列 A, lna $= 11$, n $= 4$, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_hbhfsl */

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

int main()
{
    double _Complex *a;
    int na;
    int n;
    double _Complex *b;
    int *ipvt;
    double *w1;
    int nt=2;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
    if( a == NULL )
    {
```

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

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

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

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

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

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

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    double tmp_re;
    fscanf( fp, "%lf", &tmp_re );
    b[i] = tmp_re;
}
for( i=0 ; i<n ; i++ )
{
    double tmp_im;
    fscanf( fp, "%lf", &tmp_im );
    b[i] = b[i] + tmp_im * _Complex_I;
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
}
fclose( fp );

ierr = ASL_hbhfsl(a, na, n, b, ipvt, w1, nt);

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

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

free( a );
free( b );
free( ipvt );
free( w1 );
```

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_hbhfsl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

  x[     0] = (       1 ,        0)
  x[     1] = (       1 ,        0)
  x[     2] = (-6.63e-17 ,        1)
  x[     3] = (2.09e-17 ,        1)
```

3.9.2 ASL_hbhfud, ASL_gbhfud
エルミート行列の LDL* 分解

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を修正コレスキー法を用いて LDL* 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hbhfud  (a, lna, n, ipvt, w1, nt);
```

単精度関数:

```
ierr = ASL_gbhfud  (a, lna, n, ipvt, w1, nt);
```

(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	{Z*, C*}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	$A = LDL^*$ と分解したときの, 上三角行列 L^* (注意事項 (a) 参照)
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
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 (列)i と交換した行 (列) の番号 (注意事項 (b) 参照)
5	w1	{D*, R*}	n	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 配列 a には, 上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 a には格納されない. この関数は配列 a の上三角部分のみを使用する (3.9.1 図 3−9 参照).

(b) この関数では, 係数行列 A の U*DU 分解時に, 部分軸選択 (partial pivoting) が行われている. 部分軸選択は行と列について対称に行われる. 第 i 段目のピボット行 (列) が第 j 行 (列)(i ≤ j) となった場合, ipvt[i − 1] に j が格納される. また, このとき, 行列 A の第 i 行 (列) と第 j 行 (列) の対応する列 (行) 要素のうち, 第 i 列 (行) から第 n 列 (行) までの要素が実際に交換される.

3.10 エルミート行列 (2 次元配列型) (上三角型) (複素引数型) (軸選択なし)

3.10.1 ASL_hbhesl, ASL_gbhesl
連立 1 次方程式 (エルミート行列) (軸選択なし)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を修正コレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_hbhesl (a, lna, n, b, nt);

単精度関数:

ierr = ASL_gbhesl (a, lna, n, b, nt);

(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	{Z∗ / C∗}	lna × n	入 力	係数行列 A(エルミート行列, 2 次元配列型, 上三角型)
				出 力	$A = LDL^*$ と分解したときの, 上三角行列 L^* (注意事項 (b) 参照)
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
4	b	{Z∗ / C∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
5	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない. b[0] ← b[0]/a[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 定数ベクトル***b***のみが異なる複数の連立 1 次方程式を解く場合には, この関数を一度使用した後, 続けて < 基本機能第 2 分冊 > 2.12.4 $\left\{\begin{matrix}\text{ASL_zbhels}\\ \text{ASL_cbhels}\end{matrix}\right\}$を配列 b の内容のみを変えて使用すればよい. このようにすれば行列 A の LDL* 分解が一度だけしか行われないため, 効率よく解が求まる.

(b) 配列 a の上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 a には格納されない. 行列 L は行列 L^* の随伴行列であり, 行列 D は行列 L^* の対角要素の逆数を成分とする対角行列である.

図 3−10　行列 L^* の格納状態と行列 D の内容

行列 L^*

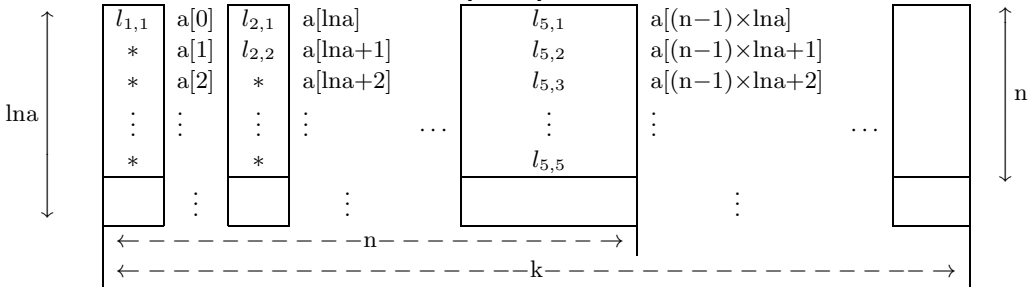
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

行列 D

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

⇓

配列a[lna×k] の格納状態



備　考

a.　lna ≥ n, n ≤ k を満たさなければならない.

b.　＊に対応する入力時の値は保証されない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+i \\ 1-i & 2-4i & 5-i & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10+6i \\ 11+2i \\ 4+6i \\ 4+6i \end{bmatrix}$$ を解く.

(b) 入力データ

係数行列 A, lna $= 11$, n $= 4$, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_hbhesl */

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

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

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

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

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

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

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

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

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "                      " );
        }
        for( j=i ; j<n ; j++ )
        {
```

```
            printf( "(%8.3g , %8.3g) ", creal(a[i+na*j]),cimag(a[i+na*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        double tmp_re;
        fscanf( fp, "%lf", &tmp_re );
        b[i] = tmp_re;
    }
    for( i=0 ; i<n ; i++ )
    {
        double tmp_im;
        fscanf( fp, "%lf", &tmp_im );
        b[i] = b[i] + tmp_im * _Complex_I;
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t(%8.3g , %8.3g)\n", creal(b[i]),cimag(b[i]) );
    }
    fclose( fp );


    ierr = ASL_hbhesl(a, na, n, b, nt);


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

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

    free( a );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_hbhesl ***

 ** Input **

 n =      4

Coefficient Matrix  (Real, Imaginary)

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

Constant Vector (Real, Imaginary)

(      10 ,        6)
(      11 ,        2)
(       4 ,        6)
(       4 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

  x[     0] = (       1 , -9.87e-17)
  x[     1] = (       1 , -6.25e-17)
  x[     2] = (5.11e-17 ,        1)
  x[     3] = (       0 ,        1)
```

3.10.2 ASL_hbheud, ASL_gbheud
エルミート行列の LDL* 分解 (軸選択なし)

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型) を修正コレスキー法を用いて LDL* 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hbheud  (a, lna, n, nt);
```

単精度関数:

```
ierr = ASL_gbheud  (a, lna, n, nt);
```

(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	{Z*, C*}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	$A = LDL^*$ と分解したときの, 上三角行列 L^* (注意事項 (a) 参照)
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
4	nt	I	1	入 力	生成するタスク数
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) $0 < \mathrm{n} \leq \mathrm{lna}$

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

(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.	配列 a の内容は変更されない.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	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 (b) を満足しなかった.	
4000+i	i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 配列 a には, 上三角部分に上三角行列 L^* が格納される. 対角行列 D, および下三角行列 L は L^* より算出されるので, 配列 a には格納されない. この関数は配列 a の上三角部分のみを使用する (3.10.1 図 3−10 参照).

第 4 章　連立 1 次方程式 (反復法)

4.1　概 要

本章では, 大次元スパース行列を係数行列とする連立 1 次方程式を反復法により解く関数について説明する.

本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.

本ライブラリで用意されている関数には, CG 法や GMRES 法などの非定常反復解法を基礎反復法として用いたものがある.

本ライブラリの非定常反復解法関数としては, 正値対称連立 1 次方程式 (正値対称行列を係数とする連立 1 次方程式) を CG 法 (共役勾配法) を用いて解くための関数と, 非対称連立 1 次方程式 (非対称行列を係数とする連立 1 次方程式) を CGS 法 (自乗共役勾配法), BiCGSTAB 法 (双共役勾配安定法) または GMRES 法を用いて解くための関数が用意されている.

また収束を加速するための前処理手法としては, スケーリング法が用意されている.

係数行列データの格納形式は, 不規則スパース行列用に ELLPACK 型の形式が用意されている.

これらの基礎反復法関数では, 関数の引数並びが統一されている. したがって同一の前処理手法および同一の係数行列データ格納形式のもとで各基礎反復法関数の性能比較を行いたいときは, 利用者は基礎反復法関数名を変更しながら計算を実行すればよい.

表 4–1　各反復解法と対応する関数

基礎反復法	関数名
CG 法 (共役勾配法)(対称行列のみ) Conjugate Gradient method	ASL_qxe010 ASL_pxe010
CGS 法 (自乗共役勾配法) Conjugate Gradient Squared method	ASL_qxe020 ASL_pxe020
BiCGSTAB 法 (双共役勾配安定法) Bi-Conjugate Gradient Stabilized method	ASL_qxe030 ASL_pxe030
GMRES 法 General Minimal Residual method	ASL_qxe040 ASL_pxe040

4.1.1 使用上の注意

(1) 反復法の使い分け

CG 法の関数は, 主に拡散方程式の有限差分近似・有限要素近似で生じる正値対称な大規模スパース連立 1 次方程式の求解を目的としたものであり, CGS 法・BiCGSTAB 法の関数は, 主に移流拡散方程式の有限差分近似・有限要素近似で生じる非対称な大規模スパース連立 1 次方程式の求解を目的としたものである. 反復法は直接法に比べて必要な記憶領域が小さく, 従って大規模な問題の解法に適しているが, 真の解に収束しないというおそれがある. そこで実用に際しては, 最大反復回数を少なめにする, あるいは問題のサイズを小さくするなどの試用を行ってみることが望ましい.

とくに大規模非対称連立 1 次方程式の求解は一般には極めて困難であり, 本章で採用されている反復法アルゴリズムによっても常に解が得られるという保証はないので, やはり関数の組み込みに先立っての試用が望ましい.

(2) 行列格納形式

2 次元矩形領域や 3 次元直方体領域を有限差分法で離散化した場合などに生じる連立 1 次方程式の係数行列は非零要素が対角方向に直線状に並ぶ. このような行列を規則 (スパース) 行列と呼ぶ.

一方, 不規則な形状の領域を有限要素法で離散化した場合に生じる連立 1 次方程式の係数行列には, 対角要素が非零要素である以外には, 一般に何の規則性もない. このような行列を不規則 (スパース) 行列と呼ぶ.

本章の関数では不規則スパース行列用の格納形式をサポートしている.

各関数は引数並びが統一されている (ただし, GMRES 法 関数には, 入力引数 (gmitr) の追加が必要). したがって, 同じ入力値を用いて別の基礎反復法アルゴリズムを試してみるには, プログラム中で呼び出す関数名を付けかえて (GMRES 法 を試す場合には, 関数名を付けかえると同時に入力引数 (gmitr) を追加して), 実行するだけでよい. ただし基礎反復法を変更する場合には次のような点に注意しなければならない.

CG 法関数は正値対称連立 1 次方程式を解くためのものであり, 係数行列が非対称である問題に対しては適用できない.

(3) 前処理手法

前処理は基礎反復法の収束性を上げるための手法である. これは連立 1 次方程式 $A\boldsymbol{u} = \boldsymbol{b}$ を行列反転が容易な A の近似行列 M を用いて等価で解きやすい方程式に変換した後, この方程式に基礎反復法の手順を適用するものである. 本章で扱う関数では, スケーリング前処理を行っている.

通常の問題の場合には, スケーリング前処理が最も効果的である. それはスケーリング前処理での反復計算はベクトル長が長く, Vector Engine の多重並列パイプラインを効率よく運用でき, また必要とする記憶領域も少ないからである.

(4) 反復の終了条件

反復計算は,

(a) 与えられた打ち切り残差ノルム以下になった

(b) 与えられた許容反復回数に達した

(c) エラーが検出され, それ以上計算できなくなった

場合に打ち切られる. ここでは, 打ち切り残差ノルムについて説明する.

方程式 $A\boldsymbol{u} = \boldsymbol{b}$ に対して, 反復計算により近似値 $\boldsymbol{u}^*$ が得られたとする. このとき

$$\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}^*$$

を残差ベクトルという. 本章の関数では, 残差ノルムとは, この残差ベクトルの通常の相対ノルム

$$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}$$

であり, それぞれのノルムは L^2 ノルム

$$\|\boldsymbol{r}\|_2 = \left(\sum_{i=1}^{n} {r_i}^2 \right)^{1/2}$$

である.

(5) 必要とする記憶領域

本章の関数は, 方程式の係数行列を格納するための領域, 解ベクトルや右辺ベクトルを格納するための領域, さらに作業領域を必要とする.

これらの領域の大きさは基礎反復法としてどれを選ぶかで異なっている.

4.1.2 使用しているアルゴリズム

4.1.2.1 非定常反復解法 (対称係数行列用)

(1) **CG 法**

n 元連立 1 次方程式

$$A\boldsymbol{u} = \boldsymbol{b} \qquad (1)$$

を考える. CG 法は係数行列 A が正定値対称な場合に方程式 (1) を解く反復法で, 以下のように表される.

$\boldsymbol{u} = \boldsymbol{u}_1$ $\quad$ ($\boldsymbol{u}_1$は初期ベクトル),
$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1$,
$\boldsymbol{p}_1 = \boldsymbol{r}_1$.
for $i = 1, 2, \cdots, n$
$\quad a_i = \dfrac{(\boldsymbol{r}_i, \boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)}$,
$\quad \boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i$,
$\quad \boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_iA\boldsymbol{p}_i$,
$\quad b_i = \dfrac{(\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, \boldsymbol{r}_i)}$,
$\quad \boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i$.

CG 法の大きな特徴として, 反復法でありながら理論的には元数 n 回の反復計算で厳密解が得られることがあげられる.

4.1.2.2 非定常反復解法 (非対称係数行列用)

(1) **CGS 法と BiCGSTAB 法**

CGS 法と BiCGSTAB 法を説明するために, まず BiCG 法について説明する. BiCG 法は式 (1) に対して, それと双対な方程式を組み合わせた $2n$ 元連立 1 次方程式

$$\tilde{A}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{b}},$$
$$\tilde{A} = \begin{bmatrix} A & O \\ O & A^T \end{bmatrix}, \tilde{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{u}^* \end{bmatrix}, \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}^* \end{bmatrix}$$

を考え,

$$F(\tilde{\boldsymbol{u}}) =< (\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}), \quad A(\tilde{\boldsymbol{u}} - \hat{\boldsymbol{u}}) > H \quad (\hat{\boldsymbol{u}}; \text{真解})$$

の停留値を共役勾配法で求めるものである. ただし,

$$< \boldsymbol{u}, \boldsymbol{v} > H \equiv (\boldsymbol{u}, H\boldsymbol{v}) = (H\boldsymbol{u}, \boldsymbol{v}), \quad H = \begin{bmatrix} O & I \\ I & O \end{bmatrix}.$$

k 回反復後の残差ベクトル $\boldsymbol{r}_k$ と方向ベクトル $\boldsymbol{p}_k$ は初期残差ベクトル $\boldsymbol{r}_0$ を用いて

$$\begin{aligned} \boldsymbol{r}_k &= R_k(A)\boldsymbol{r}_0, \\ \boldsymbol{p}_k &= P_k(A)\boldsymbol{r}_0. \end{aligned}$$

と表される. ここで $R_k(A), P_k(A)$ は A から生成される多項式である.
このとき CGS 法では, この $R_k(A), P_k(A)$ を用いて新しいベクトル $\boldsymbol{r}'_k, \boldsymbol{p}'_k$ を次のように計算する.

$$\begin{aligned} \boldsymbol{r}'_k &= R_k^2(A)\boldsymbol{r}_0, \\ \boldsymbol{p}'_k &= P_k^2(A)\boldsymbol{r}_0 \end{aligned}$$

この算法では

$$\|\boldsymbol{r}_k\| = \|R_k(A)\boldsymbol{r}_0\|$$

が小さくなったとき,

$$\|\boldsymbol{r}'_k\| = \|R_k^2(A)\boldsymbol{r}_0\|$$

はさらに小さくなることが期待され, BiCG 法よりも高い収束性が期待される.
このように残差が $\boldsymbol{r}'_k$ となるように反復をすすめる方法が CGS 法である.
しかし CGS 法はきわめて不規則な収束性をしめす場合があり, 特に反復の初期値が解に近い場合など反復の初期に残差が著しく増大し, このため丸め誤差の影響により正確な反復計算が行えないことがある. BiCGSTAB 法では,

$$Q_k(A) = (1 - \omega_1 A)(1 - \omega_2 A) \cdots (1 - \omega_k A)$$

とおき, 各反復において残差

$$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{u}_k$$

が

$$\boldsymbol{r}_k = Q_k(A)R_k(A)\boldsymbol{r}_0$$

で与えられるように近似解 $\boldsymbol{u}_k$ を求める. ここでパラメータ ω_k は残差がなるべく安定して減少するように $(\boldsymbol{r}_k, \boldsymbol{r}_k)$ が最も小さくなるものを選ぶ.

(2) **GMRES 法 (GMRES(m) 法)**
GMRES 法は, j ステップ目の初期近似解ベクトルが $\boldsymbol{u}_0^{(j)}$, 初期残差ベクトルが

$$\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)}$$

であるとき,

$$\boldsymbol{r}_0, \quad A\boldsymbol{r}_0, \quad A^2\boldsymbol{r}_0, \quad \cdots, \quad A^i\boldsymbol{r}_0 \quad (i = 1, 2, \cdots, m)$$

で張られる空間内から

$$\boldsymbol{b} - A(\boldsymbol{u}_0^{(j)} + \boldsymbol{z}^{(j)}) = \boldsymbol{r}^{(j)} - A\boldsymbol{z}^{(j)}$$

の L^2 ノルムを最小にするようなベクトル $\boldsymbol{z}^{(j)}$ を $j = 1, 2, \cdots$ について探していく反復法である.

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0$.
for $j = 1, 2, \cdots$
  $\boldsymbol{r}^{(j)} = \boldsymbol{b} - A\boldsymbol{u}_0^{(j)}$,
  $\boldsymbol{v}_1 = \boldsymbol{r}^{(j)} / \|\boldsymbol{r}^{(j)}\|_2$.
  for $i = 1, 2, \cdots m$
    $\boldsymbol{w} = A\boldsymbol{v}_i$.
    for $k = 1, 2, \cdots i$
      $\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k$,
    $\boldsymbol{v}_{i+1} = \boldsymbol{w} / \|\boldsymbol{w}\|_2$,
    $\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ が最小になるように
      $\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$ の係数 $y_1, y_2, \cdots, y_i$ を選ぶ.
  $\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}$.

このアルゴリズムに示されているように, GMRES 法では外側の反復計算 (j に関するループ部分) と, 内側の反復計算 (i に関するループ部分) とから構成される. GMRES 法は, 内部反復数の上限値を表すパラメータ m に依存する反復法なので厳密には GMRES(m) 法と言い表される.
なお, ASL では GMRES(m) 法における反復回数を (内部反復数 i) + (外部反復数 j) $\times$ m で数えることにする.

4.1.2.3 前処理付き反復法

反復法の実際の計算では誤差の混入のために収束特性が悪くなる. また偏微分方程式の差分スキームにおいて, 差分メッシュに疎密がある, 係数 (拡散, 熱伝導性) の値の空間的な変動が大きい, 差分メッシュ幅や上記の係数に異方性があるなどの場合, 行列 A の固有値が分散してしまうが, こういった状況では反復法は収束しにくくなる. 逆に, 固有値が密集していると収束が速いことが知られている.
前処理付き反復法とは, 上記の難点を克服するために, 元の行列に前処理を施して条件のよい行列に変換してから反復法を行うことにより収束性を向上させる手法であり, 以下のように言い表すことができる. いま, $A = (a_{ij})$ を何らかの意味で近似する行列 $M = M_1 M_2$ が与えられたとする.

$$A \sim M = M_1 M_2$$

このとき, 式 (1) と等価な方程式

$$M_1^{-1} A M_2^{-1} \boldsymbol{u}' = \boldsymbol{b}' \qquad (2)$$
$$\left(\quad \boldsymbol{u}' = M_2 \boldsymbol{u} \quad \right)$$
$$\left(\quad \boldsymbol{b}' = M_1^{-1} \boldsymbol{b} \quad \right)$$

をつくると, この方程式の係数行列はもとの行列に較べ単位行列に近いので解き易くなる.
式 (2) を前処理方程式, 行列 M を前処理行列と呼ぶ.
式 (1) に対する前処理付き反復法は式 (2) に対する (基礎) 反復法として定義・導出される.
前処理付き反復法では M_1 や M_2, あるいは M に関する反転 (逆行列による列ベクトルへの左作用) を実行する必要がある. これが前処理演算 (または単に前処理) と呼ばれるものである.
通常は前処理行列 $M = M_1 M_2$ として, 前処理演算が容易に行えるものが使用される. 前処理行列 M が元の行列 A を近似する度合が高いほど, 前処理付き反復法の収束特性は向上するが, 一般に収束特性がよい前処理行列ほど, 一回の前処理演算に手間がかかる傾向がある.
ASL で用意されている前処理手法のアルゴリズムについては 4.1.2.4 で具体的に説明する.
以下に 前処理付き CG 法 (PCG 法), 前処理付き CGS 法 (PCGS 法), 前処理付き BiCGSTAB 法 (PBiCGSTAB 法), および 前処理付き GMRES(m) 法 (PGMRES(m) 法) のアルゴリズムを示す.

(1) **PCG 法**

$\boldsymbol{u} = \boldsymbol{u}_1$　($\boldsymbol{u}_1$は初期ベクトル),
$\boldsymbol{r}_1 = \boldsymbol{b} - A\boldsymbol{u}_1$,
$\boldsymbol{p}_1 = M^{-1}\boldsymbol{r}_1$.
for　$i = 1, 2, \cdots, n$
$a_i = \dfrac{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)}{(\boldsymbol{p}_i, A\boldsymbol{p}_i)}$,
$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + a_i\boldsymbol{p}_i$,
$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - a_iA\boldsymbol{p}_i$,
$b_i = \dfrac{(\boldsymbol{r}_{i+1}, M^{-1}\boldsymbol{r}_{i+1})}{(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)}$,
$\boldsymbol{p}_i = M^{-1}\boldsymbol{r}_{i+1} + b_i\boldsymbol{p}_i$.

(2) **PCGS 法** (文献 (1))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0$,
$\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$,
$\boldsymbol{e}_0 = \boldsymbol{r}_0$.
for　$k = 0, 1, 2, \cdots$
$\alpha_{k+1} = \dfrac{(\boldsymbol{r}_0, \boldsymbol{r}_k)}{(\boldsymbol{r}_0, A\boldsymbol{p}_k)}$,
$\boldsymbol{h}_{k+1} = \boldsymbol{e}_k - \alpha_{k+1}A\boldsymbol{p}_k$,
$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha_{k+1}M^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1})$,
$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_{k+1}AM^{-1}(\boldsymbol{e}_k + \boldsymbol{h}_{k+1})$,
$\beta_{k+1} = \dfrac{(\boldsymbol{r}_0, \boldsymbol{r}_{k+1})}{(\boldsymbol{r}_0, \boldsymbol{r}_k)}$,
$\boldsymbol{e}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1}$,
$\boldsymbol{p}_{k+1} = M^{-1}(\boldsymbol{e}_{k+1} + \beta_{k+1}\boldsymbol{h}_{k+1}) + \beta_{k+1}^2\boldsymbol{p}_k$.

(3) **PBiCGSTAB 法** (文献 (2))

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{u}_0$,
for　$k = 0, 1, 2, \cdots$
$\rho_{k+1} = (\boldsymbol{r}_0, \boldsymbol{r}_k)$,
if $k = 0$ then
　$\boldsymbol{p}_1 = \boldsymbol{r}_0$.
else
　$\beta = (\dfrac{\rho_{k+1}}{\rho_k})(\dfrac{\alpha}{\omega_k})$,
　$\boldsymbol{p}_{k+1} = \boldsymbol{r}_k + \beta(\boldsymbol{p}_k - \omega_k\boldsymbol{v}_k)$.
end if
$\boldsymbol{y} = M^{-1}\boldsymbol{p}_{k+1}$,
$\boldsymbol{v}_{k+1} = A\boldsymbol{y}$,
$\alpha = \dfrac{\rho_{k+1}}{(\boldsymbol{r}_0, \boldsymbol{v}_{k+1})}$,
$\boldsymbol{s} = \boldsymbol{r}_k - \alpha\boldsymbol{v}_{k+1}$,
$\boldsymbol{z} = M^{-1}\boldsymbol{s}$,
$\boldsymbol{t} = A\boldsymbol{z}$,
$\omega_{k+1} = \dfrac{(\boldsymbol{t}, \boldsymbol{s})}{(\boldsymbol{t}, \boldsymbol{t})}$,
$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha\boldsymbol{y} + \omega_{k+1}\boldsymbol{z}$,
$\boldsymbol{r}_{k+1} = \boldsymbol{s} - \omega_{k+1}\boldsymbol{t}$.

(4) **GMRES(m) 法** (文献 (3))

$\boldsymbol{u}_0^{(1)} = \boldsymbol{u}_0$.

for $j = 1, 2, \cdots$

$\boldsymbol{r}^{(j)} = M^{-1}(\boldsymbol{b} - A\boldsymbol{u}_0^{(j)})$,

$\boldsymbol{v}_1 = \dfrac{\boldsymbol{r}^{(j)}}{\|\boldsymbol{r}^{(j)}\|_2}$.

for $i = 1, 2, \cdots m$

$\boldsymbol{w} = M^{-1}A\boldsymbol{v_i}$.

for $k = 1, 2, \cdots i$

$\boldsymbol{w} = \boldsymbol{w} - (\boldsymbol{w}, \boldsymbol{v}_k)\boldsymbol{v}_k$.

$\boldsymbol{v}_{i+1} = \dfrac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$,

$\|\boldsymbol{b} - A\tilde{\boldsymbol{u}}^{(j)}\|_2$ が最小になるように

$\tilde{\boldsymbol{u}}^{(j)} = \boldsymbol{u}_0^{(j)} + y_1\boldsymbol{v}_1 + y_2\boldsymbol{v}_2 + \cdots + y_i\boldsymbol{v}_i$の係数 $y_1, y_2, \cdots, y_i$を選ぶ.

$\boldsymbol{u}_0^{(j+1)} = \tilde{\boldsymbol{u}}^{(j)}$

備考: GMRES(m) 法の反復回数は (内部反復数 i) + (外部反復数 j) $\times$ m で数える.

4.1.2.4 前処理手法

本章の関数が 4.1.2.3 で述べた前処理付き反復法の前処理として採用しているアルゴリズムについて説明する.

(1) スケーリング法

スケーリング法とは, 前処理行列 M として行列 A の対角成分からなる行列

$$D = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn})$$

を使用する手法である (文献 (4), (5)). スケーリング法による前処理付き CG 法は通称 SCG 法と呼ばれている.

Vector Engine 上では多くの場合, スケーリング法は他の前処理手法よりも高速である.

4.1.2.5 性能を上げるための高度な手法

(1) 反復改良法

反復改良法とは, 何らかの解法で得られた近似解をもとに, より精度の高い解を構成する手法であり, 単精度関数を用いた場合には倍精度並みの解が, 倍精度関数を用いた場合には 4 倍精度並みの解が得られる (そのかわり計算時間も 2 から 4 倍程度費やされる). また, 反復改良法のもう一つのメリットは得られた解の誤差が推定できることである. 反復改良法を用いない場合, 残差がわかっても誤差を推定することはむずかしい. さらに悪いことに, PCG 法で, 残差をどんなに 0 に近づけても, ある程度以上の誤差の改善はない. 従って, 解の精度に特に注意を要する場合や単精度関数を使用した場合などは, 解の反復改良を行うことが望ましい.

反復改良法は直感的には次のような式変形から導かれる. ただし, $\boldsymbol{u}$ は方程式 $A\boldsymbol{u} = \boldsymbol{b}$ の真の解, $\boldsymbol{u}'$ は近似解である.

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{u}' + \boldsymbol{u} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}\boldsymbol{b} - \boldsymbol{u}' \\ &= \boldsymbol{u}' + A^{-1}(\boldsymbol{b} - A\boldsymbol{u}') \end{aligned}$$

これより, 以下のアルゴリズムが得られる.

(a) 与えられた方程式 $A\boldsymbol{u} = \boldsymbol{b}$ を本章の関数で解く. 得られた解を $\boldsymbol{u}'$ とする.

(b) 解の残差を高精度で求める. すなわち $\boldsymbol{r}' = \boldsymbol{b} - A\boldsymbol{u}'$ を単精度関数の場合は倍精度で, 倍精度関数の場合は 4 倍精度で計算する.

(c) $\boldsymbol{r}'$ を元の精度に戻して $\boldsymbol{r}$ に代入する. すなわち, $\boldsymbol{r}'$ を単精度関数の場合は単精度に, 倍精度関数の場合は倍精度にする.

(d) 方程式 $A\boldsymbol{v} = \boldsymbol{r}$ を本章の関数で解く. 得られた解を $\boldsymbol{v}'$ とする.

(e) $\boldsymbol{u}' + \boldsymbol{v}'$ を計算し, 改めて $\boldsymbol{u}'$ とおく.

(f) $\boldsymbol{v}'$ が推定誤差であり, $\boldsymbol{v}'$ が十分小さければ終了し, そうでなければ (b) に戻る.

ここで最も重要な部分は (b) であり, 残差を高精度で計算することにより誤差の改善が図られる. 反復改良法は, 一般に数回の反復で十分な精度が得られるといわれている.

(2) 有限要素法における番号付けの入力について

レベルスケジューリングによるベクトル化において, 長いベクトル長を得るには, 例えば Odd-Even Ordering や多色 Ordering を行えばよいことは知られているが, 前処理付き反復法ではこのようなトリッキーな番号付けは収束性を著しく低下させることがあるので注意を要する (文献 (6)). 最も安全な番号付けは行列の帯幅を小さくするような番号付けであり, それを自動的に行うには Cuthill-McKee 法などを用いるとよい (文献 (7)).

4.1.3 参考文献

(1) Sonneveld, P. , “CGS, a Fast Lanczos–type Solver for Nonsymmetric Linear Systems”, Delft University of Technology, Report No. 84–16, Delft The Netherland (1984).

(2) Van Der Vorst, H, A. , “Bi–CGSTAB:A more smoothly converging variant of CGS for the solution of nonsymmetric linear systems”, SIAM J. Sci. Stat. Comput. 13, pp631–644 (1992).

(3) Y. Saad and M. H. SCHULTZ, “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”, SIAM J. Sci. Stat. Comput. , vol. 7, pp856-869 (1986).

(4) 速水, 原田, “ベクトル計算機における Scaled CG 法の有効性について”, 情報処理学会数値解析研究会資料, 17–4 (1986).

(5) 速水, 原田, “対角項スケーリングを施した共役勾配法のベクトル計算機における有効性について”, 情報処理学会論文誌第 30 巻第 11 号 (1989).

(6) Duff, I. S. and Meurant, G. A. , “The Effect of Ordering on Preconditioned Conjugate Gradients”, CERFACS, TR88/2, Toulouse, France (1988).

(7) Cuthill, E. and McKee, J. , “Reducing the Bandwidth of Sparse Symmetric Matrices”, Proc. of the 24th National Conference of the Association of Computing Machinery, Prandon Press, New Jersey, pp. 157–172 (1969).

4.2 スパース行列—非定常反復 (基礎反復法関数)

4.2.1 ASL_qxe010, ASL_pxe010
正値対称行列 (ELLPACK 型)(CG 法)

(1) **機 能**

スパース対称行列を係数行列とする連立 1 次方程式 $A\boldsymbol{u} = \boldsymbol{b}$ をスケーリング前処理付き CG 反復法で解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qxe010  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

単精度関数:

```
ierr = ASL_pxe010  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

(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}

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×m	入 力	係数行列の非零要素値の配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
2	lna	I	1	入 力	配列 a および ja の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	配列 a および ja の列数 (注意事項 (d) 参照)
5	ja	I*	lna×m	入 力	係数行列の非零構造データを格納する配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
6	b	{D∗ R∗}	n	入 力	方程式 $A\boldsymbol{u} = \boldsymbol{b}$ の右辺ベクトル $\boldsymbol{b}$
7	u	{D∗ R∗}	n	出 力	方程式の解ベクトル $\boldsymbol{u}$
8	itrmax	I	1	入 力	最大反復回数 (既定値 n)
9	itr	I*	1	出 力	実際の反復回数
10	epsmax	{D R}	1	入 力	打ち切り残差ノルム (既定値 { 10^{-12} (倍精度) / 10^{-6} (単精度) })
11	eps	{D∗ R∗}	1	出 力	最終残差ノルム
12	isw	I	1	入 力	処理スイッチ (既定値 0) (注意事項 (e) 参照) isw=0 : ja の値が全て異なるかチェックしない isw=1 : ja の値が全て異なるかチェックする
13	nt	I	1	入 力	生成するタスク数 (既定値 1)
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c)
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $\mathrm{ja}[i-1] = i, \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \neq i \ \ (i = 1, \cdots, \mathrm{n}; j = 2, \cdots, j_i)$,
  $1 \leq \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \leq \mathrm{n} \ \ (j = 2, \cdots, j_i)$
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $j_i < \mathrm{m}$ ならば,
  $\mathrm{ja}[(i-1) + \mathrm{lna} \times j_i] = 0$

(d) $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
同じ i に対して, $\mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \ \ (j = 1, \cdots, j_i)$ は全て異なる.

(e) $\mathrm{a}[i-1] \neq 0.0 \ \ (i = 1, \cdots, \mathrm{n})$

(f) $\mathrm{itrmax} \geq 1$

(g) epsmax $>$ アンダフロー判定値

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

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

(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	制限条件 (f) を満足しなかった.	itrmax = n として, 処理を続ける.
1200	制限条件 (g) を満足しなかった.	epsmax $= \left\{ \begin{array}{ll} 10^{-12} & \text{(倍精度)} \\ 10^{-6} & \text{(単精度)} \end{array} \right\}$ として, 処理を続ける.
1300	制限条件 (h) を満足しなかった.	isw=0 として, 制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
1400	制限条件 (i) を満足しなかった.	nt = 1 として, 処理を続ける.
2000	右辺 $\boldsymbol{b}$ の絶対ノルムが アンダフロー判定値より小さい.	$\mathrm{u}[i-1] \leftarrow 0.0 \ \ (i = 1, \cdots, \mathrm{n})$ を解とする.
2100	制限条件 (e) を満足しなかった.	処理を続ける.
2200	isw には 0 が入力されていた.	制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3500	許容反復数に達した.	その時点で得られた結果を返す.

戻り値	意　　味	処　理　内　容
4000	a の対角項に絶対値が アンダフロー判定値よりも小さいものがある.	処理を打ち切る.
4100	右辺 $\boldsymbol{b}$ のノルムが オーバフロー判定値より大きい.	
4210	残差 $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ のノルムが オーバフロー判定値より大きい.	
4220	残差 $\boldsymbol{r}$ の相対ノルムが オーバフロー判定値より大きい.	
4310	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ が アンダフロー判定値より小さい .	
4320	$\lvert(\boldsymbol{r}_i, M^{-1}\boldsymbol{r}_i)\rvert$ が オーバフロー判定値より大きい.	
4410	$\lvert(\boldsymbol{p}_i, A\boldsymbol{p}_i)\rvert$ が アンダフロー判定値より小さい.	
4420	$\lvert\ (\boldsymbol{p}_i, A\boldsymbol{p}_i)\ \rvert$ が オーバフロー判定値より大きい.	
5000	実行に必要な作業領域の確保に失敗した (注意事項 (f) 参照).	

(6) **注意事項**

(a) 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, 最大値などを以下に示す.
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 4–2　ASL C 言語インタフェースで使用している数値

	倍精度	単精度
最大値	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
正の最小値	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
負の最大値	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
最小値	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
オーバフロー判定値	最大値 $\times 10^{-3}$	
アンダフロー判定値	正の最小値 $\times 10^{3}$	

(b) 配列 a および ja の格納方法は以下のとおりである.

図 4–1　入力データの格納形式

格納する行列 A

$$\begin{array}{|ccccccc|}\hline a_{1,1} & a_{1,2} & 0.0 & a_{1,4} & 0.0 & a_{1,6} & 0.0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & 0.0 & 0.0 & a_{2,7} \\ 0.0 & a_{3,2} & a_{3,3} & 0.0 & 0.0 & a_{3,6} & 0.0 \\ a_{4,1} & a_{4,2} & 0.0 & a_{4,4} & a_{4,5} & a_{4,6} & 0.0 \\ 0.0 & 0.0 & 0.0 & a_{5,4} & a_{5,5} & 0.0 & 0.0 \\ a_{6,1} & 0.0 & a_{6,3} & a_{6,4} & 0.0 & a_{6,6} & 0.0 \\ 0.0 & a_{7,2} & 0.0 & 0.0 & 0.0 & 0.0 & a_{7,7} \\ \hline \end{array}$$

$\Downarrow$

配列 a[lna×m] の格納状態

lna												n
	$a_{1,1}$	a[0]	$a_{1,2}$	a[lna]	$a_{1,6}$	a[2 ∗ lna]	$a_{1,4}$	a[3 ∗ lna]	∗	a[4 ∗ lna]		
	$a_{2,2}$	a[1]	$a_{2,1}$	a[lna + 1]	$a_{2,3}$	a[2 ∗ lna + 1]	$a_{2,4}$	a[3 ∗ lna + 1]	$a_{2,7}$	a[4 ∗ lna + 1]		
	$a_{3,3}$	a[2]	$a_{3,2}$	a[lna + 2]	$a_{3,6}$	a[2 ∗ lna + 2]	∗	a[3 ∗ lna + 2]	∗	a[4 ∗ lna + 2]		
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
	$a_{6,6}$		$a_{6,1}$		$a_{6,3}$		$a_{6,4}$		∗			
	$a_{7,7}$		$a_{7,2}$		∗		∗		∗			

< − − − − −	m	− − − − − >

配列 ja[lna ∗ m] の格納状態

lna												n
	1	ja[0]	2	ja[lna]	6	ja[2 ∗ lna]	4	ja[3 ∗ lna]	0	ja[4 ∗ lna]		
	2	ja[1]	1	ja[lna + 1]	3	ja[2 ∗ lna + 1]	4	ja[3 ∗ lna + 1]	7	ja[4 ∗ lna + 1]		
	3	ja[2]	2	ja[lna + 2]	6	ja[2 ∗ lna + 2]	0	ja[3 ∗ lna + 2]	∗	ja[4 ∗ lna + 2]		
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
	6		1		3		4		0			
	7		2		0		∗		∗			

< − − − − −	m	− − − − − >

備 考

a. n は, 行列 A の次数.

b. lna $\geq$ n を満たさなければならない.

c. m は, 行列 A の非零要素を格納する配列 a の列数.

d. 配列 a には, 行列 A の非零要素を次のように格納する.
- 第 1 列に対角要素を格納する.
- 第 2〜m 列には下三角部分および上三角部分の非零要素をつめて各行ごとに格納する. ここで, 各行の非零要素を格納する順序は, 順不同である.
- 残りの部分の ∗ となっている位置に対応する要素は任意の値でよい.

e. 配列 ja には, 配列 a に格納した各要素に対応する箇所に行列 A 上での列番号を格納する.
m − 1 が行内の下三角部分および下三角部分の非零要素数より大きくなるような行については, 行列 A の非零要素の列番号を詰めた ja 内の領域の最右端の右隣の位置には 0 を格納する.
残りの部分の ∗ となっている位置には任意の値を格納する.

(c) 演算速度性能を最適化するために, 配列 a および ja に入力されたデータは一部変更される.

(d) m は, 各行の非零要素数の最大値より大きな値にしても構わないが, 無駄な領域をとらないほうがよりよい計算効率が得られる.

(e) isw = 0 を指定するほうがよりよい計算効率が得られる (isw = 1 を指定した場合著しく計算効率が落ちる). したがって, ja が制限条件 (d) を確実に満足する場合, isw = 0 を指定すべきである.
isw = 0 の場合, 制限条件 (d) のチェックは省略されるため, 係数行列データの入力を利用者自身が注意して行うこと.

ja が制限条件 (d) を満足しないかぎり, 得られた結果は保証されない.

(f) この関数では, 作業用領域を内部で自動的に確保している. 作業用領域が確保できない場合, 処理が打ち切られ, ierr = 5000 となる.
この場合は, 問題規模を縮小するか, または, マシンの環境を変更しなければ, この関数を使用して問題を解くことはできない.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 3 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & -3 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & -4 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & -5 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & -6 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

に対して $A\boldsymbol{u} = \boldsymbol{b}$ を解く.

(b) 入力データ
入力配列 a, ja, b,
$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) 主プログラム

```
/*      C interface example for ASL_qxe010 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;

    printf( "    *** ASL_qxe010 ***\n" );

    lna = 11;
    n   = 10;
    m   =  5;

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

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;
a[  lna*2] = -1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;
a[1+lna*3] = -2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;
a[2+lna*3] = -3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;
a[3+lna*4] = -4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;
a[4+lna*4] = -5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;
a[5+lna*4] = -6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;
ja[  lna*2] =  4;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;
ja[1+lna*3] =  5;

ja[2      ] =  3;
ja[2+lna  ] =  2;
ja[2+lna*2] =  4;
ja[2+lna*3] =  6;

ja[3      ] =  4;
ja[3+lna  ] =  1;
ja[3+lna*2] =  3;
ja[3+lna*3] =  5;
ja[3+lna*4] =  7;

ja[4      ] =  5;
ja[4+lna  ] =  2;
ja[4+lna*2] =  4;
ja[4+lna*3] =  6;
ja[4+lna*4] =  8;
```

```
    ja[5      ] =  6;
    ja[5+lna  ] =  3;
    ja[5+lna*2] =  5;
    ja[5+lna*3] =  7;
    ja[5+lna*4] =  9;

    ja[6      ] =  7;
    ja[6+lna  ] =  4;
    ja[6+lna*2] =  6;
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe010
    (a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) 出力結果

```
 *** ASL_qxe010 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =3.46e-16

 u[ 0 ] =   0.335
 u[ 1 ] =    0.22
 u[ 2 ] = -0.0808
 u[ 3 ] =   0.224
 u[ 4 ] =   0.136
 u[ 5 ] =  -0.151
 u[ 6 ] =   0.247
 u[ 7 ] =  0.0408
 u[ 8 ] = -0.0926
```

```
u[ 9 ] =    0.17
```

4.2.2 ASL_qxe020, ASL_pxe020 非対称行列 (ELLPACK 型)(CGS 法)

(1) **機 能**

スパース非対称行列を係数行列とする連立 1 次方程式 $A\boldsymbol{u} = \boldsymbol{b}$ をスケーリング前処理付き CGS 反復法で解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qxe020  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

