科学技術計算ライブラリ
ASL C 言語インタフェース
ユーザーズガイド
＜基本機能編 第2分冊＞

はしがき

本書は, 科学技術計算ライブラリ ASL (Advanced Scientific Library) C 言語インタフェースの概念, 機能, 利用方法などについて説明したものです.
当製品に対応する説明書は 7 分冊からなっており, 構成は次のとおりです. このうち本書は, 基本機能第 2 分冊について記述したものです.

基本機能 第 1 分冊

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

基本機能 第 2 分冊

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

基本機能 第 3 分冊

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

基本機能 第 4 分冊

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

基本機能 第 5 分冊

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

基本機能 第 6 分冊

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

共有メモリ並列機能

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

2023 年　3 月　ASL 付属説明書　3.0.0-230301

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

(2) 製品名などの固有名詞は, 各メーカーの登録商標または商標です.

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

目 次

第 1 章　使用の手引

1.1　概　説

1.1.1　科学技術計算ライブラリ ASL C 言語インタフェースの概要

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

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

表 1–1　ASL C 言語インタフェース の機能分類

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

1.1.2　ASL C 言語インタフェース の特長

ASL C 言語インタフェースの特長は, 次のとおりである.

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

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

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

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

1.2 ライブラリの種類

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

表 1–2　ASL C 言語インタフェース で提供しているライブラリの種類

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

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

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

1.3 マニュアルについて

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

1.3.1 『概　要』

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

1.3.2 関数説明文の構成

各章の第 2 節では, 関数ごとに以下の順で説明している.

(1) 機能

(2) 使用法

(3) 引数と戻り値

(4) 制限条件

(5) エラーインディケータ (戻り値)

(6) 注意事項

(7) 使用例

各項目は次に述べる原則に従って記述されている.

1.3.3 各項目の内容

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

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

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

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

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

(3) **引　数**

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

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

(a) 引数と戻り値

引数と戻り値が記載されている.

(b) 型

引数と戻り値のデータの型を示す. 次の記号のいずれかに示されている.

I ：整数型

D ：倍精度実数型

R ：単精度実数型

Z ：倍精度複素数型

C ：単精度複素数型

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

(c) 大きさ

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

1　：変数であることを示す.

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

(d) 入出力

引数の内容説明が入力時であるか出力時であるかを示す.

i. 「入力」とだけある場合：

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

ii. 「出力」とだけある場合：

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

iii. 「入力」と「出力」の両方に説明がある場合：

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

iv. 「ワーク」とある場合：

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

(e) 内 容

入力時あるいは出力時に, 引数が保持している情報について説明される.

- 「引 数」の説明の例を次に示す.

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

倍精度関数:

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

単精度関数:

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

この場合の引数と戻り値の説明は次のようになる.

表 1−3 引数と戻り値の例

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

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

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

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

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

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

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

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

(4) **制限条件**

関数の引数の制限範囲を明確にしてある.

(5) **エラーインディケータ (戻り値)**

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

表 1–4　エラーインディケータ (戻り値) の出力値区分

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

(6) **注意事項**

関数を使用するときの注意点およびあいまいな点を明確にしてある.

(7) **使用例**

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

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

1.4 関数名

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

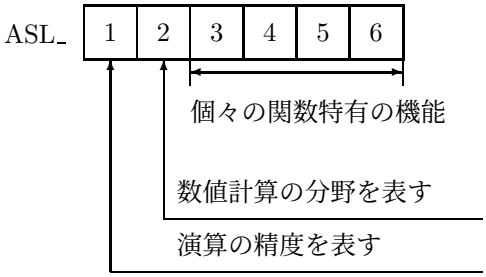


図 1−1　関数名の構成要素

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

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

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

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

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

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

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

1.5 ASL C言語インタフェースの複素数型

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

表 1–7 ASL C 言語インタフェースの複素引数の型

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

C 言語ならびに C++言語での使用例を以下に示す.

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

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

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

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

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

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

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

1.6 注意事項

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