単精度関数:

```
ierr = ASL_pxe020  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

(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}

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×m	入 力	係数行列の非零要素値の配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
2	lna	I	1	入 力	配列 a および ja の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	配列 a および ja の列数 (注意事項 (d) 参照)
5	ja	I*	lna×m	入 力	係数行列の非零構造データを格納する配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
6	b	{D∗ R∗}	n	入 力	方程式 $A\boldsymbol{u} = \boldsymbol{b}$ の右辺ベクトル $\boldsymbol{b}$
7	u	{D∗ R∗}	n	出 力	方程式の解ベクトル $\boldsymbol{u}$
8	itrmax	I	1	入 力	最大反復回数 (既定値 n)
9	itr	I*	1	出 力	実際の反復回数
10	epsmax	{D R}	1	入 力	打ち切り残差ノルム (既定値 { 10^{-12} (倍精度) ; 10^{-6} (単精度) })
11	eps	{D∗ R∗}	1	出 力	最終残差ノルム
12	isw	I	1	入 力	処理スイッチ (既定値 0) (注意事項 (e) 参照) isw=0 : ja の値が全て異なるかチェックしない isw=1 : ja の値が全て異なるかチェックする
13	nt	I	1	入 力	生成するタスク数 (既定値 1)
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c)
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $\mathrm{ja}[i-1] = i, \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \neq i \ \ (i = 1, \cdots, \mathrm{n}; j = 2, \cdots, j_i)$,
  $1 \leq \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \leq \mathrm{n} \ \ (j = 2, \cdots, j_i)$
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $j_i < \mathrm{m}$ ならば,
  $\mathrm{ja}[(i-1) + \mathrm{lna} \times j_i] = 0$

(d) $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
同じ i に対して, $\mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \ \ (j = 1, \cdots, j_i)$ は全て異なる.

(e) $\mathrm{a}[i-1] \neq 0.0 \ \ (i = 1, \cdots, \mathrm{n})$

(f) $\mathrm{itrmax} \geq 1$

(g) epsmax $>$ アンダフロー判定値

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

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

(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	制限条件 (f) を満足しなかった.	itrmax = n として, 処理を続ける.
1200	制限条件 (g) を満足しなかった.	epsmax $= \left\{ \begin{array}{ll} 10^{-12} & \text{(倍精度)} \\ 10^{-6} & \text{(単精度)} \end{array} \right\}$ として, 処理を続ける.
1300	制限条件 (h) を満足しなかった.	isw=0 として, 制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
1400	制限条件 (i) を満足しなかった.	nt = 1 として, 処理を続ける.
2000	右辺 $\boldsymbol{b}$ の絶対ノルムが アンダフロー判定値より小さい.	$\mathrm{u}[i-1] \leftarrow 0.0 \ \ (i = 1, \cdots, \mathrm{n})$ を解とする.
2100	制限条件 (e) を満足しなかった.	処理を続ける.
2200	isw には 0 が入力されていた.	制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3500	許容反復数に達した.	その時点で得られた結果を返す.

戻り値	意　　味	処　理　内　容
4000	a の対角項に絶対値が アンダフロー判定値よりも小さいものがある.	処理を打ち切る.
4100	右辺 $\boldsymbol{b}$ のノルムが オーバフロー判定値より大きい.	
4210	残差 $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ のノルムが オーバフロー判定値より大きい.	
4220	残差 $\boldsymbol{r}$ の相対ノルムが オーバフロー判定値より大きい.	
4310	$\lvert(\boldsymbol{r}_0, \boldsymbol{r}_i)\rvert$ が アンダフロー判定値より小さい .	
4320	$\lvert(\boldsymbol{r}_0, \boldsymbol{r}_i)\rvert$ が オーバフロー判定値より大きい.	
4410	$\lvert(\boldsymbol{r}_0, A\boldsymbol{p}_i)\rvert$ が アンダフロー判定値より小さい.	
4420	$\lvert(\boldsymbol{r}_0, A\boldsymbol{p}_i)\rvert$ が オーバフロー判定値より大きい.	
5000	実行に必要な作業領域の確保に失敗した (注意事項 (f) 参照).	

(6) **注意事項**

(a) 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, 最大値などを以下に示す.
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 4–3　ASL C 言語インタフェースで使用している数値

	倍精度	単精度
最大値	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
正の最小値	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
負の最大値	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
最小値	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
オーバフロー判定値	最大値 $\times 10^{-3}$	
アンダフロー判定値	正の最小値 $\times 10^{3}$	

(b) 配列 a および ja の格納方法は以下のとおりである.

図 4–2　入力データの格納形式

格納する行列 A

$$\begin{array}{|ccccccc|}\hline a_{1,1} & a_{1,2} & 0.0 & a_{1,4} & 0.0 & a_{1,6} & 0.0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & 0.0 & 0.0 & a_{2,7} \\ 0.0 & a_{3,2} & a_{3,3} & 0.0 & 0.0 & a_{3,6} & 0.0 \\ a_{4,1} & a_{4,2} & 0.0 & a_{4,4} & a_{4,5} & a_{4,6} & 0.0 \\ 0.0 & 0.0 & 0.0 & a_{5,4} & a_{5,5} & 0.0 & 0.0 \\ a_{6,1} & 0.0 & a_{6,3} & a_{6,4} & 0.0 & a_{6,6} & a_{6,7} \\ 0.0 & a_{7,2} & 0.0 & 0.0 & 0.0 & 0.0 & a_{7,7} \\ \hline \end{array}$$

⇓

配列 a[lna×m] の格納状態

lna ↕	$a_{1,1}$	a[0]	$a_{1,2}$	a[lna]	$a_{1,6}$	a[2 ∗ lna]	$a_{1,4}$	a[3 ∗ lna]	∗	a[4 ∗ lna]	n ↕
	$a_{2,2}$	a[1]	$a_{2,1}$	a[lna + 1]	$a_{2,3}$	a[2 ∗ lna + 1]	$a_{2,4}$	a[3 ∗ lna + 1]	$a_{2,7}$	a[4 ∗ lna + 1]	
	$a_{3,3}$	a[2]	$a_{3,2}$	a[lna + 2]	$a_{3,6}$	a[2 ∗ lna + 2]	∗	a[3 ∗ lna + 2]	∗	a[4 ∗ lna + 2]	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	$a_{6,6}$		$a_{6,1}$		$a_{6,3}$		$a_{6,4}$		$a_{6,7}$		
	$a_{7,7}$		$a_{7,2}$		∗		∗		∗		

< − − − − −	m	− − − − − >

配列 ja[lna ∗ m] の格納状態

lna ↕	1	ja[0]	2	ja[lna]	6	ja[2 ∗ lna]	4	ja[3 ∗ lna]	0	ja[4 ∗ lna]	n ↕
	2	ja[1]	1	ja[lna + 1]	3	ja[2 ∗ lna + 1]	4	ja[3 ∗ lna + 1]	7	ja[4 ∗ lna + 1]	
	3	ja[2]	2	ja[lna + 2]	6	ja[2 ∗ lna + 2]	0	ja[3 ∗ lna + 2]	∗	ja[4 ∗ lna + 2]	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	6		1		3		4		7		
	7		2		0		∗		∗		

< − − − − −	m	− − − − − >

備　考

a. n は, 行列 A の次数.

b. lna $\geq$ n を満たさなければならない.

c. m は, 行列 A の非零要素を格納する配列 a の列数.

d. 配列 a には, 行列 A の非零要素を次のように格納する.
- 第 1 列に対角要素を格納する.
- 第 2〜m 列には下三角部分および上三角部分の非零要素をつめて各行ごとに格納する. ここで, 各行の非零要素を格納する順序は, 順不同である.
- 残りの部分の ∗ となっている位置に対応する要素は任意の値でよい.

e. 配列 ja には, 配列 a に格納した各要素に対応する箇所に行列 A 上での列番号を格納する.
m − 1 が行内の下三角部分および下三角部分の非零要素数より大きくなるような行については, 行列 A の非零要素の列番号を詰めた ja 内の領域の最右端の右隣の位置には 0 を格納する.
残りの部分の ∗ となっている位置には任意の値を格納する.

(c) 演算速度性能を最適化するために, 配列 a および ja に入力されたデータは一部変更される.

(d) m は, 各行の非零要素数の最大値より大きな値にしても構わないが, 無駄な領域をとらないほうがよりよい計算効率が得られる.

(e) 係数行列データの入力を利用者自身が注意して行うこと.
isw = 0 を指定するほうがよりよい計算効率が得られる (isw = 1 を指定した場合著しく計算効率が落ちる). したがって, ja が制限条件 (d) を確実に満足する場合, isw = 0 を指定すべきである.
isw = 0 の場合, 制限条件 (d) のチェックは省略されるため, 係数行列データの入力を利用者自身が注意し

て行うこと.
ja が制限条件 (d) を満足しないかぎり, 得られた結果は保証されない.

(f) この関数では, 作業用領域を内部で自動的に確保している. 作業用領域が確保できない場合, 処理が打ち切られ, ierr = 5000 となる.
この場合は, 問題規模を縮小するか, または, マシンの環境を変更しなければ, この関数を使用して問題を解くことはできない.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

に対して $A\boldsymbol{u} = \boldsymbol{b}$ を解く.

(b) 入力データ
入力配列 a, ja, b,
$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) 主プログラム

```
/*      C interface example for ASL_qxe020 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;

    printf( "    *** ASL_qxe020 ***\n" );

    lna = 11;
    n   = 10;
    m   =  5;

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

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;

ja[2      ] =  3;
ja[2+lna  ] =  2;
ja[2+lna*2] =  4;

ja[3      ] =  4;
ja[3+lna  ] =  1;
ja[3+lna*2] =  3;
ja[3+lna*3] =  5;

ja[4      ] =  5;
ja[4+lna  ] =  2;
ja[4+lna*2] =  4;
ja[4+lna*3] =  6;

ja[5      ] =  6;
ja[5+lna  ] =  3;
ja[5+lna*2] =  5;
ja[5+lna*3] =  7;

ja[6      ] =  7;
ja[6+lna  ] =  4;
ja[6+lna*2] =  6;
```

```
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe020
    (a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) 出力結果

```
 *** ASL_qxe020 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =6.19e-16

 u[ 0 ] =   0.296
 u[ 1 ] =   0.111
 u[ 2 ] =  0.0732
 u[ 3 ] =  0.0882
 u[ 4 ] =  0.0707
 u[ 5 ] =  0.0185
 u[ 6 ] =   0.104
 u[ 7 ] =  0.0335
 u[ 8 ] = 0.00671
 u[ 9 ] =  0.0795
```

4.2.3 ASL_qxe030, ASL_pxe030 非対称行列 (ELLPACK 型)(BiCGSTAB 法)

(1) **機 能**

スパース非対称行列を係数行列とする連立 1 次方程式 $A\boldsymbol{u} = \boldsymbol{b}$ をスケーリング前処理付き BiCGSTAB 反復法で解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qxe030  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

単精度関数:

```
ierr = ASL_pxe030  (a, lna, n, m, ja, b, u, itrmax, & itr, epsmax, & eps, isw, nt);
```

(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}

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×m	入 力	係数行列の非零要素値の配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
2	lna	I	1	入 力	配列 a および ja の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	配列 a および ja の列数 (注意事項 (d) 参照)
5	ja	I*	lna×m	入 力	係数行列の非零構造データを格納する配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
6	b	{D∗ R∗}	n	入 力	方程式 $A\boldsymbol{u} = \boldsymbol{b}$ の右辺ベクトル $\boldsymbol{b}$
7	u	{D∗ R∗}	n	出 力	方程式の解ベクトル $\boldsymbol{u}$
8	itrmax	I	1	入 力	最大反復回数 (既定値 n)
9	itr	I*	1	出 力	実際の反復回数
10	epsmax	{D R}	1	入 力	打ち切り残差ノルム (既定値 { 10^{-12} (倍精度) / 10^{-6} (単精度) })
11	eps	{D∗ R∗}	1	出 力	最終残差ノルム
12	isw	I	1	入 力	処理スイッチ (既定値 0) (注意事項 (e) 参照) isw=0 : ja の値が全て異なるかチェックしない isw=1 : ja の値が全て異なるかチェックする
13	nt	I	1	入 力	生成するタスク数 (既定値 1)
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c)
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $\mathrm{ja}[i-1] = i, \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \neq i \ \ (i = 1, \cdots, \mathrm{n}; j = 2, \cdots, j_i)$,
  $1 \leq \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \leq \mathrm{n} \ \ (j = 2, \cdots, j_i)$
- $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $j_i < \mathrm{m}$ ならば,
  $\mathrm{ja}[(i-1) + \mathrm{lna} \times j_i] = 0$

(d) $j_i \ \ (i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
同じ i に対して, $\mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \ \ (j = 1, \cdots, j_i)$ は全て異なる.

(e) $\mathrm{a}[i-1] \neq 0.0 \ \ (i = 1, \cdots, \mathrm{n})$

(f) $\mathrm{itrmax} \geq 1$

(g) epsmax > アンダフロー判定値

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

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

(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	制限条件 (f) を満足しなかった.	$\mathrm{itrmax} = \mathrm{n}$ として, 処理を続ける.
1200	制限条件 (g) を満足しなかった.	$\mathrm{epsmax} = \left\{ \begin{array}{ll} 10^{-12} & \text{(倍精度)} \\ 10^{-6} & \text{(単精度)} \end{array} \right\}$ として, 処理を続ける.
1300	制限条件 (h) を満足しなかった.	isw=0 として, 制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
1400	制限条件 (i) を満足しなかった.	$\mathrm{nt} = 1$ として, 処理を続ける.
2000	右辺 $\boldsymbol{b}$ の絶対ノルムが アンダフロー判定値より小さい.	$\mathrm{u}[i-1] \leftarrow 0.0 \ \ (i = 1, \cdots, \mathrm{n})$ を解とする.
2100	制限条件 (e) を満足しなかった.	処理を続ける.
2200	isw には 0 が入力されていた.	制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3500	許容反復数に達した.	その時点で得られた結果を返す.

戻り値	意　　味	処　理　内　容
4000	a の対角項に絶対値が アンダフロー判定値よりも小さいものがある.	処理を打ち切る.
4100	右辺 $\boldsymbol{b}$ のノルムが オーバフロー判定値より大きい.	
4210	残差 $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ のノルムが オーバフロー判定値より大きい.	
4220	残差 $\boldsymbol{r}$ の相対ノルムが オーバフロー判定値より大きい.	
4310	$\|\boldsymbol{\rho}_i\|$ が アンダフロー判定値より小さい.	
4320	$\|\boldsymbol{\rho}_i\|$ が オーバフロー判定値より大きい.	
4410	$\|(\boldsymbol{r}_0, \boldsymbol{v}_i)\|$ が アンダフロー判定値より小さい.	
4420	$\|(\boldsymbol{r}_0, \boldsymbol{v}_i)\|$ が オーバフロー判定値より大きい.	
4510	$\|(\boldsymbol{t}, \boldsymbol{t})\|$ が アンダフロー判定値より小さい.	
4520	$\|(\boldsymbol{t}, \boldsymbol{t})\|$ が オーバフロー判定値より大きい.	
4610	$\|\boldsymbol{\omega}_i\|$ が アンダフロー判定値より小さい.	
4620	$\|\boldsymbol{\omega}_i\|$ が オーバフロー判定値より大きい.	
4700	$\|\boldsymbol{\beta}\|$ が オーバフロー判定値より大きい.	
4800	$\|\boldsymbol{\alpha}\|$ が オーバフロー判定値より大きい.	
4900	$\|(\boldsymbol{t}, \boldsymbol{s})\|$ が オーバフロー判定値より大きい.	
5000	実行に必要な作業領域の確保に失敗した (注意事項 (f) 参照).	

(6) **注意事項**

(a) 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, 最大値などを以下に示す.
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 4–4　ASL C 言語インタフェースで使用している数値

	倍精度	単精度
最大値	$2^{1023}(2-2^{-52}) \simeq (1.80 \times 10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40 \times 10^{38})$
正の最小値	$2^{-1022} \simeq (2.23 \times 10^{-308})$	$2^{-126} \simeq (1.17 \times 10^{-38})$
負の最大値	$-2^{-1022} \simeq (-2.23 \times 10^{-308})$	$-2^{-126} \simeq (-1.17 \times 10^{-38})$
最小値	$-2^{1023}(2-2^{-52}) \simeq (-1.80 \times 10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40 \times 10^{38})$
オーバフロー判定値	最大値 $\times 10^{-3}$	
アンダフロー判定値	正の最小値 $\times 10^{3}$	

(b) 配列 a および ja の格納方法は以下のとおりである.

図 4–3　入力データの格納形式

格納する行列 A

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

配列 a[lna×m] の格納状態

$a_{1,1}$	a[0]	$a_{1,2}$	a[lna]	$a_{1,6}$	a[2 ∗ lna]	$a_{1,4}$	a[3 ∗ lna]	∗	a[4 ∗ lna]	↑	
$a_{2,2}$	a[1]	$a_{2,1}$	a[lna + 1]	$a_{2,3}$	a[2 ∗ lna + 1]	$a_{2,4}$	a[3 ∗ lna + 1]	$a_{2,7}$	a[4 ∗ lna + 1]		
$a_{3,3}$	a[2]	$a_{3,2}$	a[lna + 2]	$a_{3,6}$	a[2 ∗ lna + 2]	∗	a[3 ∗ lna + 2]	∗	a[4 ∗ lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
$a_{6,6}$		$a_{6,1}$		$a_{6,3}$		$a_{6,4}$		$a_{6,7}$			
$a_{7,7}$		$a_{7,2}$		∗		∗		∗			

lna ↕　　n ↕

< − − − − −	m	− − − − − >

配列 ja[lna ∗ m] の格納状態

1	ja[0]	2	ja[lna]	6	ja[2 ∗ lna]	4	ja[3 ∗ lna]	0	ja[4 ∗ lna]	↑	
2	ja[1]	1	ja[lna + 1]	3	ja[2 ∗ lna + 1]	4	ja[3 ∗ lna + 1]	7	ja[4 ∗ lna + 1]		
3	ja[2]	2	ja[lna + 2]	6	ja[2 ∗ lna + 2]	0	ja[3 ∗ lna + 2]	∗	ja[4 ∗ lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
6		1		3		4		7			
7		2		0		∗		∗			

lna ↕　　n ↕

< − − − − −	m	− − − − − >

備 考

a. n は, 行列 A の次数.

b. lna $\geq$ n を満たさなければならない.

c. m は, 行列 A の非零要素を格納する配列 a の列数.

d. 配列 a には, 行列 A の非零要素を次のように格納する.
- 第 1 列に対角要素を格納する.
- 第 2〜m 列には下三角部分および上三角部分の非零要素をつめて各行ごとに格納する. ここで, 各行の非零要素を格納する順序は, 順不同である.
- 残りの部分の ∗ となっている位置に対応する要素は任意の値でよい.

e. 配列 ja には, 配列 a に格納した各要素に対応する箇所に行列 A 上での列番号を格納する.
m − 1 が行内の下三角部分および下三角部分の非零要素数より大きくなるような行については, 行列 A の非零要素の列番号を詰めた ja 内の領域の最右端の右隣の位置には 0 を格納する.
残りの部分の ∗ となっている位置には任意の値を格納する.

(c) 演算速度性能を最適化するために, 配列 a および ja に入力されたデータは一部変更される.

(d) m は, 各行の非零要素数の最大値より大きな値にしても構わないが, 無駄な領域をとらないほうがよりよい計算効率が得られる.

(e) isw = 0 を指定するほうがよりよい計算効率が得られる (isw = 1 を指定した場合著しく計算効率が落ちる). したがって, ja が制限条件 (d) を確実に満足する場合, isw = 0 を指定すべきである.
isw = 0 の場合, 制限条件 (d) のチェックは省略されるため, 係数行列データの入力を利用者自身が注意して行うこと.

ja が制限条件 (d) を満足しないかぎり, 得られた結果は保証されない.

(f) この関数では, 作業用領域を内部で自動的に確保している. 作業用領域が確保できない場合, 処理が打ち切られ, ierr = 5000 となる.
この場合は, 問題規模を縮小するか, または, マシンの環境を変更しなければ, この関数を使用して問題を解くことはできない.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

に対して $A\boldsymbol{u} = \boldsymbol{b}$ を解く.

(b) 入力データ
入力配列 a, ja, b,
$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) 主プログラム

```
/*      C interface example for ASL_qxe030 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;

    printf( "    *** ASL_qxe030 ***\n" );

    lna = 11;
    n   = 10;
    m   =  5;

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

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;

ja[2      ] =  3;
ja[2+lna  ] =  2;
ja[2+lna*2] =  4;

ja[3      ] =  4;
ja[3+lna  ] =  1;
ja[3+lna*2] =  3;
ja[3+lna*3] =  5;

ja[4      ] =  5;
ja[4+lna  ] =  2;
ja[4+lna*2] =  4;
ja[4+lna*3] =  6;

ja[5      ] =  6;
ja[5+lna  ] =  3;
ja[5+lna*2] =  5;
ja[5+lna*3] =  7;

ja[6      ] =  7;
ja[6+lna  ] =  4;
ja[6+lna*2] =  6;
ja[6+lna*3] =  8;
```

```
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe030
    (a,lna,n,m,ja,b,u,itrmax,&itr,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) 出力結果

```
 *** ASL_qxe030 ***

***   Input   ***

 itrmax =     100
 epsmax =   1e-12
 nt     =       2

***   Output   ***

 ierr   =       0
 itr    =      10
 eps    =3.88e-16

 u[ 0 ] =   0.296
 u[ 1 ] =   0.111
 u[ 2 ] =  0.0732
 u[ 3 ] =  0.0882
 u[ 4 ] =  0.0707
 u[ 5 ] =  0.0185
 u[ 6 ] =   0.104
 u[ 7 ] =  0.0335
 u[ 8 ] = 0.00671
 u[ 9 ] =  0.0795
```

4.2.4 ASL_qxe040, ASL_pxe040
非対称行列 (ELLPACK 型)(GMRES(m) 法)

(1) **機 能**

スパース非対称行列を係数行列とする連立 1 次方程式 $A\boldsymbol{u} = \boldsymbol{b}$ をスケーリング前処理付き GMRES(m) 反復法で解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qxe040  (a, lna, n, m, ja, b, u, itrmax, & itr, mgmrs, epsmax, & eps, isw, nt);
```

単精度関数:

```
ierr = ASL_pxe040  (a, lna, n, m, ja, b, u, itrmax, & itr, mgmrs, epsmax, & eps, isw, nt);
```

(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}

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×m	入 力	係数行列の非零要素値の配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
2	lna	I	1	入 力	配列 a および ja の整合寸法
3	n	I	1	入 力	行列 A の次数
4	m	I	1	入 力	配列 a および ja の列数 (注意事項 (d) 参照)
5	ja	I*	lna×m	入 力	係数行列の非零構造データを格納する配列 (格納形式については 注意事項 (b) 参照)
				出 力	演算最適化のため更新された値 (注意事項 (c) 参照)
6	b	{D∗ R∗}	n	入 力	方程式 $A\boldsymbol{u} = \boldsymbol{b}$ の右辺ベクトル $\boldsymbol{b}$
7	u	{D∗ R∗}	n	出 力	方程式の解ベクトル $\boldsymbol{u}$
8	itrmax	I	1	入 力	最大反復回数 (既定値 n)
9	itr	I*	1	出 力	実際の反復回数
10	mgmrs	I	1	入 力	GMRES(m) のパラメータ m (既定値 10)
11	epsmax	{D R}	1	入 力	打ち切り残差ノルム (既定値 { 10^{-12} (倍精度) ; 10^{-6} (単精度) })
12	eps	{D∗ R∗}	1	出 力	最終残差ノルム
13	isw	I	1	入 力	処理スイッチ (既定値 0) (注意事項 (e) 参照) isw=0 : ja の値が全て異なるかチェックしない isw=1 : ja の値が全て異なるかチェックする

項番	引数と戻り値	型	大きさ	入出力	内　　容
14	nt	I	1	入 力	生成するタスク数 (既定値 1)
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq \mathrm{n} \leq \mathrm{lna}$

(b) $1 \leq \mathrm{m} \leq \mathrm{n}$

(c)
- j_i $(i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $\mathrm{ja}[i-1] = i, \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \neq i$ $(i = 1, \cdots, \mathrm{n}; j = 2, \cdots, j_i)$,
  $1 \leq \mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)] \leq \mathrm{n}$ $(j = 2, \cdots, j_i)$
- j_i $(i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
  $j_i < \mathrm{m}$ ならば,
  $\mathrm{ja}[(i-1) + \mathrm{lna} \times j_i] = 0$

(d) j_i $(i = 1, \cdots, \mathrm{n})$ を行列 A の各 i 行の非零要素数とすると,
同じ i に対して, $\mathrm{ja}[(i-1) + \mathrm{lna} \times (j-1)]$ $(j = 1, \cdots, j_i)$ は全て異なる.

(e) $\mathrm{a}[i-1] \neq 0.0$ $(i = 1, \cdots, \mathrm{n})$

(f) $\mathrm{itrmax} \geq 1$

(g) $\mathrm{mgmrs} \geq 1$

(h) epsmax > アンダフロー判定値

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

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

(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	制限条件 (f) を満足しなかった.	itrmax = n として, 処理を続ける.
1100	制限条件 (g) を満足しなかった.	mgmrs = 10 として, 処理を続ける.
1200	制限条件 (h) を満足しなかった.	epsmax $= \left\{ \begin{array}{ll} 10^{-12} & \text{(倍精度)} \\ 10^{-6} & \text{(単精度)} \end{array} \right\}$ として, 処理を続ける.
1300	制限条件 (i) を満足しなかった.	isw=0 として, 制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).
1400	制限条件 (j) を満足しなかった.	nt = 1 として, 処理を続ける.
2000	右辺 $\boldsymbol{b}$ の絶対ノルムが アンダフロー判定値より小さい.	$\mathrm{u}[i-1] \leftarrow 0.0$ $(i = 1, \cdots, \mathrm{n})$ を解とする.
2100	制限条件 (e) を満足しなかった.	処理を続ける.
2200	isw には 0 が入力されていた.	制限条件 (d) のチェックを行わずに処理を続ける (注意事項 (e) 参照).

戻り値	意　　味	処　理　内　容
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3500	許容反復数に達した.	その時点で得られた結果を返す.
4000	a の対角項に絶対値が アンダフロー判定値よりも小さいものがある.	処理を打ち切る.
4100	右辺 $\boldsymbol{b}$ のノルムが オーバフロー判定値より大きい.	
4210	初期残差 $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{u}$ のノルムが オーバフロー判定値より大きい.	
4220	最終解の残差 $\boldsymbol{r}$ のノルムが オーバフロー判定値より大きい.	
4310	$\|AM^{-1}\boldsymbol{v}_i\|_2{}^2$ が アンダフロー判定値より小さい. この場合 AM^{-1} が退化している可能性が高い.	$(i-1)$ 回目までで得られた解を返して処理を打ち切る.
4320	$\|AM^{-1}\boldsymbol{v}_i\|_2{}^2$ が オーバフロー判定値より大きい.	処理を打ち切る.
4410	Gram-Schmidt の直交化後 $\|\boldsymbol{w}\|_2{}^2$ が アンダフロー判定値より小さくなり, 続行不可能となった. これは反復解がすでに収束していることを示す. 収束判定条件が厳しすぎる可能性が高い.	その時点までに得られた解を返して処理を打ち切る.
5000	実行に必要な作業領域の確保に失敗した (注意事項 (f) 参照).	処理を打ち切る.

(6) **注意事項**

(a) 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, 最大値などを以下に示す.
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 4–5　ASL C 言語インタフェースで使用している数値

	倍精度	単精度
最大値	$2^{1023}(2-2^{-52}) \simeq (1.80\times10^{308})$	$2^{127}(2-2^{-23}) \simeq (3.40\times10^{38})$
正の最小値	$2^{-1022} \simeq (2.23\times10^{-308})$	$2^{-126} \simeq (1.17\times10^{-38})$
負の最大値	$-2^{-1022} \simeq (-2.23\times10^{-308})$	$-2^{-126} \simeq (-1.17\times10^{-38})$
最小値	$-2^{1023}(2-2^{-52}) \simeq (-1.80\times10^{308})$	$-2^{127}(2-2^{-23}) \simeq (-3.40\times10^{38})$
オーバフロー判定値	最大値 $\times 10^{-3}$	
アンダフロー判定値	正の最小値 $\times 10^{3}$	

(b) 配列 a および ja の格納方法は以下のとおりである.

図 4–4　入力データの格納形式

格納する行列 A

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

配列 a[lna×m] の格納状態

$a_{1,1}$	a[0]	$a_{1,2}$	a[lna]	$a_{1,6}$	a[2 ∗ lna]	$a_{1,4}$	a[3 ∗ lna]	∗	a[4 ∗ lna]		
$a_{2,2}$	a[1]	$a_{2,1}$	a[lna + 1]	$a_{2,3}$	a[2 ∗ lna + 1]	$a_{2,4}$	a[3 ∗ lna + 1]	$a_{2,7}$	a[4 ∗ lna + 1]		
$a_{3,3}$	a[2]	$a_{3,2}$	a[lna + 2]	$a_{3,6}$	a[2 ∗ lna + 2]	∗	a[3 ∗ lna + 2]	∗	a[4 ∗ lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
$a_{6,6}$		$a_{6,1}$		$a_{6,3}$		$a_{6,4}$		$a_{6,7}$			
$a_{7,7}$		$a_{7,2}$		∗		∗		∗			

lna　　n

< − − − − −　　m　　− − − − − >

配列 ja[lna ∗ m] の格納状態

1	ja[0]	2	ja[lna]	6	ja[2 ∗ lna]	4	ja[3 ∗ lna]	0	ja[4 ∗ lna]		
2	ja[1]	1	ja[lna + 1]	3	ja[2 ∗ lna + 1]	4	ja[3 ∗ lna + 1]	7	ja[4 ∗ lna + 1]		
3	ja[2]	2	ja[lna + 2]	6	ja[2 ∗ lna + 2]	0	ja[3 ∗ lna + 2]	∗	ja[4 ∗ lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
6		1		3		4		7			
7		2		0		∗		∗			

lna　　n

< − − − − −　　m　　− − − − − >

備　考

a. n は, 行列 A の次数.

b. lna $\geq$ n を満たさなければならない.

c. m は, 行列 A の非零要素を格納する配列 a の列数.

d. 配列 a には, 行列 A の非零要素を次のように格納する.
- 第 1 列に対角要素を格納する.
- 第 2〜m 列には下三角部分および上三角部分の非零要素をつめて各行ごとに格納する. ここで, 各行の非零要素を格納する順序は, 順不同である.
- 残りの部分の ∗ となっている位置に対応する要素は任意の値でよい.

e. 配列 ja には, 配列 a に格納した各要素に対応する箇所に行列 A 上での列番号を格納する.
m − 1 が行内の下三角部分および下三角部分の非零要素数より大きくなるような行については, 行列 A の非零要素の列番号を詰めた ja 内の領域の最右端の右隣の位置には 0 を格納する.
残りの部分の ∗ となっている位置には任意の値を格納する.

(c) 演算速度性能を最適化するために, 配列 a および ja に入力されたデータは一部変更される.

(d) m は, 各行の非零要素数の最大値より大きな値にしても構わないが, 無駄な領域をとらないほうがよりよい計算効率が得られる.

(e) isw = 0 を指定するほうがよりよい計算効率が得られる (isw = 1 を指定した場合著しく計算効率が落ちる). したがって, ja が制限条件 (d) を確実に満足する場合, isw = 0 を指定すべきである.
isw = 0 の場合, 制限条件 (d) のチェックは省略されるため, 係数行列データの入力を利用者自身が注意して行うこと.

ja が制限条件 (d) を満足しないかぎり, 得られた結果は保証されない.

(f) この関数では, 作業用領域を内部で自動的に確保している. 作業用領域が確保できない場合, 処理が打ち切られ, ierr = 5000 となる.
この場合は, 問題規模を縮小するか, または, マシンの環境を変更しなければ, この関数を使用して問題を解くことはできない.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 7 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & 9 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 11 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 0 & 5 & 13 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 6 & 15 & 7 & 0 & -7 \\ 0 & 0 & 0 & 0 & -5 & 0 & 7 & 17 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & 8 & 19 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 9 & 21 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

に対して $A\boldsymbol{u} = \boldsymbol{b}$ を解く.

(b) 入力データ
入力配列 a, ja, b,
$\mathrm{lna} = 11, \mathrm{n} = 10, \mathrm{m} = 11, \mathrm{itrmax} = 100, \mathrm{mgmrs} = 5, \mathrm{epsmax} = 10^{-12}, \mathrm{isw} = 0, \mathrm{nt} = 2$

(c) 主プログラム

```
/*      C interface example for ASL_qxe040 */

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

int main()
{
    double *a;
    int lna;
    int n;
    int m;
    int *ja;
    double *b;
    double *u;
    int itrmax;
    int itr;
    int mgmrs;
    double epsmax;
    double eps;
    int isw;
    int nt;
    int ierr;
    int i,j;

    printf( "    *** ASL_qxe040 ***\n" );

    lna = 11;
    n   = 10;
    m   =  5;

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

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

for( j=0 ; j < m ; j++ )
{
    for( i=0 ; i < n ; i++ )
    {
        a[i+lna*j] = 0.0;
       ja[i+lna*j] = 0;
    }
}

a[0      ] =  3.0;
a[  lna  ] =  1.0;

a[1      ] =  5.0;
a[1+lna  ] =  1.0;
a[1+lna*2] =  2.0;

a[2      ] =  7.0;
a[2+lna  ] =  2.0;
a[2+lna*2] =  3.0;

a[3      ] =  9.0;
a[3+lna  ] = -1.0;
a[3+lna*2] =  3.0;
a[3+lna*3] =  4.0;

a[4      ] = 11.0;
a[4+lna  ] = -2.0;
a[4+lna*2] =  4.0;
a[4+lna*3] =  5.0;

a[5      ] = 13.0;
a[5+lna  ] = -3.0;
a[5+lna*2] =  5.0;
a[5+lna*3] =  6.0;

a[6      ] = 15.0;
a[6+lna  ] = -4.0;
a[6+lna*2] =  6.0;
a[6+lna*3] =  7.0;
a[6+lna*4] = -7.0;

a[7      ] = 17.0;
a[7+lna  ] = -5.0;
a[7+lna*2] =  7.0;
a[7+lna*3] =  8.0;

a[8      ] = 19.0;
a[8+lna  ] = -6.0;
a[8+lna*2] =  8.0;
a[8+lna*3] =  9.0;

a[9      ] = 21.0;
a[9+lna  ] = -7.0;
a[9+lna*2] =  9.0;

ja[0      ] =  1;
ja[  lna  ] =  2;

ja[1      ] =  2;
ja[1+lna  ] =  1;
ja[1+lna*2] =  3;

ja[2      ] =  3;
ja[2+lna  ] =  2;
ja[2+lna*2] =  4;

ja[3      ] =  4;
ja[3+lna  ] =  1;
ja[3+lna*2] =  3;
ja[3+lna*3] =  5;

ja[4      ] =  5;
ja[4+lna  ] =  2;
ja[4+lna*2] =  4;
ja[4+lna*3] =  6;

ja[5      ] =  6;
ja[5+lna  ] =  3;
ja[5+lna*2] =  5;
ja[5+lna*3] =  7;

ja[6      ] =  7;
ja[6+lna  ] =  4;
ja[6+lna*2] =  6;
```

```
    ja[6+lna*3] =  8;
    ja[6+lna*4] = 10;

    ja[7      ] =  8;
    ja[7+lna  ] =  5;
    ja[7+lna*2] =  7;
    ja[7+lna*3] =  9;

    ja[8      ] =  9;
    ja[8+lna  ] =  6;
    ja[8+lna*2] =  8;
    ja[8+lna*3] = 10;

    ja[9      ] = 10;
    ja[9+lna  ] =  7;
    ja[9+lna*2] =  9;

    for( i=0 ; i < n ; i++ )
    {
        b[i] = 1.0;
    }

    for( i=0 ; i < n ; i++ )
    {
        u[i] = 0.0;
    }

    itrmax = 100;
    mgmrs  = 10;
    epsmax = 1.0e-12;
    isw    = 1;
    nt     = 2;

    printf( "\n   ***   Input   ***\n\n" );
    printf( "\titrmax =  %6d\n", itrmax );
    printf( "\tepsmax =%8.3g\n", epsmax );
    printf( "\tmgmrs  =  %6d\n", mgmrs );
    printf( "\tnt     =  %6d\n", nt );


        ierr = ASL_qxe040
    (a,lna,n,m,ja,b,u,itrmax,&itr,mgmrs,epsmax,&eps,isw,nt);


    printf( "\n   ***   Output   ***\n\n" );
    printf( "\tierr   =  %6d\n", ierr );
    printf( "\titr    =  %6d\n", itr );
    printf( "\teps    =%8.3g\n\n", eps );

    for( i=0 ; i < n ; i++ )
    {
        printf( "\tu[%2d ] =%8.3g\n", i, u[i] );
    }

    free( a );
    free( ja );
    free( b );
    free( u );

    return 0;
}
```

(d) 出力結果

```
    *** ASL_qxe040 ***

   ***   Input   ***

    itrmax =     100
    epsmax =   1e-12
    mgmrs  =      10
    nt     =       2

   ***   Output   ***

    ierr   =       0
    itr    =      10
    eps    =3.02e-16

    u[ 0 ] =   0.296
    u[ 1 ] =   0.111
    u[ 2 ] =  0.0732
    u[ 3 ] =  0.0882
    u[ 4 ] =  0.0707
    u[ 5 ] =  0.0185
    u[ 6 ] =   0.104
    u[ 7 ] =  0.0335
    u[ 8 ] = 0.00671
    u[ 9 ] =  0.0795
```

第 5 章　固有値・固有ベクトル

5.1　概 要

本章では, 実対称行列の標準固有値問題等についてについて説明する.

本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.

標準固有値問題では, 与えられた行列 A について

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

を満たす値 λ およびベクトル $\boldsymbol{x}$ を求める. この λ は行列 A の固有値, $\boldsymbol{x}$ はこれに対応する固有ベクトルと呼ばれる.

また, 一般化固有値問題では与えられた行列 A ならびに B について,

$A\boldsymbol{x} = \lambda B\boldsymbol{x}$,

$AB\boldsymbol{x} = \lambda\boldsymbol{x}$ (A と B は両方ともエルミート行列, B は正定値),

$BA\boldsymbol{x} = \lambda\boldsymbol{x}$ (A と B は両方ともエルミート行列, B は正定値)

のいずれかを満たす値 λ およびベクトル $\boldsymbol{x}$ を求める. これらの λ, $\boldsymbol{x}$ も同様に固有値, 固有ベクトルと呼ばれる.

本章に属する関数では, つぎの 4 つのカテゴリーに対応する機能を用意している.

全固有値・全固有ベクトル

すべての固有値と対応する固有ベクトルを求める.

全固有値

固有値のみをすべて求める.

固有値・固有ベクトル

固有値を大きい方から数個, または固有値を小さい方から数個求め, 対応する固有ベクトルを求める.

固有値

固有値を大きい方から数個, または固有値を小さい方から数個求める.

5.1.1 使用上の注意

(1) 一般に, 「全固有値・全固有ベクトル」または「固有値・固有ベクトル」に対応する機能はそれぞれ「全固有値」または「固有値」に対応する機能より多くの処理時間, メモリ量を必要とする.

(2) 一般に, 「固有値・固有ベクトル」および「固有値」に対応する機能を使用して効果があるのは, 求める固有値の個数が, たかだか全体の 2 割程度までの場合で, それ以上求めたい場合は, 「全固有値・全固有ベクトル」または「全固有値」に対応する機能を使用したほうが計算時間が少なくてすむ.

(3) 本ライブラリにおける一般化固有値問題の関数では, 行列 B が正定値であるという条件が課せられている. しかし, 以下のような場合は, 行列 B が正定値でなくても固有値・固有ベクトルを求めることができる.

(a) 行列 B が正定値でないが, 行列 A は正定値の場合

$$Bv = \lambda^{-1} Av$$

によって固有値 $\lambda(\neq 0)$ と固有ベクトルが得られる.

(b) A, B が共に正定値でないが, $A + B$ は正定値の場合

$$Av = \frac{\lambda}{1+\lambda}(A + B)v$$

によって固有値 $\lambda(\neq -1)$ と固有ベクトルが得られる.

(4) 入力行列が実対称行列またはエルミート行列である場合, 専用の関数を使用したほうが計算時間は少なくてすむ.

(5) 行列の次元数が小さいと, 演算コストに対して並列処理オーバヘッドの影響が大きいため, 非並列処理関数を用いた場合よりも性能が低下する.

5.1.2 使用しているアルゴリズム

5.1.2.1 実対称行列の実対称 3 重対角行列への変換

ハウスホルダー法によって, $n \times n$ 実対称行列 A を実対称 3 重対角行列 T に変換する.
すなわち, $A_1 = A$ として, $k = 1, 2, \cdots, n-2$ に対して, あるベクトル $\boldsymbol{u}_k$ を

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

かつ,

$$A_{k+1} = P_k A_k P_k$$

の第 k 列の副対角成分より下がすべて 0 になるように取れる. A_{n-1} が求める実対称 3 重対角行列となる. なお, 変換行列 P_k は直交かつ対称な行列である.

5.1.2.2 エルミート (Hermitian) 行列の実対称 3 重対角行列への変換

まず, ハウスホルダー法によって, $n \times n$ エルミート行列 A をエルミート 3 重対角行列 S に変換する.

$$S = P_{n-2} \cdots P_2 P_1 A P_1 P_2 \cdots P_{n-2}$$

さらに, 正則な複素対角行列 D によって (相似変換), 実対称 3 重対角行列 T に変換する.

$$T = D^* S D$$

5.1.2.3 ブロックアルゴリズムによるハウスホルダー変換

ハウスホルダー変換には, ブロックアルゴリズムを用いる. この手法は, もとの実対称行列を実対称 3 重対角行列に変換する際, 行列の積和によるランク-1 の行列更新を単純化する.
対称行列 A について k 回相似変換した行列を A_{k+1} とすると,

$$A_{k+1} = P_k A_k P_k = A_k - \boldsymbol{u}_k \boldsymbol{v}_k^T - \boldsymbol{v}_k \boldsymbol{u}_k^T$$

となる. ここで, P_k は, 直交行列である. ただし,

$$A_1 = A$$

$$P_k = I - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2}\boldsymbol{u}_k^T \boldsymbol{u}_k$$

$$\boldsymbol{y}_k = A_k \boldsymbol{u}_k$$

$$\boldsymbol{v}_k = \frac{(\boldsymbol{y}_k - \frac{(\boldsymbol{u}_k^T \boldsymbol{y}_k)\boldsymbol{u}_k}{2H_k})}{H_k}$$

ハウスホルダー変換では, 1 回の相似変換に対し, 2 段の行列更新を行う. この行列 A_{k+1} は, A_k を直接使わずに表すことができる.

$$A_{k+1} = A_{k-1} - \boldsymbol{u}_{k-1}{\boldsymbol{v}_{k-1}}^T - \boldsymbol{v}_{k-1}{\boldsymbol{u}_{k-1}}^T - \boldsymbol{u}_k {\boldsymbol{v}_k}^T - \boldsymbol{v}_k {\boldsymbol{u}_k}^T$$

同様の操作によって, p 回の鏡像変換を行って得られる行列 A_{p+1} は,

$$A_{p+1} = A_1 - \sum_{i=1}^{p}(\boldsymbol{u}_i\boldsymbol{v}_i^T + \boldsymbol{v}_i\boldsymbol{u}_i^T)$$

となるので, A_1 に $2p$ 段の行列更新をまとめて行うことにより A_{p+1} が高速に求まる. なお, 行列 A がエルミート行列の場合, 転置記号 ^{T} はエルミート共役記号 * に読み替える. 詳細は, 参考文献 (6)(7) を参照されたい.

5.1.2.4 QR 法

3 重対角行列 T はユニタリ行列 Q と上三角行列 R(対角成分がすべて正になる) に

$$T = QR$$

と一意に分解される.

$$T_k = T$$

とおくと, T_k は $T_k \Rightarrow Q_kR_k$
と分解される. それを逆順に掛けて,

$$T_{k+1} \Leftarrow R_kQ_k = Q_k^*T_kQ_k \quad (Q_k^*\text{は } Q_k\text{の随伴行列}) \quad (k = 1, \cdots)$$

とすると, $T_1, T_2, \cdots, T_k, T_{k+1}$ はすべて 3 重対角行列であり, $k \to \infty$ とすると T_k は対角行列に収束し, その対角成分には T の固有値が並ぶ. 反復計算は非対角成分が十分小さくなったとき, 収束したと判断し, 反復を打ち切る.
実際の QR 法では, 収束を加速するために, μ_k を固有値の近似値として, T_k の代りに原点移動を行った $T_k - \mu_kI$ を作り,

$$T_k - \mu_kI = Q_kR_k$$

と分解する.
固有値の近似値の算定方法としては, 隣接固有値 (または絶対値の近い固有値) がある場合を考えて, 右下すみ小行列の固有値を無平方根 QR 法で求めて μ_k とする.
これから,

$$T_{k+1} = R_kQ_k + \mu_kI$$

を作ると,

$$T_{k+1} = Q_k^*T_kQ_k$$

となる. この操作を繰り返して収束させた後に, 原点移動量で補正したものが固有値となる.
3 重対角行列の変換する前の, もとの行列の固有ベクトルは, ハウスホルダー変換により 3 重対角行列 T を求めた際の変換行列を順に掛けてゆき, さらに QR 法によって得られた変換行列 $Q_1, Q_2, \cdots, Q_k$ を掛けあわせれば得られる. その行列の第 i 列は, 第 i 番目の固有値に対応する固有ベクトルに収束する.

5.1.2.5 無平方根 QR 法

実対称 3 重行列の固有値のみを求める場合は, QR 法の平方根計算を省いた無平方 QR 法が高速である.
対角要素を $\alpha_1, \cdots, \alpha_n$, 副対角要素を $\beta_1, \cdots, \beta_{n-1}$ とする. 計算中の変換行列の一成分を $P^{(i)}$ として $P^{(i)}$ の中の $\sin\theta$ と $\cos\theta$ を S_i, C_i とする.
QR 法では,

$$P_i = \alpha_iC_{i-1} - \beta_{i-1}S_{i-1}C_{i-2}$$

$$S_i = \frac{\beta_i}{\sqrt{P_i^2 + \beta_i^2}}$$

$$C_i = \frac{P_i}{\sqrt{P_i^2 + \beta_i^2}}$$

新$\alpha_{i-1} = \alpha_i + P_{i-1}C_{i-2} - P_iC_{i-1}$

新$\beta_{i-2} = S_{i-2}\sqrt{P_{i-1}^2 + \beta_{i-1}^2}$

と平方根の計算をしなければならないが, これを $P_i\beta_iS_iC_i$ 全て 2 乗の形で計算すると,

$$C_0 = 1, S_0 = 0, \gamma_1 = \alpha_1, P_1^2 = \alpha_1^2, \alpha_{n+1} = \beta_{n+1} = 0$$

とし

$$t_i^2 = P_i^2 + \beta_{i+1}^2$$

新$\beta_i^2 = S_{i-1}^2 t_i^2$

$$S_i^2 = \frac{\beta_{i+1}^2}{t_i^2}, C_i^2 = \frac{P_i^2}{t_i^2}$$

$$P_{i+1}^2 = \alpha_{i+1}^2 C_i^2 - 2\alpha_i + S_i^2\gamma_i + \beta_{i+1}^2 S_i^2 C_{i-1}^2$$

$$\gamma_{i+1} = \alpha_{i+1}C_i^2 = S_i^2\gamma_i$$

新$\alpha_i = \alpha_{i+1} + \gamma_i - \gamma_{i+1}$

となり, 平方根なしで計算できる. 詳細は, 文献 (13) を参照されたい.

5.1.2.6 バイセクション (Bisection) 法

実対称 3 重対角行列 T の固有値を大きい方, または小さい方から数個求める.
T の対角成分を $d_1, d_2, \cdots, d_n$, 副対角成分を $s_1, s_2, \cdots, s_{n-1}$ とし, λ を変数として,

$$f_0(\lambda) = 1$$

$$f_1(\lambda) = d_1 - \lambda$$

$$f_i(\lambda) = (d_i - \lambda)f_{i-1}(\lambda) - s_{i-1}^2 f_{i-2}(\lambda) \quad (i = 2, \cdots, n)$$

という関数列を作ると, $f_0(\lambda), f_1(\lambda), \cdots, f_m(\lambda)$ はスツルム (Sturm) 列をなす. つまり, ある λ に対する引き続く関数列の符号の不一致の個数を $L(\lambda)$ とすると, この $L(\lambda)$ は λ より小さな固有値の個数に等しい.
オーバフロー, アンダフローを防ぐため, 実際は $g_i(\lambda)$ を

$$g_i(\lambda) = \frac{f_i(\lambda)}{f_{i-1}(\lambda)} \quad (i = 1, 2, \cdots, n)$$

で定義すれば, 負になる $g_i(\lambda)$ の個数が $L(\lambda)$ となる. なお $g_i(\lambda)$ は,

$$g_1(\lambda) = d_1 - \lambda$$

$$g_i = (d_i - \lambda) - \frac{s_{i-1}^2}{g_{i-1}(\lambda)} \quad (i = 2, \cdots, n)$$

を満足する. もし, $g_{i-1}(\lambda) = 0$ となったときは

$$g_i(\lambda) = (d_i - \lambda) - \frac{|s_{i-1}|}{\varepsilon} \quad (\varepsilon : \text{誤差判定のための単位})$$

とする.
T の固有値を $\lambda_1 \leq \lambda_2 \leq \cdots\cdots \leq \lambda_m$ とする. **ゲルシュゴリン (Gerschgorin) の定理**より固有値全体の下限 ($x_{\min}$) および上限 ($x_{\max}$) は,

$$x_{\max} = \max(d_i + (|s_{i-1}| + |s_i|)) \quad (1 \leq i \leq n)$$

$$x_{\min} = \min(d_i - (|s_{i-1}| + |s_i|)) \quad (1 \leq i \leq n)$$

で与えられる. ただし, $s_0 = s_n = 0$ とする.
この $x_{\min}$, $x_{\max}$ をもとにして, 上記のように固有値の個数を数えながら区間分割を繰り返して, 固有値の存在範囲を小さくしていく. そうすれば微小区間の両端をある固有値に収束させることができる. この過程を大きい方 ($x_{\max}$) から, または小さい方 ($x_{\min}$) から繰り返せば, 必要なだけ固有値を求めることができる.
スツルム関数列, ゲルシュゴリンの定理については, 参考文献 (2), (8) 等を参照されたい.

5.1.2.7 ブロックアルゴリズムによる相似 (ユニタリ) 変換の累積

QR 法および逆反復法を用いて実対称行列の固有ベクトルを求める場合, ハウスホルダー変換の際に用いた相似 (ユニタリ) 変換行列の累積が必要になる. この累積計算を高速に行うには, ブロックアルゴリズムの適用が非常に有効である.
ハウスホルダー変換により実対称行列から 3 重対角行列を求めた際の変換行列 P_k を

$$P_k = \boldsymbol{I} - \frac{\boldsymbol{u}_k \boldsymbol{u}_k^T}{H_k}$$

$$H_k = \frac{1}{2} \boldsymbol{u}_k^T \boldsymbol{u}_k$$

とする. この変換行列 P_k の累積は, 次式のようになる.

$$P_1 P_2 \cdots P_{n-2} = \boldsymbol{I} - \Sigma_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T$$

ただし, $\boldsymbol{w}_i^T$ は, 以下の漸化式によって表される.

$$\boldsymbol{w}_{n-2}^T = \frac{\boldsymbol{u}_{n-2}^T}{H_{n-2}}$$

$$\boldsymbol{w}_i^T = \frac{\boldsymbol{u}_i^T - \sum_{j=i-1}^{n-2} (\boldsymbol{u}_i^T \boldsymbol{u}_j) \boldsymbol{w}_j^T}{H_i}$$

QR 法または逆反復法を用いて得られた実対称 3 重対角行列の固有ベクトルを $\boldsymbol{V}$ とする. もとの行列の固有ベクトル $\boldsymbol{X}$ は次式により得られる.

$$\boldsymbol{X} = P_1 \cdots P_{n-2} \boldsymbol{V}$$

$$= \boldsymbol{V} - \sum_{i=1}^{n-2} \boldsymbol{u}_i \boldsymbol{w}_i^T \boldsymbol{V}$$

相似 (ユニタリ) 変換行列 P_k と固有ベクトル $\boldsymbol{V}$ の積は, ランク-1 の行列更新である. したがって, 変換行列の累積は, 行列更新をまとめて行うことによって高速に計算できる. なお, もとの行列がエルミート行列の場合, 転置記号 ^{T} はエルミート共役記号 * に読み替える.

5.1.2.8 逆反復法

無平方根 QR 法またはバイセクション法で求まった固有値に対応する固有ベクトルを逆反復法で求める.
実対称 3 重対角行列 T のある固有値 λ_k の近似値 μ_k が求まったとする. このとき初期ベクトル $\boldsymbol{v}_0$ を適当にとって連立 1 次方程式,

$$(T-\mu_k I)\boldsymbol{v}_i = \boldsymbol{v}_{i-1} \quad (i=1,2,\cdots)$$

を繰り返し反復的に解いて, $\boldsymbol{v}_i$ が収束条件を満足したならそれを固有ベクトルとする.
連立 1 次方程式を解くのには, 部分軸選択を伴うガウス法による LU 分解を行い, 前進代入, 後退代入を用いて解く.

5.1.2.9 一般化固有値問題

エルミート行列の一般化固有値問題

$$A\boldsymbol{x} = \lambda B\boldsymbol{x} \quad (A:\text{エルミート}, B:\text{正値エルミート})$$

で, B をコレスキー分解して,

$$B = LL^*$$

とし,

$$(L^{-1}A(L^*)^{-1})(L^*\boldsymbol{x}) = \lambda(L^*\boldsymbol{x})$$

とする.

$$P = L^{-1}A(L^*)^{-1}$$

$$L^*\boldsymbol{x} = \boldsymbol{y}$$

とおけば, P はエルミート行列で

$$P\boldsymbol{y} = \lambda\boldsymbol{y}$$

という標準の固有値問題に変換される. 行列 A の固有ベクトルは,

$$\boldsymbol{x} = (L^*)^{-1}\boldsymbol{y}$$

である.
さらに, エルミート行列に対する一般化固有値問題 $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (B: 正定値) 以外の一般化固有値問題を, 正定値エルミート行列 B の位置で分けて

$$AB\boldsymbol{x} = \lambda\boldsymbol{x}$$

と

$$BA\boldsymbol{x} = \lambda\boldsymbol{x}$$

に分類する. これらを以下の手順で標準固有値問題に帰着し, その固有値 λ と固有ベクトル $\boldsymbol{x}$ を求める. エルミート行列の標準固有値問題に帰着させるために以下のような手順をとる.

① 正定値行列 B をコレスキー分解し, $B = L^*L$ とする (ここで, L は下三角行列である).

② $AB\boldsymbol{x} = \lambda\boldsymbol{x}$ は $C = LAL^*$ の固有値問題に帰着し, 固有ベクトルは L の逆行列を掛けることで得られる.

③ $BA\boldsymbol{x} = \lambda\boldsymbol{x}$ は $C = LAL^*$ の固有値問題に帰着し, 固有ベクトルは L^* を掛けることで得られる.

5.1.3 参考文献

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

(2) Wilkinson, J. H. , “The Algebraic Eigenvalue Problem”, Clarendon Press, Oxford, (1965).

(3) 別府良孝, “スーパーコンピュータに適した固有値ルーチン”, bit 臨時増刊 (名取, 野寺編), 共立出版, (1987).

(4) 別府良孝, “固有値問題の高速算法”, 中央情報教育研究所, TN871-12, (1987).

(5) 別府良孝, 井坂秀高, 竹内聖彦, “3 重対角行列の固有値を求めるための諸算法について”, 情報処理学会第 42 回全国大会論文誌, Vol. 1, PP. 63-64(1991).

(6) Dongarra J. J. , Sorensen D. C. , and Hammarling A. J. , “Block reduction of matrices to condensed forms for eigenvalue computations”, Journal of Computational and Applied Mathematics, Vol. 27, PP. 215-227(1989).

(7) Dongarra J. J. and van de Geijn R. A. , “Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures”, LAPACK Working Note 30, PP. 1-12(1991).

(8) Francis, J. G. F. , “The QR transformation, I, II”, Comput. J. 4, pp. 265-271, pp. 332-345(1961, 1962).

(9) Cuppen, J. J. M., “A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem”, Numer. Math. 36, pp. 177-195(1981).

(10) Gu, M. and Eisenstat, S. C., “A Stable and Efficient Algorithm for the rank-1 modification of the symmetric eigenproblem”, SIAM J. Matrix Anal. Appl. 15, pp. 1266-1276(1994).

(11) Gu, M. and Eisenstat, S. C., “A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem”, SIAM J. Matrix Anal. Appl. 16, pp. 172-191(1995).

(12) 戸川隼人, “マトリクスの数値計算”, オーム社, (1971).

(13) Y. Beppu and I. Ninomiya, “HQRII—A Fast Diagonalization Subroutine”, Computers and Chemistry Vol. 6(1982).

(14) 井坂秀高, 別府良孝, 竹内聖彦, “QR 法の原点移動方法の比較”, 第 20 回数値解析シンポジウム講演予稿集. (1991)

5.2 実対称行列 (2 次元配列型)(上三角型)

5.2.1 ASL_qcsmaa, ASL_pcsmaa
実対称行列の全固有値・全固有ベクトル

(1) **機 能**

実対称行列 A(2 次元配列型)(上三角型) の全固有値とそれに対応する全固有ベクトルをハウスホルダー法, QR 法により求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcsmaa (a, lna, n, e, w1, nt);

単精度関数:

ierr = ASL_pcsmaa (a, lna, n, e, w1, nt);

(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∗}	lna×n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	各固有値に対応する固有ベクトル (列ベクトル)
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
4	e	{D∗ / R∗}	n	出 力	固有値
5	w1	{D∗ / R∗}	n	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna

(b) nt≥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.	e [0] ←a [0] , a [0] ←1.0 とする.
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.	
5000+i	固有値, 固有ベクトルを求める段階で収束しなかった. (1≤i≤n)	e [0] , ···, e [i−2]) にそれまでに求まった固有値, a にそれに対応する固有ベクトルが入る. (ただし順不同)

(6) **注意事項**

(a) 配列 a には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルは正規直交系である.

(d) 固有ベクトルを必要としないときは, 5.2.2 $\left\{\begin{matrix}\text{ASL_qcsman}\\ \text{ASL_pcsman}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A = \begin{bmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{bmatrix}$$

の全固有値とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A, lna=11, n=4, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qcsmaa */

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

int main()
{
    double *a;
    int lda=11;
    int n;
    double *e;
    double *w1;
    int nt = 2;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

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

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

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

```
mod = n % 4;

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

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

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

printf( "\tn = %6d\n", n );
printf( "\tnt= %6d\n\n", nt );
printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+lda*j] );
    }
}

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

fclose( fp );


ierr = ASL_qcsmaa(a, lda, n, e, w1, nt);


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

for( k=0 ; k<n-3 ; k = k+4 )
{
    printf( "\n" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvalue " );
    }
    printf( "\n" );
    for( i=k ; i< k+4 ; i++ )
    {
        printf( "\t%8.3g     ", e[i] );
    }
    printf( "\n" );

    for( i=0 ; i<4 ; i++ )
    {
        printf( "\tEigenvector " );
    }
    printf( "\n" );
    for( j=0 ; j<n ; j++ )
    {
        for( i=k ; i< k+4 ; i++ )
        {
            printf( "\t%8.3g     ", a[j+lda*i] );
        }
        printf( "\n" );
    }
}

if( mod != 0 )
{
    printf( "\n" );
    for( i=n-mod ; i<n ; i++ )
    {
        printf( "\tEigenvalue " );
    }
    printf( "\n" );
    for( i=n-mod ; i<n ; i++ )
    {
```

```
                printf( "\t%8.3g     ", e[i] );
            }
            printf( "\n" );

            for( i=n-mod ; i<n ; i++ )
            {
                printf( "\tEigenvector " );
            }
            printf( "\n" );
            for( j=1 ; j<n ; j++ )
            {
                for( i=n-mod ; i<n ; i++ )
                {
                    printf( "\t%8.3g     ", a[j+lda*i] );
                }
                printf( "\n" );
            }
        }

        free( a );
        free( e );
        free( w1 );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_qcsmaa ***

 ** Input **

n =      4
nt=      2

Input Matrix a

       6         4         4         1
       4         6         1         4
       4         1         6         4
       1         4         4         6

 ** Output **

ierr =      0

Eigenvalue  Eigenvalue  Eigenvalue  Eigenvalue
      -1           5           5          15
Eigenvector     Eigenvector     Eigenvector     Eigenvector
     0.5           0.707           0             0.5
    -0.5        2.78e-17          -0.707         0.5
    -0.5        -1.67e-16          0.707         0.5
     0.5          -0.707        8.33e-17         0.5
```

5.2.2 ASL_qcsman, ASL_pcsman 実対称行列の全固有値

(1) **機 能**

実対称行列 A(2 次元配列型)(上三角型) の全固有値をハウスホルダー法, 無平方根 QR 法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qcsman  (a, lna, n, e, w1, nt);
```

単精度関数:

```
ierr = ASL_pcsman  (a, lna, n, e, w1, nt);
```

(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∗}	lna×n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
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
4	e	{D∗ / R∗}	n	出 力	固有値
5	w1	{D∗ / R∗}	n	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) $0<n\leq lna$

(b) $nt\geq 1$

(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.	e [0] ←aa [0] とする.
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.	
5000+i	固有値を求める段階で収束しなかった. (1≤i≤n).	e [0] , ⋯, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 配列 a には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

5.2.3 ASL_qcsmss, ASL_pcsmss
実対称行列の固有値・固有ベクトル

(1) **機 能**

実対称行列 A(2 次元配列型)(上三角型) の固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求め, それに対応する固有ベクトルを逆反復法により求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcsmss (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, nt);

単精度関数:

ierr = ASL_pcsmss (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, nt);

(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∗}	lna×n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
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
4	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
5	e	{D∗ / R∗}	m	出 力	固有値
6	m	I	1	入 力	求めたい固有値の個数 m
7	ve	{D∗ / R∗}	lnv×m	出 力	各固有値に対応する固有ベクトル (列ベクトル)
8	lnv	I	1	入 力	配列 ve の整合寸法
9	isw	I	1	入 力	処理スイッチ isw≥0 : 大きい方から固有値を求める. isw<0 : 小さい方から固有値を求める.
10	iw1	I*	m	出 力	固有ベクトルフラグ (注意事項 (e) 参照)
11	w1	{D∗ / R∗}	8×n	ワーク	作業領域
12	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna, lnv

(b) 0<m≤n

(c) nt≥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.	e [0] ←a [0] , ve [0] ←1.0 とする.
2000	固有ベクトルを求める逆反復で最大反復回数をこえた.	固有ベクトルに一部精度の低いものがあるが, 処理は続行する. (注意事項 (e) 参照)
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 a には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分け計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

(e) 逆反復法で最大反復回数をこえた場合 (ierr=2000 出力時) について

- iw1[$i-1$] = 0 の場合 :
  i 番目の固有ベクトル計算は正常終了している.
- iw1[$i-1$] ≠ 0 の場合 :
  i 番目の固有ベクトル計算は収束条件が満たされず, 固有ベクトルの精度は低い.
  この場合 iw1[$i-1$] は, 反復回数が設定される.

なお正常終了時 (ierr=0 出力時) は, iw1[$i-1$] ← 0 が設定される.

(f) 固有ベクトルは正規直交系である.

(g) 固有ベクトルを必要としないときは, 5.2.4 $\left\{\begin{matrix}\text{ASL_qcsmsn}\\ \text{ASL_pcsmsn}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

の固有値を小さい順に 3 個とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A, lna=11, n=6, eps=-1.0, m=3, lnv=11, isw=-1, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qcsmss */

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

int main()
{
    double *a;
    int lda=11;
    int n;
    double ceps = -1.0;
    double *e;
    int m;
    double *ve;
    int ldv=11;
    int isw = -1;
    int *iw1;
    double *w1;
    int nt = 2;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 4;

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

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

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

    iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( iw1 == NULL )
    {
        printf( "no enough memory for array iw1\n" );
        return -1;
    }

    w1 = ( double * )malloc((size_t)( sizeof(double) * (8*n) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

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

    printf( "\n\tInput Matrix a\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=i ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+lda*j] );
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
```

```
            for( j=0 ; j<i ; j++ )
            {
                printf( "%8.3g  ", a[j+lda*i] );
            }
            for( j=i ; j<n ; j++ )
            {
                printf( "%8.3g  ", a[i+lda*j] );
            }
            printf( "\n" );
        }

        fclose( fp );

        ierr = ASL_qcsmss(a, lda, n, ceps, e, m, ve, ldv, isw, iw1, w1, nt);

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

        for( k=0 ; k<m-3 ; k = k+4 )
        {
            printf( "\n" );
            for( i=0 ; i<4 ; i++ )
            {
                printf( "\tEigenvalue  " );
            }
            printf( "\n" );
            for( i=k ; i<k+4 ; i++ )
            {
                printf( "\t%8.3g     ", e[i] );
            }
            printf( "\n" );

            for( i=0 ; i<4 ; i++ )
            {
                printf( "\tEigenvector " );
            }
            printf( "\n" );
            for( j=0 ; j<n ; j++ )
            {
                for( i=k ; i<k+4 ; i++ )
                {
                    printf( "\t%8.3g     ", ve[j+ldv*i] );
                }
                printf( "\n" );
            }
        }

        if( mod != 0 )
        {
            printf( "\n" );
            for( i= m-mod ; i<m ; i++ )
            {
                printf( "\tEigenvalue  " );
            }
            printf( "\n" );
            for( i= m-mod ; i<m ; i++ )
            {
                printf( "\t%8.3g     ", e[i] );
            }
            printf( "\n" );

            for( i= m-mod ; i<m ; i++ )
            {
                printf( "\tEigenvector " );
            }
            printf( "\n" );
            for( j=0 ; j<n ; j++ )
            {
                for( i= m-mod ; i<m ; i++ )
                {
                    printf( "\t%8.3g     ", ve[j+ldv*i] );
                }
                printf( "\n" );
            }

        }

        free( a );
        free( e );
        free( ve );
        free( iw1 );
        free( w1 );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_qcsmss ***

     ** Input **

    n =      4
```

```
m =       3
nt=       2


Input Matrix a

       4         1         0         0
       1         3         1         0
       0         1         3         1
       0         0         1         4

 ** Output **

ierr =       0

Eigenvalue       Eigenvalue       Eigenvalue
     1.59                3             4.41
Eigenvector      Eigenvector      Eigenvector
  -0.271             0.5           -0.653
   0.653            -0.5           -0.271
  -0.653            -0.5            0.271
   0.271             0.5            0.653
```

5.2.4 ASL_qcsmsn, ASL_pcsmsn
実対称行列の固有値

(1) **機 能**

実対称行列 A(2 次元配列型)(上三角型) の固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qcsmsn  (a, lna, n, eps, e, m, isw, w1, nt);
```

単精度関数:

```
ierr = ASL_pcsmsn  (a, lna, n, eps, e, m, isw, w1, nt);
```

(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∗}	lna×n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
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
4	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
5	e	{D∗ / R∗}	m	出 力	固有値
6	m	I	1	入 力	求めたい固有値の個数 m
7	isw	I	1	入 力	処理スイッチ isw≥0 : 大きい方から固有値を求める. isw<0 : 小さい方から固有値を求める.
8	w1	{D∗ / R∗}	5×n	ワーク	作業領域
9	nt	I	1	入 力	生成するタスク数
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) $0<n\leq lna$

(b) $0<m\leq n$

(c) $nt\geq 1$

(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.	e [0] ←a [0] とする.
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 a には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

5.3 エルミート行列 (2 次元配列型) (上三角型) (実数引数型)

5.3.1 ASL_hchraa, ASL_gchraa
エルミート行列の全固有値・全固有ベクトル

(1) **機 能**

エルミート行列 A=(ar, ai)(2 次元配列型)(上三角型)(実数引数型) の全固有値とそれに対応する全固有ベクトルをハウスホルダー法, QR 法により求める.

(2) **使用法**

倍精度関数:

ierr = ASL_hchraa (ar, ai, lna, n, e, vr, vi, lnv, w1, nt);

単精度関数:

ierr = ASL_gchraa (ar, ai, lna, n, e, vr, vi, lnv, w1, nt);

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	e	{D∗ / R∗}	n	出 力	固有値
6	vr	{D∗ / R∗}	lnv × n	出 力	各固有値に対応する固有ベクトルの実部 (列ベクトル)
7	vi	{D∗ / R∗}	lnv × n	出 力	各固有値に対応する固有ベクトルの虚部 (列ベクトル)
8	lnv	I	1	入 力	配列 vr, vi の整合寸法
9	w1	{D∗ / R∗}	3×n	ワーク	作業領域
10	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna, lnv

(b) nt≥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.	e [0] ←a [0] , vr [0] ←1.0, vi [0] ←0.0 とする.
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.	
5000+i	固有値を求める段階で収束しなかった. (1≤i≤n)	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同). このとき固有ベクトルは求まらない.

(6) **注意事項**

(a) 配列 ar, ai には, エルミート行列の実部, 虚部のそれぞれ上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルは正規直交系である.

(d) 固有ベクトルを必要としないときは, 5.3.2 $\left\{\begin{matrix}\text{ASL_hchran}\\ \text{ASL_gchran}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A=\begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

の全固有値とそれに対応する固有ベクトルを求める.

(b) 入力データ
行列 A の実部 ar, 虚部 ai, lna=11, n=4, lnv=10, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_hchraa */

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

int main()
{
    double *ar;
    double *ai;
    int lda=11;
    int n;
    double *e;
    double *vr;
    double *vi;
    int ldv=11;
    double *w1;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

    fp = fopen( "hchraa.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
```

```
    return -1;
}

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

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

mod = n % 2;

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

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

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

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

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

w1 = ( double * )malloc((size_t)( sizeof(double) * (3*n) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\tn = %6d\n", n );
printf( "\tnt= %6d\n\n", nt );

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+lda*j] );
        fscanf( fp, "%lf", &ai[i+lda*j] );
    }
}

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

fclose( fp );


ierr = ASL_hchraa(ar, ai, lda, n, e, vr, vi, ldv, w1, nt);


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

for( j=0 ; j<n-1 ; j = j+2 )
{
    printf( "\n" );
    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvalue              " );
    }
    printf( "\n" );
```

```
            printf( "\t%8.3g                \t%8.3g\n",
                e[j], e[j+1] );

            for( i=0 ; i<2 ; i++ )
            {
                printf( "\tEigenvector             " );
            }
            printf( "\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
                    vr[i+ldv*j], vi[i+ldv*j], vr[i+ldv*(j+1)],vi[i+ldv*(j+1)] );
            }
        }

        if( mod != 0 )
        {
            printf( "\n" );
            printf( "\tEigenvalue\n" );
            printf( "\t%8.3g\n", e[n-1] );
            printf( "\tEigenvector\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t%8.3g , %8.3g\n",
                        vr[i+ldv*(n-1)], vi[i+ldv*(n-1)] );
            }
        }

        free( ar );
        free( ai );
        free( e );
        free( vr );
        free( vi );
        free( w1 );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_hchraa ***

 ** Input **

n =      4
nt=      2


Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
       0                           8
Eigenvector                 Eigenvector
     0.5 ,        0          -0.707 ,        0
    -0.5 ,        0         6.38e-16 ,        0
1.67e-16 ,      0.5            0.354 ,    0.354
-1.11e-16 ,      0.5          -0.354 ,    0.354

Eigenvalue                  Eigenvalue
       8                          12
Eigenvector                 Eigenvector
       0 ,        0             0.5 ,        0
 -0.0999 ,      0.7             0.5 ,        0
    -0.3 ,     -0.4             0.5 , -5.55e-16
    -0.4 ,      0.3            -0.5 , -4.44e-16
```

5.3.2 ASL_hchran, ASL_gchran
エルミート行列の全固有値

(1) **機 能**

エルミート行列 A=(ar, ai)(2 次元配列型)(上三角型)(実数引数型) の全固有値をハウスホルダー法, 無平方根 QR 法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchran  (ar, ai, lna, n, e, w1, nt);
```

単精度関数:

```
ierr = ASL_gchran  (ar, ai, lna, n, e, w1, nt);
```

(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	ar	{D∗ R∗}	lna × n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
2	ai	{D∗ R∗}	lna × n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	e	{D∗ R∗}	n	出 力	固有値
6	w1	{D∗ R∗}	3×n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna

(b) nt≥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.	e [0] ←ar [0] とする.
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.	
5000+i	固有値を求める段階で収束しなかった. (1≤i≤n)	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 配列 ar, ai には, エルミート行列の実部, 虚部のそれぞれ上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は小さい順に格納される.

5.3.3 ASL_hchrss, ASL_gchrss
エルミート行列の固有値・固有ベクトル

(1) **機 能**

エルミート行列 A=(ar, ai)(2 次元配列型)(上三角型)(実数引数型) の全固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求め, それに対応する固有ベクトルを逆反復法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchrss  (ar, ai, lna, n, eps, e, m, vr, vi, lnv, isw, iw1, w1, nt);
```

単精度関数:

```
ierr = ASL_gchrss  (ar, ai, lna, n, eps, e, m, vr, vi, lnv, isw, iw1, w1, nt);
```

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
6	e	{D∗ / R∗}	m	出 力	固有値
7	m	I	1	入 力	求めたい固有値の個数 m
8	vr	{D∗ / R∗}	lnv × m	出 力	各固有値に対応する固有ベクトルの実部 (列ベクトル)
9	vi	{D∗ / R∗}	lnv × m	出 力	各固有値に対応する固有ベクトルの虚部 (列ベクトル)
10	lnv	I	1	入 力	配列 vr, vi の整合寸法
11	isw	I	1	入 力	処理スイッチ isw≥0:大きい方から固有値を求める. isw<0:小さい方から固有値を求める.
12	iw1	I*	m	出 力	固有ベクトルフラグ (注意事項 (e) 参照)
13	w1	{D∗ / R∗}	10×n	ワーク	作業領域
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) 0<n≤lna, lnv

(b) 0<m≤n

(c) nt≥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.	e [0] ←ar [0] , vr [0] ←1.0, vi [0] ←0.0 とする.
2000	固有ベクトルを求める逆反復で最大反復回数をこえた.	固有ベクトルに一部精度の低いものがあるが, 処理は続行する. (注意事項 (e) 参照)
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 ar, ai には, エルミート行列の実部, 虚部のそれぞれ上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

(e) 逆反復法で最大反復回数をこえた場合 (ierr=2000 出力時) について

- iw1$[i-1] = 0$ の場合 :
  i 番目の固有ベクトル計算は正常終了している.
- iw1$[i-1] \neq 0$ の場合 :
  i 番目の固有ベクトル計算は収束条件が満たされず, 固有ベクトルの精度は低い.
  この場合 iw1$[i-1]$ は, 反復回数が設定される.

なお正常終了時 (ierr=0 出力時) は, iw1$[i-1] \leftarrow 0$ が設定される.

(f) 固有ベクトルは正規直交系である.

(g) 固有ベクトルを必要としないときは, 5.3.4 $\left\{\begin{matrix}\text{ASL_hchrsn}\\ \text{ASL_gchrsn}\end{matrix}\right\}$を使用する.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

のとき, エルミート行列 A の固有値を大きい順に 3 個とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A の実部 ar, 虚部 ai, lna=11, n=4, eps=−1.0, m=3, lnv=11,

isw=1, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_hchrss */

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

int main()
{
    double *ar;
    double *ai;
    int lda=11;
    int n;
    double ceps= -1.0;
    double *e;
    int m;
    double *vr;
    double *vi;
    int ldv=11;
    int isw=1;
    int *iw1;
    double *w1;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 2;

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

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

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

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

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

iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
if( iw1 == NULL )
{
    printf( "no enough memory for array iw1\n" );
    return -1;
}

w1 = ( double * )malloc((size_t)( sizeof(double) * (10*n) ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

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

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+lda*j] );
        fscanf( fp, "%lf", &ai[i+lda*j] );
    }
}

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

fclose( fp );

ierr = ASL_hchrss(ar, ai, lda, n, ceps, e, m, vr, vi, ldv, isw, iw1, w1, nt);

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

for( j=0 ; j<m-1 ; j = j+2 )
{
    printf( "\n" );
    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvalue              " );
    }
    printf( "\n" );
    printf( "\t%8.3g                \t%8.3g\n",
        e[j], e[j+1] );

    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvector             " );
    }
    printf( "\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
            vr[i+ldv*j], vi[i+ldv*j], vr[i+ldv*(j+1)],vi[i+ldv*(j+1)] );
    }
}

if( mod != 0 )
{
    printf( "\n" );
    printf( "\tEigenvalue\n" );
    printf( "\t%8.3g\n", e[m-1] );
    printf( "\tEigenvector\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g , %8.3g\n",
                vr[i+ldv*(m-1)], vi[i+ldv*(m-1)] );
    }
}

free( ar );
```

```
    free( ai );
    free( e );
    free( vr );
    free( vi );
    free( iw1 );
    free( w1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_hchrss ***

 ** Input **

n =      4
m =      3
nt=      2


Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
      12                          8
Eigenvector                 Eigenvector
     0.5 ,        0             0 ,        0
     0.5 , 1.02e-16          -0.0999 ,      0.7
     0.5 , -5.41e-16            -0.3 ,     -0.4
    -0.5 , -4.58e-16            -0.4 ,      0.3

Eigenvalue
       8
Eigenvector
   0.707 ,        0
-6.66e-16 , -5.09e-17
  -0.354 ,   -0.354
   0.354 ,   -0.354
```

5.3.4 ASL_hchrsn, ASL_gchrsn
エルミート行列の固有値

(1) **機 能**

エルミート行列 A=(ar, ai)(2 次元配列型)(上三角型)(実数引数型) の固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchrsn  (ar, ai, lna, n, eps, e, m, isw, w1, nt);
```

単精度関数:

```
ierr = ASL_gchrsn  (ar, ai, lna, n, eps, e, m, isw, w1, nt);
```

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚部 (2 次元配列型) (上三角型)
				出 力	入力時の内容は保存されない.
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
6	e	{D∗ / R∗}	m	出 力	固有値
7	m	I	1	入 力	求めたい固有値の個数 m
8	isw	I	1	入 力	処理スイッチ isw≥0:大きい方から固有値を求める. isw<0:小さい方から固有値を求める.
9	w1	{D∗ / R∗}	5×n	ワーク	作業領域
10	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna

(b) 0<m≤n

(c) nt≥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.	e [0] ←ar [0] とする.
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 ar, ai には, エルミート行列の実部, 虚部のそれぞれ上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

5.4 エルミート行列 (2 次元配列型)(上三角型)(複素引数型)

5.4.1 ASL_hcheaa, ASL_gcheaa
エルミート行列の全固有値・全固有ベクトル

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型)(複素引数型) の全固有値とそれに対応する全固有ベクトルをハウスホルダー法, QR 法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hcheaa  (a, lna, n, e, w1, w2, nt);
```

単精度関数:

```
ierr = ASL_gcheaa  (a, lna, n, e, w1, w2, nt);
```

(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	{Z∗ / C∗}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	各固有値に対応する固有ベクトル (列ベクトル)
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
4	e	{D∗ / R∗}	n	出 力	固有値
5	w1	{D∗ / R∗}	n	ワーク	作業領域
6	w2	{Z∗ / C∗}	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0<n\leq lna$

(b) $nt\geq 1$

(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.	e[0] ← creal(a[0]), a [0] ←(1.0, 0.0) とする.
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.	
5000+i	固有値を求める段階で収束しなかった. (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同). このとき固有ベクトルは求まらない.

(6) **注意事項**

(a) 配列 a には, エルミート行列の上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルは正規直交系である.

(d) 固有ベクトルを必要としないときは, 5.4.2 $\left\{\begin{matrix}\text{ASL_hchean}\\ \text{ASL_gchean}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A = \begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

の全固有値とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A, lna=11, n=4

(c) 主プログラム

```
/*      C interface example for ASL_hcheaa */

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

int main()
{
    double _Complex *a;
    int lda=11;
    int n;
    double *e;
    double *w1;
    double _Complex *w2;
    int nt = 2;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

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

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

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

mod = n % 2;

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

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

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

w2 = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
if( w2 == NULL )
{
    printf( "no enough memory for array w2\n" );
    return -1;
}

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

printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        double tmp_re, tmp_im;
        fscanf( fp, "%lf", &tmp_re );
        fscanf( fp, "%lf", &tmp_im );
        a[i+lda*j] = tmp_re + tmp_im * _Complex_I;
    }
}

for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<i ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", creal(a[j+lda*i]), -cimag(a[j+lda*i]) );
    }
    for( j=i ; j<n ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", creal(a[i+lda*j]), cimag(a[i+lda*j]) );
    }
    printf( "\n" );
}

fclose( fp );


ierr = ASL_hcheaa(a, lda, n, e, w1, w2, nt);


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

for( j=0 ; j<n-1 ; j = j+2 )
{
    printf( "\n" );
    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvalue              " );
    }
    printf( "\n" );
    printf( "\t%8.3g                  \t%8.3g\n",
        e[j], e[j+1] );

    for( i=0 ; i<2 ; i++ )
    {
        printf( "\tEigenvector             " );
    }
    printf( "\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
            creal(a[i+lda*j]), cimag(a[i+lda*j]), creal(a[i+lda*(j+1)]), cimag(a[i+lda*(j+1)]) );
    }
}

if( mod != 0 )
{
```

```
            printf( "\n" );
            printf( "\tEigenvalue\n" );
            printf( "\t%8.3g\n", e[n-1] );
            printf( "\tEigenvector\n" );
            for( i=0 ; i<n ; i++ )
            {
                printf( "\t%8.3g , %8.3g\n",
                        creal(a[i+lda*(n-1)]), cimag(a[i+lda*(n-1)]) );
            }
        }

        free( a );
        free( e );
        free( w1 );
        free( w2 );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_hcheaa ***

 ** Input **

n =      4

Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
       0                           8
Eigenvector                 Eigenvector
     0.5 ,        0           -0.707 ,        0
    -0.5 ,        0          6.11e-16 ,        0
2.22e-16 ,      0.5            0.354 ,    0.354
-1.11e-16 ,      0.5           -0.354 ,    0.354

Eigenvalue                  Eigenvalue
       8                          12
Eigenvector                 Eigenvector
       0 ,        0             0.5 ,        0
 -0.0999 ,      0.7             0.5 ,        0
    -0.3 ,     -0.4             0.5 , -5.55e-16
    -0.4 ,      0.3            -0.5 , -4.44e-16
```

5.4.2 ASL_hchean, ASL_gchean
エルミート行列の全固有値

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型)(複素引数型) の全固有値をハウスホルダー法, 無平方根 QR 法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchean  (a, lna, n, e, w1, w2, nt);
```

単精度関数:

```
ierr = ASL_gchean  (a, lna, n, e, w1, w2, nt);
```

(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	{Z∗ / C∗}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保存されない.
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
4	e	{D∗ / R∗}	n	出 力	固有値
5	w1	{D∗ / R∗}	n	ワーク	作業領域
6	w2	{Z∗ / C∗}	n	ワーク	作業領域
7	nt	I	1	入 力	生成するタスク数
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) $0<n\leq lna$

(b) $nt\geq 1$

(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.	e[0] ← creal(a[0]) とする.
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.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 配列 a には, エルミート行列の上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は小さい順に格納される.

5.4.3 ASL_hchess, ASL_gchess
エルミート行列の固有値・固有ベクトル

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型)(複素引数型) の固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求め, それに対応する固有ベクトルを逆反復法により求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchess  (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, w2, nt);
```

単精度関数:

```
ierr = ASL_gchess  (a, lna, n, eps, e, m, ve, lnv, isw, iw1, w1, w2, nt);
```

(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	{Z∗ / C∗}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保存されない.
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
4	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
5	e	{D∗ / R∗}	m	出 力	固有値
6	m	I	1	入 力	求めたい固有値の個数 m
7	ve	{Z∗ / C∗}	lnv × m	出 力	各固有値に対応する固有ベクトル (列ベクトル)
8	lnv	I	1	入 力	配列 ve の整合寸法
9	isw	I	1	入 力	処理スイッチ isw≥0 : 大きい方から固有値を求める. isw<0 : 小さい方から固有値を求める.
10	iw1	I*	m	出 力	固有ベクトルフラグ (注意事項 (e) 参照)
11	w1	{D∗ / R∗}	8×n	ワーク	作業領域
12	w2	{Z∗ / C∗}	n	ワーク	作業領域
13	nt	I	1	入 力	生成するタスク数
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) 0<n≤lna, lnv

(b) 0<m≤n

(c) nt≥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.	e[0] ← creal(a[0]), ve [0] ←(1.0, 0.0) とする.
2000	固有ベクトルを求める逆反復で最大反復回数をこえた.	固有ベクトルに一部精度の低いものがあるが, 処理は続行する. (注意事項 (e) 参照)
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 a には, エルミート行列の上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

(e) 逆反復法で最大反復回数をこえた場合 (ierr=2000 出力時) について

- iw1[$i-1$] $=0$ の場合：
  i 番目の固有ベクトル計算は正常終了している.
- iw1[$i-1$] $\neq 0$ の場合：
  i 番目の固有ベクトル計算は収束条件が満たされず, 固有ベクトルの精度は低い.
  この場合 iw1[$i-1$] は, 反復回数が設定される.

なお正常終了時 (ierr=0 出力時) は, iw1[$i-1$] ← 0 が設定される.

(f) 固有ベクトルは正規直交系である.

(g) 固有ベクトルを必要としないときは, 5.4.4 $\left\{\begin{matrix}\text{ASL_hchesn}\\ \text{ASL_gchesn}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A=\begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$$

のとき, エルミート行列 A の固有値を大きい順に 3 個とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A, lna=11, n=4, eps=-1.0, m=3, lnv=11, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_hchess */

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

int main()
{
    double _Complex *a;
    int lda=11;
    int n;
    double ceps= -1.0;
    double *e;
    int m;
    double _Complex *ve;
    int ldv=11;
    int isw=1;
    int *iw1;
    double *w1;
    double _Complex *w2;
    int nt=2;
    int ierr;
    int i,j;
    FILE *fp;

    int mod;

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

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

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

    mod = m % 2;

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

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

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

    iw1 = ( int * )malloc((size_t)( sizeof(int) * m ));
    if( iw1 == NULL )
    {
        printf( "no enough memory for array iw1\n" );
        return -1;
    }

    w1 = ( double * )malloc((size_t)( sizeof(double) * (8*n) ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        return -1;
    }

    w2 = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * n ));
    if( w2 == NULL )
    {
        printf( "no enough memory for array w2\n" );
        return -1;
    }

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

    printf( "\n\tInput Matrix a ( Real , Imaginary )\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=i ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
```

```
            a[i+lda*j] = tmp_re + tmp_im * _Complex_I;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<i ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[j+lda*i]), -cimag(a[j+lda*i]) );
        }
        for( j=i ; j<n ; j++ )
        {
            printf( "(%8.3g , %8.3g) ", creal(a[i+lda*j]), cimag(a[i+lda*j]) );
        }
        printf( "\n" );
    }

    fclose( fp );


    ierr = ASL_hchess(a, lda, n, ceps, e, m, ve, ldv, isw, iw1, w1, w2, nt);


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

    for( j=0 ; j<m-1 ; j = j+2 )
    {
        printf( "\n" );
        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvalue              " );
        }
        printf( "\n" );
        printf( "\t%8.3g                 \t%8.3g\n",
            e[j], e[j+1] );

        for( i=0 ; i<2 ; i++ )
        {
            printf( "\tEigenvector             " );
        }
        printf( "\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g      \t%8.3g , %8.3g\n",
                creal(ve[i+ldv*j]), cimag(ve[i+ldv*j]), creal(ve[i+ldv*(j+1)]), cimag(ve[i+ldv*(j+1)]) );
        }
    }

    if( mod != 0 )
    {
        printf( "\n" );
        printf( "\tEigenvalue\n" );
        printf( "\t%8.3g\n", e[m-1] );
        printf( "\tEigenvector\n" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "\t%8.3g , %8.3g\n",
                    creal(ve[i+ldv*(m-1)]), cimag(ve[i+ldv*(m-1)]) );
        }
    }

    free( a );
    free( e );
    free( ve );
    free( iw1 );
    free( w1 );
    free( w2 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_hchess ***

 ** Input **

n =      4
m =      3

Input Matrix a ( Real , Imaginary )

(       7 ,        0) (       3 ,        0) (       1 ,        2) (      -1 ,        2)
(       3 ,        0) (       7 ,        0) (       1 ,       -2) (      -1 ,       -2)
(       1 ,       -2) (       1 ,        2) (       7 ,        0) (      -3 ,        0)
(      -1 ,       -2) (      -1 ,        2) (      -3 ,        0) (       7 ,        0)

 ** Output **

ierr =      0

Eigenvalue                  Eigenvalue
```

```
      12                        8
Eigenvector               Eigenvector
     0.5 ,        0             0 ,        0
     0.5 , 1.02e-16       -0.0999 ,      0.7
     0.5 ,   -5e-16          -0.3 ,     -0.4
    -0.5 , -4.44e-16         -0.4 ,      0.3

Eigenvalue
       8
Eigenvector
   0.707 ,        0
-6.66e-16 ,        0
  -0.354 ,   -0.354
   0.354 ,   -0.354
```

5.4.4 ASL_hchesn, ASL_gchesn
エルミート行列の固有値

(1) **機 能**

エルミート行列 A(2 次元配列型)(上三角型)(複素引数型) の固有値をハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hchesn  (a, lna, n, eps, e, m, isw, w1, w2, nt);
```

単精度関数:

```
ierr = ASL_gchesn  (a, lna, n, eps, e, m, isw, w1, w2, nt);
```

(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	{Z∗ / C∗}	lna × n	入 力	エルミート行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保存されない.
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
4	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
5	e	{D∗ / R∗}	m	出 力	固有値
6	m	I	1	入 力	求めたい固有値の個数 m
7	isw	I	1	入 力	処理スイッチ isw≥0 : 大きい方から固有値を求める. isw<0 : 小さい方から固有値を求める.
8	w1	{D∗ / R∗}	2×n	ワーク	作業領域
9	w2	{Z∗ / C∗}	n	ワーク	作業領域
10	nt	I	1	入 力	生成するタスク数
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) $0<n\leq lna$

(b) $0<m\leq n$

(c) $nt\geq 1$

(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.	e[0] ← creal(a[0]) とする.
3000	制限条件 (a) または (b) 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.	

(6) **注意事項**

(a) 配列 a には, エルミート行列の上三角部分のみが格納されていればよい (付録 A 参照).

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

5.5 実対称行列 (2 次元配列型) (上三角型) の一般化固有値問題 ($Ax = \lambda Bx$)

5.5.1 ASL_qcgsaa, ASL_pcgsaa
実対称行列 (一般化固有値問題 $Ax = \lambda Bx$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型) の一般化固有値問題 $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: 実対称行列 , B: 正値実対称行列) をコレスキー法を用いて標準の固有値問題に変換し, ハウスホルダー法, QR 法により全固有値とそれに対応する全固有ベクトルを求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qcgsaa  (a, lna, n, b, lnb, e, w1, nt);
```

単精度関数:

```
ierr = ASL_pcgsaa  (a, lna, n, b, lnb, e, w1, nt);
```

(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∗}	lna × n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	各固有値に対応する固有ベクトル (列ベクトル)
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 A, B の次数
4	b	{D∗ R∗}	lnb × n	入 力	正値対称行列 B(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ R∗}	n	出 力	固有値
7	w1	{D∗ R∗}	2×n	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna, lnb

(b) nt≥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.	e [0] ← a [0] /b [0] , a [0] ← 1.0/$\sqrt{\text{b[0]}}$ とする.
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	固有ベクトルの精度が低い可能性があるが, 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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同). このとき固有ベクトルは求まらない.

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(d) 固有ベクトルを必要としないときは, 5.5.2 $\left\{\begin{matrix}\text{ASL_qcgsan}\\ \text{ASL_pcgsan}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

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

$$B = \begin{bmatrix} 153 & 31 & 58 & -58 \\ 31 & 153 & -53 & 58 \\ 58 & -58 & 153 & 31 \\ -58 & 58 & 31 & 153 \end{bmatrix}$$

のとき, $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ の全固有値とそれに対応する固有ベクトルを求める.

(b) 入力データ
行列 A, lna=11, n=4, 行列 B, lnb=11, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qcgsaa */

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

int main()
{
    double *a;
    int na=11;
    int nn;
    double *b;
    int nb=11;
```

```
double *e;
double *wk;
int nt;
int ierr;
int i,j,k;
FILE *fp;

int mod;

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

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

fscanf( fp, "%d", &nn );
fscanf( fp, "%d", &nt );

mod = nn % 4;

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

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

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

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

printf( "\tn = %6d\n", nn );
printf( "\tnt= %6d\n\n", nt );

printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &a[i+na*j] );
    }
}

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

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

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

```
            printf( "%8.3g", b[i+nb*j] );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_qcgsaa(a, na, nn, b, nb, e, wk, nt);

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

    for( k=0 ; k<nn-3 ; k = k+4 )
    {
        printf( "\n\t" );
        for( i=0 ; i<4 ; i++ )
        {
            printf( "Eigenvalue   " );
        }
        printf( "\n\t" );
        for( i=k ; i<k+4 ; i++ )
        {
            printf( " %8.3g    ", e[i] );
        }
        printf( "\n" );

        printf( "\t" );
        for( i=0 ; i<4 ; i++ )
        {
            printf( "Eigenvector  " );
        }
        printf( "\n" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "\t" );
            for( i=k ; i<k+4 ; i++ )
            {
                printf( " %8.3g    ", a[j+na*i] );
            }
            printf( "\n" );
        }
    }

    if( mod != 0 )
    {
        printf( "\n\t" );
        for( i= nn-mod ; i<nn ; i++ )
        {
            printf( "Eigenvalue  " );
        }
        printf( "\n\t" );
        for( i= nn-mod ; i<nn ; i++ )
        {
            printf( " %8.3g    ", e[i] );
        }
        printf( "\n" );

        printf( "\t" );
        for( i= nn-mod ; i<nn ; i++ )
        {
            printf( "Eigenvector " );
        }
        printf( "\n" );
        for( j=0 ; j<nn ; j++ )
        {
            printf( "\t" );
            for( i= nn-mod ; i<nn ; i++ )
            {
                printf( " %8.3g    ", a[j+na*i] );
            }
            printf( "\n" );
        }

    }

    free( a );
    free( b );
    free( e );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qcgsaa ***

 ** Input **

n =      4
nt=      2

Input Matrix a
```

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

Input Matrix b

     153      31      58     -58
      31     153     -58      58
      58     -58     153      31
     -58      58      31     153

 ** Output **

ierr =      0

Eigenvalue   Eigenvalue   Eigenvalue   Eigenvalue
 0.000648      0.00537       0.0274        0.217
Eigenvector  Eigenvector  Eigenvector  Eigenvector
   0.0294       0.0498      -0.0161        0.205
  -0.0469       0.0377       0.0686       -0.193
   0.0311      -0.0194        0.086       -0.192
  -0.0196      -0.0332      0.00145         0.21
```

5.5.2 ASL_qcgsan, ASL_pcgsan
実対称行列 (一般化固有値問題 $Ax = \lambda Bx$, B: 正定値) の全固有値

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型) の一般化固有値問題 $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: 実対称行列 , B: 正値実対称行列) をコレスキー法を用いて標準の固有値問題に変換し, ハウスホルダー法, QR 法により全固有値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcgsan (a, lna, n, b, lnb, e, w1, nt);

単精度関数:

ierr = ASL_pcgsan (a, lna, n, b, lnb, e, w1, nt);

(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∗}	lna × n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保存されない.
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 A, B の次数
4	b	{D∗ / R∗}	lnb × n	入 力	正値対称行列 B(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ / R∗}	n	出 力	固有値
7	w1	{D∗ / R∗}	n	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna, lnb

(b) nt≥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.	e [0] ← a [0] /b [0] とする.
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ⋯, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

5.5.3 ASL_qcgsss, ASL_pcgsss
実対称行列 (一般化固有値問題 $Ax = \lambda Bx$, B: 正定値) の固有値・固有ベクトル

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型) の一般化固有値問題 $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: 実対称行列 , B: 正値実対称行列) をコレスキー法を用いて標準の固有値問題に変換し, ハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個の固有値を求め, それに対応する固有ベクトルを逆反復法により求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcgsss (a, lna, n, b, lnb, eps, e, m, ve, lnv, isw, iw1, w1, nt);

単精度関数:

ierr = ASL_pcgsss (a, lna, n, b, lnb, eps, e, m, ve, lnv, isw, iw1, w1, nt);

(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∗}	lna × n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 A, B の次数
4	b	{D∗ / R∗}	lnb × n	入 力	正値対称行列 B(2 次元配列型)(上三角型)
				出 力	狭義の上三角部分は保存されない.
5	lnb	I	1	入 力	配列 b の整合寸法
6	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
7	e	{D∗ / R∗}	m	出 力	固有値
8	m	I	1	入 力	求めたい固有値の個数 m
9	ve	{D∗ / R∗}	lnv × m	出 力	各固有値に対応する固有ベクトル (列ベクトル)
10	lnv	I	1	入 力	配列 ve の整合寸法
11	isw	I	1	入 力	処理スイッチ isw≥0:大きい方から固有値を求める. isw<0:小さい方から固有値を求める.
12	iw1	I*	m	出 力	固有ベクトルフラグ (注意事項 (e) 参照)
13	w1	{D∗ / R∗}	9×n	ワーク	作業領域
14	nt	I	1	入 力	生成するタスク数
15	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) 0<n≤lna, lnb, lnv

(b) 0<m≤n

(c) nt≥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.	e [0] ← a [0] /b [0] , ve [0] ← $1.0/\sqrt{\text{b[0]}}$ とする.
2000	固有ベクトルを求める逆反復で最大反復回数をこえた.	固有ベクトルに一部精度の低いものがあるが, 処理は続ける. (注意事項 (e) 参照)
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	固有ベクトルの精度が低い可能性があるが, 処理を続ける.
3000	制限条件 (a) または (b) 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.	
4000	B が正定値でなかった.	

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

(e) 逆反復法で最大反復回数をこえた場合 (ierr=2000 出力時) について

- iw1$[i-1] = 0$ の場合 :
  i 番目の固有ベクトル計算は正常終了している.
- iw1$[i-1] \neq 0$ の場合 :
  i 番目の固有ベクトル計算は収束条件が満たされず, 固有ベクトルの精度は低い.
  この場合 iw1$[i-1]$ は, 反復回数が設定される.

なお正常終了時 (ierr=0 出力時) は, iw1$[i-1] \leftarrow 0$ が設定される.

(f) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(g) 固有ベクトルを必要としないときは, 5.5.4 $\left\{\begin{matrix}\text{ASL_qcgssn}\\ \text{ASL_pcgssn}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

$$A = \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix}$$

$$B = \begin{bmatrix} 170 & 18 & 33 & -21 & -17 & 13 & 25 & -36 \\ 18 & 171 & -21 & 22 & 13 & -17 & -36 & 25 \\ 33 & -21 & 171 & 18 & 25 & -36 & -17 & 13 \\ -21 & 22 & 18 & 171 & -36 & 25 & 13 & -17 \\ -17 & 13 & 25 & -36 & 171 & 18 & 33 & -21 \\ 13 & -17 & -36 & 25 & 18 & 171 & -21 & -3 \\ 25 & -36 & -17 & 13 & 33 & -21 & 171 & 18 \\ -36 & 25 & 13 & -17 & -21 & -3 & 18 & 171 \end{bmatrix}$$

のとき, $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ の固有値を小さい順に 2 個とそれに対応する固有ベクトルを求める.

(b) 入力データ

行列 A, lna=11, n=8, 行列 B, lnb=11, eps=-1.0, m=2, lnv=10, isw=-1, nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qcgsss */

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

int main()
{
    double *a;
    int na=11;
    int nn;
    double *b;
    int nb=11;
    double ceps= -1.0;
    double *e;
    int mm;
    double *ve;
    int nv=10;
    int ksw= -1;
    int *kw1;
    double *wk;
    int nt;
    int ierr;
    int i,j,k;
    FILE *fp;

    int mod;

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

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

    fscanf( fp, "%d", &nn );
    fscanf( fp, "%d", &mm );
    fscanf( fp, "%d", &nt );

    mod = mm % 4;

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

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

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

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

```
kw1 = ( int * )malloc((size_t)( sizeof(int) * mm ));
if( kw1 == NULL )
{
    printf( "no enough memory for array kw1\n" );
    return -1;
}

wk = ( double * )malloc((size_t)( sizeof(double) * (9*nn) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

printf( "\tn = %6d\n", nn );
printf( "\tm = %6d\n", mm );
printf( "\tnt= %6d\n\n", nt );

printf( "\tInput Matrix a\n\n" );
for( i=0 ; i<nn ; i++ )
{
    for( j=i ; j<nn ; j++ )
    {
        fscanf( fp, "%lf", &a[i+na*j] );
    }
}

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

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

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

fclose( fp );

ierr = ASL_qcgsss(a, na, nn, b, nb, ceps, e, mm, ve, nv, ksw, kw1, wk, nt);

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

for( k=0 ; k<mm-3 ; k = k+4 )
{
    printf( "\n" );
    printf( "\t" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "Eigenvalue   " );
    }
    printf( "\n" );
    printf( "\t" );
    for( i=k ; i<k+4 ; i++ )
    {
        printf( " %8.3g   ", e[i] );
    }
    printf( "\n" );

    printf( "\t" );
    for( i=0 ; i<4 ; i++ )
    {
        printf( "Eigenvector  " );
    }
    printf( "\n" );
```

```
            for( j=0 ; j<nn ; j++ )
            {
                printf( "\t" );
                for( i=k ; i<k+4 ; i++ )
                {
                    printf( " %8.3g   ", ve[j+nv*i] );
                }
                printf( "\n" );
            }
        }

        if( mod != 0 )
        {
            printf( "\n" );
            printf( "\t" );
            for( i= mm-mod ; i<mm ; i++ )
            {
                printf( "Eigenvalue   " );
            }
            printf( "\n" );
            printf( "\t" );
            for( i= mm-mod ; i<mm ; i++ )
            {
                printf( " %8.3g   ", e[i] );
            }
            printf( "\n" );

            printf( "\t" );
            for( i= nn-mod ; i<nn ; i++ )
            {
                printf( "Eigenvector  " );
            }
            printf( "\n" );
            for( j=0 ; j<nn ; j++ )
            {
                printf( "\t" );
                for( i= mm-mod ; i<mm ; i++ )
                {
                    printf( " %8.3g   ", ve[j+nv*i] );
                }
                printf( "\n" );
            }

        }

        free( a );
        free( b );
        free( e );
        free( ve );
        free( kw1 );
        free( wk );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_qcgsss ***

 ** Input **

n =      8
m =      2
nt=      2

Input Matrix a

     611     196    -192     407      -8     -52     -49      29
     196     899     113    -192     -71     -43      -8     -44
    -192     113     899     196      61      49       8      52
     407    -192     196     611       8      44      59     -23
      -8     -71      61       8     411    -599     208     208
     -52     -43      49      44    -599     411     208     208
     -49      -8       8      59     208     208      99    -911
      29     -44      52     -23     208     208    -911      99

Input Matrix b

     170      18      33     -21     -17      13      25     -36
      18     171     -21      22      13     -17     -36      25
      33     -21     171      18      25     -36     -17      13
     -21      22      18     171     -36      25      13     -17
     -17      13      25     -36     171      18      33     -21
      13     -17     -36      25      18     171     -21      -3
      25     -36     -17      13      33     -21     171      18
     -36      25      13     -17     -21      -3      18     171

 ** Output **

ierr =      0

Eigenvalue   Eigenvalue
      -5.3    -1.04e-15
Eigenvector  Eigenvector
 0.000789    -0.00329
  0.00146    -0.00658
 0.000624     0.00658
```

```
-0.00168      0.00329
 -0.0247      -0.0461
  -0.019      -0.0461
  0.0479       -0.023
  0.0445       -0.023
```

5.5.4 ASL_qcgssn, ASL_pcgssn
実対称行列 (一般化固有値問題 $Ax = \lambda Bx$, B: 正定値) の固有値

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型) の一般化固有値問題 $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ (A: 実対称行列 , B: 正値実対称行列) をコレスキー法を用いて標準の固有値問題に変換し, ハウスホルダー法, 無平方根 QR 法またはバイセクション法により, 大きい方から m 個, または小さい方から m 個の固有値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcgssn (a, lna, n, b, lnb, eps, e, m, isw, w1, nt);

単精度関数:

ierr = ASL_pcgssn (a, lna, n, b, lnb, eps, e, m, isw, w1, nt);

(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∗}	lna × n	入 力	実対称行列 A(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 A, B の次数
4	b	{D∗ / R∗}	lnb × n	入 力	正値対称行列 B(2 次元配列型)(上三角型)
				出 力	入力時の内容は保持されない.
5	lnb	I	1	入 力	配列 b の整合寸法
6	eps	{D / R}	1	入 力	固有値の収束判定のための絶対誤差の上限を与えるパラメータ (注意事項 (d) 参照)
7	e	{D∗ / R∗}	m	出 力	固有値
8	m	I	1	入 力	求めたい固有値の個数 m
9	isw	I	1	入 力	処理スイッチ isw≥0:大きい方から固有値を求める. isw<0:小さい方から固有値を求める.
10	w1	{D∗ / R∗}	5×n	ワーク	作業領域
11	nt	I	1	入 力	生成するタスク数
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) 0<n≤lna, lnb

(b) 0<m≤n

(c) nt≥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.	e [0] ← a [0] /b [0] とする.
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	処理を続ける.
3000	制限条件 (a) または (b) 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.	
4000	B が正定値でなかった.	

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は isw≥0 のときには大きい順に, isw<0 のときには小さい順に格納される.

(c) 固有値は無平方根 QR 法とバイセクション法を内部で適切に切り分けて計算している.

(d) eps ≤ 0 のときは内部で自動的に最適値を設定する. 通常は自動設定されるように負の値を設定されたい. なお, eps はバイセクション法で固有値を求めるときに使用される.

5.6 実対称行列 (2 次元配列型) (上三角型) の一般化固有値問題 ($ABx = \lambda x$)

5.6.1 ASL_qcgjaa, ASL_pcgjaa
実対称行列 (一般化固有値問題 $ABx = \lambda x$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型)(実引数型) の一般化固有値問題

$AB\boldsymbol{x} = \lambda\boldsymbol{x}$ (A : 実対称行列 , B : 正定値実対称行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ とそれに対応する全固有ベクトル $\boldsymbol{x}$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcgjaa (a, lna, n, b, lnb, e, work, nt);

単精度関数:

ierr = ASL_pcgjaa (a, lna, n, b, lnb, e, work, nt);

(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∗}	lna × n	入 力	実対称行列 A
				出 力	固有ベクトル $\boldsymbol{x}$
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 a,b の次数
4	b	{D∗ / R∗}	lnb × n	入 力	実対称行列 B
				出 力	入力時の内容は保持されない
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ / R∗}	n	出 力	固有値 λ
7	work	{D∗ / R∗}	2×n	ワーク	作業領域
8	nt	I	1	入 力	タスク数
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) $1 \leq \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(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	e [0] ← a [0] *b [0] , a [0] ← $1.0/\sqrt{\mathrm{b[0]}}$ とする.
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	結果の精度が低い可能性があるが, 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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同). このとき固有ベクトルは求まらない.

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(d) 固有ベクトルを必要としないときは, 5.6.2 $\left\{\begin{matrix}\mathrm{ASL_qcgjan}\\ \mathrm{ASL_pcgjan}\end{matrix}\right\}$を使用する.

(e) 行列 A のみが正定値行列であるときは, 5.7.1 $\left\{\begin{matrix}\mathrm{ASL_qcgkaa}\\ \mathrm{ASL_pcgkaa}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

4 次実正定値対称行列 A, B

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

に対して, 非対称行列 AB の固有値と固有ベクトルを求める.

注 ここで, A, B は, 以下で定義される実正定値対称行列である.

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

ただし,

$$P(a^2, \mathrm{n})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt \quad (1 \le i \le \mathrm{n}; 1 \le j \le \mathrm{n})$$

また各々の行列の固有値は, 全て, 次数 n に無関係に定まる有界区間内に存在する.

(b) 入力データ

n=4, lna=lnb=4, nt=2, 実正定値対称行列 A, B

(c) 主プログラム

```
/*       C interface example for ASL_qcgjaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *e, *work;
    double one,tre,fiv;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4;nt=4;
    one=1.0;tre=3.0;fiv=5.0;
    printf( "    *** ASL_qcgjaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    a = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n");
        return -1;
    }
    b = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n");
        return -1;
    }
    e = ( double * )malloc((size_t)(sizeof(double)*n));
    if ( e == NULL )
    {
        printf( "no enough memory for array e\n");
        return -1;
    }
    work = ( double * )malloc((size_t)(sizeof(double)*(2*n)));
    if ( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tlna = %6d\n",lna );
    printf( "\tlnb = %6d\n",lnb );
    printf( "\tnt  = %6d\n",nt  );
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            a[i+n*j]=one/(one+tre*(i+j+2)*(i+j+2))
                +one/(one+tre*(i-j)*(i-j));
            b[i+n*j]=one/(one+fiv*(i+j+2)*(i+j+2))
                +one/(one+fiv*(i-j)*(i-j));
        }
    }
    printf( "\n\tInput Matrix a\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
        printf( "\n" );
    }
    printf( "\n\tInput Matrix b\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",b[i+n*j]);
        }
        printf( "\n" );
    }
```

```
    ierr = ASL_qcgjaa(a, lna, n, b, lnb, e, work, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t          Eigenvalue   " );
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "%8.3g",e[j]);
    }
    printf( "\n\t          Eigenvector     " );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
    }
    printf( "\n" );
    free(a);
    free(b);
    free(e);
    free(work);
    return 0;
}
```

(d) 出力結果

```
*** ASL_qcgjaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

    1.08   0.286  0.0973  0.0489
   0.286    1.02   0.263  0.0861
  0.0973   0.263    1.01   0.257
  0.0489  0.0861   0.257    1.01

Input Matrix b

    1.05   0.188    0.06  0.0297
   0.188    1.01   0.175  0.0531
    0.06   0.175    1.01   0.171
  0.0297  0.0531   0.171       1

 ** Output **

ierr =      0

          Eigenvalue
   0.503   0.706    1.15    2.13
          Eigenvector
   -0.36  -0.573     0.6   0.414
   0.702   0.497   0.257   0.494
  -0.718    0.42  -0.389    0.46
   0.406  -0.626  -0.604   0.324
```

5.6.2 ASL_qcgjan, ASL_pcgjan
実対称行列 (一般化固有値問題 $ABx = \lambda x$, B: 正定値) の全固有値

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型)(実引数型) の一般化固有値問題

$$AB\boldsymbol{x} = \lambda\boldsymbol{x} \quad (A:\ \text{実対称行列},\ B:\ \text{正定値実対称行列})$$

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qcgjan  (a, lna, n, b, lnb, e, work, nt);
```

単精度関数:

```
ierr = ASL_pcgjan  (a, lna, n, b, lnb, e, work, nt);
```

(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∗}	lna × n	入 力	実対称行列 A
				出 力	入力時の内容は保持されない
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 a,b の次数
4	b	{D∗ / R∗}	lnb × n	入 力	実対称行列 B
				出 力	入力時の内容は保持されない
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ / R∗}	n	出 力	固有値 λ
7	work	{D∗ / R∗}	2×n	ワーク	作業領域
8	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0]
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	結果の精度が低い可能性があるが, 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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ⋯, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 配列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルを必要とするときは, 5.6.1 $\left\{\begin{matrix}\text{ASL_qcgjaa}\\ \text{ASL_pcgjaa}\end{matrix}\right\}$を使用する.

(d) 行列 A のみが正定値行列であるときは, 5.7.2 $\left\{\begin{matrix}\text{ASL_qcgkan}\\ \text{ASL_pcgkan}\end{matrix}\right\}$を使用する.

5.7 実対称行列 (2 次元配列型) (上三角型) の一般化固有値問題 ($BAx = \lambda x$)

5.7.1 ASL_qcgkaa, ASL_pcgkaa
実対称行列 (一般化固有値問題 $BAx = \lambda x$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型)(実引数型) の一般化固有値問題

$BA\boldsymbol{x} = \lambda\boldsymbol{x}$ (A : 実対称行列 , B : 正定値実対称行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ とそれに対応する全固有ベクトル $\boldsymbol{x}$ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qcgkaa  (a, lna, n, b, lnb, e, work, nt);
```

単精度関数:

```
ierr = ASL_pcgkaa  (a, lna, n, b, lnb, e, work, nt);
```

(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∗}	lna × n	入 力	実対称行列 A
				出 力	固有ベクトル $\boldsymbol{x}$
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 a,b の次数
4	b	{D∗ / R∗}	lnb × n	入 力	実対称行列 B
				出 力	入力時の内容は保持されない
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ / R∗}	n	出 力	固有値 λ
7	work	{D∗ / R∗}	2×n	ワーク	作業領域
8	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0] , a [0] ← $\sqrt{\text{b[0]}}$ とする.
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	結果の精度が低い可能性があるが, 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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同). このとき固有ベクトルは求まらない.

(6) **注意事項**

(a) 行列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^T B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(c) 固有値は小さい順に格納される.

(d) 固有ベクトルを必要としないときは, 5.7.2 $\left\{\begin{matrix}\text{ASL_qcgkan}\\ \text{ASL_pcgkan}\end{matrix}\right\}$を使用する.

(e) 行列 A のみが正定値行列であるときは, 5.6.1 $\left\{\begin{matrix}\text{ASL_qcgjaa}\\ \text{ASL_pcgjaa}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

4 次実正定値対称行列 A, B

$$A = \begin{bmatrix} 1.07692 & 0.28571 & 0.09733 & 0.04887 \\ 0.28571 & 1.02041 & 0.26316 & 0.08610 \\ 0.09733 & 0.26316 & 1.00917 & 0.25676 \\ 0.04887 & 0.08610 & 0.25676 & 1.00518 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.04762 & 0.18841 & 0.05996 & 0.02968 \\ 0.18841 & 1.01235 & 0.17460 & 0.05314 \\ 0.05996 & 0.17460 & 1.00552 & 0.17073 \\ 0.02968 & 0.05314 & 0.17073 & 1.00312 \end{bmatrix}$$

に対して, 非対称行列 AB の固有値と固有ベクトルを求める.

注 ここで, A, B は, 以下で定義される実正定値対称行列である.

$$A = P(3.0, 4), \quad B = P(5.0, 4)$$

ただし,

$$P(a^2, \mathrm{n})_{i,j} = 2\int_0^\infty \cos(ait)\cos(ajt)e^{-t}dt \quad (1 \le i \le \mathrm{n}; 1 \le j \le \mathrm{n})$$

また各々の行列の固有値は, 全て, 次数 n に無関係に定まる有界区間内に存在する.

(b) 入力データ

n=4,lna=lnb=4,nt=2, 実正定値対称行列 A, B

(c) 主プログラム

```
/*       C interface example for ASL_qcgkaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *e, *work;
    double one,tre,fiv;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4;nt=4;
    one=1.0;tre=3.0;fiv=5.0;
    printf( "    *** ASL_qcgkaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    a = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n");
        return -1;
    }
    b = ( double * )malloc((size_t)(sizeof(double)*n*n));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n");
        return -1;
    }
    e = ( double * )malloc((size_t)(sizeof(double)*n));
    if ( e == NULL )
    {
        printf( "no enough memory for array e\n");
        return -1;
    }
    work = ( double * )malloc((size_t)(sizeof(double)*(2*n)));
    if ( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tlna = %6d\n",lna );
    printf( "\tlnb = %6d\n",lnb );
    printf( "\tnt  = %6d\n",nt  );
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            a[i+n*j]=one/(one+tre*(i+j+2)*(i+j+2))
                +one/(one+tre*(i-j)*(i-j));
            b[i+n*j]=one/(one+fiv*(i+j+2)*(i+j+2))
                +one/(one+fiv*(i-j)*(i-j));
        }
    }
    printf( "\n\tInput Matrix a\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
        printf( "\n" );
    }
    printf( "\n\tInput Matrix b\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",b[i+n*j]);
        }
        printf( "\n" );
    }
```

```
    ierr = ASL_qcgkaa(a, lna, n, b, lnb, e, work, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t          Eigenvalue   " );
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "%8.3g",e[j]);
    }
    printf( "\n\t          Eigenvector     " );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "%8.3g",a[i+n*j]);
        }
    }
    printf( "\n" );
    free(a);
    free(b);
    free(e);
    free(work);
    return 0;
}
```

(d) 出力結果

```
*** ASL_qcgkaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

    1.08   0.286  0.0973  0.0489
   0.286    1.02   0.263  0.0861
  0.0973   0.263    1.01   0.257
  0.0489  0.0861   0.257    1.01

Input Matrix b

    1.05   0.188    0.06  0.0297
   0.188    1.01   0.175  0.0531
    0.06   0.175    1.01   0.171
  0.0297  0.0531   0.171       1

 ** Output **

ierr =      0

          Eigenvalue
   0.503   0.706    1.15    2.13
          Eigenvector
  -0.276    -0.5   0.635   0.564
    0.54   0.436   0.273   0.676
  -0.551   0.368  -0.414   0.629
   0.311  -0.547  -0.641   0.442
```

5.7.2 ASL_qcgkan, ASL_pcgkan
実対称行列 (一般化固有値問題 $BAx = \lambda x$, B: 正定値) の全固有値

(1) **機 能**

実対称行列 (2 次元配列型)(上三角型)(実引数型) の一般化固有値問題

$BA\boldsymbol{x} = \lambda\boldsymbol{x}$ (A : 実対称行列 , B : 正定値実対称行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_qcgkan (a, lna, n, b, lnb, e, work, nt);

単精度関数:

ierr = ASL_pcgkan (a, lna, n, b, lnb, e, work, nt);

(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∗}	lna × n	入 力	実対称行列 A
				出 力	入力時の内容は保持されない
2	lna	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	行列 a,b の次数
4	b	{D∗ R∗}	lnb × n	入 力	実対称行列 B
				出 力	入力時の内容は保持されない
5	lnb	I	1	入 力	配列 b の整合寸法
6	e	{D∗ R∗}	n	出 力	固有値 λ
7	work	{D∗ R∗}	2×n	ワーク	作業領域
8	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0]
2100	B の対角要素に非常に小さい絶対値をもつものがあった.	結果の精度が低い可能性があるが, 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 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000+i	固有値を求める段階で収束しなかった (1≤i≤n).	e [0] , ···, e [i−2] にそれまでに求まった固有値が入る (ただし順不同).

(6) **注意事項**

(a) 行列 a, b には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルを必要とするときは, 5.7.1 $\left\{\begin{matrix}\text{ASL_qcgkaa}\\ \text{ASL_pcgkaa}\end{matrix}\right\}$を使用する.

(d) 行列 A のみが正定値行列であるときは, 5.6.2 $\left\{\begin{matrix}\text{ASL_qcgjan}\\ \text{ASL_pcgjan}\end{matrix}\right\}$を使用する.

5.8 エルミート行列 (2 次元配列型) (上三角型) (実数引数型) の一般化固有値問題 ($Az = \lambda Bz$)

5.8.1 ASL_hcgraa, ASL_gcgraa
エルミート行列 (一般化固有値問題 $Az = \lambda Bz$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$$A\boldsymbol{z} = \lambda B\boldsymbol{z} \ (A: \text{エルミート行列}, \ B: \text{正定値エルミート行列})$$

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ とそれに対応する全固有ベクトル $\boldsymbol{z}$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_hcgraa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

単精度関数:

ierr = ASL_gcgraa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実数部
				出 力	固有ベクトル $\boldsymbol{x}$ の実数部
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚数部
				出 力	固有ベクトル $\boldsymbol{x}$ の虚数部
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	入力時の内容は保持されない
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	{D∗ / R∗}	n	出 力	固有値 λ
9	work	{D∗ / R∗}	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] /b [0] , a [0] ← $1.0/\sqrt{\text{b[0]}}$ とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) 注意事項

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(d) 固有ベクトルを必要としないときは, 5.8.2 $\left\{\begin{matrix}\text{ASL_hcgran}\\ \text{ASL_gcgran}\end{matrix}\right\}$を使用する.

(7) 使用例

(a) 問 題

4 次エルミート行列

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

と, 各成分が共役である 4 次エルミート行列

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

A, B に対して, $A\boldsymbol{x} = \lambda B\boldsymbol{x}$ の全固有値とそれに対応する固有ベクトルを求める.

(b) 入力データ

n=4,lna=lnb=4,nt=2, エルミート行列 A, B

(c) 主プログラム

```
/*       C interface example for ASL_hcgraa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4,nt=4;
    printf( "    *** ASL_hcgraa ***\n" );
    printf( "\n     ** Input **\n\n" );
    ar = ( double * )malloc(sizeof(double)*n*n);
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n");
        return -1;
    }
    br = ( double * )malloc(sizeof(double)*n*n);
    if( br == NULL )
    {
        printf( "no enough memory for array br\n");
        return -1;
    }
    ai = ( double * )malloc(sizeof(double)*n*n);
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n");
        return -1;
    }
    bi = ( double * )malloc(sizeof(double)*n*n);
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n");
        return -1;
    }
    e = ( double * )malloc(sizeof(double)*n);
    if ( e == NULL )
    {
```

```
        printf( "no enough memory for array e\n");
        return -1;
    }
    work = ( double * )malloc(sizeof(double)*4*n);
    if ( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tlna = %6d\n",lna );
    printf( "\tlnb = %6d\n",lnb );
    printf( "\tnt  = %6d\n",nt  );
    br[0+n*0]=8.0;bi[0+n*0]=0.0;
    br[1+n*1]=9.0;bi[1+n*1]=0.0;
    br[2+n*2]=10.0;bi[2+n*2]=0.0;
    br[3+n*3]=11.0,bi[3+n*3]=0.0;
    br[0+n*1]=3.0;bi[0+n*1]=0.0;
    br[0+n*2]=1.0;bi[0+n*2]=2.0;
    br[0+n*3]=-1.0;bi[0+n*3]=2.0;
    br[1+n*2]=1.0;bi[1+n*2]=-2.0;
    br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
    br[2+n*3]=-3.0;bi[2+n*3]=0.0;
    for(i=1;i<n;i++)
    {
        for(j=0;j<i;j++)
        {
            br[i+n*j]= br[j+n*i];
            bi[i+n*j]=-bi[j+n*i];
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            ar[i+n*j]= br[i+n*j];
            ai[i+n*j]=-bi[i+n*j];
        }
    }
    printf( "\n\tInput Matrix a\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "(%8.3g,",ar[i+n*j]);
            printf( "%8.3g)  ",ai[i+n*j]);
        }
        printf( "\n" );
    }
    printf( "\n\tInput Matrix b\n\n" );
    for(i=0; i<n; i++)
    {
        printf( "\t" );
        for(j=0; j<n; j++)
        {
            printf( "(%8.3g,",br[i+n*j]);
            printf( "%8.3g)  ",bi[i+n*j]);
        }
        printf( "\n" );
    }

    ierr = ASL_hcgraa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t          Eigenvalue   " );
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "       %8.3g       ",e[j]);
    }
    printf( "\n\t          Eigenvector     " );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        for(j=0; j<n; j++)
        {
            printf( "(%8.3g,",ar[i+n*j]);
            printf( "%8.3g)   ",ai[i+n*j]);
        }
    }
    printf( "\n" );
    free(ar);
    free(br);
    free(ai);
    free(bi);
    free(e);
    free(work);
    return 0;
}
```

(d) 出力結果

```
*** ASL_hcgraa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
          0.231                    1                        1                    4.33
          Eigenvector
(   0.175,        0)   (        0,        0)   (   0.211,        0)   (   0.364,        0)
(   -0.16, 0.00182)   (-6.89e-19,-2.03e-16)   (   0.211,-1.71e-18)   (  -0.333,-0.00378)
(-0.00199,  -0.149)   (  -0.192, 0.00302)   ( -0.0313,-4.35e-17)   (-0.00414,     0.31)
( 0.00029,  -0.138)   (   0.192,-0.00302)   (  0.0313,-1.66e-16)   (0.000603,    0.287)
```

5.8.2 ASL_hcgran, ASL_gcgran
エルミート行列 (一般化固有値問題 $Az = \lambda Bz$, B: 正定値) の全固有値

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$$A\boldsymbol{z} = \lambda B\boldsymbol{z} \quad (A:\ \text{エルミート行列}\ ,\ B:\ \text{正定値エルミート行列})$$

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hcgran  (ar, ai, lna, n, br, bi, lnb, e, work, nt);
```

単精度関数:

```
ierr = ASL_gcgran  (ar, ai, lna, n, br, bi, lnb, e, work, nt);
```

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実数部
				出 力	入力時の内容は保持されない
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚数部
				出 力	入力時の内容は保持されない
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	入力時の内容は保持されない
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	{D∗ / R∗}	n	出 力	固有値 λ
9	work	{D∗ / R∗}	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(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	e [0] ← a [0] /b [0] ,
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) **注意事項**

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルを必要とするときは, 5.8.1 $\left\{\begin{matrix}\mathrm{ASL_hcgraa}\\ \mathrm{ASL_gcgraa}\end{matrix}\right\}$を使用する.

5.9 エルミート行列 (2 次元配列型) (上三角型) (実数引数型) の一般化固有値問題 ($ABx = \lambda z$)

5.9.1 ASL_hcgjaa, ASL_gcgjaa
エルミート行列 (一般化固有値問題 $ABz = \lambda z$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$ABz = \lambda z$ (A : エルミート行列 , B : 正定値エルミート行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ とそれに対応する全固有ベクトル z を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_hcgjaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

単精度関数:

ierr = ASL_gcgjaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(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	ar	D∗ / R∗	lna × n	入 力	エルミート行列 A の実数部
				出 力	固有ベクトル $\boldsymbol{x}$ の実数部
2	ai	D∗ / R∗	lna × n	入 力	エルミート行列 A の虚数部
				出 力	固有ベクトル $\boldsymbol{x}$ の虚数部
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	D∗ / R∗	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	D∗ / R∗	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	入力時の内容は保持されない
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	D∗ / R∗	n	出 力	固有値 λ
9	work	D∗ / R∗	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0] , a [0] ← $1.0/\sqrt{\text{b[0]}}$ とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) **注意事項**

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^* B \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(d) 固有ベクトルを必要としないときは, 5.9.2 $\left\{\begin{matrix}\text{ASL_hcgjan}\\ \text{ASL_gcgjan}\end{matrix}\right\}$を使用する.

(e) 行列 A のみが正定値行列であるときは, 5.10.1 $\left\{\begin{matrix}\text{ASL_hcgkaa}\\ \text{ASL_gcgkaa}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

4 次エルミート行列

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

と, 各成分が共役である 4 次エルミート行列

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

A, B に対して, 非エルミート行列 AB の固有値と固有ベクトルを求める.

(b) 入力データ

n=4,lna=lnb=4,nt=2, エルミート行列 A, B

(c) 主プログラム

```
/*       C interface example for ASL_hcgjaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4,nt=4;
    printf( "    *** ASL_hcgjaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    ar = ( double * )malloc(sizeof(double)*n*n);
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n");
        return -1;
    }
    br = ( double * )malloc(sizeof(double)*n*n);
    if( br == NULL )
    {
        printf( "no enough memory for array br\n");
        return -1;
    }
    ai = ( double * )malloc(sizeof(double)*n*n);
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n");
        return -1;
    }
    bi = ( double * )malloc(sizeof(double)*n*n);
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n");
```

```
    return -1;
}
e = ( double * )malloc(sizeof(double)*n);
if ( e == NULL )
{
    printf( "no enough memory for array e\n");
    return -1;
}
work = ( double * )malloc(sizeof(double)*4*n);
if ( work == NULL )
{
    printf( "no enough memory for array work\n");
    return -1;
}
printf( "\tn   = %6d\n" , n );
printf( "\tlna = %6d\n",lna );
printf( "\tlnb = %6d\n",lnb );
printf( "\tnt  = %6d\n",nt  );
br[0+n*0]=8.0;bi[0+n*0]=0.0;
br[1+n*1]=9.0;bi[1+n*1]=0.0;
br[2+n*2]=10.0;bi[2+n*2]=0.0;
br[3+n*3]=11.0,bi[3+n*3]=0.0;
br[0+n*1]=3.0;bi[0+n*1]=0.0;
br[0+n*2]=1.0;bi[0+n*2]=2.0;
br[0+n*3]=-1.0;bi[0+n*3]=2.0;
br[1+n*2]=1.0;bi[1+n*2]=-2.0;
br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
br[2+n*3]=-3.0;bi[2+n*3]=0.0;
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        br[i+n*j]= br[j+n*i];
        bi[i+n*j]=-bi[j+n*i];
    }
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        ar[i+n*j]= br[i+n*j];
        ai[i+n*j]=-bi[i+n*j];
    }
}
printf( "\n\tInput Matrix a\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)  ",ai[i+n*j]);
    }
    printf( "\n" );
}
printf( "\n\tInput Matrix b\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",br[i+n*j]);
        printf( "%8.3g)  ",bi[i+n*j]);
    }
    printf( "\n" );
}

ierr = ASL_hcgjaa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\t          Eigenvalue   " );
printf( "\n\t" );
for(j=0; j<n; j++)
{
    printf( "       %8.3g       ",e[j]);
}
printf( "\n\t          Eigenvector     " );
for(i=0; i<n; i++)
{
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)   ",ai[i+n*j]);
    }
}
printf( "\n" );
free(ar);
free(br);
free(ai);
free(bi);
free(e);
free(work);
```

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_hcgjaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
           16.7                    37                    106                    218
          Eigenvector
(  -0.399,       0)   (  -0.109,       0)   (    0.17,       0)   (  0.0917,       0)
(   0.345,0.000936)   (   0.102,   0.016)   (   0.203, 0.00834)   (   0.101,-0.00404)
( 0.00726,  -0.132)   (  0.0187,  -0.327)   ( -0.0997,-0.00601)   (   0.147,-0.00401)
(-0.00421,  -0.125)   (  0.0156,  -0.281)   (    0.13,-0.000644)   (  -0.166, 0.00269)
```

5.9.2 ASL_hcgjan, ASL_gcgjan
エルミート行列 (一般化固有値問題 $ABz = \lambda z$, B: 正定値) の全固有値

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$ABz = \lambda z$ (A : エルミート行列 , B : 正定値エルミート行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_hcgjan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

単精度関数:

ierr = ASL_gcgjan (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実数部
				出 力	入力時の内容は保持されない
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚数部
				出 力	入力時の内容は保持されない
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	入力時の内容は保持されない
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	{D∗ / R∗}	n	出 力	固有値 λ
9	work	{D∗ / R∗}	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(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	e [0] ← a [0] *b [0] ,
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) **注意事項**

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルを必要とするときは, 5.9.1 $\left\{\begin{matrix}\mathrm{ASL_hcgjaa}\\ \mathrm{ASL_gcgjaa}\end{matrix}\right\}$を使用する.

(d) 行列 A のみが正定値行列であるときは, 5.10.2 $\left\{\begin{matrix}\mathrm{ASL_hcgkan}\\ \mathrm{ASL_gcgkan}\end{matrix}\right\}$を使用する.

5.10 エルミート行列 (2 次元配列型) (上三角型) (実数引数型) の一般化固有値問題 ($BAx = \lambda z$)

5.10.1 ASL_hcgkaa, ASL_gcgkaa
エルミート行列 (一般化固有値問題 $BAz = \lambda z$, B: 正定値) の全固有値・全固有ベクトル

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$BA\boldsymbol{z} = \lambda\boldsymbol{z}$ (A : エルミート行列 , B : 正定値エルミート行列)

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ とそれに対応する全固有ベクトル $\boldsymbol{z}$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_hcgkaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

単精度関数:

ierr = ASL_gcgkaa (ar, ai, lna, n, br, bi, lnb, e, work, nt);

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実数部
				出 力	固有ベクトル $\boldsymbol{x}$ の実数部
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚数部
				出 力	固有ベクトル $\boldsymbol{x}$ の虚数部
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	{D∗ / R∗}	n	出 力	固有値 λ
9	work	{D∗ / R∗}	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0] , a [0] ← $\sqrt{\text{b[0]}}$ とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) **注意事項**

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 各固有ベクトル $\boldsymbol{v}_i$ は $\boldsymbol{v}_j^* B^{-1} \boldsymbol{v}_k = \delta_{j,k}$ となるように正規直交化される.

(d) 固有ベクトルを必要としないときは, 5.10.2 $\left\{\begin{matrix}\text{ASL_hcgkan}\\ \text{ASL_gcgkan}\end{matrix}\right\}$を使用する.

(e) 行列 A のみが正定値行列であるときは, 5.9.1 $\left\{\begin{matrix}\text{ASL_hcgjaa}\\ \text{ASL_gcgjaa}\end{matrix}\right\}$を使用する.

(7) **使用例**

(a) 問 題

4 次エルミート行列

$$A = \begin{bmatrix} 8 & 3 & 1-2i & -1-2i \\ 3 & 9 & 1+2i & -1+2i \\ 1+2i & 1-2i & 10 & -3 \\ -1+2i & -1-2i & -3 & 11 \end{bmatrix}$$

と, 各成分が共役である 4 次エルミート行列

$$B = \begin{bmatrix} 8 & 3 & 1+2i & -1+2i \\ 3 & 9 & 1-2i & -1-2i \\ 1-2i & 1+2i & 10 & -3 \\ -1-2i & -1+2i & -3 & 11 \end{bmatrix}$$

A, B に対して, 非エルミート行列 BA の固有値と固有ベクトルを求める.

(b) 入力データ

n=4,lna=lnb=4,nt=2, エルミート行列 A, B

(c) 主プログラム

```
/*       C interface example for ASL_hcgkaa */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *ar, *br, *ai, *bi, *e, *work;
    int    i,j,n,ierr;
    int    nt;
    int    lna, lnb;
    n=4;
    lna=4;lnb=4,nt=4;
    printf( "    *** ASL_hcgkaa ***\n" );
    printf( "\n     ** Input **\n\n" );
    ar = ( double * )malloc(sizeof(double)*n*n);
    if( ar == NULL )
    {
        printf( "no enough memory for array ar\n");
        return -1;
    }
    br = ( double * )malloc(sizeof(double)*n*n);
    if( br == NULL )
    {
        printf( "no enough memory for array br\n");
        return -1;
    }
    ai = ( double * )malloc(sizeof(double)*n*n);
    if( ai == NULL )
    {
        printf( "no enough memory for array ai\n");
        return -1;
    }
    bi = ( double * )malloc(sizeof(double)*n*n);
    if( bi == NULL )
    {
        printf( "no enough memory for array bi\n");
```

```
    return -1;
}
e = ( double * )malloc(sizeof(double)*n);
if ( e == NULL )
{
    printf( "no enough memory for array e\n");
    return -1;
}
work = ( double * )malloc(sizeof(double)*4*n);
if ( work == NULL )
{
    printf( "no enough memory for array work\n");
    return -1;
}
printf( "\tn   = %6d\n" , n );
printf( "\tlna = %6d\n",lna );
printf( "\tlnb = %6d\n",lnb );
printf( "\tnt  = %6d\n",nt  );
br[0+n*0]=8.0;bi[0+n*0]=0.0;
br[1+n*1]=9.0;bi[1+n*1]=0.0;
br[2+n*2]=10.0;bi[2+n*2]=0.0;
br[3+n*3]=11.0,bi[3+n*3]=0.0;
br[0+n*1]=3.0;bi[0+n*1]=0.0;
br[0+n*2]=1.0;bi[0+n*2]=2.0;
br[0+n*3]=-1.0;bi[0+n*3]=2.0;
br[1+n*2]=1.0;bi[1+n*2]=-2.0;
br[1+n*3]=-1.0;bi[1+n*3]=-2.0;
br[2+n*3]=-3.0;bi[2+n*3]=0.0;
for(i=1;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        br[i+n*j]= br[j+n*i];
        bi[i+n*j]=-bi[j+n*i];
    }
}
for(i=0;i<n;i++)
{
    for(j=0;j<n;j++)
    {
        ar[i+n*j]= br[i+n*j];
        ai[i+n*j]=-bi[i+n*j];
    }
}
printf( "\n\tInput Matrix a\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)  ",ai[i+n*j]);
    }
    printf( "\n" );
}
printf( "\n\tInput Matrix b\n\n" );
for(i=0; i<n; i++)
{
    printf( "\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",br[i+n*j]);
        printf( "%8.3g)  ",bi[i+n*j]);
    }
    printf( "\n" );
}

ierr = ASL_hcgkaa(ar,ai, lna, n, br,bi, lnb, e, work, nt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );
printf( "\n\t          Eigenvalue   " );
printf( "\n\t" );
for(j=0; j<n; j++)
{
    printf( "       %8.3g       ",e[j]);
}
printf( "\n\t          Eigenvector     " );
for(i=0; i<n; i++)
{
    printf( "\n\t" );
    for(j=0; j<n; j++)
    {
        printf( "(%8.3g,",ar[i+n*j]);
        printf( "%8.3g)   ",ai[i+n*j]);
    }
}
printf( "\n" );
free(ar);
free(br);
free(ai);
free(bi);
free(e);
free(work);
```

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_hcgkaa ***

 ** Input **

n   =      4
lna =      4
lnb =      4
nt  =      4

Input Matrix a

(       8,       0)  (       3,       0)  (       1,      -2)  (      -1,      -2)
(       3,       0)  (       9,       0)  (       1,       2)  (      -1,       2)
(       1,       2)  (       1,      -2)  (      10,       0)  (      -3,       0)
(      -1,       2)  (      -1,      -2)  (      -3,       0)  (      11,       0)

Input Matrix b

(       8,       0)  (       3,       0)  (       1,       2)  (      -1,       2)
(       3,       0)  (       9,       0)  (       1,      -2)  (      -1,      -2)
(       1,      -2)  (       1,       2)  (      10,       0)  (      -3,       0)
(      -1,      -2)  (      -1,       2)  (      -3,       0)  (      11,       0)

 ** Output **

ierr =      0

          Eigenvalue
           16.7                  37                  106                 218
          Eigenvector
(   -1.63, 0.00149)   (   0.656,  0.0713)   (    1.76,    0.08)   (    1.35, -0.0577)
(    1.41,-0.00511)   (  -0.625,  0.0299)   (    2.09, 0.00933)   (     1.5,-0.00416)
(  0.0301,    0.54)   (   0.102,   -1.99)   (   -1.03,  0.0151)   (    2.16, -0.0331)
( -0.0167,    0.51)   (  0.0905,   -1.71)   (    1.34,  0.0676)   (   -2.45,  0.0648)
```

5.10.2 ASL_hcgkan, ASL_gcgkan
エルミート行列 (一般化固有値問題 $BAz = \lambda z$, B: 正定値) の全固有値

(1) **機 能**

エルミート行列 (2 次元配列型)(上三角型)(実数引数型) の一般化固有値問題

$$BA\boldsymbol{z} = \lambda\boldsymbol{z} \ (A: \text{ エルミート行列 }, \ B: \text{ 正定値エルミート行列})$$

をコレスキー分解, ハウスホルダー法, QR 法で解いて, 全固有値 λ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_hcgkan  (ar, ai, lna, n, br, bi, lnb, e, work, nt);
```

単精度関数:

```
ierr = ASL_gcgkan  (ar, ai, lna, n, br, bi, lnb, e, work, nt);
```

(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	ar	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の実数部
				出 力	入力時の内容は保持されない
2	ai	{D∗ / R∗}	lna × n	入 力	エルミート行列 A の虚数部
				出 力	入力時の内容は保持されない
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A, B の次数
5	br	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の実数部
				出 力	入力時の内容は保持されない
6	bi	{D∗ / R∗}	lnb × n	入 力	エルミート行列 B の虚数部
				出 力	入力時の内容は保持されない
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	e	{D∗ / R∗}	n	出 力	固有値 λ
9	work	{D∗ / R∗}	4×n	ワーク	作業領域
10	nt	I	1	入 力	タスク数
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) $1 \leq$ n $\leq$ lna, lnb

(b) nt ≥ 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	e [0] ← a [0] *b [0] ,
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
4000	B が正定値でなかった.	
5000	通常の固有値問題の解法過程において, 処理が続行不能となった.	

(6) **注意事項**

(a) 行列 ar, ai, br, bi には, 上三角部分のみにデータが格納されていればよい.

(b) 固有値は小さい順に格納される.

(c) 固有ベクトルを必要とするときは, 5.10.1 $\left\{\begin{matrix}\text{ASL_hcgkaa}\\ \text{ASL_gcgkaa}\end{matrix}\right\}$を使用する.

(d) 行列 A のみが正定値行列であるときは, 5.9.2 $\left\{\begin{matrix}\text{ASL_hcgjan}\\ \text{ASL_gcgjan}\end{matrix}\right\}$を使用する.

第 6 章　フーリエ変換とその応用

6.1　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・スペクトル解析を行う関数について説明する.

本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.

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. 利用者は, 入力データが次のいずれかの性質を満たしているかにより効率のよい処理を行うことができる.

(1) 多重 1 次元複素フーリエ変換 (任意基数)
　入力データが複素数で, かつ 1 次元のデータが複数ある.

(2) 多重 1 次元実フーリエ変換 (任意基数)
　入力データが実数で, かつ 1 次元である.

(3) 2 次元複素フーリエ変換 (任意基数)
　入力データが複素数で, かつ 2 次元である.

(4) 2 次元実フーリエ変換 (任意基数)
　入力データが実数で, かつ 2 次元である.

(5) 3 次元複素フーリエ変換 (任意基数)
　入力データが複素数で, かつ 3 次元である.

(6) 3 次元実フーリエ変換 (任意基数)
　入力データが実数で, かつ 3 次元である.

また, 本章で扱う高速フーリエ変換では, 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) 2 次元データ

(2) 3 次元データ

なお, これ以外のフーリエ変換を実行したい場合や, 関連機能を利用したい場合は, ＜基本機能第 3 分冊＞ の関数を使用されたい.

6.1.1 使用上の注意

(1) 2 次元複素フーリエ変換の入力データ数 nx, ny (3 次元複素フーリエ変換では nx, ny, nz) は, それぞれ 1 周期分 $[0, 2\pi)$ のデータに相当する.

(2) 利用者は, まず, 初期値設定を行う必要がある. ここでは, 三角関数テーブルの生成, および基数分けを行う. 次にフーリエ変換を実行する. このとき初期値を格納した配列は保存しておかなければならない.

(3) 入力データ数が小さいと, 演算コストに対して並列処理オーバヘッドの影響が大きいため, 非並列処理関数を用いた場合よりも性能が低下することがある.

6.1.2 使用しているアルゴリズム

6.1.2.1 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.

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

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.

6.1.2.2 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

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

従って, 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.

6.1.2.3 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.

6.1.2.4 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,

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

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.

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

従って, 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.

6.1.3 参考文献

(1) Pease, M. C, “An Adaption of the Fast Fourier Transform for Parallel Processing”, J. Assn. Comput. Mach. , 15, 252(1968).

(2) Stockham, T. G. , “High Speed Convolution and Correlation”, AFIPS Conf. Proc. , 28, 229(1966).

(3) 花村光泰, 萬淳一, 津和義昭, 片山博, 渡辺貞, “NEC スーパーコンピュータ SX-1/SX-2 に 適した FFT プログラム”, 情報処理学会 第 29 回全国大会 講演集, 133 (1984).

(4) 花村光泰, 萬淳一, 宮平知博, 津和義昭, 宍戸俊一, “NEC スーパーコンピュータ SX-1/SX-2 に 適した高速フーリエ変換プログラム [II]”, 情報処理学会 第 31 回全国大会 講演集, 73 (1985).

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

(6) Singleton, R. C. , “An Algorithm for Computing the Mixed Radix Fast Fourier Transform”, IEEE Trans. Audio and Electroacoust. , AU-17, 93(1969).

(7) Singleton, R. C. , “ALGOL Procedures for the Fast Fourier Transform”, Commun. ACM, 11, 773(1968).

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

(9) Brigham, E. O. , “The Fast Fourier Transform”, Prentice-Hall Inc. , 1974.

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

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

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

(13) Willemstein, T. , “Two-dimensional Fourier Transforms on the Cray-1S”, Supercomputer, 10(1985).

(14) 萬 淳一, 花村光泰, 宮平知博, “SX システムに有効なプログラミング例”, 大阪大学 大型計算機 センターニュース (Vol. 12 No. 2) 1986.

(15) Brigham, E. O. , (宮川 洋, 今井秀樹共訳), “高速フーリエ変換”, 科学技術出版社 (1978).

6.2 多重 1 次元複素フーリエ変換 (実数引数型)

6.2.1 [非推奨]ASL_qfcmfb, ASL_pfcmfb 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_qfcmfb (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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	nt	I	1	入 力	生成するタスク数
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) $\mathrm{n} \geq 1$, $\mathrm{m} \geq 1$

(b) $\mathrm{incn} \geq 1$, $\mathrm{incm} \geq 1$

(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) isw=0 または isw=1, isw=−1

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

(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.	
3040	制限条件 (e) 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 6.2.2 $\left\{\begin{matrix}\mathrm{ASL\_qfcmbf}\\ \mathrm{ASL\_pfcmbf}\end{matrix}\right\}$ 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 6.2.2 $\left\{\begin{matrix}\mathrm{ASL\_qfcmbf}\\ \mathrm{ASL\_pfcmbf}\end{matrix}\right\}$ 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) **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**

6.2.2 (7) 使用例参照.

6.2.2 [非推奨]ASL_qfcmbf, ASL_pfcmbf 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_qfcmbf (n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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**: incn × (n − 1) + incm × (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**: incn × (n − 1) + incm × (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 × n	入 力	三角関数テーブル (注意事項 (a) 参照)
10	wk	{D∗ / R∗}	2 × m × n	ワーク	作業領域
11	nt	I	1	入 力	生成するタスク数
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) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(c) incn ≥ m × gcm (incn, incm) または
incm ≥ n × gcm (incn, incm)
ただし, gcm(i, j) は i, j の最大公約数を表す.

(d) isw=1 または isw=−1

(e) nt ≥ 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.	
3040	制限条件 (e) 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 6.2.1 $\left\{\begin{matrix}\text{ASL\_qfcmfb}\\ \text{ASL\_pfcmfb}\end{matrix}\right\}$ 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 $c_{k,l}$ $(k = 0, \cdots, n-1;\ l = 1, \cdots, m)$ 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) 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) **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), nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qfcmfb , ASL_qfcmbf */

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

```
int main()
{
    int ld=35;
    int n;      int m;
    double *cr; double *ci;
    int incn;   int incm;
    int isw;
    int ifax[20];   double *trigs;
    double *wk;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_qfcmfb , ASL_qfcmbf ***\n" );
    printf( "\n     ** Input **\n\n" );

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

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

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

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

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

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

    printf("\t   n    = %d \n", n );
    printf("\t   m    = %d \n", m );
    printf("\t   incn = %d \n", incn );
    printf("\t   incm = %d \n", incm );
    printf("\t   nt   = %d \n\n", nt );

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

    fclose( fp );


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

    isw = 1;
    ierr = ASL_qfcmfb(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            cr[i*incn+j*incm] /= n ;
            ci[i*incn+j*incm] /= n ;
        }
    }
```

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

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

    isw = -1;
    ierr = ASL_qfcmbf(n, m, cr, ci, incn, incm, isw, ifax, trigs, wk, nt);

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

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



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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfcmfb , ASL_qfcmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 9
   nt   = 2

  Real Part                  Imaginary Part
  cr[  0] =        1         ci[  0] =        4
  cr[  1] =        2         ci[  1] =        3
  cr[  2] =        3         ci[  2] =        2
  cr[  3] =        4         ci[  3] =        1
  cr[  4] =        4         ci[  4] =        1
  cr[  5] =        3         ci[  5] =        2
  cr[  6] =        2         ci[  6] =        3
  cr[  7] =        1         ci[  7] =        4
  cr[  9] =        1         ci[  9] =        2
  cr[ 10] =        1         ci[ 10] =        2
  cr[ 11] =        2         ci[ 11] =        1
  cr[ 12] =        2         ci[ 12] =        1
  cr[ 13] =        2         ci[ 13] =        1
  cr[ 14] =        2         ci[ 14] =        1
  cr[ 15] =        1         ci[ 15] =        2
  cr[ 16] =        1         ci[ 16] =        2
  cr[ 18] =        1         ci[ 18] =        2
  cr[ 19] =        1         ci[ 19] =        2
  cr[ 20] =        1         ci[ 20] =        2
  cr[ 21] =        1         ci[ 21] =        2
  cr[ 22] =        2         ci[ 22] =        1
  cr[ 23] =        2         ci[ 23] =        1
  cr[ 24] =        2         ci[ 24] =        1
  cr[ 25] =        2         ci[ 25] =        1
  cr[ 27] =        1         ci[ 27] =        1
  cr[ 28] =        1         ci[ 28] =        1
  cr[ 29] =        1         ci[ 29] =        1
  cr[ 30] =        1         ci[ 30] =        1
  cr[ 31] =        1         ci[ 31] =        1
  cr[ 32] =        1         ci[ 32] =        1
  cr[ 33] =        1         ci[ 33] =        1
  cr[ 34] =        1         ci[ 34] =        1

 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
```

```
  cr[  0] =      2.5         ci[  0] =      2.5
  cr[  1] =    -1.03         ci[  1] =    0.427
  cr[  2] =        0         ci[  2] =        0
  cr[  3] =  -0.0732         ci[  3] =  -0.0303
  cr[  4] =        0         ci[  4] =        0
  cr[  5] =   0.0303         ci[  5] =   0.0732
  cr[  6] =        0         ci[  6] =        0
  cr[  7] =   -0.427         ci[  7] =     1.03
  cr[  9] =      1.5         ci[  9] =      1.5
  cr[ 10] =   -0.427         ci[ 10] =    0.177
  cr[ 11] =        0         ci[ 11] =        0
  cr[ 12] =    0.177         ci[ 12] =   0.0732
  cr[ 13] =        0         ci[ 13] =        0
  cr[ 14] =  -0.0732         ci[ 14] =   -0.177
  cr[ 15] =        0         ci[ 15] =        0
  cr[ 16] =   -0.177         ci[ 16] =    0.427
  cr[ 18] =      1.5         ci[ 18] =      1.5
  cr[ 19] =    0.177         ci[ 19] =    0.427
  cr[ 20] =        0         ci[ 20] =        0
  cr[ 21] =  -0.0732         ci[ 21] =    0.177
  cr[ 22] =        0         ci[ 22] =        0
  cr[ 23] =   -0.177         ci[ 23] =   0.0732
  cr[ 24] =        0         ci[ 24] =        0
  cr[ 25] =   -0.427         ci[ 25] =   -0.177
  cr[ 27] =        1         ci[ 27] =        1
  cr[ 28] =        0         ci[ 28] =        0
  cr[ 29] =        0         ci[ 29] =        0
  cr[ 30] =        0         ci[ 30] =        0
  cr[ 31] =        0         ci[ 31] =        0
  cr[ 32] =        0         ci[ 32] =        0
  cr[ 33] =        0         ci[ 33] =        0
  cr[ 34] =        0         ci[ 34] =        0

< Backward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
  cr[  0] =        1         ci[  0] =        4
  cr[  1] =        2         ci[  1] =        3
  cr[  2] =        3         ci[  2] =        2
  cr[  3] =        4         ci[  3] =        1
  cr[  4] =        4         ci[  4] =        1
  cr[  5] =        3         ci[  5] =        2
  cr[  6] =        2         ci[  6] =        3
  cr[  7] =        1         ci[  7] =        4
  cr[  9] =        1         ci[  9] =        2
  cr[ 10] =        1         ci[ 10] =        2
  cr[ 11] =        2         ci[ 11] =        1
  cr[ 12] =        2         ci[ 12] =        1
  cr[ 13] =        2         ci[ 13] =        1
  cr[ 14] =        2         ci[ 14] =        1
  cr[ 15] =        1         ci[ 15] =        2
  cr[ 16] =        1         ci[ 16] =        2
  cr[ 18] =        1         ci[ 18] =        2
  cr[ 19] =        1         ci[ 19] =        2
  cr[ 20] =        1         ci[ 20] =        2
  cr[ 21] =        1         ci[ 21] =        2
  cr[ 22] =        2         ci[ 22] =        1
  cr[ 23] =        2         ci[ 23] =        1
  cr[ 24] =        2         ci[ 24] =        1
  cr[ 25] =        2         ci[ 25] =        1
  cr[ 27] =        1         ci[ 27] =        1
  cr[ 28] =        1         ci[ 28] =        1
  cr[ 29] =        1         ci[ 29] =        1
  cr[ 30] =        1         ci[ 30] =        1
  cr[ 31] =        1         ci[ 31] =        1
  cr[ 32] =        1         ci[ 32] =        1
  cr[ 33] =        1         ci[ 33] =        1
  cr[ 34] =        1         ci[ 34] =        1
```

6.3 多重 1 次元複素フーリエ変換 (複素引数型)

6.3.1 [非推奨]ASL_hfcmfb, ASL_gfcmfb 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_hfcmfb (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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	nt	I	1	入 力	生成するタスク数
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) $\mathrm{n} \geq 1$, $\mathrm{m} \geq 1$

(b) $\mathrm{incn} \geq 1$, $\mathrm{incm} \geq 1$

(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) isw=0 または isw=1, isw=−1

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

(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.	
3040	制限条件 (e) 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 6.3.2 $\left\{\begin{matrix}\mathrm{ASL\_hfcmbf}\\ \mathrm{ASL\_gfcmbf}\end{matrix}\right\}$ 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 6.3.2 $\left\{\begin{matrix}\mathrm{ASL\_hfcmbf}\\ \mathrm{ASL\_gfcmbf}\end{matrix}\right\}$ 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) **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**

6.3.2 (7) 使用例参照.

6.3.2 [非推奨]ASL_hfcmbf, ASL_gfcmbf
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_hfcmbf (n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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	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	nt	I	1	入 力	生成するタスク数
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) n $\geq$ 1, m $\geq$ 1

(b) incn $\geq$ 1, incm $\geq$ 1

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

(d) isw=1 または isw=-1

(e) nt $\geq$ 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.	
3040	制限条件 (e) 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 6.3.1 $\left\{\begin{matrix}\text{ASL\_hfcmfb}\\ \text{ASL\_gfcmfb}\end{matrix}\right\}$ 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) **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), nt=2

(c) 主プログラム

```
/*      C interface example for ASL_hfcmfb , ASL_hfcmbf */

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

int main()
{
    int ld=35;
    int n;      int m;
    double _Complex *c;
    int incn;   int incm;
    int isw;
    int ifax[20];   double *trigs;
    double _Complex *wk;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_hfcmfb , ASL_hfcmbf ***\n" );
    printf( "\n     ** Input **\n\n" );

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

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

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

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

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

    printf("\t   n    = %d \n", n );
    printf("\t   m    = %d \n", m );
    printf("\t   incn = %d \n", incn );
    printf("\t   incm = %d \n", incm );
    printf("\t   nt   = %d \n\n", nt );

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

    fclose( fp );


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

    isw = 1;
    ierr = ASL_hfcmfb(n, m, c, incn, incm, isw, ifax, trigs, wk, nt);
```

```
    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            c[i*incn+j*incm] /= n ;
        }
    }

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

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

    isw = -1;
    ierr = ASL_hfcmbf(n, m, c, incn, incm, isw, ifax, trigs, wk, nt);

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

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


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

    return 0;
}
```

(d) 出力結果

```
*** ASL_hfcmfb , ASL_hfcmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 9
   nt   = 2

  Real Part                  Imaginary Part
  creal(c[  0]) =        1         cimag(c[  0]) =        4
  creal(c[  1]) =        2         cimag(c[  1]) =        3
  creal(c[  2]) =        3         cimag(c[  2]) =        2
  creal(c[  3]) =        4         cimag(c[  3]) =        1
  creal(c[  4]) =        4         cimag(c[  4]) =        1
  creal(c[  5]) =        3         cimag(c[  5]) =        2
  creal(c[  6]) =        2         cimag(c[  6]) =        3
  creal(c[  7]) =        1         cimag(c[  7]) =        4
  creal(c[  9]) =        1         cimag(c[  9]) =        2
  creal(c[ 10]) =        1         cimag(c[ 10]) =        2
  creal(c[ 11]) =        2         cimag(c[ 11]) =        1
  creal(c[ 12]) =        2         cimag(c[ 12]) =        1
  creal(c[ 13]) =        2         cimag(c[ 13]) =        1
  creal(c[ 14]) =        2         cimag(c[ 14]) =        1
  creal(c[ 15]) =        1         cimag(c[ 15]) =        2
  creal(c[ 16]) =        1         cimag(c[ 16]) =        2
  creal(c[ 18]) =        1         cimag(c[ 18]) =        2
  creal(c[ 19]) =        1         cimag(c[ 19]) =        2
  creal(c[ 20]) =        1         cimag(c[ 20]) =        2
  creal(c[ 21]) =        1         cimag(c[ 21]) =        2
  creal(c[ 22]) =        2         cimag(c[ 22]) =        1
  creal(c[ 23]) =        2         cimag(c[ 23]) =        1
  creal(c[ 24]) =        2         cimag(c[ 24]) =        1
  creal(c[ 25]) =        2         cimag(c[ 25]) =        1
  creal(c[ 27]) =        1         cimag(c[ 27]) =        1
  creal(c[ 28]) =        1         cimag(c[ 28]) =        1
  creal(c[ 29]) =        1         cimag(c[ 29]) =        1
  creal(c[ 30]) =        1         cimag(c[ 30]) =        1
```

```
  creal(c[ 31]) =        1         cimag(c[ 31]) =        1
  creal(c[ 32]) =        1         cimag(c[ 32]) =        1
  creal(c[ 33]) =        1         cimag(c[ 33]) =        1
  creal(c[ 34]) =        1         cimag(c[ 34]) =        1

 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                   Imaginary Part
  creal(c[  0]) =      2.5         cimag(c[  0]) =      2.5
  creal(c[  1]) =    -1.03         cimag(c[  1]) =    0.427
  creal(c[  2]) =        0         cimag(c[  2]) =        0
  creal(c[  3]) =  -0.0732         cimag(c[  3]) =  -0.0303
  creal(c[  4]) =        0         cimag(c[  4]) =        0
  creal(c[  5]) =   0.0303         cimag(c[  5]) =   0.0732
  creal(c[  6]) =        0         cimag(c[  6]) =        0
  creal(c[  7]) =   -0.427         cimag(c[  7]) =     1.03
  creal(c[  9]) =      1.5         cimag(c[  9]) =      1.5
  creal(c[ 10]) =   -0.427         cimag(c[ 10]) =    0.177
  creal(c[ 11]) =        0         cimag(c[ 11]) =        0
  creal(c[ 12]) =    0.177         cimag(c[ 12]) =   0.0732
  creal(c[ 13]) =        0         cimag(c[ 13]) =        0
  creal(c[ 14]) =  -0.0732         cimag(c[ 14]) =   -0.177
  creal(c[ 15]) =        0         cimag(c[ 15]) =        0
  creal(c[ 16]) =   -0.177         cimag(c[ 16]) =    0.427
  creal(c[ 18]) =      1.5         cimag(c[ 18]) =      1.5
  creal(c[ 19]) =    0.177         cimag(c[ 19]) =    0.427
  creal(c[ 20]) =        0         cimag(c[ 20]) =        0
  creal(c[ 21]) =  -0.0732         cimag(c[ 21]) =    0.177
  creal(c[ 22]) =        0         cimag(c[ 22]) =        0
  creal(c[ 23]) =   -0.177         cimag(c[ 23]) =   0.0732
  creal(c[ 24]) =        0         cimag(c[ 24]) =        0
  creal(c[ 25]) =   -0.427         cimag(c[ 25]) =   -0.177
  creal(c[ 27]) =        1         cimag(c[ 27]) =        1
  creal(c[ 28]) =        0         cimag(c[ 28]) =        0
  creal(c[ 29]) =        0         cimag(c[ 29]) =        0
  creal(c[ 30]) =        0         cimag(c[ 30]) =        0
  creal(c[ 31]) =        0         cimag(c[ 31]) =        0
  creal(c[ 32]) =        0         cimag(c[ 32]) =        0
  creal(c[ 33]) =        0         cimag(c[ 33]) =        0
  creal(c[ 34]) =        0         cimag(c[ 34]) =        0

< Backward Transform >
ierr =      0

Solution

  Real Part                   Imaginary Part
  creal(c[  0]) =        1         cimag(c[  0]) =        4
  creal(c[  1]) =        2         cimag(c[  1]) =        3
  creal(c[  2]) =        3         cimag(c[  2]) =        2
  creal(c[  3]) =        4         cimag(c[  3]) =        1
  creal(c[  4]) =        4         cimag(c[  4]) =        1
  creal(c[  5]) =        3         cimag(c[  5]) =        2
  creal(c[  6]) =        2         cimag(c[  6]) =        3
  creal(c[  7]) =        1         cimag(c[  7]) =        4
  creal(c[  9]) =        1         cimag(c[  9]) =        2
  creal(c[ 10]) =        1         cimag(c[ 10]) =        2
  creal(c[ 11]) =        2         cimag(c[ 11]) =        1
  creal(c[ 12]) =        2         cimag(c[ 12]) =        1
  creal(c[ 13]) =        2         cimag(c[ 13]) =        1
  creal(c[ 14]) =        2         cimag(c[ 14]) =        1
  creal(c[ 15]) =        1         cimag(c[ 15]) =        2
  creal(c[ 16]) =        1         cimag(c[ 16]) =        2
  creal(c[ 18]) =        1         cimag(c[ 18]) =        2
  creal(c[ 19]) =        1         cimag(c[ 19]) =        2
  creal(c[ 20]) =        1         cimag(c[ 20]) =        2
  creal(c[ 21]) =        1         cimag(c[ 21]) =        2
  creal(c[ 22]) =        2         cimag(c[ 22]) =        1
  creal(c[ 23]) =        2         cimag(c[ 23]) =        1
  creal(c[ 24]) =        2         cimag(c[ 24]) =        1
  creal(c[ 25]) =        2         cimag(c[ 25]) =        1
  creal(c[ 27]) =        1         cimag(c[ 27]) =        1
  creal(c[ 28]) =        1         cimag(c[ 28]) =        1
  creal(c[ 29]) =        1         cimag(c[ 29]) =        1
  creal(c[ 30]) =        1         cimag(c[ 30]) =        1
  creal(c[ 31]) =        1         cimag(c[ 31]) =        1
  creal(c[ 32]) =        1         cimag(c[ 32]) =        1
  creal(c[ 33]) =        1         cimag(c[ 33]) =        1
  creal(c[ 34]) =        1         cimag(c[ 34]) =        1
```

6.4 多重 1 次元実フーリエ変換

6.4.1 [非推奨]ASL_qfrmfb, ASL_pfrmfb 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_qfrmfb (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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	nt	I	1	入 力	生成するタスク数
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) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(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 または isw=1, isw=−1

(e) nt ≥ 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.	
3040	制限条件 (e) 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 6.4.2 $\left\{\begin{matrix}\text{ASL\_qfrmbf}\\ \text{ASL\_pfrmbf}\end{matrix}\right\}$ 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 6.4.2 $\left\{\begin{matrix}\text{ASL_qfrmbf}\\ \text{ASL_pfrmbf}\end{matrix}\right\}$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) **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**

6.4.2 (7) 使用例参照.

6.4.2 [非推奨]ASL_qfrmbf, ASL_pfrmbf 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_qfrmbf (n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

単精度関数:

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

(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	nt	I	1	入 力	生成するタスク数
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) n ≥ 1, m ≥ 1

(b) incn ≥ 1, incm ≥ 1

(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 または isw=−1

(e) nt ≥ 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.	
3040	制限条件 (e) 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 6.4.1 $\left\{\begin{matrix}\text{ASL\_qfrmfb}\\ \text{ASL\_pfrmfb}\end{matrix}\right\}$ 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 $r_{k,l}(k=0,\cdots,n-1;\ l=1,\cdots,m)$ 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) 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) **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), nt=2

(c) 主プログラム

```
/*      C interface example for ASL_qfrmfb , ASL_qfrmbf */

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

int main()
{
    int ld=46;
    int n;      int m;
    double *r;
    int incn;   int incm;
    int isw;
    int ifax[20];   double *trigs;
    double *wk;
    int nt;
    int ierr;
    int i,j;
    FILE *fp;

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

    printf( "    *** ASL_qfrmfb , ASL_qfrmbf ***\n" );
    printf( "\n     ** Input **\n\n" );

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

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

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

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

    for( j=0 ; j<m ; j++ )
    {
        for( i=0 ; i<n ; i++ )
        {
            fscanf( fp, "%lf", &r[i*incn+j*incm] );
        }
    }

    printf("\t   n    = %d \n", n );
    printf("\t   m    = %d \n", m );
    printf("\t   incn = %d \n", incn );
    printf("\t   incm = %d \n", incm );
    printf("\t   nt   = %d \n\n", nt );

    printf( "\t  Real Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "\t" );
        for( i=0 ; i<n ; i++ )
        {
            printf( "    r[%3d] =%4.1f",
                i*incn+j*incm, r[i*incn+j*incm] );
            if((i+1)%4==0) printf( "\n\t" );
        }
        printf( "\n" );
    }

    fclose( fp );


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

    isw = 1;
    ierr = ASL_qfrmfb(n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

    for( j=0 ; j<m ; j++ )
```

```
    {
        for( i=0 ; i<n+2 ; i++ )
        {
            r[i*incn+j*incm] /= n ;
        }
    }

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

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

    isw = -1;
    ierr = ASL_qfrmbf(n, m, r, incn, incm, isw, ifax, trigs, wk, nt);

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

    printf( "\n\tSolution\n\n" );
    printf( "\t  Real Part\n" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "\t" );
        for( i=0 ; i<n+2 ; i++ )
        {
            printf( "    r[%3d] =%4.1f",
                i*incn+j*incm, r[i*incn+j*incm] );
            if((i+1)%4==0) printf( "\n\t" );
        }
        printf( "\n" );
    }


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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfrmfb , ASL_qfrmbf ***

 ** Input **

   n    = 8
   m    = 4
   incn = 1
   incm = 12
   nt   = 2

  Real Part
    r[  0] = 1.0    r[  1] = 2.0    r[  2] = 3.0    r[  3] = 4.0
    r[  4] = 5.0    r[  5] = 6.0    r[  6] = 7.0    r[  7] = 8.0

    r[ 12] = 1.0    r[ 13] = 1.0    r[ 14] = 2.0    r[ 15] = 2.0
    r[ 16] = 3.0    r[ 17] = 3.0    r[ 18] = 4.0    r[ 19] = 4.0

    r[ 24] = 1.0    r[ 25] = 1.0    r[ 26] = 1.0    r[ 27] = 1.0
    r[ 28] = 2.0    r[ 29] = 2.0    r[ 30] = 2.0    r[ 31] = 2.0

    r[ 36] = 1.0    r[ 37] = 1.0    r[ 38] = 1.0    r[ 39] = 1.0
    r[ 40] = 1.0    r[ 41] = 1.0    r[ 42] = 1.0    r[ 43] = 1.0


 ** Output **

< Forward Transform >
ierr =      0

Solution

  Real Part                  Imaginary Part
  r[  0] =      4.5           r[  1] =        0
  r[  2] =     -0.5           r[  3] =     1.21
  r[  4] =     -0.5           r[  5] =      0.5
  r[  6] =     -0.5           r[  7] =    0.207
  r[  8] =     -0.5           r[  9] =        0

  r[ 12] =      2.5           r[ 13] =        0
  r[ 14] =    -0.25           r[ 15] =    0.604
  r[ 16] =    -0.25           r[ 17] =     0.25
  r[ 18] =    -0.25           r[ 19] =    0.104
```

```
  r[ 20] =        0           r[ 21] =        0

  r[ 24] =      1.5           r[ 25] =        0
  r[ 26] =   -0.125           r[ 27] =    0.302
  r[ 28] =        0           r[ 29] =        0
  r[ 30] =   -0.125           r[ 31] =   0.0518
  r[ 32] =        0           r[ 33] =        0

  r[ 36] =        1           r[ 37] =        0
  r[ 38] =        0           r[ 39] =        0
  r[ 40] =        0           r[ 41] =        0
  r[ 42] =        0           r[ 43] =        0
  r[ 44] =        0           r[ 45] =        0


< Backward Transform >
ierr =      0

Solution

  Real Part
    r[  0] = 1.0    r[  1] = 2.0    r[  2] = 3.0    r[  3] = 4.0
    r[  4] = 5.0    r[  5] = 6.0    r[  6] = 7.0    r[  7] = 8.0
    r[  8] = 0.0    r[  9] = 0.0
    r[ 12] = 1.0    r[ 13] = 1.0    r[ 14] = 2.0    r[ 15] = 2.0
    r[ 16] = 3.0    r[ 17] = 3.0    r[ 18] = 4.0    r[ 19] = 4.0
    r[ 20] = 0.0    r[ 21] = 0.0
    r[ 24] = 1.0    r[ 25] = 1.0    r[ 26] = 1.0    r[ 27] = 1.0
    r[ 28] = 2.0    r[ 29] = 2.0    r[ 30] = 2.0    r[ 31] = 2.0
    r[ 32] = 0.0    r[ 33] = 0.0
    r[ 36] = 1.0    r[ 37] = 1.0    r[ 38] = 1.0    r[ 39] = 1.0
    r[ 40] = 1.0    r[ 41] = 1.0    r[ 42] = 1.0    r[ 43] = 1.0
    r[ 44] = 0.0    r[ 45] = 0.0
```

6.5 2 次元複素フーリエ変換 (実数引数型)

6.5.1 [非推奨]ASL_qfc2fb, ASL_pfc2fb
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_qfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfc2fb (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw=0, isw=1 または isw=−1

(d) nt ≥ 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.	
3030	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) 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 6.5.2 $\left\{\begin{matrix}\mathrm{ASL_qfc2bf}\\ \mathrm{ASL_pfc2bf}\end{matrix}\right\}$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 6.5.2 $\left\{\begin{matrix}\mathrm{ASL_qfc2bf}\\ \mathrm{ASL_pfc2bf}\end{matrix}\right\}$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) **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**

6.5.2 (7) 使用例参照.

6.5.2 [非推奨]ASL_qfc2bf, ASL_pfc2bf
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_qfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfc2bf (nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw=1 または isw=−1

(d) nt ≥ 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.	
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 6.5.1 $\left\{\begin{matrix}\text{ASL\_qfc2fb}\\ \text{ASL\_pfc2fb}\end{matrix}\right\}$ 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) **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 cr, ci, nx=5, ny=4, lx=5, ly=5, isw=1(順変換) および isw=−1(逆変換), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_qfc2fb , ASL_qfc2bf */

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

int main()
{
```

```
int nx = 5; int ny = 4;
int lx = 5; int ly = 5;
double *cr; double *ci;
int isw;
int ifax[40];
double *trigs;
double *wk;
int nt = 2;
int ierr;
int i,j;

printf( "    *** ASL_qfc2fb , ASL_qfc2bf ***\n" );
printf( "\n     ** Input **\n" );

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

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

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

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

printf( "\tnx = %6d\n", nx );
printf( "\tny = %6d\n", ny );
printf( "\tnt = %6d\n", nt );

for( j=1 ; j<=ny ; j++ )
{
    for( i=1 ; i<=nx ; i++ )
    {
        cr[(i-1)+lx*(j-1)]=(double)(i+j) ;
        ci[(i-1)+lx*(j-1)]=(double)(i*j)/(double)(nx*ny) ;
    }
}

printf( "\tcr[ix][iy]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
    }
    printf( "\n" );
}


isw = 1;
ierr = ASL_qfc2fb(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

for( i=0 ; i<lx*ly ; i++)
{
    cr[i] /= (double)(nx*ny);
    ci[i] /= (double)(nx*ny);
}

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

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

printf( "\tcr[ix][iy]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
    }
    printf( "\n" );
}

isw = -1;
ierr = ASL_qfc2bf(nx, ny, cr, ci, lx, ly, isw, ifax, trigs, wk, nt);

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

```
        printf( "\tcr[ix][iy]\n" );
        for( i=0 ; i<nx ; i++ )
        {
            for( j=0 ; j<ny ; j++ )
            {
                printf( "\t(%8.3g,%8.3g)", cr[i+lx*j], ci[i+lx*j] );
            }
            printf( "\n" );
        }


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

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_qfc2fb , ASL_qfc2bf ***

         ** Input **
        nx =      5
        ny =      4
        nt =      2
        cr[ix][iy]
        (       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
        (       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
        (       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
        (       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
        (       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)

         ** Output **
        < Forward Transform >
        ierr =      0
        cr[ix][iy]
        (     5.5,   0.375) (  -0.575,   0.425) (    -0.5,  -0.075) (  -0.425,  -0.575)
        (  -0.586,   0.626) (  0.0297, -0.0047) (  0.0172,  0.0125) (  0.0047,  0.0297)
        (   -0.52,     0.1) (  0.0166, 0.00844) ( 0.00406,  0.0125) (-0.00844,  0.0166)
        (   -0.48,  -0.225) ( 0.00844,  0.0166) (-0.00406,  0.0125) ( -0.0166, 0.00844)
        (  -0.414,  -0.751) ( -0.0047,  0.0297) ( -0.0172,  0.0125) ( -0.0297, -0.0047)
        < Backward Transform >
        ierr =      0
        cr[ix][iy]
        (       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
        (       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
        (       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
        (       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
        (       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)
```

6.6 2 次元複素フーリエ変換 (複素引数型)

6.6.1 [非推奨]ASL_hfc2fb, ASL_gfc2fb 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_hfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_gfc2fb (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

(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	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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw=0, isw=1 または isw=−1

(d) nt ≥ 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.	
3030	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) 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 6.6.2 $\left\{\begin{matrix}\mathrm{ASL_hfc2bf}\\ \mathrm{ASL_gfc2bf}\end{matrix}\right\}$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 6.6.2 $\left\{\begin{matrix}\mathrm{ASL_hfc2bf}\\ \mathrm{ASL_gfc2bf}\end{matrix}\right\}$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) **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**

6.6.2 (7) 使用例参照.

6.6.2 [非推奨]ASL_hfc2bf, ASL_gfc2bf 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_hfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_gfc2bf (nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

(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	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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx≤lx, ny≤ly

(c) isw=1 または isw=−1

(d) nt ≥ 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.	
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 6.6.1 $\left\{\begin{matrix}\text{ASL\_hfc2fb}\\ \text{ASL\_gfc2fb}\end{matrix}\right\}$ 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) **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), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_hfc2fb , ASL_hfc2bf */

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

int main()
{
    int nx = 5; int ny = 4;
    int lx = 5; int ly = 5;
    double _Complex *c;
    int isw;
    int ifax[40];
    double *trigs;
    double _Complex *wk;
    int nt = 2;
    int ierr;
    int i,j;

    printf( "    *** ASL_hfc2fb , ASL_hfc2bf ***\n" );
    printf( "\n     ** Input **\n" );

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

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

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

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnt = %6d\n", nt );

    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            c[(i-1)+lx*(j-1)]=(double)(i+j) + (double)(i*j)/(double)(nx*ny) * _Complex_I;
        }
    }

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }


    isw = 1;
    ierr = ASL_hfc2fb(nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly ; i++)
    {
        c[i] /= (double)(nx*ny);
    }
```

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

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

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_hfc2bf(nx, ny, c, lx, ly, isw, ifax, trigs, wk, nt);

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

    printf( "\tc[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j]), cimag(c[i+lx*j]) );
        }
        printf( "\n" );
    }


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

    return 0;
}
```

(d) 出力結果

```
*** ASL_hfc2fb , ASL_hfc2bf ***

 ** Input **
nx =      5
ny =      4
nt =      2
c[ix][iy]
(       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
(       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
(       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
(       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
(       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)

 ** Output **
< Forward Transform >
ierr =      0
c[ix][iy]
(     5.5,   0.375) (  -0.575,   0.425) (    -0.5,  -0.075) (  -0.425,  -0.575)
(  -0.586,   0.626) (  0.0297, -0.0047) (  0.0172,  0.0125) (  0.0047,  0.0297)
(   -0.52,     0.1) (  0.0166, 0.00844) ( 0.00406,  0.0125) (-0.00844,  0.0166)
(   -0.48,  -0.225) ( 0.00844,  0.0166) (-0.00406,  0.0125) ( -0.0166, 0.00844)
(  -0.414,  -0.751) ( -0.0047,  0.0297) ( -0.0172,  0.0125) ( -0.0297, -0.0047)
< Backward Transform >
ierr =      0
c[ix][iy]
(       2,    0.05) (       3,     0.1) (       4,    0.15) (       5,     0.2)
(       3,     0.1) (       4,     0.2) (       5,     0.3) (       6,     0.4)
(       4,    0.15) (       5,     0.3) (       6,    0.45) (       7,     0.6)
(       5,     0.2) (       6,     0.4) (       7,     0.6) (       8,     0.8)
(       6,    0.25) (       7,     0.5) (       8,    0.75) (       9,       1)
```

6.7　2 次元実フーリエ変換

6.7.1　[非推奨]ASL_qfr2fb, ASL_pfr2fb
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_qfr2fb (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfr2fb (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx が奇数の時, nx+1≤lx, ny≤ly
nx が偶数の時, nx+2≤lx, ny≤ly

(c) isw=0, isw=1 または isw=−1

(d) nt ≥ 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.	
3030	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.

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

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 6.7.2 $\left\{\begin{matrix}\mathrm{ASL\_qfr2bf} \\ \mathrm{ASL\_pfr2bf}\end{matrix}\right\}$ 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 6.7.2 $\left\{\begin{matrix}\text{ASL_qfr2bf}\\ \text{ASL_pfr2bf}\end{matrix}\right\}$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) **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**

6.7.2 (7) 使用例参照.

6.7.2 [非推奨]ASL_qfr2bf, ASL_pfr2bf 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_qfr2bf (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfr2bf (nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2

(b) nx が奇数の時, nx+1≤lx, ny≤ly
nx が偶数の時, nx+2≤lx, ny≤ly

(c) isw=1 または isw=−1

(d) nt ≥ 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.	
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 6.7.1 $\left\{\begin{matrix}\text{ASL\_qfr2fb}\\ \text{ASL\_pfr2fb}\end{matrix}\right\}$ 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, $c_{j_x,j_y}\quad (j_x = \lfloor \frac{n_x}{2} \rfloor + 1, \cdots, n_x - 1;\ j_y = 0, \cdots, n_y - 1)$ 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.

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

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) **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), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_qfr2fb , ASL_qfr2bf */

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

int main()
{
    int nx =  6;    int ny =  4;
    int lx = 10;    int ly =  5;
    double *r;
    int isw;
    int ifax[40];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j;

    printf( "    *** ASL_qfr2fb , ASL_qfr2bf ***\n" );
    printf( "\n     ** Input **\n" );

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

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

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

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnt = %6d\n", nt );

    for( j=1 ; j<=ny ; j++ )
    {
        for( i=1 ; i<=nx ; i++ )
        {
            r[i-1+lx*(j-1)]=(double)(nx+ny)/(double)(i+j) ;
        }
    }

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }


    isw = 1;
    ierr = ASL_qfr2fb(nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly ; i++)
    {
        r[i] /= (double)(nx*ny);
    }

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

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

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx+2 ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_qfr2bf(nx, ny, r, lx, ly, isw, ifax, trigs, wk, nt);

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

    printf( "\tr[ix][iy]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j] );
        }
        printf( "\n" );
    }


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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfr2fb , ASL_qfr2bf ***

 ** Input **
nx =      6
ny =      4
nt =      2
r[ix][iy]
          5         3.33          2.5            2
       3.33          2.5            2         1.67
        2.5            2         1.67         1.43
          2         1.67         1.43         1.25
       1.67         1.43         1.25         1.11
       1.43         1.25         1.11            1

 ** Output **
< Forward Transform >
ierr =      0
r[ix][iy]
       1.94        0.249        0.219        0.249
          0       -0.155            0        0.155
      0.296       0.0585       0.0761        0.119
     -0.247      -0.0939      -0.0447     -0.00945
      0.229       0.0557        0.058       0.0794
    -0.0928      -0.0535      -0.0186       0.0102
      0.219       0.0637       0.0547       0.0637
          0      -0.0301            0       0.0301
< Backward Transform >
ierr =      0
r[ix][iy]
          5         3.33          2.5            2
       3.33          2.5            2         1.67
        2.5            2         1.67         1.43
          2         1.67         1.43         1.25
       1.67         1.43         1.25         1.11
       1.43         1.25         1.11            1
```

6.8 3 次元複素フーリエ変換 (実数引数型)

6.8.1 [非推奨]ASL_qfc3fb, ASL_pfc3fb
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_qfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfc3fb (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
14	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≥2, ny≥2, nz≥2

(b) nx≤lx,　ny≤ly,　nz≤lz

(c) isw=0, isw=1 または isw=−1

(d) nt ≥ 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.	
3030	制限条件 (d) 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 6.8.2 $\left\{\begin{matrix}\mathrm{ASL_qfc3bf}\\ \mathrm{ASL_pfc3bf}\end{matrix}\right\}$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 6.8.2 $\left\{\begin{matrix}\mathrm{ASL_qfc3bf}\\ \mathrm{ASL_pfc3bf}\end{matrix}\right\}$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) **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**

6.8.2 (7) 使用例参照.

6.8.2 [非推奨]ASL_qfc3bf, ASL_pfc3bf 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_qfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfc3bf (nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
14	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≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw=1 または isw=−1

(d) nt ≥ 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.	
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
6.8.1 $\left\{\begin{matrix}\text{ASL_qfc3fb}\\ \text{ASL_pfc3fb}\end{matrix}\right\}$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

$$\begin{aligned}&\hat{c}_{k_x,k_y,k_z} = n_x n_y n_z c_{k_x,k_y,k_z}\\ &\quad(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1,\ k_z = 0,\cdots,n_z-1)\end{aligned}$$

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) 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) **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

$$\begin{aligned}&c_{k_x,k_y,k_z} = \frac{n_x+n_y+n_z}{(k_x+1)+(k_y+1)+(k_z+1)} + \sqrt{-1}\frac{(k_x+1)(k_y+1)(k_z+1)}{n_x n_y n_z}\\ &\quad(k_x = 0,\cdots,n_x-1;\ k_y = 0,\cdots,n_y-1;\ k_z = 0,\cdots,n_z-1)\end{aligned}$$

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), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_qfc3fb , ASL_qfc3bf */

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

int main()
{
    int nx = 5; int ny = 4; int nz = 3;
    int lx = 5; int ly = 5; int lz = 3;
    double *cr; double *ci;
    int isw;
    int ifax[60];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j,k;

    printf( "    *** ASL_qfc3fb , ASL_qfc3bf ***\n" );
    printf( "\n     ** Input **\n" );

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

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

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

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

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnz = %6d\n", nz );
    printf( "\tnt = %6d\n", nt );

    for( k=1 ; k<=nz ; k++ )
    {
        for( j=1 ; j<=ny ; j++ )
        {
            for( i=1 ; i<=nx ; i++ )
            {
                cr[i-1+lx*(j-1)+lx*ly*(k-1)]=(double)(nx+ny+nz)/(double)(i+j+k) ;
                ci[i-1+lx*(j-1)+lx*ly*(k-1)]=(double)(i*j*k)/(double)(nx*ny*nz) ;
            }
        }
    }

    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
```

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


    isw = 1;
    ierr = ASL_qfc3fb(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

    for( i=0 ; i<lx*ly*lz ; i++)
    {
        cr[i] /= (double)(nx*ny*nz);
        ci[i] /= (double)(nx*ny*nz);
    }

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

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

    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }

    isw = -1;
    ierr = ASL_qfc3bf(nx, ny, nz, cr, ci, lx, ly, lz, isw, ifax, trigs, wk, nt);

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

    printf( "\tcr[ix][iy][1] ci[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j        ], ci[i+lx*j        ] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][2] ci[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*1], ci[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tcr[ix][iy][3] ci[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", cr[i+lx*j+lx*ly*2], ci[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }



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

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfc3fb , ASL_qfc3bf ***

 ** Input **
nx =      5
ny =      4
nz =      3
nt =      2
cr[ix][iy][1] ci[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
cr[ix][iy][2] ci[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
cr[ix][iy][3] ci[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)

 ** Output **
< Forward Transform >
ierr =      0
cr[ix][iy][1] ci[ix][iy][1]
(    1.74,    0.25) (   0.102,   -0.16) (   0.137,   -0.05) (   0.202,    0.06)
(   0.108,  -0.189) (  0.0379, -0.0469) (  0.0406, -0.0125) (  0.0525,   0.016)
(   0.125, -0.0784) (   0.034, -0.0168) (  0.0261, 0.00288) (  0.0254,  0.0209)
(   0.152,-0.00492) (  0.0366, 0.00116) (  0.0207,  0.0138) (   0.012,   0.028)
(   0.223,   0.106) (  0.0462,  0.0236) (  0.0177,  0.0292) (-0.00166,  0.0406)
cr[ix][iy][2] ci[ix][iy][2]
(   0.106,  -0.127) (  0.0407, -0.0223) (  0.0315, 0.00295) (  0.0297,  0.0255)
(  0.0419, -0.0317) (-0.00167,-0.00877) (  0.0025,-0.00799) ( 0.00901,-0.00976)
(  0.0317,-0.00698) ( 0.00134,-0.00743) ( 0.00423,-0.00524) ( 0.00924,-0.00424)
(  0.0297,  0.0084) ( 0.00473,-0.00711) ( 0.00655,-0.00336) (  0.0108,  0.0001)
(  0.0318,  0.0285) (  0.0112,-0.00921) (  0.0118,-0.00179) (   0.016, 0.00627)
cr[ix][iy][3] ci[ix][iy][3]
(   0.178, 0.00231) (  0.0403,   0.014) (   0.017,   0.022) ( 0.00125,  0.0329)
(  0.0484, 0.00885) ( 0.00516, -0.0104) ( 0.00849,-0.00569) (  0.0153, -0.0016)
(  0.0244,  0.0163) ( 0.00692, -0.0061) (  0.0076,-0.00159) (  0.0107, 0.00322)
(  0.0129,  0.0239) ( 0.00961, -0.0029) ( 0.00799, 0.00185) ( 0.00849,  0.0078)
( 0.00117,   0.036) (  0.0163,0.000768) (  0.0106, 0.00714) ( 0.00733,  0.0161)
< Backward Transform >
ierr =      0
cr[ix][iy][1] ci[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
cr[ix][iy][2] ci[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
cr[ix][iy][3] ci[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)
```

6.9 3 次元複素フーリエ変換 (複素引数型)

6.9.1 [非推奨]ASL_hfc3fb, ASL_gfc3fb
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_hfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_gfc3fb (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

(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
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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw=0, isw=1 または isw=−1

(d) nt ≥ 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.	
3030	制限条件 (d) 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 6.9.2 $\left\{\begin{matrix}\mathrm{ASL_hfc3bf}\\ \mathrm{ASL_gfc3bf}\end{matrix}\right\}$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 6.9.2 $\left\{\begin{matrix}\mathrm{ASL_hfc3bf}\\ \mathrm{ASL_gfc3bf}\end{matrix}\right\}$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) **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**

6.9.2 (7) 使用例参照.

6.9.2 [非推奨]ASL_hfc3bf, ASL_gfc3bf 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_hfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_gfc3bf (nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2, nz≥2

(b) nx≤lx, ny≤ly, nz≤lz

(c) isw=1 または isw=−1

(d) nt ≥ 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.	
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
6.9.1 $\left\{\begin{matrix}\text{ASL_hfc3fb}\\ \text{ASL_gfc3fb}\end{matrix}\right\}$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) **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), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_hfc3fb , ASL_hfc3bf */

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

int main()
{
    int nx = 5; int ny = 4; int nz = 3;
    int lx = 5; int ly = 5; int lz = 3;
    double _Complex *c;
    int isw;
    int ifax[60];
    double *trigs;
    double _Complex *wk;
    int nt = 2;
    int ierr;
    int i,j,k;

    printf( "    *** ASL_hfc3fb , ASL_hfc3bf ***\n" );
    printf( "\n     ** Input **\n" );

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

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

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

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnz = %6d\n", nz );
    printf( "\tnt = %6d\n", nt );

    for( k=1 ; k<=nz ; k++ )
    {
        for( j=1 ; j<=ny ; j++ )
        {
            for( i=1 ; i<=nx ; i++ )
            {
                c[(i-1)+lx*(j-1)+lx*ly*(k-1)]=(double)(nx+ny+nz)/(double)(i+j+k)
                                              +(double)(i*j*k)/(double)(nx*ny*nz) * _Complex_I;
            }
        }
    }

    printf( "\tc[ix][iy][1]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
        }
        printf( "\n" );
    }

    printf( "\tc[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
```

```
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
    }
    printf( "\n" );
}

printf( "\tc[ix][iy][3]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
    }
    printf( "\n" );
}


isw = 1;
ierr = ASL_hfc3fb(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

for( i=0 ; i<lx*ly*lz ; i++)
{
    c[i] /= (double)(nx*ny*nz);
}

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

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

printf( "\tc[ix][iy][1]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
    }
    printf( "\n" );
}

printf( "\tc[ix][iy][2]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
    }
    printf( "\n" );
}

printf( "\tc[ix][iy][3]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
    }
    printf( "\n" );
}

isw = -1;
ierr = ASL_hfc3bf(nx, ny, nz, c, lx, ly, lz, isw, ifax, trigs, wk, nt);

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

printf( "\tc[ix][iy][1]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j        ]), cimag(c[i+lx*j        ]) );
    }
    printf( "\n" );
}

printf( "\tc[ix][iy][2]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*1]), cimag(c[i+lx*j+lx*ly*1]) );
    }
    printf( "\n" );
}

printf( "\tc[ix][iy][3]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t(%8.3g,%8.3g)", creal(c[i+lx*j+lx*ly*2]), cimag(c[i+lx*j+lx*ly*2]) );
    }
```

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


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

    return 0;
}
```

(d) 出力結果

```
*** ASL_hfc3fb , ASL_hfc3bf ***

 ** Input **
nx =      5
ny =      4
nz =      3
nt =      2
c[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
c[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
c[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)

 ** Output **
< Forward Transform >
ierr =      0
c[ix][iy][1]
(    1.74,    0.25) (   0.102,   -0.16) (   0.137,   -0.05) (   0.202,    0.06)
(   0.108,  -0.189) (  0.0379, -0.0469) (  0.0406, -0.0125) (  0.0525,   0.016)
(   0.125, -0.0784) (   0.034, -0.0168) (  0.0261, 0.00288) (  0.0254,  0.0209)
(   0.152,-0.00492) (  0.0366, 0.00116) (  0.0207,  0.0138) (   0.012,   0.028)
(   0.223,   0.106) (  0.0462,  0.0236) (  0.0177,  0.0292) (-0.00166,  0.0406)
c[ix][iy][2]
(   0.106,  -0.127) (  0.0407, -0.0223) (  0.0315, 0.00295) (  0.0297,  0.0255)
(  0.0419, -0.0317) (-0.00167,-0.00877) (  0.0025,-0.00799) ( 0.00901,-0.00976)
(  0.0317,-0.00698) ( 0.00134,-0.00743) ( 0.00423,-0.00524) ( 0.00924,-0.00424)
(  0.0297,  0.0084) ( 0.00473,-0.00711) ( 0.00655,-0.00336) (  0.0108,  0.0001)
(  0.0318,  0.0285) (  0.0112,-0.00921) (  0.0118,-0.00179) (   0.016, 0.00627)
c[ix][iy][3]
(   0.178, 0.00231) (  0.0403,   0.014) (   0.017,   0.022) ( 0.00125,  0.0329)
(  0.0484, 0.00885) ( 0.00516, -0.0104) ( 0.00849,-0.00569) (  0.0153, -0.0016)
(  0.0244,  0.0163) ( 0.00692, -0.0061) (  0.0076,-0.00159) (  0.0107, 0.00322)
(  0.0129,  0.0239) ( 0.00961, -0.0029) ( 0.00799, 0.00185) ( 0.00849,  0.0078)
( 0.00117,   0.036) (  0.0163,0.000768) (  0.0106, 0.00714) ( 0.00733,  0.0161)
< Backward Transform >
ierr =      0
c[ix][iy][1]
(       4,  0.0167) (       3,  0.0333) (     2.4,    0.05) (       2,  0.0667)
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,  0.0667) (    1.71,   0.133) (     1.5,     0.2) (    1.33,   0.267)
(    1.71,  0.0833) (     1.5,   0.167) (    1.33,    0.25) (     1.2,   0.333)
c[ix][iy][2]
(       3,  0.0333) (     2.4,  0.0667) (       2,     0.1) (    1.71,   0.133)
(     2.4,  0.0667) (       2,   0.133) (    1.71,     0.2) (     1.5,   0.267)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,   0.133) (     1.5,   0.267) (    1.33,     0.4) (     1.2,   0.533)
(     1.5,   0.167) (    1.33,   0.333) (     1.2,     0.5) (    1.09,   0.667)
c[ix][iy][3]
(     2.4,    0.05) (       2,     0.1) (    1.71,    0.15) (     1.5,     0.2)
(       2,     0.1) (    1.71,     0.2) (     1.5,     0.3) (    1.33,     0.4)
(    1.71,    0.15) (     1.5,     0.3) (    1.33,    0.45) (     1.2,     0.6)
(     1.5,     0.2) (    1.33,     0.4) (     1.2,     0.6) (    1.09,     0.8)
(    1.33,    0.25) (     1.2,     0.5) (    1.09,    0.75) (       1,       1)
```

6.10 3 次元実フーリエ変換

6.10.1 [非推奨]ASL_qfr3fb, ASL_pfr3fb
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_qfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

単精度関数:

ierr = ASL_pfr3fb (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2, nz≥2

(b) nx が奇数の時, nx+1≤lx, ny≤ly, nz≤lz
nx が偶数の時, nx+2≤lx, ny≤ly, nz≤lz

(c) lx は偶数である.

(d) isw=1 または isw=−1

(e) nt ≥ 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.	
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.	

(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 $=300(=2^2\times3\times5^2)$ 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_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.

$$\Re\{c_{j_x,j_y,j_z}\} \quad\leftrightarrow\quad \mathrm{r}[2*j_x+\mathrm{lx}*(j_y+\mathrm{ly}*j_z)]$$
$$\Im\{c_{j_x,j_y,j_z}\} \quad\leftrightarrow\quad \mathrm{r}[2*j_x+1+\mathrm{lx}*(j_y+\mathrm{ly}*j_z)]$$

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.

$$c^*_{n_x-j_x,n_y-j_y,n_z-j_z}=c_{j_x,j_y,j_z}$$
$$c^*_{n_x-j_x,j_y,j_z}=c_{j_x,n_y-j_y,n_z-j_z}$$
$$c^*_{n_x-j_x,n_y-j_y,j_z}=c_{j_x,j_y,n_z-j_z}$$

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 6.10.2 $\left\{\begin{matrix}\text{ASL\_qfr3bf}\\ \text{ASL\_pfr3bf}\end{matrix}\right\}$ 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 6.10.2 $\left\{\begin{matrix}\text{ASL\_qfr3bf}\\ \text{ASL\_pfr3bf}\end{matrix}\right\}$ 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) **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**

6.10.2 (7) 使用例参照.

6.10.2 [非推奨]ASL_qfr3bf, ASL_pfr3bf 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_qfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);
```

単精度関数:

```
ierr = ASL_pfr3bf  (nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);
```

(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	nt	I	1	入 力	生成するタスク数
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≥2, ny≥2, nz≥2

(b) nx が奇数の時, nx+1≤lx, ny≤ly, nz≤lz
nx が偶数の時, nx+2≤lx, ny≤ly, nz≤lz

(c) lx は偶数である.

(d) isw=1 または isw=−1

(e) nt ≥ 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.	

戻り値	意　　味	処　理　内　容
3030	制限条件 (d) を満足しなかった.	処理を打ち切る.
3050	制限条件 (e) 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
6.10.1 $\left\{\begin{array}{l}\text{ASL_qfr3fb}\\ \text{ASL_pfr3fb}\end{array}\right\}$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_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

$$\begin{aligned}&\hat{r}_{k_x,k_y,k_z}=n_xn_yn_zr_{k_x,k_y,k_z}\\ &\quad(k_x=0,\cdots,n_x-1;\ k_y=0,\cdots,n_y-1;\ k_z=0,\cdots,n_z-1)\end{aligned}$$

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) 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) **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), nt=2

(c) 主プログラム

```
/*      C Interface example for ASL_qfr3fb , ASL_qfr3bf */

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

int main()
{
    int nx =  6;    int ny =  4;    int nz =  3;
    int lx = 10;    int ly =  5;    int lz =  3;
    double *r;
    int isw;
    int ifax[60];
    double *trigs;
    double *wk;
    int nt = 2;
    int ierr;
    int i,j,k;

    printf( "    *** ASL_qfr3fb , ASL_qfr3bf ***\n" );
    printf( "\n     ** Input **\n" );

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

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

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

    printf( "\tnx = %6d\n", nx );
    printf( "\tny = %6d\n", ny );
    printf( "\tnz = %6d\n", nz );
    printf( "\tnt = %6d\n", nt );

    for( k=1 ; k<=nz ; k++ )
    {
        for( j=1 ; j<=ny ; j++ )
        {
            for( i=1 ; i<=nx ; i++ )
            {
```

```
            r[(i-1)+lx*(j-1)+lx*ly*(k-1)]=(double)(i*j*k)/(double)(nx*ny*nz) ;
        }
    }
}

printf( "\tr[ix][iy][1]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j        ] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][2]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][3]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
    }
    printf( "\n" );
}


isw = 1;
ierr = ASL_qfr3fb(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

for( i=0 ; i<lx*ly*lz ; i++)
{
    r[i] /= (double)(nx*ny*nz);
}

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

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

printf( "\tr[ix][iy][1]\n" );
for( i=0 ; i<nx+2 ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j        ] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][2]\n" );
for( i=0 ; i<nx+2 ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
    }
    printf( "\n" );
}

printf( "\tr[ix][iy][3]\n" );
for( i=0 ; i<nx+2 ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
    }
    printf( "\n" );
}

isw = -1;
ierr = ASL_qfr3bf(nx, ny, nz, r, lx, ly, lz, isw, ifax, trigs, wk, nt);

printf( "\t< Backward Transform >\n" );
printf( "\tierr = %6d\n", ierr );

printf( "\tr[ix][iy][1]\n" );
for( i=0 ; i<nx ; i++ )
{
    for( j=0 ; j<ny ; j++ )
    {
        printf( "\t%8.3g", r[i+lx*j        ] );
    }
    printf( "\n" );
}
```

```
    printf( "\tr[ix][iy][2]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*1] );
        }
        printf( "\n" );
    }

    printf( "\tr[ix][iy][3]\n" );
    for( i=0 ; i<nx ; i++ )
    {
        for( j=0 ; j<ny ; j++ )
        {
            printf( "\t%8.3g", r[i+lx*j+lx*ly*2] );
        }
        printf( "\n" );
    }


    free( r );
    free( trigs );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfr3fb , ASL_qfr3bf ***

 ** Input **
nx =      6
ny =      4
nz =      3
nt =      2
r[ix][iy][1]
  0.0139      0.0278      0.0417      0.0556
  0.0278      0.0556      0.0833       0.111
  0.0417      0.0833       0.125       0.167
  0.0556       0.111       0.167       0.222
  0.0694       0.139       0.208       0.278
  0.0833       0.167        0.25       0.333
r[ix][iy][2]
  0.0278      0.0556      0.0833       0.111
  0.0556       0.111       0.167       0.222
  0.0833       0.167        0.25       0.333
   0.111       0.222       0.333       0.444
   0.139       0.278       0.417       0.556
   0.167       0.333         0.5       0.667
r[ix][iy][3]
  0.0417      0.0833       0.125       0.167
  0.0833       0.167        0.25       0.333
   0.125        0.25       0.375         0.5
   0.167       0.333         0.5       0.667
   0.208       0.417       0.625       0.833
    0.25         0.5        0.75           1

 ** Output **
< Forward Transform >
ierr =      0
r[ix][iy][1]
   0.243     -0.0486     -0.0486     -0.0486
       0      0.0486           0     -0.0486
 -0.0347    -0.00508     0.00694       0.019
  0.0601      -0.019      -0.012    -0.00508
 -0.0347     0.00294     0.00694       0.011
    0.02      -0.011    -0.00401     0.00294
 -0.0347     0.00694     0.00694     0.00694
       0    -0.00694           0     0.00694
r[ix][iy][2]
 -0.0608     0.00514      0.0122      0.0192
  0.0351     -0.0192    -0.00702     0.00514
4.63e-18     0.00401    -1.54e-18   -0.00401
   -0.02     0.00401     0.00401     0.00401
 0.00579    0.000847    -0.00116    -0.00316
   -0.01     0.00316       0.002    0.000847
 0.00868    -0.000734   -0.00174    -0.00274
-0.00501     0.00274       0.001    -0.000734
r[ix][iy][3]
 -0.0608      0.0192      0.0122     0.00514
 -0.0351    -0.00514     0.00702      0.0192
  0.0174    -0.00147    -0.00347    -0.00548
   -0.01     0.00548       0.002    -0.00147
  0.0116    -0.00231    -0.00231    -0.00231
       0     0.00231           0    -0.00231
 0.00868    -0.00274    -0.00174    -0.000734
 0.00501    0.000734      -0.001    -0.00274
< Backward Transform >
ierr =      0
r[ix][iy][1]
  0.0139      0.0278      0.0417      0.0556
  0.0278      0.0556      0.0833       0.111
  0.0417      0.0833       0.125       0.167
  0.0556       0.111       0.167       0.222
```

```
  0.0694       0.139       0.208       0.278
  0.0833       0.167        0.25       0.333
r[ix][iy][2]
  0.0278      0.0556      0.0833       0.111
  0.0556       0.111       0.167       0.222
  0.0833       0.167        0.25       0.333
   0.111       0.222       0.333       0.444
   0.139       0.278       0.417       0.556
   0.167       0.333         0.5       0.667
r[ix][iy][3]
  0.0417      0.0833       0.125       0.167
  0.0833       0.167        0.25       0.333
   0.125        0.25       0.375         0.5
   0.167       0.333         0.5       0.667
   0.208       0.417       0.625       0.833
    0.25         0.5        0.75           1
```

6.11 畳み込み

6.11.1 ASL_qfcn2d, ASL_pfcn2d
2 次元畳み込み

(1) **機 能**

任意の整数 L_x, L_y に対して

$$\begin{aligned} f(i_x, i_y) &= f(i_x + L_x m_x, i_y + L_y m_y), \\ g(j_x, j_y) &= g(j_x + L_x m_x, j_y + L_y m_y), \\ & (i_x, j_x = 0, \cdots, m_x - 1;\ i_y, j_y = 0, \cdots, m_y - 1) \end{aligned}$$

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.

$$\begin{aligned} p(k_x, k_y) &= \sum_{i_x=0}^{m_x-1} \sum_{i_y=0}^{m_y-1} f(i_x, i_y) g(k_x - i_x, k_y - i_y) \\ &= \sum_{j_x=0}^{m_x-1} \sum_{j_y=0}^{m_y-1} g(j_x, j_y) f(k_x - j_x, k_y - j_y) \\ & (k_x = 0, \cdots, m_x - 1;\ k_y = 0, \cdots, m_y - 1) \end{aligned}$$

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_qfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

単精度関数:

ierr = ASL_pfcn2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

(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	nt	I	1	入 力	生成するタスク数
17	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 $\in \{0, 1, 2\}$

(b) nx1 > 1 かつ ny1 > 1

(c) nx2 > 1 かつ ny2 > 1

(d) mx $\geq$ MAX(nx1, nx2) かつ my $\geq$ MAX(ny1, ny2)

(e) lx1 $\geq$ nx1 かつ ly1 $\geq$ ny1 (isw=0 のとき)
lx1 $\geq$ mx $+ 1$ かつ ly1 $\geq$ my (isw ≥ 1 で mx が奇数のとき)
lx1 $\geq$ mx $+ 2$ かつ ly1 $\geq$ my (isw ≥ 1 で mx が偶数のとき)

(f) lx2 $\geq$ mx かつ ly2 $\geq$ my (isw=0 のとき)
lx2 $\geq$ mx $+ 1$ かつ ly2 $\geq$ my (isw ≥ 1 で mx が奇数のとき)
lx2 $\geq$ mx $+ 2$ かつ ly2 $\geq$ my (isw ≥ 1 で mx が偶数のとき)

(g) nt ≥ 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	mx $<$ nx1+nx2-1 または my $<$ ny1+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.	
3060	制限条件 (g) 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.

$$f(i_x, i_y) \quad \leftrightarrow \quad \mathrm{r1}[i_x + \mathrm{lx1} * i_y]$$
$$g(j_x, j_y) \quad \leftrightarrow \quad \mathrm{r2}[j_x + \mathrm{lx2} * j_y]$$

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.

$$p(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

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 $P(j_x, j_y)$:

$$P(j_x, j_y) \quad = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} p(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})}$$
$$(j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor)$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\begin{aligned} \Re\{P(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + \mathrm{lx2} * j_y] \\ \Im\{P(j_x, j_y)\} &\leftrightarrow \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * j_y] \end{aligned}$$

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

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.

(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

$r1[i_x + lx1 * i_y] = f(i_x\Delta, i_y\Delta)$ $(i_x = 0,1,\cdots,nx1-1;\ i_y = 0,1,\cdots,ny1-1)$

$r2[j_x + lx2 * j_y] = g(j_x\Delta, j_y\Delta)$ $(j_x = 0,1,\cdots,nx2-1;\ j_y = 0,1,\cdots,ny2-1)$

ただし, Δ = 0.5

nx1, ny1, nx2, ny2, mx, my, isw

(c) 主プログラム

```
/*      C interface example for ASL_qfcn2d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nx2;
    int ny2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    double *r2;
    int lx2;
    int ly2;
    int mx;
    int my;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
    int nwk;
    int ierr;
    int i,j;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0;
    double xg=2.0,yg=2.0;


    printf( "    *** ASL_qfcn2d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
    ny1=(int) yf/dt;
    nx2=(int) xg/dt;
    ny2=(int) yg/dt;
    mx=my=m0;
    lx1=lx2=m0+2;
    ly1=ly2=m0;
    nwk=mx+2*my+lx2*my;

    r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1) ));
```

```
    if( r1 == NULL )
    {
        printf( "no enough memory for array r1\n" );
        return -1;
    }
    r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2) ));
    if( r2 == NULL )
    {
        printf( "no enough memory for array r2\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\t isw = %6d\n\t (nx1, ny1) = (%3d,%3d)\n",
        isw, nx1, ny1);
    printf( "\t (nx2, ny2) = (%3d,%3d)\n",
        nx2, ny2);
    printf( "\t (mx , my ) = (%3d,%3d)\n\n",
        mx, my);

    for( j=0 ; j<ny1 ; j++ )
        for( i=0 ; i<nx1 ; i++ )
        {
            t=i*dt;
            r1[i+lx1*j]=t;
        }
    for( j=0 ; j<ny2 ; j++ )
        for( i=0 ; i<nx2 ; i++ )
        {
            t=i*dt;
            r2[i+lx2*j]=xg-t;
        }
    printf( "\tData r1[i+%3d*j]\n", lx1 );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t---------------------------------------\n" );
    for( i=0 ; i<nx1 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny1 ; j++ )
            printf( "%8.3g", r1[i+lx1*j] );
        printf( "\n");
    }
    printf( "\n");
    printf( "\tData r2[i+%3d*j]\n", lx2 );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t---------------------------------------\n" );
    for( i=0 ; i<nx2 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny2 ; j++ )
            printf( "%8.3g", r2[i+lx2*j] );
        printf( "\n");
    }


    ierr = ASL_qfcn2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
         r2, lx2, ly2, mx, my, isw, iwk, wk, nt);


    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tConvolution r2[i+%3d*j]\n", lx2 );
    printf( "\ti/j   0      1      2      3      4" );
    printf( "      5      6      7\n" );
    printf( "\t---------------------------------------");
    printf( "--------------------\n" );
    for( i=0 ; i<mx ; i++ )
    {
        printf( "\t%2d", i );
        for( j=0 ; j<my ; j++ )
            printf( "%7.2lf", r2[i+lx2*j] );
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfcn2d ***

 ** Input **

 isw =      1
 (nx1, ny1) = (  4,  4)
 (nx2, ny2) = (  4,  4)
 (mx , my ) = (  8,  8)

Data r1[i+ 10*j]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*j]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

 ** Output **

ierr =      0
Convolution r2[i+ 10*j]
i/j   0      1      2      3      4      5      6      7
-----------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
 1   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
 2   2.75   5.50   8.25  11.00   8.25   5.50   2.75  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   3.50   7.00  10.50  14.00  10.50   7.00   3.50  -0.00
 5   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 6   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 7  -0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00  -0.00
```

6.11.2 ASL_qfcn3d, ASL_pfcn3d
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,

$$\begin{aligned} m_x &= \min(n_x^{(f)} + n_x^{(g)} - 1, M_x), \\ m_y &= \min(n_y^{(f)} + n_y^{(g)} - 1, M_y), \\ m_z &= \min(n_z^{(f)} + n_z^{(g)} - 1, M_z) \end{aligned}$$

であり, M_x, M_y, M_z はそれぞれ

$$\begin{aligned} M_x &\geq \max(n_x^{(f)}, n_x^{(g)}), \\ M_y &\geq \max(n_y^{(f)}, n_y^{(g)}), \\ M_z &\geq \max(n_z^{(f)}, n_z^{(g)}) \end{aligned}$$

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_qfcn3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

単精度関数:

ierr = ASL_pfcn3d (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz, isw, iwk, wk, nt);

(3) 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)
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)

項番	引数と戻り値	型	大きさ	入出力	内　　容
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)$\times$(ny2+1)$\times$nz2 (isw= 0, nx2:偶数, ny2:偶数のとき) nx2 $\times$ (ny2 + 1) $\times$ nz2 (isw= 0, nx2:奇数, ny2:偶数のとき) (nx2 + 1) $\times$ ny2 $\times$ nz2 (isw= 0, nx2:偶数, ny2:奇数のとき) nx2 $\times$ ny2 $\times$ nz2 (isw= 0, nx2:奇数, ny2:奇数のとき) mx + 2 $\times$ (my + mz) + max(lx1 $\times$ ly1 $\times$ lz1, lx2 $\times$ ly2 $\times$ lz2) (isw $\geq$ 1 のとき)
21	nt	I	1	入 力	生成するタスク数
22	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 $\in \{0, 1, 2\}$

(b) nx1 $>$ 1 かつ ny1 $>$ 1 かつ nz1 $>$ 1

(c) nx2 $>$ 1 かつ ny2 $>$ 1 かつ nz2 $>$ 1

(d) mx $\geq$ MAX(nx1, nx2) かつ my $\geq$ MAX(ny1, ny2) かつ mz $\geq$ MAX(nz1, nz2)

(e) lx1 $\geq$ nx1 かつ ly1 $\geq$ ny1 かつ lz1 $\geq$ nz1 (isw=0 のとき)
lx1 $\geq$ mx + 1 かつ lx1 は偶数かつ ly1 $\geq$ my かつ lz1 $\geq$ mz (isw $\geq$ 1 で mx が奇数のとき)
lx1 $\geq$ mx + 2 かつ lx1 は偶数かつ ly1 $\geq$ my かつ lz1 $\geq$ mz (isw $\geq$ 1 で mx が偶数のとき)

(f) lx2 $\geq$ mx かつ ly2 $\geq$ ny2 かつ lz2 $\geq$ nz2 (isw=0 のとき)
lx2 $\geq$ mx + 1 かつ lx2 は偶数かつ ly2 $\geq$ my かつ lz2 $\geq$ mz (isw $\geq$ 1 で mx が奇数のとき)
lx2 $\geq$ mx + 2 かつ lx2 は偶数かつ ly2 $\geq$ my かつ lz2 $\geq$ mz (isw $\geq$ 1 で mx が偶数のとき)

(g) nt $\geq$ 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	mx < nx1+nx2−1 または 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.	
3060	制限条件 (g) 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 $P(j_x, j_y, j_z)$:

$$\begin{aligned} P(j_x, j_y, j_z) &= \frac{1}{M_x M_y M_z} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} \sum_{k_z=0}^{M_z-1} p(k_x, k_y, k_z) e^{-2\pi\sqrt{-1}\left(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y} + \frac{j_z k_z}{M_z}\right)} \\ &\quad (j_x = 0, \cdots, \lfloor \tfrac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \tfrac{M_y}{2} \rfloor; j_z = 0, \cdots, \lfloor \tfrac{M_z}{2} \rfloor) \end{aligned}$$

($\lfloor x \rfloor$ 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) 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. mx > nx1 + nx2 − 1 または my > ny1 + ny2 − 1 または mz > nz1 + 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, mx = nx1 + nx2 − 1, my = ny1 + ny2 − 1, mz = nz1 + nz2 − 1 とするのがよい. 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, mx = 289(=17^2) とした方が良いが, isw $\geq$ 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 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,
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 $\geq$ 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, nwk = mx + 2 × (my + mz) + lx1 × my × mz である.

(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

$\mathrm{r1}[\mathrm{i_x} + \mathrm{lx1} * (\mathrm{i_y} + \mathrm{ly1} * \mathrm{i_z})] = \mathrm{f}(\mathrm{i_x}\Delta, \mathrm{i_y}\Delta, \mathrm{i_z}\Delta)$ $(\mathrm{i_x} = 0, 1, \cdots, \mathrm{nx1} - 1;\ \mathrm{i_y} = 0, 1, \cdots, \mathrm{ny1} - 1;\ \mathrm{i_z} = 0, 1, \cdots, \mathrm{nz1} - 1)$

$\mathrm{r2}[\mathrm{j_x} + \mathrm{lx2} * (\mathrm{j_y} + \mathrm{ly2} * \mathrm{j_z})] = \mathrm{g}(\mathrm{j_x}\Delta, \mathrm{j_y}\Delta, \mathrm{j_z}\Delta)$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx2} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny2} - 1;\ \mathrm{j_z} = 0, 1, \cdots, \mathrm{nz2} - 1)$

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_qfcn3d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nz1;
    int nx2;
    int ny2;
    int nz2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    int lz1;
    double *r2;
    int lx2;
    int ly2;
    int lz2;
    int mx;
    int my;
    int mz;
    int isw;
    int *iwk;
    int niwk=60;
    double *wk;
    int nwk;
    int ierr;
    int i,j,k;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0,zf=2.0;
    double xg=2.0,yg=2.0,zg=2.0;


    printf( "    *** ASL_qfcn3d ***\n" );
    printf( "\n     ** Input **\n\n" );

    isw=1;
    nx1=(int) xf/dt;
    ny1=(int) yf/dt;
    nz1=(int) zf/dt;
    nx2=(int) xg/dt;
    ny2=(int) yg/dt;
    nz2=(int) zg/dt;
    mx=my=mz=m0;
    lx1=lx2=(m0+2)/2*2;
    ly1=ly2=my;
    lz1=lz2=mz;
    nwk=mx+2*(my+mz)+lx2*my*mz;

    r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1*lz1) ));
    if( r1 == NULL )
    {
```

```
    printf( "no enough memory for array r1\n" );
    return -1;
}
r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2*lz2) ));
if( r2 == NULL )
{
    printf( "no enough memory for array r2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw = %6d\n\t (nx1, ny1, nz1) = (%3d,%3d,%3d)\n",
    isw, nx1, ny1, nz1);
printf( "\t (nx2, ny2, nz2) = (%3d,%3d,%3d)\n",
    nx2, ny2, nz2);
printf( "\t (mx , my , mz ) = (%3d,%3d,%3d)\n\n",
    mx, my, mz);

for( k=0 ; k<nz1 ; k++ )
    for( j=0 ; j<ny1 ; j++ )
        for( i=0 ; i<nx1 ; i++ )
        {
            t=i*dt;
            r1[i+lx1*(j+ly1*k)]=t;
        }
for( k=0 ; k<nz2 ; k++ )
    for( j=0 ; j<ny2 ; j++ )
        for( i=0 ; i<nx2 ; i++ )
        {
            t=i*dt;
            r2[i+lx2*(j+ly2*k)]=xg-t;
        }
for( k=0 ; k<nz1 ; k++ )
{
    printf( "\tData r1[i+%3d*(j+%3d*%3d)]\n", lx1, ly1, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t--------------------------------------\n" );
    for( i=0 ; i<nx1 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny1 ; j++ )
            printf( "%8.3g", r1[i+lx1*(j+ly1*k)] );
        printf( "\n");
    }
    printf( "\n");
}
for( k=0 ; k<nz2 ; k++ )
{
    printf( "\tData r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t--------------------------------------\n" );
    for( i=0 ; i<nx2 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny2 ; j++ )
            printf( "%8.3g", r2[i+lx2*(j+ly2*k)] );
        printf( "\n");
    }
    printf( "\n");
}


ierr = ASL_qfcn3d(nx1, ny1, nz1, nx2, ny2, nz2,
    r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
    mx, my, mz, isw, iwk, wk, nt);


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

for( k=0 ; k<mz ; k++ )
{
    printf( "\tConvolution r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
    printf( "\ti/j   0      1      2      3      4" );
    printf( "      5      6      7\n" );
    printf( "\t--------------------------------------");
    printf( "--------------------\n" );
    for( i=0 ; i<mx ; i++ )
    {
        printf( "\t%2d", i );
        for( j=0 ; j<my ; j++ )
            printf( "%7.2lf", r2[i+lx2*(j+ly2*k)] );
```

```
            printf( "\n");
        }
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfcn3d ***

 ** Input **

 isw =      1
 (nx1, ny1, nz1) = (  4,  4,  4)
 (nx2, ny2, nz2) = (  4,  4,  4)
 (mx , my , mz ) = (  8,  8,  8)

Data r1[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5


 ** Output **

ierr =      0
Convolution r2[i+ 10*(j+  8*  0)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00   0.00
 1   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
 2   2.75   5.50   8.25  11.00   8.25   5.50   2.75   0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00   0.00
```

```
 4   3.50   7.00  10.50  14.00  10.50   7.00   3.50   0.00
 5   2.00   4.00   6.00   8.00   6.00   4.00   2.00   0.00
 6   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 7   0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00   0.00

Convolution r2[i+ 10*(j+  8*  1)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
 1   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 2   5.50  11.00  16.50  22.00  16.50  11.00   5.50  -0.00
 3  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
 4   7.00  14.00  21.00  28.00  21.00  14.00   7.00  -0.00
 5   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 6   1.50   3.00   4.50   6.00   4.50   3.00   1.50  -0.00
 7   0.00  -0.00  -0.00   0.00  -0.00  -0.00  -0.00  -0.00

Convolution r2[i+ 10*(j+  8*  2)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00   0.00  -0.00
 1   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 2   8.25  16.50  24.75  33.00  24.75  16.50   8.25  -0.00
 3  15.00  30.00  45.00  60.00  45.00  30.00  15.00  -0.00
 4  10.50  21.00  31.50  42.00  31.50  21.00  10.50  -0.00
 5   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 6   2.25   4.50   6.75   9.00   6.75   4.50   2.25  -0.00
 7   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00  -0.00

Convolution r2[i+ 10*(j+  8*  3)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00  -0.00   0.00
 1   4.00   8.00  12.00  16.00  12.00   8.00   4.00   0.00
 2  11.00  22.00  33.00  44.00  33.00  22.00  11.00  -0.00
 3  20.00  40.00  60.00  80.00  60.00  40.00  20.00  -0.00
 4  14.00  28.00  42.00  56.00  42.00  28.00  14.00  -0.00
 5   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
 6   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 7  -0.00  -0.00   0.00  -0.00  -0.00   0.00  -0.00   0.00

Convolution r2[i+ 10*(j+  8*  4)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0   0.00  -0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
 1   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 2   8.25  16.50  24.75  33.00  24.75  16.50   8.25  -0.00
 3  15.00  30.00  45.00  60.00  45.00  30.00  15.00  -0.00
 4  10.50  21.00  31.50  42.00  31.50  21.00  10.50   0.00
 5   6.00  12.00  18.00  24.00  18.00  12.00   6.00   0.00
 6   2.25   4.50   6.75   9.00   6.75   4.50   2.25  -0.00
 7  -0.00  -0.00   0.00   0.00   0.00  -0.00  -0.00  -0.00

Convolution r2[i+ 10*(j+  8*  5)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0   0.00   0.00   0.00   0.00   0.00   0.00  -0.00   0.00
 1   2.00   4.00   6.00   8.00   6.00   4.00   2.00   0.00
 2   5.50  11.00  16.50  22.00  16.50  11.00   5.50  -0.00
 3  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
 4   7.00  14.00  21.00  28.00  21.00  14.00   7.00  -0.00
 5   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 6   1.50   3.00   4.50   6.00   4.50   3.00   1.50  -0.00
 7   0.00  -0.00   0.00  -0.00  -0.00  -0.00  -0.00   0.00

Convolution r2[i+ 10*(j+  8*  6)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0  -0.00   0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 1   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
 2   2.75   5.50   8.25  11.00   8.25   5.50   2.75  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   3.50   7.00  10.50  14.00  10.50   7.00   3.50  -0.00
 5   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 6   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 7  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00

Convolution r2[i+ 10*(j+  8*  7)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------------
 0  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 1  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 2  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 3  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 4  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 5  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 6  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 7  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
```

6.12 相関

6.12.1 ASL_qfcr2d, ASL_pfcr2d
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_qfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

単精度関数:

ierr = ASL_pfcr2d (nx1, ny1, nx2, ny2, r1, lx1, ly1, r2, lx2, ly2, mx, my, isw, iwk, wk, nt);

(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)

項番	引数と戻り値	型	大きさ	入出力	内　　容
16	nt	I	1	入 力	生成するタスク数
17	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 $\in \{0, 1, 2\}$

(b) nx1 > 1 かつ ny1 > 1

(c) nx2 > 1 かつ ny2 > 1

(d) mx $\geq$ MAX(nx1, nx2) かつ my $\geq$ MAX(ny1, ny2)

(e) lx1 $\geq$ nx1 かつ ly1 $\geq$ ny1 (isw=0 のとき)
lx1 $\geq$ mx $+ 1$ かつ ly1 $\geq$ my (isw ≥ 1 で mx が奇数のとき)
lx1 $\geq$ mx $+ 2$ かつ ly1 $\geq$ my (isw ≥ 1 で mx が偶数のとき)

(f) lx2 $\geq$ mx かつ ly2 $\geq$ my (isw=0 のとき)
lx2 $\geq$ mx $+ 1$ かつ ly2 $\geq$ my (isw ≥ 1 で mx が奇数のとき)
lx2 $\geq$ mx $+ 2$ かつ ly2 $\geq$ my (isw ≥ 1 で mx が偶数のとき)

(g) nt ≥ 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	mx $<$ nx1+nx2-1 または my $<$ ny1+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.	
3060	制限条件 (g) 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.

$$f(i_x, i_y) \quad \leftrightarrow \quad \mathrm{r1}[i_x + \mathrm{lx1} * i_y]$$
$$g(j_x, j_y) \quad \leftrightarrow \quad \mathrm{r2}[j_x + \mathrm{lx2} * j_y]$$

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.

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

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 $Q(j_x, j_y)$:

$$Q(j_x, j_y) = \frac{1}{M_x M_y} \sum_{k_x=0}^{M_x-1} \sum_{k_y=0}^{M_y-1} q(k_x, k_y) e^{-2\pi\sqrt{-1}(\frac{j_x k_x}{M_x} + \frac{j_y k_y}{M_y})}$$
$$(j_x = 0, \cdots, \lfloor \frac{M_x}{2} \rfloor;\ j_y = 0, \cdots, \lfloor \frac{M_y}{2} \rfloor)$$

($\lfloor x \rfloor$ は x を超えない最大の整数) を求める場合には,

$$\Re\{Q(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2}[2 * j_x + \mathrm{lx2} * j_y]$$
$$\Im\{Q(j_x, j_y)\} \quad \leftrightarrow \quad \mathrm{r2}[2 * j_x + 1 + \mathrm{lx2} * j_y]$$

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

$$Q(M_x - j_x, M_y - j_y)^* = Q(j_x, j_y)$$
$$Q(M_x - j_x, j_y)^* = Q(j_x, M_y - j_y)$$

(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

$$\tilde{q}(k_x, k_y) \quad \leftrightarrow \quad \mathrm{r2}[k_x + \mathrm{lx2} * k_y]$$

$k_x = \mathrm{nx1} + \mathrm{nx2} - 1, \cdots, \mathrm{mx} - 1$; $k_y = 0, \cdots, \mathrm{my} - 1$ 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.

(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_qfcr2d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nx2;
    int ny2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    double *r2;
    int lx2;
    int ly2;
    int mx;
    int my;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
```

```
int nwk;
int ierr;
int i,j;
int nt = 2;
double t;
double dt=0.5;
double xf=2.0,yf=2.0;
double xg=2.0,yg=2.0;


printf( "    *** ASL_qfcr2d ***\n" );
printf( "\n     ** Input **\n\n" );

isw=1;
nx1=(int) xf/dt;
ny1=(int) yf/dt;
nx2=(int) xg/dt;
ny2=(int) yg/dt;
mx=my=m0;
lx1=lx2=m0+2;
ly1=ly2=m0;
nwk=mx+2*my+lx2*my;

r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1) ));
if( r1 == NULL )
{
    printf( "no enough memory for array r1\n" );
    return -1;
}
r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2) ));
if( r2 == NULL )
{
    printf( "no enough memory for array r2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw = %6d\n\t (nx1, ny1) = (%3d,%3d)\n",
    isw, nx1, ny1);
printf( "\t (nx2, ny2) = (%3d,%3d)\n",
    nx2, ny2);
printf( "\t (mx , my ) = (%3d,%3d)\n\n",
    mx, my);

for( j=0 ; j<ny1 ; j++ )
    for( i=0 ; i<nx1 ; i++ )
    {
        t=i*dt;
        r1[i+lx1*j]=t;
    }
for( j=0 ; j<ny2 ; j++ )
    for( i=0 ; i<nx2 ; i++ )
    {
        t=i*dt;
        r2[i+lx2*j]=xg-t;
    }
printf( "\tData r1[i+%3d*j]\n", lx1 );
printf( "\ti/j       0       1       2       3\n" );
printf( "\t-------------------------------------\n" );
for( i=0 ; i<nx1 ; i++ )
{
    printf( "\t%3d", i );
    for( j=0 ; j<ny1 ; j++ )
        printf( "%8.3g", r1[i+lx1*j] );
    printf( "\n");
}
printf( "\n");
printf( "\tData r2[i+%3d*j]\n", lx2 );
printf( "\ti/j       0       1       2       3\n" );
printf( "\t-------------------------------------\n" );
for( i=0 ; i<nx2 ; i++ )
{
    printf( "\t%3d", i );
    for( j=0 ; j<ny2 ; j++ )
        printf( "%8.3g", r2[i+lx2*j] );
    printf( "\n");
}


ierr = ASL_qfcr2d(nx1, ny1, nx2, ny2, r1, lx1, ly1,
     r2, lx2, ly2, mx, my, isw, iwk, wk, nt);
```

```
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\tCorrelation r2[i+%3d*j]\n", lx2 );
    printf( "\ti/j   0      1      2      3      4" );
    printf( "      5      6      7\n" );
    printf( "\t-------------------------------------");
    printf( "--------------------\n" );
    for( i=0 ; i<mx ; i++ )
    {
        printf( "\t%2d", i );
        for( j=0 ; j<my ; j++ )
            printf( "%7.2lf", r2[i+lx2*j] );
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfcr2d ***

 ** Input **

 isw =      1
 (nx1, ny1) = (  4,  4)
 (nx2, ny2) = (  4,  4)
 (mx , my ) = (  8,  8)

Data r1[i+ 10*j]
i/j       0       1       2       3
-------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*j]
i/j       0       1       2       3
-------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

 ** Output **

ierr =      0
Correlation r2[i+ 10*j]
i/j   0      1      2      3      4      5      6      7
-----------------------------------------------------------
 0   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 1   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
 2   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 3   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
 4   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
 5   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
 6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
```

6.12.2 ASL_qfcr3d, ASL_pfcr3d
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,

$$\begin{aligned} m_x &= \min(n_x^{(f)}+n_x^{(g)}-1, M_x), \\ m_y &= \min(n_y^{(f)}+n_y^{(g)}-1, M_y), \\ m_z &= \min(n_z^{(f)}+n_z^{(g)}-1, M_z) \end{aligned}$$

であり, M_x, M_y, M_z はそれぞれ

$$\begin{aligned} M_x &\geq \max(n_x^{(f)}, n_x^{(g)}), \\ M_y &\geq \max(n_y^{(f)}, n_y^{(g)}), \\ M_z &\geq \max(n_z^{(f)}, n_z^{(g)}) \end{aligned}$$

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_qfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk, nt);
```

単精度関数:

```
ierr = ASL_pfcr3d  (nx1, ny1, nz1, nx2, ny2, nz2, r1, lx1, ly1, lz1, r2, lx2, ly2, lz2, mx, my, mz,
                    isw, iwk, wk, nt);
```

(3) 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	nt	I	1	入 力	生成するタスク数
22	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 $\in \{0, 1, 2\}$

(b) nx1 $>$ 1 かつ ny1 $>$ 1 かつ nz1 $>$ 1

(c) nx2 $>$ 1 かつ ny2 $>$ 1 かつ nz2 $>$ 1

(d) mx $\geq$ MAX(nx1, nx2) かつ my $\geq$ MAX(ny1, ny2) かつ mz $\geq$ MAX(nz1, nz2)

(e) lx1 $\geq$ nx1 かつ ly1 $\geq$ ny1 かつ lz1 $\geq$ nz1 (isw=0 のとき)
lx1 $\geq$ mx + 1 かつ lx1 は偶数かつ ly1 $\geq$ my かつ lz1 $\geq$ mz (isw $\geq$ 1 で mx が奇数のとき)
lx1 $\geq$ mx + 2 かつ lx1 は偶数かつ ly1 $\geq$ my かつ lz1 $\geq$ mz (isw $\geq$ 1 で mx が偶数のとき)

(f) lx2 $\geq$ mx かつ ly2 $\geq$ ny2 かつ lz2 $\geq$ nz2 (isw=0 のとき)
lx2 $\geq$ mx + 1 かつ lx2 は偶数かつ ly2 $\geq$ my かつ lz2 $\geq$ mz (isw $\geq$ 1 で mx が奇数のとき)
lx2 $\geq$ mx + 2 かつ lx2 は偶数かつ ly2 $\geq$ my かつ lz2 $\geq$ mz (isw $\geq$ 1 で mx が偶数のとき)

(g) nt $\geq$ 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	mx < nx1+nx2−1 または 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.	
3060	制限条件 (g) 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.

(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

$r1[i_x + lx1 * (i_y + ly1 * i_z)] = f(i_x\Delta, i_y\Delta, i_z\Delta)$ $(i_x = 0,1,\cdots,nx1-1;\ i_y = 0,1,\cdots,ny1-1;\ i_z = 0,1,\cdots,nz1-1)$

$r2[j_x + lx2 * (j_y + ly2 * j_z)] = g(j_x\Delta, j_y\Delta, j_z\Delta)$ $(j_x = 0,1,\cdots,nx2-1;\ j_y = 0,1,\cdots,ny2-1;\ j_z = 0,1,\cdots,nz2-1)$

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_qfcr3d */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nx1;
    int ny1;
    int nz1;
    int nx2;
    int ny2;
    int nz2;
    double *r1;
    int m0=8;
    int lx1;
    int ly1;
    int lz1;
    double *r2;
    int lx2;
    int ly2;
    int lz2;
    int mx;
    int my;
    int mz;
    int isw;
    int *iwk;
    int niwk=60;
    double *wk;
    int nwk;
    int ierr;
    int i,j,k;
    int nt = 2;
    double t;
    double dt=0.5;
    double xf=2.0,yf=2.0,zf=2.0;
    double xg=2.0,yg=2.0,zg=2.0;


    printf( "    *** ASL_qfcr3d ***\n" );
    printf( "\n     ** Input **\n\n" );
```

```
isw=1;
nx1=(int) xf/dt;
ny1=(int) yf/dt;
nz1=(int) zf/dt;
nx2=(int) xg/dt;
ny2=(int) yg/dt;
nz2=(int) zg/dt;
mx=my=mz=m0;
lx1=lx2=(m0+2)/2*2;
ly1=ly2=my;
lz1=lz2=mz;
nwk=mx+2*(my+mz)+lx2*my*mz;

r1 = ( double * )malloc((size_t)( sizeof(double) * (lx1*ly1*lz1) ));
if( r1 == NULL )
{
    printf( "no enough memory for array r1\n" );
    return -1;
}
r2 = ( double * )malloc((size_t)( sizeof(double) * (lx2*ly2*lz2) ));
if( r2 == NULL )
{
    printf( "no enough memory for array r2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw = %6d\n\t (nx1, ny1, nz1) = (%3d,%3d,%3d)\n",
    isw, nx1, ny1, nz1);
printf( "\t (nx2, ny2, nz2) = (%3d,%3d,%3d)\n",
    nx2, ny2, nz2);
printf( "\t (mx , my , mz ) = (%3d,%3d,%3d)\n\n",
    mx, my, mz);

for( k=0 ; k<nz1 ; k++ )
    for( j=0 ; j<ny1 ; j++ )
        for( i=0 ; i<nx1 ; i++ )
        {
            t=i*dt;
            r1[i+lx1*(j+ly1*k)]=t;
        }
for( k=0 ; k<nz2 ; k++ )
    for( j=0 ; j<ny2 ; j++ )
        for( i=0 ; i<nx2 ; i++ )
        {
            t=i*dt;
            r2[i+lx2*(j+ly2*k)]=xg-t;
        }
for( k=0 ; k<nz1 ; k++ )
{
    printf( "\tData r1[i+%3d*(j+%3d*%3d)]\n", lx1, ly1, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx1 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny1 ; j++ )
            printf( "%8.3g", r1[i+lx1*(j+ly1*k)] );
        printf( "\n");
    }
    printf( "\n");
}
for( k=0 ; k<nz2 ; k++ )
{
    printf( "\tData r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
    printf( "\ti/j       0       1       2       3\n" );
    printf( "\t------------------------------------\n" );
    for( i=0 ; i<nx2 ; i++ )
    {
        printf( "\t%3d", i );
        for( j=0 ; j<ny2 ; j++ )
            printf( "%8.3g", r2[i+lx2*(j+ly2*k)] );
        printf( "\n");
    }
    printf( "\n");
}


ierr = ASL_qfcr3d(nx1, ny1, nz1, nx2, ny2, nz2,
    r1, lx1, ly1, lz1, r2, lx2, ly2, lz2,
    mx, my, mz, isw, iwk, wk, nt);
```

```
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    for( k=0 ; k<mz ; k++ )
    {
        printf( "\tCorrelation r2[i+%3d*(j+%3d*%3d)]\n", lx2, ly2, k );
        printf( "\ti/j   0      1      2      3      4" );
        printf( "      5      6      7\n" );
        printf( "\t-------------------------------------");
        printf( "--------------------\n" );
        for( i=0 ; i<mx ; i++ )
        {
            printf( "\t%2d", i );
            for( j=0 ; j<my ; j++ )
                printf( "%7.2lf", r2[i+lx2*(j+ly2*k)] );
            printf( "\n");
        }
        printf( "\n");
    }
    free( iwk );
    free( wk );
    free( r2 );
    free( r1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qfcr3d ***

 ** Input **

 isw =      1
 (nx1, ny1, nz1) = (  4,  4,  4)
 (nx2, ny2, nz2) = (  4,  4,  4)
 (mx , my , mz ) = (  8,  8,  8)

Data r1[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-----------------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-----------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-----------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r1[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
-----------------------------------
  0       0       0       0       0
  1     0.5     0.5     0.5     0.5
  2       1       1       1       1
  3     1.5     1.5     1.5     1.5

Data r2[i+ 10*(j+  8*  0)]
i/j       0       1       2       3
-----------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  1)]
i/j       0       1       2       3
-----------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  2)]
i/j       0       1       2       3
-----------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5

Data r2[i+ 10*(j+  8*  3)]
i/j       0       1       2       3
```

```
------------------------------------
  0       2       2       2       2
  1     1.5     1.5     1.5     1.5
  2       1       1       1       1
  3     0.5     0.5     0.5     0.5


 ** Output **

ierr =       0
Correlation r2[i+ 10*(j+  8*  0)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 1   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
 2   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 3   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
 4   1.00   2.00   3.00   4.00   3.00   2.00   1.00  -0.00
 5   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
 6   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00  -0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  1)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 1   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
 2   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 5   0.50   1.00   1.50   2.00   1.50   1.00   0.50  -0.00
 6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  2)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
 1  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
 2  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 3   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
 4   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 5   0.75   1.50   2.25   3.00   2.25   1.50   0.75  -0.00
 6   0.00   0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
 7   0.00  -0.00   0.00  -0.00   0.00   0.00  -0.00   0.00

Correlation r2[i+ 10*(j+  8*  3)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 1  17.00  34.00  51.00  68.00  51.00  34.00  17.00  -0.00
 2  16.00  32.00  48.00  64.00  48.00  32.00  16.00  -0.00
 3  10.00  20.00  30.00  40.00  30.00  20.00  10.00  -0.00
 4   4.00   8.00  12.00  16.00  12.00   8.00   4.00   0.00
 5   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
 6  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00   0.00
 7  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00

Correlation r2[i+ 10*(j+  8*  4)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   9.00  18.00  27.00  36.00  27.00  18.00   9.00  -0.00
 1  12.75  25.50  38.25  51.00  38.25  25.50  12.75  -0.00
 2  12.00  24.00  36.00  48.00  36.00  24.00  12.00  -0.00
 3   7.50  15.00  22.50  30.00  22.50  15.00   7.50  -0.00
 4   3.00   6.00   9.00  12.00   9.00   6.00   3.00  -0.00
 5   0.75   1.50   2.25   3.00   2.25   1.50   0.75   0.00
 6   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00   0.00  -0.00   0.00  -0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  5)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   6.00  12.00  18.00  24.00  18.00  12.00   6.00  -0.00
 1   8.50  17.00  25.50  34.00  25.50  17.00   8.50  -0.00
 2   8.00  16.00  24.00  32.00  24.00  16.00   8.00  -0.00
 3   5.00  10.00  15.00  20.00  15.00  10.00   5.00  -0.00
 4   2.00   4.00   6.00   8.00   6.00   4.00   2.00  -0.00
 5   0.50   1.00   1.50   2.00   1.50   1.00   0.50   0.00
 6   0.00   0.00   0.00  -0.00   0.00   0.00   0.00   0.00
 7   0.00   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  6)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0   3.00   6.00   9.00  12.00   9.00   6.00   3.00   0.00
 1   4.25   8.50  12.75  17.00  12.75   8.50   4.25  -0.00
 2   4.00   8.00  12.00  16.00  12.00   8.00   4.00  -0.00
 3   2.50   5.00   7.50  10.00   7.50   5.00   2.50  -0.00
 4   1.00   2.00   3.00   4.00   3.00   2.00   1.00   0.00
 5   0.25   0.50   0.75   1.00   0.75   0.50   0.25   0.00
 6   0.00  -0.00  -0.00   0.00   0.00   0.00   0.00   0.00
 7   0.00  -0.00  -0.00  -0.00   0.00   0.00   0.00   0.00

Correlation r2[i+ 10*(j+  8*  7)]
i/j   0      1      2      3      4      5      6      7
-------------------------------------------------------------
 0  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00   0.00   0.00
 1  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
```

```
2  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
3  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
4  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00  -0.00
5  -0.00  -0.00   0.00   0.00   0.00   0.00  -0.00  -0.00
6  -0.00  -0.00   0.00   0.00   0.00   0.00   0.00   0.00
7  -0.00   0.00  -0.00   0.00   0.00   0.00   0.00   0.00
```

6.13 パワー・スペクトル解析

6.13.1 ASL_qfps2d, ASL_pfps2d
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_qfps2d (nx, ny, r, lx, ly, isw, iwk, wk, nt);

単精度関数:

ierr = ASL_pfps2d (nx, ny, r, lx, ly, isw, iwk, wk, nt);

(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 × ny + lx × ly
9	nt	I	1	入 力	生成するタスク数
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) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx >1 かつ ny >1

(c) lx $\geq$ nx + 1 かつ ly $\geq$ ny (nx が奇数の時)
lx $\geq$ nx + 2 かつ ly $\geq$ ny (nx が偶数の時)

(d) nt ≥ 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.	
3030	制限条件 (c) を満足しなかった.	
3040	制限条件 (d) を満足しなかった.	
4000	isw=1 の場合にユーザ定義窓関数が $w_{j_x}^{(x)} = 0$ $(j_x = 0, \cdots, n_x - 1)$ 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.	

(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} $(k_x = 0, 1, \cdots, \lfloor \tfrac{n_x}{2} \rfloor;\ k_y = 0, 1, \cdots, n_y - 1)$ 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.

(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

$\mathrm{r[j_x + lx * j_y]} = \mathrm{f(j_x\Delta, j_y\Delta)}$ $(\mathrm{j_x} = 0, 1, \cdots, \mathrm{nx} - 1;\ \mathrm{j_y} = 0, 1, \cdots, \mathrm{ny} - 1)$

ただし, $\Delta = 0.5$

nx, ny, isw

(c) 主プログラム

```
/*      C interface example for ASL_qfps2d */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int n0=8,isw0=4;
    int nx;
    int ny;
    double *r;
    int lx;
    int ly;
    int isw;
    int *iwk;
    int niwk=40;
    double *wk;
    int nwk;
    int ierr;
    int i,j,m,nd2,is;
    int nt = 2;
    double *p,t,tx,ty,dt,dfx,dfy,f0,f1,f2;

    printf( "    *** ASL_qfps2d ***\n" );
    printf( "\n     ** Input **\n\n" );

    nx=n0;
    ny=n0;
    lx=n0+2;
    ly=ny;
    nwk=nx+2*ny+lx*ly;

    r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*(isw0+2)) ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    p = ( double * )malloc((size_t)( sizeof(double) * (isw0+2) ));
    if( p == NULL )
    {
        printf( "no enough memory for array p\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\t isw=0, 2 to %6d\n", isw0+1 );
    printf( "\t nx=%6d\n\t ny=%6d\n\n", nx,ny );
    dt=0.5;
    f0=1.0/(2.0*dt);
    f1=0.62*f0;
    f2=0.14*f0;
    nd2=(int) (nx+1)/2;
    dfx=1.0/(dt*nx);
    dfy=1.0/(dt*ny);
    p[isw0+1]=0.0;
    for( j=0 ; j<ny ; j++ )
    {
```

```
    ty=(double) j*dt;
    for( i=0 ; i<nx ; i++ )
    {
        tx=(double) i*dt;
        t=cos(2.0*M_PI*f1*tx)+cos(2.0*M_PI*f2*ty);
        r[i+lx*(j+ly*(isw0+1))]=t;
        p[isw0+1] += (t*t);
    }
}
p[isw0+1] /= (double) (nx*ny);
printf( "\tTime series data r[i+%3d*j]\n", lx);
printf( "  i/j");
for( j=0 ; j<ny ; j++ )
    printf( "%9d", j);
printf( "\n" );
printf( "  ------------------------------------");
printf( "----------------------------------------\n");
for( i=0 ; i<nx ; i++ )
{
    printf( "%5d", i );
    for( j=0 ; j<ny ; j++ )
        printf( "%9.4lf", r[i+lx*(j+ly*(isw0+1))] );
    printf( "\n" );
}
printf( "\n");
printf( "\tTime domain power =%9.4lf\n", p[isw0+1]);
printf( "\tSignal frequency =(%9.4lf, %9.4lf)\n", f1, f2);
is=0;

for( isw=0 ; isw<=isw0 ; isw++ )
{
    for( j=0 ; j<ny ; j++ )
        for( i=0 ; i<nx ; i++ )
            r[i+lx*(j+ly*isw)]=r[i+lx*(j+ly*(isw0+1))];
    if (isw != 0)
        is=isw+1;


    ierr = ASL_qfps2d(nx, ny, &r[lx*ly*isw], lx, ly, is, iwk, wk, nt);
    p[isw]=0.0;
    if (nx%2==0)
    {
        m=nd2-1;
        for( j=0 ; j<ny ; j++ )
            for( i=1 ; i<m ; i++ )
                p[isw]+=2.0*r[i+lx*(j+ly*isw)];
        for( j=0 ; j<ny ; j++ )
            p[isw]+=r[lx*(j+ly*isw)]+r[m+lx*(j+ly*isw)];

    }
    else
    {
        m=nd2;
        for( j=0 ; j<ny ; j++ )
            for( i=1 ; i<m ; i++ )
                p[isw]+=2.0*r[i+lx*(j+ly*isw)];
        for( j=0 ; j<ny ; j++ )
            p[isw]+=r[lx*(j+ly*isw)];
    }
}


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

isw=0;
printf( "\t(Modified) periodogram (Raw)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "----------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=1;
printf( "\t(Modified) periodogram (Hanning)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
```

```
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=2;
printf( "\t(Modified) periodogram (Bartlett)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=3;
printf( "\t(Modified) periodogram (Welch)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}
printf( "\n");

isw=4;
printf( "\t(Modified) periodogram (Parzen)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
printf( "  x/y-freq");
for( j=(ny+1)/2 ; j<ny ; j++ )
    printf( "%8.2lf", (j-ny)*dfy );
for( j=0 ; j<(ny+1)/2 ; j++ )
    printf( "%8.2lf", j*dfy );
printf( "\n");
printf( "  ------------------------------------");
printf( "------------------------------------\n");
for( i=0 ; i<nd2 ; i++ )
{
    printf( "  %8.2lf", i*dfx );
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.4lf", r[i+lx*(j+ly*isw)] );
    printf( "\n" );
}

free( iwk );
free( p );
free( wk );
free( r );
```

```
    return 0;
}
```

(d) 出力結果

```
  *** ASL_qfps2d ***

   ** Input **

   isw=0, 2 to      5
   nx=     8
   ny=     8

  Time series data r[i+ 10*j]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   2.0000   1.9048   1.6374   1.2487   0.8126   0.4122   0.1237   0.0020
  1   0.6319   0.5367   0.2693  -0.1194  -0.5555  -0.9559  -1.2444  -1.3662
  2   0.2710   0.1759  -0.0915  -0.4803  -0.9163  -1.3168  -1.6053  -1.7270
  3   1.9048   1.8097   1.5423   1.1535   0.7174   0.3170   0.0285  -0.0932
  4   1.0628   0.9676   0.7002   0.3115  -0.1246  -0.5250  -0.8135  -0.9352
  5   0.0489  -0.0462  -0.3136  -0.7024  -1.1384  -1.5388  -1.8274  -1.9491
  6   1.6374   1.5423   1.2748   0.8861   0.4500   0.0496  -0.2389  -0.3606
  7   1.4818   1.3866   1.1192   0.7304   0.2944  -0.1060  -0.3946  -0.5163

  Time domain power =   1.0626
  Signal frequency =(   0.6200,    0.1400)

   ** Output **

  ierr =      0
  (Modified) periodogram (Raw)
  Frequency domain power=   0.9717
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0158  0.0188  0.0350  0.2150  0.0218  0.2150  0.0350  0.0188
    0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

  (Modified) periodogram (Hanning)
  Frequency domain power=   0.5980
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0054  0.0632  0.0105  0.0632  0.0054  0.0001
    0.25  0.0000  0.0000  0.0013  0.0095  0.0056  0.0236  0.0013  0.0000
    0.50  0.0000  0.0000  0.0000  0.0204  0.0814  0.0204  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0205  0.0820  0.0205  0.0000  0.0000

  (Modified) periodogram (Bartlett)
  Frequency domain power=   0.5835
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0820  0.0109  0.0820  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0122  0.0025  0.0178  0.0002  0.0000
    0.50  0.0000  0.0005  0.0000  0.0156  0.0855  0.0156  0.0000  0.0005
    0.75  0.0000  0.0004  0.0000  0.0095  0.0762  0.0191  0.0000  0.0004

  (Modified) periodogram (Welch)
  Frequency domain power=   0.7072
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.1263  0.0124  0.1263  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0140  0.0014  0.0127  0.0000  0.0000
    0.50  0.0002  0.0003  0.0010  0.0195  0.1065  0.0054  0.0009  0.0003
    0.75  0.0002  0.0003  0.0008  0.0064  0.0941  0.0142  0.0009  0.0003

  (Modified) periodogram (Parzen)
  Frequency domain power=   0.4909
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0093  0.0253  0.0070  0.0253  0.0093  0.0002
    0.25  0.0000  0.0001  0.0022  0.0022  0.0127  0.0279  0.0064  0.0001
    0.50  0.0000  0.0000  0.0006  0.0171  0.0558  0.0323  0.0030  0.0000
    0.75  0.0000  0.0000  0.0013  0.0206  0.0485  0.0214  0.0014  0.0000
```

6.13.2 ASL_qfps3d, ASL_pfps3d
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_qfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk, nt);

単精度関数:

ierr = ASL_pfps3d (nx, ny, nz, r, lx, ly, lz, isw, iwk, wk, nt);

(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	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	nt	I	1	入 力	生成するタスク数
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) isw $\in \{0, \pm1, \pm2, \pm3, \pm4, \pm5\}$

(b) nx > 1 かつ ny > 1 かつ nz > 1

(c) lx $\geq$ nx $+ 1$ かつ lx は偶数かつ ly $\geq$ ny かつ lz $\geq$ nz (nx が奇数の時)
lx $\geq$ nx $+ 2$ かつ lx は偶数かつ ly $\geq$ ny かつ lz $\geq$ nz (nx が偶数の時)

(d) nt ≥ 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.	
3030	制限条件 (c) を満足しなかった.	
3040	制限条件 (d) を満足しなかった.	
4000	isw=1 の場合にユーザ定義窓関数が $w_{j_x}^{(x)} = 0$ $(j_x = 0, \cdots, n_x - 1)$ 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(=$2^2\times3\times5^2$) や 320(=$2^6\times5$), 384(=$2^7\times3$) 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.

$$w_j=\begin{cases}\sin^2(\pi v_j) & \text{isw}=\pm2\ \text{(Hanning 窓)}\\ 1-|2v_j-1| & \text{isw}=\pm3\ \text{(Bartlett 窓)}\\ 1-(2v_j-1)^2 & \text{isw}=\pm4\ \text{(Welch 窓)}\\ \left\{\begin{array}{ll}16{v_j}^3 & 0\le v_j<\frac14\\ 1-6v_j(v_j-1)^2 & \frac14\le v_j\le\frac12\\ 1-6v_j(v_{n-j+1}-1)^2 & \frac12\le v_j\le\frac34\\ 16{v_{n-j+1}}^3 & \frac34\le v_j<1\end{array}\right\} & \text{isw}=\pm5\ \text{(Parzen 窓)}\end{cases}$$

ただし, $v_j=\frac{j}{n}$ で, $w^{(x)}_{j_x}$ については $j=j_x, n=n_x$ とし, $w^{(y)}_{j_y}$ については $j=j_y, n=n_y$, $w^{(z)}_{j_z}$ 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 $|x|\le1$ 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.

$$w(x)=\begin{cases}\dfrac{1+\cos\pi x}{2}=\cos^2\dfrac{\pi x}{2} & \text{Hanning 窓}\\ 1-|x| & \text{Bartlett 窓}\\ 1-x^2 & \text{Welch 窓}\\ \left\{\begin{array}{ll}1-6x^2+6|x|^3 & |x|\le\frac12\\ 2(1-|x|)^3 & \frac12\le|x|\le1\end{array}\right\} & \text{Parzen 窓}\end{cases}$$

また, ユーザ独自の窓関数値 $w^{(x)}_{j_x}$, $w^{(y)}_{j_y}$, $w^{(z)}_{j_z}$ を利用したい場合には isw = ±1 として作業配列 wk に

$$\begin{aligned}&\text{wk}[j_x]=w^{(x)}_{j_x}\quad(j_x=0,\cdots,n_x-1),\\ &\text{wk}[n_x+j_y]=w^{(y)}_{j_y}\quad(j_y=0,\cdots,n_y-1),\\ &\text{wk}[n_x+n_y+j_z]=w^{(z)}_{j_z}\quad(j_z=0,\cdots,n_z-1)\end{aligned}$$

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.

(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_qfps3d */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>

int main()
{
    int n0=8,isw0=4;
    int nx;
    int ny;
    int nz;
    double *r;
    int lx;
    int ly;
    int lz;
    int isw;
    int *iwk;
    int niwk=60;
    double *wk;
    int nwk;
    int ierr;
    int i,j,k,m,nd2,is;
    int nt = 2;
    double *p,t,tx,ty,tz,dt,dfx,dfy,dfz,f0,f1,f2,f3;

    printf( "    *** ASL_qfps3d ***\n" );
    printf( "\n     ** Input **\n\n" );

    nx=n0;
    ny=n0;
    nz=n0;
    lx=(n0+2)/2*2;
    ly=ny;
    lz=nz;
```

```
nwk=nx+2*(ny+nz)+lx*ly*lz;

r = ( double * )malloc((size_t)( sizeof(double) * (lx*ly*lz*(isw0+2)) ));
if( r == NULL )
{
    printf( "no enough memory for array r\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
p = ( double * )malloc((size_t)( sizeof(double) * (isw0+2) ));
if( p == NULL )
{
    printf( "no enough memory for array p\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

printf( "\t isw=0, 2 to %6d\n", isw0+1 );
printf( "\t nx=%6d\n\t ny=%6d\n\t nz=%6d\n\n", nx,ny,nz );
dt=0.5;
f0=1.0/(2.0*dt);
f1=0.62*f0;
f2=0.14*f0;
f3=0.55*f0;
nd2=(int) (nx+1)/2;
dfx=1.0/(dt*nx);
dfy=1.0/(dt*ny);
dfz=1.0/(dt*nz);
p[isw0+1]=0.0;
for( k=0 ; k<nz ; k++ )
{
    tz=(double) k*dt;
    for( j=0 ; j<ny ; j++ )
    {
        ty=(double) j*dt;
        for( i=0 ; i<nx ; i++ )
        {
            tx=(double) i*dt;
            t=cos(2.0*M_PI*f1*tx)+cos(2.0*M_PI*f2*ty)
                +cos(2.0*M_PI*f3*tz);
            r[i+lx*(j+ly*(k+lz*(isw0+1)))]=t;
            p[isw0+1] += (t*t);
        }
    }
}
p[isw0+1] /= (double) (nx*ny*nz);
printf( "\tTime series data\n");
for( k=0 ; k<nz ; k++ )
{
    printf( "\t r[i+%3d*(j+%3d*%3d)]\n", lx,ly,k);
    printf( "  i/j");
    for( j=0 ; j<ny ; j++ )
        printf( "%9d", j);
    printf( "\n" );
    printf( "  ------------------------------------");
    printf( "----------------------------------------\n");
    for( i=0 ; i<nx ; i++ )
    {
        printf( "%5d", i );
        for( j=0 ; j<ny ; j++ )
            printf( "%9.4lf", r[i+lx*(j+ly*(k+lz*(isw0+1)))] );
        printf( "\n" );
    }
    printf( "\n" );
}
printf( "\n");
printf( "\tTime domain power =%9.4lf\n", p[isw0+1]);
printf( "\tSignal frequency =(%9.4lf, %9.4lf, %9.4lf)\n", f1, f2, f3);
is=0;

for( isw=0 ; isw<=isw0 ; isw++ )
{
    for( k=0 ; k<nz ; k++ )
        for( j=0 ; j<ny ; j++ )
            for( i=0 ; i<nx ; i++ )
                r[i+lx*(j+ly*(k+lz*isw))]=
                    r[i+lx*(j+ly*(k+lz*(isw0+1)))];
    if (isw != 0)
        is=isw+1;

    ierr = ASL_qfps3d(nx, ny, nz, &r[lx*ly*lz*isw],
         lx, ly, lz, is, iwk, wk, nt);
    p[isw]=0.0;
    if (nx%2==0)
```

```
    {
        m=nd2-1;
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                for( i=1 ; i<m ; i++ )
                    p[isw]+=2.0
                        *r[i+lx*(j+ly*(k+lz*isw))];
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                p[isw]+=r[lx*(j+ly*(k+lz*isw))]
                    +r[m+lx*(j+ly*(k+lz*isw))];
    }
    else
    {
        m=nd2;
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                for( i=1 ; i<m ; i++ )
                    p[isw]+=2.0
                        *r[i+lx*(j+ly*(k+lz*isw))];
        for( k=0 ; k<nz ; k++ )
            for( j=0 ; j<ny ; j++ )
                p[isw]+=r[lx*(j+ly*(k+lz*isw))];
    }
}


printf( "\n     ** Output **\n\n" );
printf( "\tierr = %6d\n", ierr );

isw=0;
printf( "\t(Modified) periodogram (Raw)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
printf( "\n");

isw=1;
printf( "\t(Modified) periodogram (Hanning)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
```

```
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
printf( "\n");

isw=2;
printf( "\t(Modified) periodogram (Bartlett)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "------------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
printf( "\n");

isw=3;
```

```
printf( "\t(Modified) periodogram (Welch)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
printf( "\n");

isw=4;
printf( "\t(Modified) periodogram (Parzen)\n");
printf( "\tFrequency domain power=%9.4lf\n", p[isw] );
for( k=(nz+1)/2 ; k<nz ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", (k-nz)*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        for( j=0 ; j<(ny+1)/2 ; j++ )
            printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
        printf( "\n" );
    }
    printf( "\n");
}
for( k=0 ; k<(nz+1)/2 ; k++ )
{
    printf( "\tz-frq=%8.2lf\n", k*dfz );
    printf( "  x/y-freq");
    for( j=(ny+1)/2 ; j<ny ; j++ )
        printf( "%8.2lf", (j-ny)*dfy );
    for( j=0 ; j<(ny+1)/2 ; j++ )
        printf( "%8.2lf", j*dfy );
    printf( "\n");
    printf( "  ------------------------------------");
    printf( "-----------------------------------\n");
    for( i=0 ; i<nd2 ; i++ )
    {
        printf( "  %8.2lf", i*dfx );
        for( j=(ny+1)/2 ; j<ny ; j++ )
```

```
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            for( j=0 ; j<(ny+1)/2 ; j++ )
                printf( "%8.4lf", r[i+lx*(j+ly*(k+lz*isw))] );
            printf( "\n" );
        }
        printf( "\n");
    }
    printf( "\n");

    free( iwk );
    free( p );
    free( wk );
    free( r );

    return 0;
}
```

(d) 出力結果

```
  *** ASL_qfps3d ***

   ** Input **

   isw=0, 2 to      5
   nx=     8
   ny=     8
   nz=     8

  Time series data
   r[i+ 10*(j+  8*  0)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   3.0000   2.9048   2.6374   2.2487   1.8126   1.4122   1.1237   1.0020
  1   1.6319   1.5367   1.2693   0.8806   0.4445   0.0441  -0.2444  -0.3662
  2   1.2710   1.1759   0.9085   0.5197   0.0837  -0.3168  -0.6053  -0.7270
  3   2.9048   2.8097   2.5423   2.1535   1.7174   1.3170   1.0285   0.9068
  4   2.0628   1.9676   1.7002   1.3115   0.8754   0.4750   0.1865   0.0648
  5   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
  6   2.6374   2.5423   2.2748   1.8861   1.4500   1.0496   0.7611   0.6394
  7   2.4818   2.3866   2.1192   1.7304   1.2944   0.8940   0.6054   0.4837

   r[i+ 10*(j+  8*  1)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   1.8436   1.7484   1.4810   1.0923   0.6562   0.2558  -0.0327  -0.1545
  1   0.4754   0.3803   0.1129  -0.2759  -0.7119  -1.1123  -1.4009  -1.5226
  2   0.1146   0.0194  -0.2480  -0.6367  -1.0728  -1.4732  -1.7617  -1.8834
  3   1.7484   1.6532   1.3858   0.9971   0.5610   0.1606  -0.1279  -0.2496
  4   0.9064   0.8112   0.5438   0.1550  -0.2810  -0.6814  -0.9700  -1.0917
  5  -0.1075  -0.2027  -0.4701  -0.8588  -1.2949  -1.6953  -1.9838  -2.1055
  6   1.4810   1.3858   1.1184   0.7297   0.2936  -0.1068  -0.3953  -0.5170
  7   1.3253   1.2301   0.9627   0.5740   0.1379  -0.2625  -0.5510  -0.6727

   r[i+ 10*(j+  8*  2)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   1.0489   0.9538   0.6864   0.2976  -0.1384  -0.5388  -0.8274  -0.9491
  1  -0.3192  -0.4144  -0.6818  -1.0705  -1.5066  -1.9070  -2.1955  -2.3172
  2  -0.6800  -0.7752  -1.0426  -1.4313  -1.8674  -2.2678  -2.5563  -2.6781
  3   0.9538   0.8586   0.5912   0.2025  -0.2336  -0.6340  -0.9225  -1.0443
  4   0.1117   0.0166  -0.2508  -0.6396  -1.0756  -1.4761  -1.7646  -1.8863
  5  -0.9021  -0.9973  -1.2647  -1.6534  -2.0895  -2.4899  -2.7784  -2.9001
  6   0.6864   0.5912   0.3238  -0.0649  -0.5010  -0.9014  -1.1899  -1.3117
  7   0.5307   0.4355   0.1681  -0.2206  -0.6567  -1.0571  -1.3456  -1.4673

   r[i+ 10*(j+  8*  3)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   2.4540   2.3588   2.0914   1.7027   1.2666   0.8662   0.5777   0.4560
  1   1.0859   0.9907   0.7233   0.3346  -0.1015  -0.5019  -0.7904  -0.9122
  2   0.7250   0.6298   0.3624  -0.0263  -0.4624  -0.8628  -1.1513  -1.2730
  3   2.3588   2.2636   1.9962   1.6075   1.1714   0.7710   0.4825   0.3608
  4   1.5168   1.4216   1.1542   0.7655   0.3294  -0.0710  -0.3595  -0.4812
  5   0.5029   0.4078   0.1404  -0.2484  -0.6844  -1.0849  -1.3734  -1.4951
  6   2.0914   1.9962   1.7288   1.3401   0.9040   0.5036   0.2151   0.0934
  7   1.9357   1.8406   1.5732   1.1844   0.7484   0.3480   0.0594  -0.0623

   r[i+ 10*(j+  8*  4)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   2.8090   2.7138   2.4464   2.0577   1.6216   1.2212   0.9327   0.8110
  1   1.4409   1.3457   1.0783   0.6896   0.2535  -0.1469  -0.4354  -0.5571
  2   1.0800   0.9849   0.7175   0.3287  -0.1073  -0.5077  -0.7963  -0.9180
  3   2.7138   2.6187   2.3513   1.9625   1.5265   1.1261   0.8375   0.7158
  4   1.8718   1.7766   1.5092   1.1205   0.6844   0.2840  -0.0045  -0.1262
  5   0.8580   0.7628   0.4954   0.1067  -0.3294  -0.7298  -1.0183  -1.1401
  6   2.4464   2.3513   2.0839   1.6951   1.2591   0.8587   0.5701   0.4484
  7   2.2908   2.1956   1.9282   1.5395   1.1034   0.7030   0.4145   0.2927

   r[i+ 10*(j+  8*  5)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   1.2929   1.1977   0.9303   0.5416   0.1055  -0.2949  -0.5834  -0.7051
  1  -0.0752  -0.1704  -0.4378  -0.8265  -1.2626  -1.6630  -1.9515  -2.0733
  2  -0.4361  -0.5312  -0.7987  -1.1874  -1.6235  -2.0239  -2.3124  -2.4341
  3   1.1977   1.1025   0.8351   0.4464   0.0103  -0.3901  -0.6786  -0.8003
  4   0.3557   0.2605  -0.0069  -0.3956  -0.8317  -1.2321  -1.5206  -1.6423
  5  -0.6582  -0.7533  -1.0207  -1.4095  -1.8455  -2.2459  -2.5345  -2.6562
```

```
  6   0.9303   0.8351   0.5677   0.1790  -0.2571  -0.6575  -0.9460  -1.0677
  7   0.7746   0.6795   0.4121   0.0233  -0.4127  -0.8131  -1.1017  -1.2234

   r[i+ 10*(j+  8*  6)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   1.4122   1.3170   1.0496   0.6609   0.2248  -0.1756  -0.4641  -0.5858
  1   0.0441  -0.0511  -0.3185  -0.7072  -1.1433  -1.5437  -1.8322  -1.9539
  2  -0.3168  -0.4119  -0.6793  -1.0681  -1.5041  -1.9045  -2.1931  -2.3148
  3   1.3170   1.2219   0.9545   0.5657   0.1297  -0.2707  -0.5593  -0.6810
  4   0.4750   0.3798   0.1124  -0.2763  -0.7124  -1.1128  -1.4013  -1.5230
  5  -0.5388  -0.6340  -0.9014  -1.2902  -1.7262  -2.1266  -2.4151  -2.5369
  6   1.0496   0.9545   0.6871   0.2983  -0.1377  -0.5381  -0.8267  -0.9484
  7   0.8940   0.7988   0.5314   0.1427  -0.2934  -0.6938  -0.9823  -1.1041

   r[i+ 10*(j+  8*  7)]
i/j        0        1        2        3        4        5        6        7
-------------------------------------------------------------------------------
  0   2.8910   2.7958   2.5284   2.1397   1.7036   1.3032   1.0147   0.8930
  1   1.5229   1.4277   1.1603   0.7716   0.3355  -0.0649  -0.3534  -0.4751
  2   1.1620   1.0669   0.7995   0.4107  -0.0253  -0.4257  -0.7143  -0.8360
  3   2.7958   2.7007   2.4333   2.0445   1.6085   1.2080   0.9195   0.7978
  4   1.9538   1.8586   1.5912   1.2025   0.7664   0.3660   0.0775  -0.0442
  5   0.9400   0.8448   0.5774   0.1886  -0.2474  -0.6478  -0.9364  -1.0581
  6   2.5284   2.4333   2.1659   1.7771   1.3410   0.9406   0.6521   0.5304
  7   2.3728   2.2776   2.0102   1.6215   1.1854   0.7850   0.4965   0.3747


  Time domain power =   1.6439
  Signal frequency =(   0.6200,    0.1400,    0.5500)

   ** Output **

  ierr =      0
  (Modified) periodogram (Raw)
  Frequency domain power=   1.5531
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0007  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0158  0.0188  0.0350  0.2150  0.0583  0.2150  0.0350  0.0188
    0.25  0.0000  0.0000  0.0000  0.0000  0.0239  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.1352  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0781  0.0000  0.0000  0.0000

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0136  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.2513  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0071  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

```
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000


  (Modified) periodogram (Hanning)
  Frequency domain power=   1.0699
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0001  0.0004  0.0001  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0164  0.0045  0.0053  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0027  0.0004  0.0023  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0036  0.0427  0.0087  0.0427  0.0036  0.0001
    0.25  0.0000  0.0000  0.0009  0.0064  0.0041  0.0158  0.0009  0.0000
    0.50  0.0000  0.0000  0.0000  0.0136  0.0543  0.0136  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0137  0.0547  0.0137  0.0000  0.0000

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0009  0.0053  0.0045  0.0164  0.0009  0.0000
    0.25  0.0000  0.0000  0.0002  0.0006  0.0031  0.0058  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0034  0.0136  0.0034  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0034  0.0137  0.0034  0.0000  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0176  0.0704  0.0176  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0044  0.0176  0.0044  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0078  0.0310  0.0078  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0019  0.0078  0.0019  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000


  (Modified) periodogram (Bartlett)
  Frequency domain power=   1.0593
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0000  0.0001  0.0000  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0000  0.0079  0.0346  0.0050  0.0000  0.0002
    0.25  0.0000  0.0000  0.0000  0.0014  0.0062  0.0009  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0003  0.0001  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------
    0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
    0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
```

```
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0141  0.0037  0.0074  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0022  0.0005  0.0017  0.0000  0.0000
    0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
    0.75  0.0000  0.0001  0.0000  0.0012  0.0096  0.0024  0.0000  0.0001

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0007  0.0594  0.0070  0.0594  0.0007  0.0000
    0.25  0.0000  0.0000  0.0001  0.0089  0.0017  0.0129  0.0001  0.0000
    0.50  0.0000  0.0003  0.0000  0.0113  0.0622  0.0113  0.0000  0.0003
    0.75  0.0000  0.0003  0.0000  0.0069  0.0554  0.0139  0.0000  0.0003

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0074  0.0037  0.0141  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0011  0.0011  0.0030  0.0000  0.0000
    0.50  0.0000  0.0001  0.0000  0.0021  0.0113  0.0021  0.0000  0.0001
    0.75  0.0000  0.0001  0.0000  0.0014  0.0108  0.0027  0.0000  0.0001

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0005  0.0000  0.0165  0.0907  0.0165  0.0000  0.0005
    0.25  0.0000  0.0001  0.0000  0.0030  0.0165  0.0030  0.0000  0.0001
    0.50  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0005  0.0001  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0002  0.0000  0.0050  0.0346  0.0079  0.0000  0.0002
    0.25  0.0000  0.0000  0.0000  0.0009  0.0065  0.0015  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0008  0.0002  0.0000  0.0000


  (Modified) periodogram (Welch)
  Frequency domain power=   1.2154
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0005  0.0030  0.0005  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0001  0.0003  0.0001  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0001  0.0001  0.0004  0.0051  0.0357  0.0028  0.0003  0.0001
    0.25  0.0000  0.0000  0.0000  0.0005  0.0038  0.0003  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0002  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0003  0.0004  0.0011  0.0103  0.1247  0.0186  0.0011  0.0004
    0.25  0.0000  0.0000  0.0001  0.0011  0.0131  0.0020  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0009  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0001  0.0017  0.0002  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0122  0.0012  0.0090  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0013  0.0002  0.0009  0.0000  0.0000
    0.50  0.0000  0.0000  0.0001  0.0017  0.0095  0.0005  0.0001  0.0000
    0.75  0.0000  0.0000  0.0001  0.0005  0.0078  0.0012  0.0001  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.1034  0.0186  0.1034  0.0000  0.0000
    0.25  0.0000  0.0000  0.0000  0.0115  0.0021  0.0104  0.0000  0.0000
    0.50  0.0002  0.0003  0.0008  0.0158  0.0862  0.0044  0.0007  0.0003
    0.75  0.0002  0.0002  0.0007  0.0052  0.0760  0.0115  0.0007  0.0002

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0000  0.0090  0.0012  0.0122  0.0001  0.0000
    0.25  0.0000  0.0000  0.0000  0.0010  0.0001  0.0012  0.0000  0.0000
    0.50  0.0000  0.0000  0.0001  0.0016  0.0086  0.0004  0.0001  0.0000
    0.75  0.0000  0.0000  0.0001  0.0006  0.0083  0.0012  0.0001  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
-------------------------------------------------------------------------------
    0.00  0.0003  0.0004  0.0011  0.0186  0.1247  0.0103  0.0011  0.0004
    0.25  0.0000  0.0000  0.0001  0.0020  0.0133  0.0011  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0004  0.0031  0.0003  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0004  0.0001  0.0000  0.0000

  z-frq=    0.75
```

```
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0001  0.0001  0.0003  0.0028  0.0357  0.0051  0.0004  0.0001
    0.25  0.0000  0.0000  0.0000  0.0003  0.0037  0.0005  0.0000  0.0000
    0.50  0.0000  0.0000  0.0000  0.0000  0.0003  0.0000  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0005  0.0001  0.0000  0.0000


  (Modified) periodogram (Parzen)
  Frequency domain power=   0.9132
  z-frq=   -1.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0001  0.0023  0.0053  0.0023  0.0001  0.0000
    0.25  0.0000  0.0000  0.0001  0.0010  0.0023  0.0010  0.0001  0.0000
    0.50  0.0000  0.0000  0.0000  0.0001  0.0001  0.0001  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0006  0.0092  0.0209  0.0089  0.0006  0.0000
    0.25  0.0000  0.0000  0.0003  0.0039  0.0090  0.0038  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0002  0.0005  0.0002  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  z-frq=   -0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0014  0.0162  0.0313  0.0112  0.0005  0.0000
    0.25  0.0000  0.0000  0.0005  0.0062  0.0120  0.0044  0.0002  0.0000
    0.50  0.0000  0.0000  0.0000  0.0003  0.0007  0.0004  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0003  0.0007  0.0003  0.0000  0.0000

  z-frq=   -0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0030  0.0125  0.0054  0.0016  0.0012  0.0000
    0.25  0.0000  0.0000  0.0008  0.0019  0.0010  0.0030  0.0010  0.0000
    0.50  0.0000  0.0000  0.0001  0.0032  0.0106  0.0063  0.0006  0.0000
    0.75  0.0000  0.0000  0.0003  0.0046  0.0108  0.0048  0.0003  0.0000

  z-frq=    0.00
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0001  0.0047  0.0117  0.0007  0.0117  0.0047  0.0001
    0.25  0.0000  0.0000  0.0011  0.0006  0.0055  0.0140  0.0033  0.0001
    0.50  0.0000  0.0000  0.0003  0.0089  0.0291  0.0168  0.0016  0.0000
    0.75  0.0000  0.0000  0.0007  0.0107  0.0253  0.0111  0.0007  0.0000

  z-frq=    0.25
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0012  0.0016  0.0054  0.0125  0.0030  0.0001
    0.25  0.0000  0.0000  0.0002  0.0005  0.0086  0.0110  0.0019  0.0000
    0.50  0.0000  0.0000  0.0002  0.0048  0.0151  0.0085  0.0008  0.0000
    0.75  0.0000  0.0000  0.0003  0.0047  0.0110  0.0049  0.0003  0.0000

  z-frq=    0.50
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0005  0.0112  0.0313  0.0162  0.0014  0.0000
    0.25  0.0000  0.0000  0.0003  0.0055  0.0155  0.0081  0.0007  0.0000
    0.50  0.0000  0.0000  0.0001  0.0010  0.0028  0.0014  0.0001  0.0000
    0.75  0.0000  0.0000  0.0000  0.0003  0.0008  0.0003  0.0000  0.0000

  z-frq=    0.75
x/y-freq   -1.00   -0.75   -0.50   -0.25    0.00    0.25    0.50    0.75
------------------------------------------------------------------------
    0.00  0.0000  0.0000  0.0006  0.0089  0.0209  0.0092  0.0006  0.0000
    0.25  0.0000  0.0000  0.0002  0.0039  0.0091  0.0040  0.0003  0.0000
    0.50  0.0000  0.0000  0.0000  0.0003  0.0006  0.0003  0.0000  0.0000
    0.75  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

第 7 章　ソート

7.1　概 要

本章では, データのソートを行う関数について説明する.
本章の関数は, 処理を複数のスレッドに分割して割り当て, 割り当てられた処理を並列に行う.
本ライブラリでは, 以下の機能を持つ関数が用意されている.

(1) データ列のソート

(2) ペアデータ列のソート

7.1.1 使用上の注意

ソートされるデータ数が小さいと, 演算コストに対して並列処理オーバヘッドの影響が大きいため非並列処理関数を用いた場合よりも性能が低下することがある.

7.1.2 使用しているアルゴリズム

昇順にソートする場合のアルゴリズムを以下に示す. 降順にソートする場合のアルゴリズムは大小が異なるだけで同様である.

(1) シェル・ソート (shell sort)
(1) 間隔 h を設定する.
(2) データ列より, 間隔 h の部分列をすべて取り出す.
(3) 各部分列の中が小さい順に並ぶように隣同士を比較する.
逆順なら位置を交換し, 交換したデータはさらにその前のデータとの順序を確かめる.
さらに逆順ならば位置の交換が前にさかのぼって行われる.
(4) 間隔 h を小さくして (2) から (3) を繰り返し, h =1 の処理を行えばソートは終了する.

(2) ヒープ・ソート (heap sort)
(1) 与えられたデータをヒープ・ツリー (整列二分木. 親は子よりも大きいか等しい値を持っている) に構成する.
(2) ルートとツリーの一番後ろのデータを交換する.
(3) 一番後ろのデータを除いた部分を A とする.
(4) A を新しいツリーと考え, これを再びヒープ・ツリーに構成する.
(5) (2) から (4) を繰り返し, データがルートだけになればソートは終了する.

(3) クイック・ソート (quick sort)
(1) ソート区間内のデータ数を数える.
(2) データ数により以下のことをする.

- データ数が 1 以下のとき :
  何もしない.
- データ数が 2 のとき :
  逆順なら位置を交換する.
- データ数が 3 以上のとき :
  ① 区間内から枢軸値を一つ選ぶ.
  ② 区間内のデータを枢軸値より小さいものと大きいものとの二つの区間に振り分ける.

(3) (1) から (2) を繰り返し, すべてのデータ区間のデータ数が 2 以下になればソートは終了する.

(4) マージ・ソート (merge sort)
(1) ソート区間内のデータ数を数える.
(2) データ数により以下のことをする.

- データ数が 1 のとき :
  何もしない.
- データ数が 2 のとき :
  逆順なら位置を交換する.
- データ数が 3 以上のとき :
  ① 区間のデータを半分ずつ前半と後半に分ける.
  ② 前半のデータを再帰的にマージ・ソートする.
  後半のデータを再帰的にマージ・ソートする.
  ③ ソートされた前半と後半のデータをマージする.

7.1.3 参考文献

(1) Niklaus Wirth, “ALGORITHMS + DATA STRUCTURES = PROGRAMS”, Prentice–Hall Inc. (1976).

(2) 浪平博人, “ソート・検索”, CQ 出版.

(3) 近藤嘉雪, “アルゴリズムとデータ構造”, SOFTBANK.

7.2 ソート

7.2.1 ASL_qssta1, ASL_pssta1 データ列のソート

(1) **機 能**

n 個のデータ $a_{i_k}(k = 1, 2, \cdots, n)$ が与えられたとき a_i を並べ換えたデータ列 $a_{j_k}(k = 1, 2, \cdots, n)$ を求める. ただし, a_j は

昇順の場合 : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

降順の場合 : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

を満たす.

(2) **使用法**

倍精度関数:

ierr = ASL_qssta1 (a,n,isw,wk,iwk,nt);

単精度関数:

ierr = ASL_pssta1 (a,n,isw,wk,iwk,nt);

(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∗ }	n	入 力	ソートされるデータ a_i
				出 力	ソートされたデータ a_j
2	n	I	1	入 力	配列 a の大きさ
3	isw	I	1	入 力	ソート法の選択スイッチ (注意事項 (a) 参照)
4	wk	{ D∗ / R∗ }	n	ワーク	作業領域
5	iwk	I*	2×n	ワーク	作業領域
6	nt	I	1	入 力	生成するタスク数
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) n≥1

(b) isw=1, 2, 3, 4, −1, −2, −3, −4

(c) nt ≥ 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.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) isw で選択するソート法は以下の通りである.

isw	ソート法	isw	ソート法
1	シェル・ソート (昇順)	−1	シェル・ソート (降順)
2	ヒープ・ソート (昇順)	−2	ヒープ・ソート (降順)
3	クイック・ソート (昇順)	−3	クイック・ソート (降順)
4	マージ・ソート (昇順)	−4	マージ・ソート (降順)

利用者は入力データの性質により適切なソート法を選べばよい. 各ソート法の特徴を以下に示す.

· シェル・ソート

計算量は平均 $O(n^{1.5})$ 程度である. どのようなデータに対しても安定して速いソートを行う. とくに, データ系列の一部がソートされている場合は速くなる.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· ヒープ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· クイック・ソート

計算量は平均 $O(n \log n)$ だが, 最初から部分的にソートされているなど, ある種の規則性があるものに対しては, 大変非効率的なソートになる. ランダムなデータに対してはもっとも速いソート方法である.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

· マージ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

同じ値のデータ間で整列前の順序関係が保たれる.

(7) **使用例**

(a) 問 題

a [0] = 5.0

a [1] = 4.0

a [2] = 9.0

a [3] = 6.0

a [4] = 2.0

a [5] = 5.0

をシェル・ソートで昇順にソートする.

(b) 入力データ

配列 a, n=6, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_qssta1 */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*wk;
    int n,isw,*iwk,ierr;
    int nt;
    int i;
    FILE *fp;


    fp = fopen( "qssta1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw=1;
    n=6;
    nt=4;

    printf( "    *** ASL_qssta1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

    a = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*n) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\tArray a" );

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a[i] );
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    fclose( fp );

    ierr = ASL_qssta1(a, n, isw, wk, iwk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );

    printf( "\tArray a" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    free( a );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_qssta1 ***
 ** Input **
n=  6
Array a       5       4       9       6       2       5
 ** Output **
ierr =      0
Array a       2       4       5       5       6       9
```

7.2.2 ASL_qssta2, ASL_pssta2 ペアデータ列のソート

(1) **機 能**

2 組の n 個のデータ $a_{i_k}(k = 1, 2, \cdots, n), b_{i_k}(k = 1, 2, \cdots, n)$ が与えられたとき a_i を並べ換えたデータ列 $a_{j_k}(k = 1, 2, \cdots, n)$ と a_j に対応するデータ列 $b_{j_k}(k = 1, 2, \cdots, n)$ を求める. ただし, a_j は

昇順の場合 : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

降順の場合 : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

を満たす.

なお, 2 次ソートを指定した場合には, $a_{j_k} = a_{j_{k+1}}$ を満たす任意の k について

昇順の場合 : $b_{j_k} \leq b_{j_{k+1}}$

降順の場合 : $b_{j_k} \geq b_{j_{k+1}}$

を満たすように $j = j_1, j_2, \cdots, j_n$ を決める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_qssta2  (a,n,b,isw1,isw2,wk,iwk,nt);
```

単精度関数:

```
ierr = ASL_pssta2  (a,n,b,isw1,isw2,wk,iwk,nt);
```

(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∗}	n	入 力	ソートされるデータ a_i
				出 力	ソートされたデータ a_j
2	n	I	1	入 力	配列 a の大きさ
3	b	{D∗ / R∗}	n	入 力	a_i に対するデータ b_i
				出 力	ソートされた a_j に対するデータ b_j
4	isw1	I	1	入 力	ソート法の選択スイッチ (注意事項 (a) 参照)
5	isw2	I	1	入力	2 次ソートスイッチ isw2=0 : 2 次ソートは行わない isw2=1 : 2 次ソートを行う
6	wk	{D∗ / R∗}	2×n	ワーク	作業領域
7	iwk	I*	2×n	ワーク	作業領域
8	nt	I	1	入 力	生成するタスク数
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≥1

(b) isw1=1, 2, 3, 4, −1, −2, −3, −4

(c) isw2=0 または 1

(d) nt ≥ 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.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) isw1 で選択するソート法は以下の通りである.

isw1	ソート法	isw1	ソート法
1	シェル・ソート (昇順)	−1	シェル・ソート (降順)
2	ヒープ・ソート (昇順)	−2	ヒープ・ソート (降順)
3	クイック・ソート (昇順)	−3	クイック・ソート (降順)
4	マージ・ソート (昇順)	−4	マージ・ソート (降順)

利用者は入力データの性質により適切なソート法を選べばよい. 各ソート法の特徴を以下に示す.

· シェル・ソート

計算量は平均 $O(n^{1.5})$ 程度である. どのようなデータに対しても安定して速いソートを行う. とくに, データ系列の一部がソートされている場合は速くなる.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· ヒープ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· クイック・ソート

計算量は平均 $O(n \log n)$ だが, 最初から部分的にソートされているなど, ある種の規則性があるものに対しては, 大変非効率的なソートになる. ランダムなデータに対してはもっとも速いソート方法である.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

· マージ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

同じ値のデータ間で整列前の順序関係が保たれる.

(7) 使用例

(a) 問 題

a [0] = 5.0, b [0] = 3.0
a [1] = 4.0, b [1] = 4.0
a [2] = 9.0, b [2] = 2.0
a [3] = 6.0, b [3] = 3.0
a [4] = 2.0, b [4] = 8.0
a [5] = 5.0, b [5] = 1.0

の a をシェル・ソートで昇順にソートし, それに対応するように b を並べかえる. 2 次ソートも行う.

(b) 入力データ

配列 a, 配列 b, n=6, isw1=1, isw2=1

(c) 主プログラム

```
/*      C interface example for ASL_qssta2 */

#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a,*b,*wk;
    int n,isw1,isw2,*iwk,ierr;
    int nt;
    int i;
    FILE *fp;


    fp = fopen( "qssta2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw1=1;
    isw2=1;
    n=6;
    nt=4;

    printf( "    *** ASL_qssta2 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

    a = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }

    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (2*n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*n) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf", &a[i], &b[i] );
    }
    printf( "\t   <Array a>    <Array b>\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g     %8.3g\n", a[i], b[i] );
    }

    fclose( fp );
```

```
    ierr = ASL_qssta2(a, n, b, isw1, isw2, wk, iwk, nt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );

    printf( "\t   <Array a>    <Array b>\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g     %8.3g\n", a[i], b[i] );
    }

    free( a );
    free( b );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

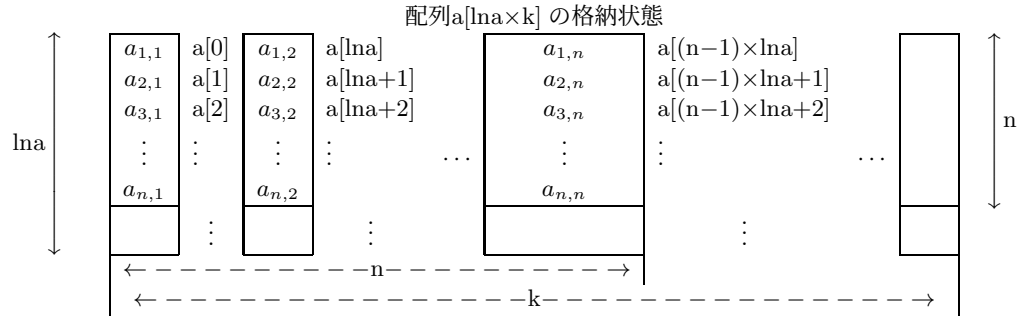
```
*** ASL_qssta2 ***

 ** Input **

n=  6

   <Array a>    <Array b>
       5            3
       4            4
       9            2
       6            3
       2            8
       5            1

 ** Output **

ierr =      0

   <Array a>    <Array b>
       2            8
       4            4
       5            1
       5            3
       6            3
       9            2
```

付 録 A　配列データの取扱い方法

A.1　行列に対応した配列データ

本ライブラリにおいては, しばしば行列に対応した配列データが使用されるが, 以下にその取扱い方法を述べる. 配列データを使用する関数を引用する場合, 利用者は引用する側のプログラム内で, その配列を宣言しておかなければならない. 宣言された配列を a [lna×k]　とすると, $n \times n$ 型行列 $A = (a_{i,j})$ $(i = 1, 2, \cdots, n;\ j = 1, 2, \cdots, n)$ は次の図のように格納される. この時の lna を**整合寸法**という. 行列に対応した配列を引数として使用する場合には, 引数

図 A−1　配列中の行列の格納形式



備 考

a.　lna≥ n, k≥ n でなければならない.

b.　行列の要素 $a_{i,j}$ は配列の要素 a[$(i-1)$+lna×$(j-1)$] に対応する.

として配列名, 次数のほかに, この整合寸法も関数に引渡さなければならない. これは, ASL C 言語インタフェースでは, 配列の格納方法は FORTRAN に準拠しており, 行列の要素 $a_{i,j}$ $(i = 1, 2, \cdots$,lna; $j = 1, 2, \cdots$,k) は, 配列の要素 a[l] (l=0, 1, 2, ···, lna×k−1) と次のように主記憶上で対応している必要があるためである.

$a_{1,1}$	$a_{2,1}$	···	$a_{\mathrm{lna},1}$	$a_{1,2}$	$a_{2,2}$	···
↕	↕	···	↕	↕	↕	···
a[0]	a[1]	···	a[lna−1]	a[lna]	a[lna+1]	···

例 ASL_dam1ad(実行列の和) の場合

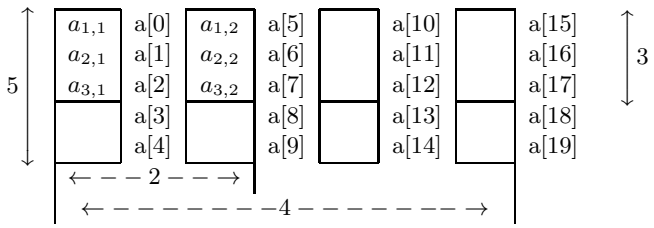
3×2 型行列 A, B の和を行列 C に求めるとする. 対応する配列 a, b, c の大きさをすべて $[5 \times 4]$ で宣言すると, 使用法は次のようになる.

```
/*      C interface example for ASL_dam1ad */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>
int main()
{
    double *a, *b, *c;
    int lma, lmb, lmc;
    int m, n, ierr;
    int k;
    lma = lmb = lmc = 5;
    k = 4;
    m = 3;
    n = 2;
    a = (double *)malloc((size_t) sizeof(double) * lma*k);
    if(a == NULL)
    {
        printf("no enough memory for array a\n");
        return -1;
```

```
    }
    b = (double *)malloc((size_t) sizeof(double) * lmb*k);
    if(b == NULL)
    {
        printf("no enough memory for array b\n");
        return -1;
    }
    c = (double *)malloc((size_t) sizeof(double) * lmc*k);
    if(c == NULL)
    {
        printf("no enough memory for array c\n");
        return -1;
    }
          ～

    ierr = ASL_dam1ad(a, lma , m, n, b, lmb, c,  lmc);

          ～
    free(a);
    free(b);
    free(c);
    return 0;
}
```

配列 a には, データが次のように格納される. 配列 b, c についても同様である.

図 A−2　配列 a 中の格納形式



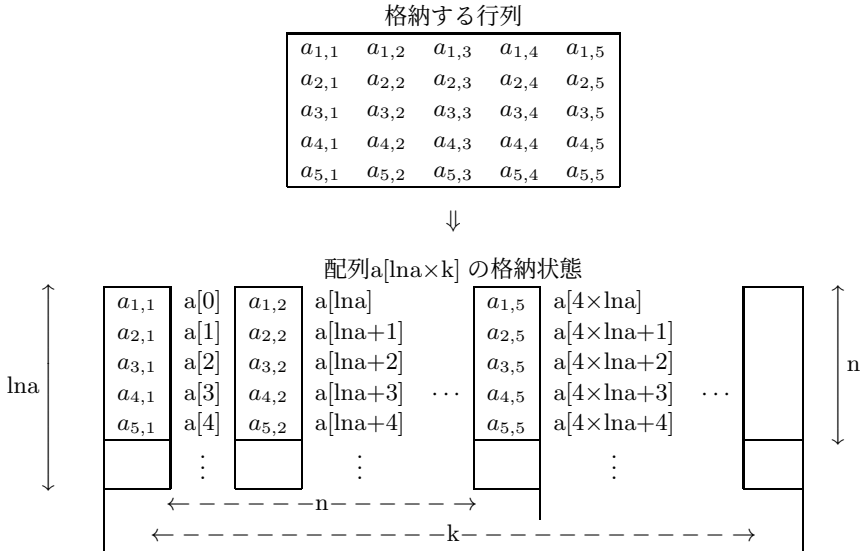
次数の異なるいくつかの配列をデータとして取り扱う場合には, そのうち最も大きな次数を lna とするような配列を一つ用意しておけば, この配列を逐次利用することができる. ただし, この時, 整合寸法として常に lna の値を与える必要がある.

A.2　データの格納方法

行列データの格納方法は, その行列の型によって異なっている. 以下にその方法を示す.

A.2.1　実行列 (2 次元配列型)

図 A−3　実行列 (2 次元配列型) の格納形式



備　考

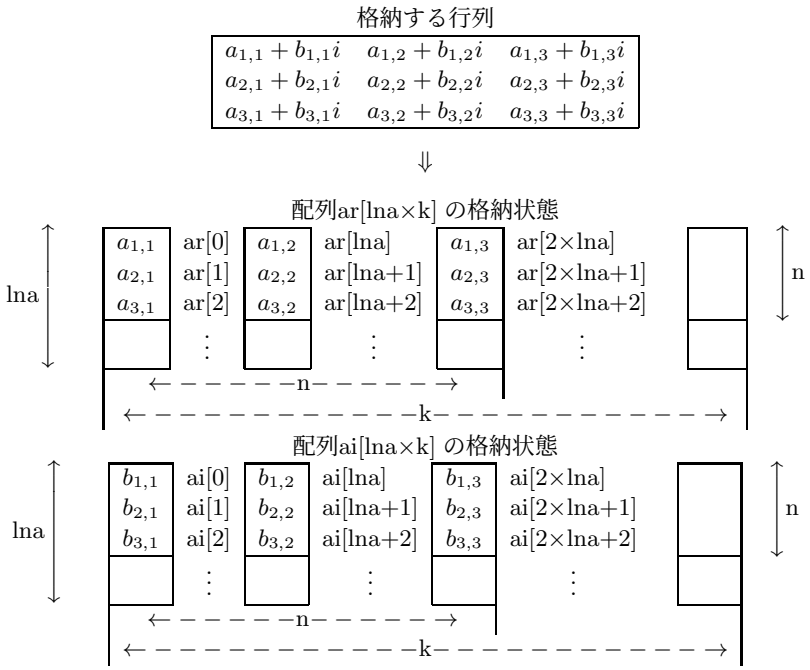
a.　lna ≥ n, k ≥ n を満たさなければならない.

A.2.2 複素行列

(1) **2 次元配列型, 実数引数型**

実部と虚部に分けて別々の配列に格納する.

図 A−4 複素行列 (2 次元配列型)(実数引数型) の格納形式

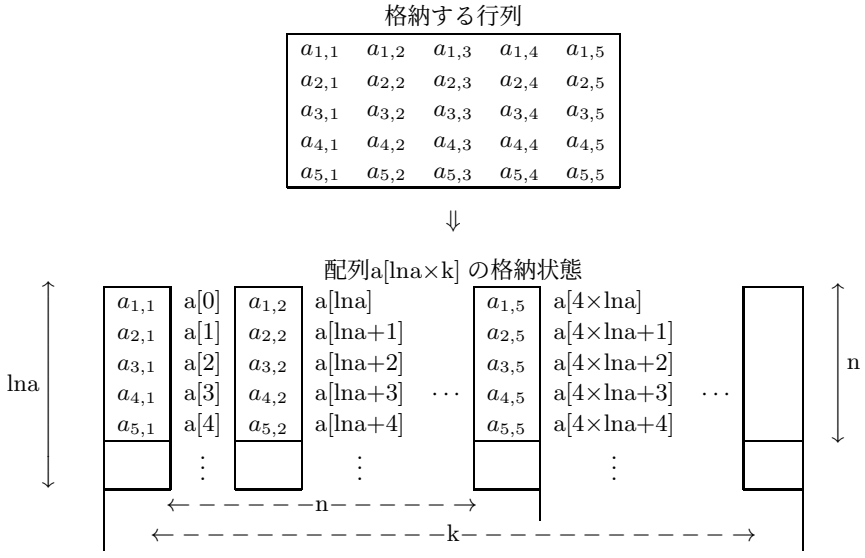


備 考

a. lna ≥ n, k ≥ n を満たさなければならない.

(2) **2 次元配列型, 複素引数型**

図 A−5 複素行列 (2 次元配列型)(複素引数型) の格納形式



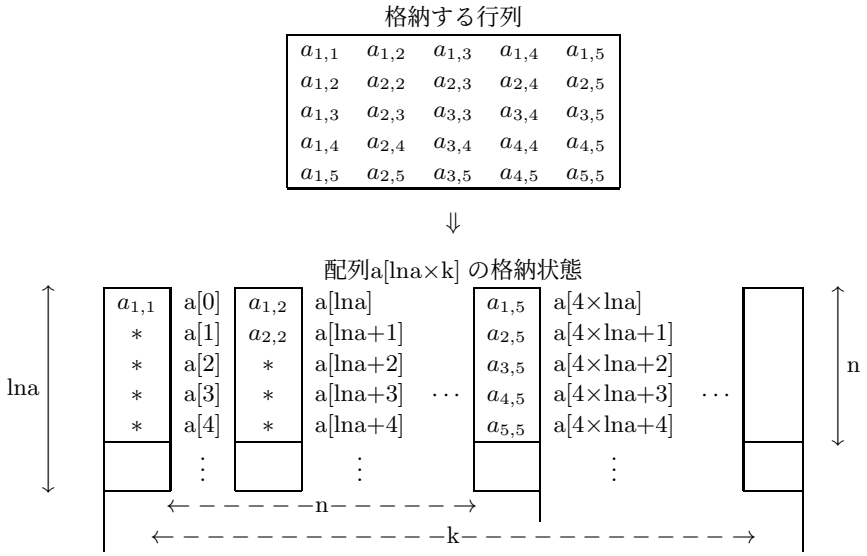
備 考

a. lna ≥ n, k ≥ n を満たさなければならない.

A.2.3 実対称行列, 正値対称行列

(1) 2 次元配列型, 上三角型

図 A−6　実対称行列 (2 次元配列型)(上三角型) の格納形式



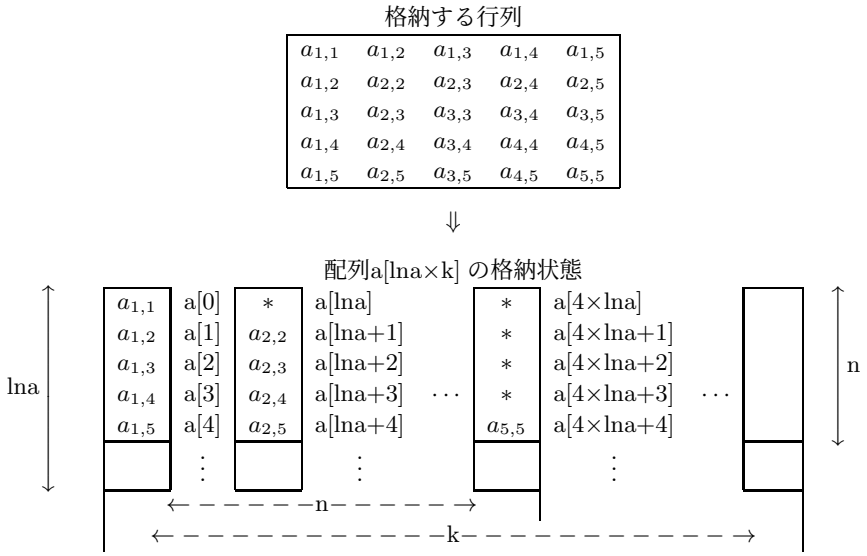
備　考

a.　∗ は, 任意の値であることを示す.

b.　lna ≥ n, k ≥ n を満たさなければならない.

(2) 2 次元配列型, 下三角型

図 A−7　実対称行列 (2 次元配列型)(下三角型) の格納形式



備　考

a.　∗ は, 任意の値であることを示す.

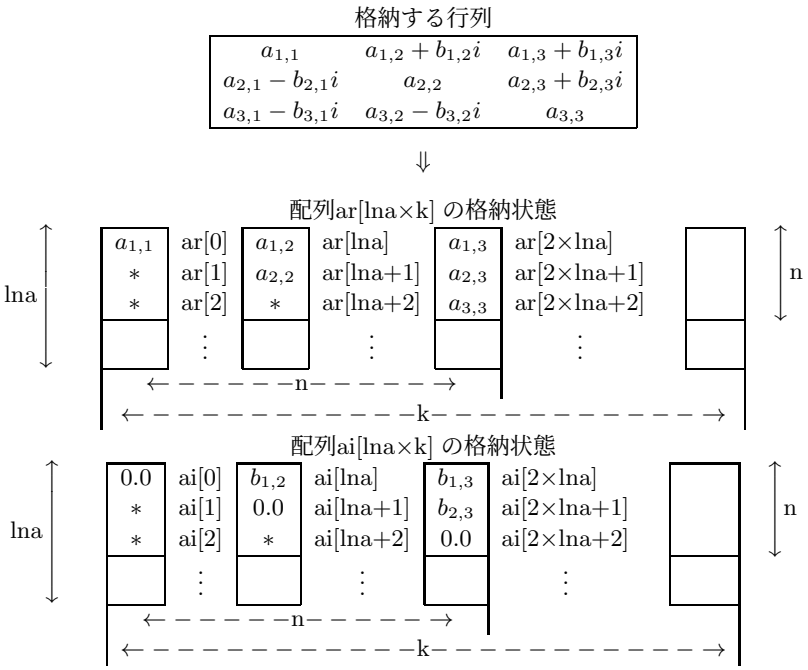
b.　lna ≥ n, k ≥ n を満たさなければならない.

A.2.4　エルミート行列

(1) **2 次元配列型, 実数引数型, 上三角型**

上三角部分の実部と虚部を別々の配列に格納する.

図 A–8　エルミート行列 (2 次元配列型)(実数引数型)(上三角型) の格納形式



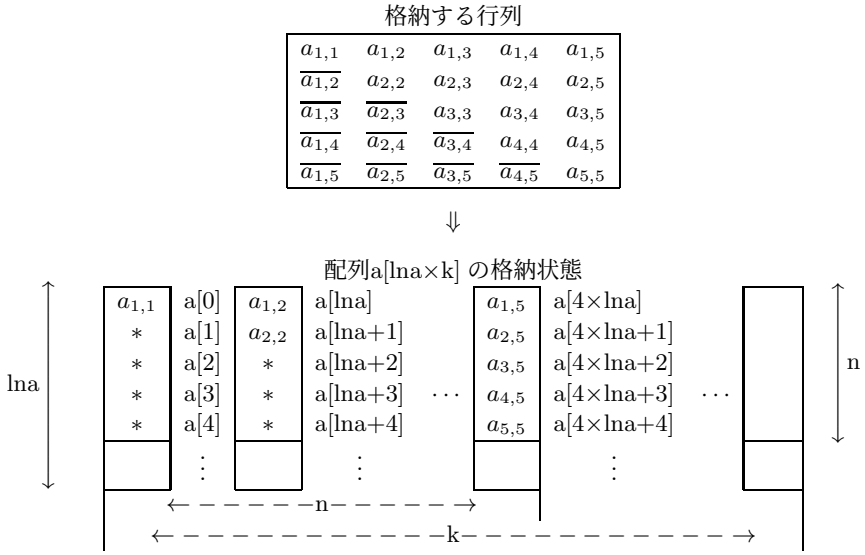
備　考

a.　∗ は, 任意の値であることを示す.

b.　lna ≥ n, k ≥ n を満たさなければならない.

(2) 2 次元配列型, 複素引数型, 上三角型

図 A−9　エルミート行列 (2 次元配列型)(複素引数型)(上三角型) の格納形式



備　考

a. x の複素共役を $\overline{x}$ で表している.

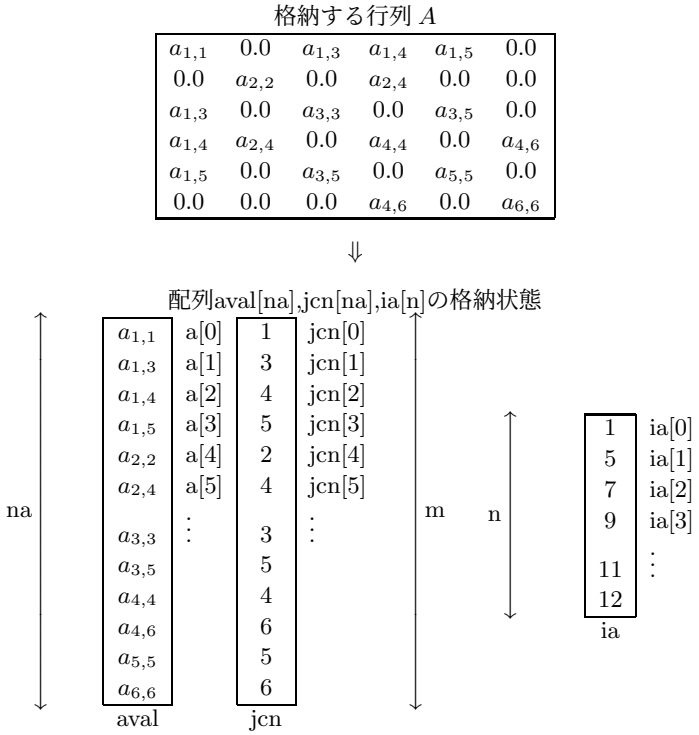
b. $*$ は, 任意の値であることを示す.

c. lna $\geq$ n, k $\geq$ n を満たさなければならない.

A.2.5 不規則スパース行列 (対称行列専用)

(1) スパース型 (対称行列の場合)

図 A−10 実対称不規則スパース行列 (スパース型) の格納形式



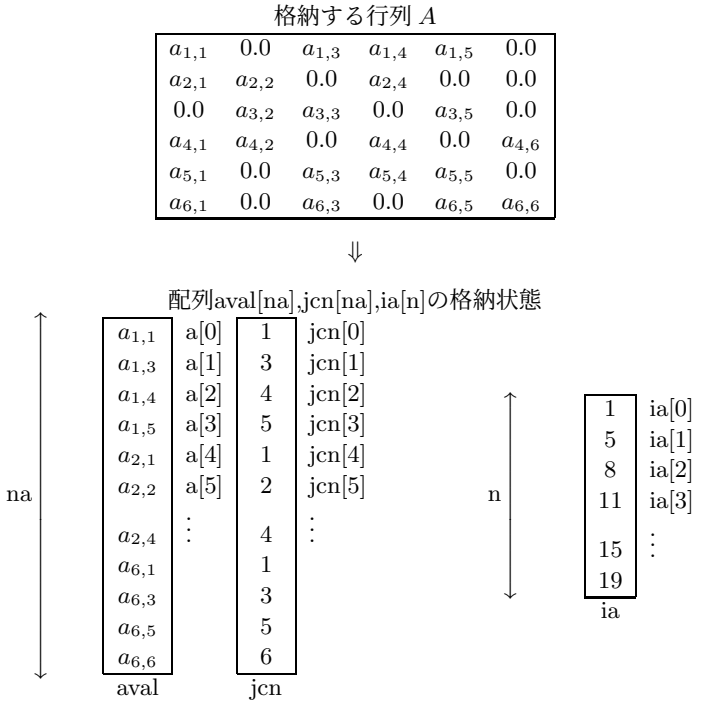
備 考

a. m は, 元の行列 A の上三角部の非零要素数.
b. 配列 aval には, 元の行列 A の上三角部の非零要素を第 1 行から順番に格納する.
c. 配列 jcn には, 配列 aval に格納した各要素の元の行列 A 上での列番号を格納する.
d. 配列 ia には, 対角要素の配列 a での位置に 1 を足した値を格納する.
e. $n \leq m < na$ を満たさなければならない.

A.2.6　不規則スパース行列

(1) スパース型

図 A−11　実非対称不規則スパース (スパース型) の格納形式



備　考

a. na は行列 A の非零要素の数.
b. 配列 aval には, 行列 A の非零要素を第 1 行から順番に格納する.
c. 配列 jcn には, 配列 aval に格納した各要素の元の行列 A 上での列番号を格納する.
d. 配列 ia には, 各行の先頭の非零要素の配列 aval での位置に 1 を足した値を格納する.
e. n < na を満たさなければならない.

(2) **ELLPACK 型**

図 A−12　非対称不規則スパース行列 (ELLPACK 型) の格納形式

格納する行列 A

$a_{1,1}$	$a_{1,2}$	0.0	$a_{1,4}$	0.0	$a_{1,6}$	0.0
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	0.0	0.0	$a_{2,7}$
0.0	$a_{3,2}$	$a_{3,3}$	0.0	0.0	$a_{3,6}$	0.0
$a_{4,1}$	$a_{4,2}$	0.0	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	0.0
0.0	0.0	0.0	$a_{5,4}$	$a_{5,5}$	0.0	0.0
$a_{6,1}$	0.0	$a_{6,3}$	$a_{6,4}$	0.0	$a_{6,6}$	$a_{6,7}$
0.0	$a_{7,2}$	0.0	0.0	0.0	0.0	$a_{7,7}$

⇓

配列 a[lna × m] の格納状態

$a_{1,1}$	a[0]	$a_{1,2}$	a[lna]	$a_{1,6}$	a[2 × lna]	$a_{1,4}$	a[3 × lna]	∗	a[4 × lna]	lna	n
$a_{2,2}$	a[1]	$a_{2,1}$	a[lna + 1]	$a_{2,3}$	a[2 × lna + 1]	$a_{2,4}$	a[3 × lna + 1]	$a_{2,7}$	a[4 × lna + 1]		
$a_{3,3}$	a[2]	$a_{3,2}$	a[lna + 2]	$a_{3,6}$	a[2 × lna + 2]	∗	a[3 × lna + 2]	∗	a[4 × lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
$a_{6,6}$		$a_{6,1}$		$a_{6,3}$		$a_{6,4}$		$a_{6,7}$			
$a_{7,7}$		$a_{7,2}$		∗		∗		∗			

< − − − − − m − − − − − >

配列 ja[lna × m] の格納状態

1	ja[0]	2	ja[lna]	6	ja[2 × lna]	4	ja[3 × lna]	0	ja[4 × lna]	lna	n
2	ja[1]	1	ja[lna + 1]	3	ja[2 × lna + 1]	4	ja[3 × lna + 1]	7	ja[4 × lna + 1]		
3	ja[2]	2	ja[lna + 2]	6	ja[2 × lna + 2]	0	ja[3 × lna + 2]	∗	ja[4 × lna + 2]		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		
6		1		3		4		7			
7		2		0		∗		∗			

< − − − − − m − − − − − >

備　考

a. n は, 行列 A の次数.

b. lna $\geq$ n を満たさなければならない.

c. m は, 行列 A の非零要素を格納する配列 a の列数.

d. 配列 a には, 行列 A の非零要素を次のように格納する.
- 第 1 列に対角要素を格納する.
- 第 2〜m 列には下三角部分および上三角部分の非零要素をつめて各行ごとに格納する. ここで, 各行の非零要素を格納する順序は, 順不同である.
- 残りの部分の ∗ となっている位置に対応する要素は任意の値でよい.

e. 配列 ja には, 配列 a に格納した各要素に対応する箇所に行列 A 上での列番号を格納する.
m − 1 が行内の下三角部分および下三角部分の非零要素数より大きくなるような行については, 行列 A の非零要素の列番号を詰めた ja 内の領域の最右端の右隣の位置には 0 を格納する.
残りの部分の ∗ となっている位置には任意の値を格納する.

付 録 B　　ASL で使用している計算機依存定数

B.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 B–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.

B.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 B–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

〈 共有メモリ並列機能編 〉

2023 年 3 月 ASL (1.1)
付属説明書 3.0.0-230301

日 本 電 気 株 式 会 社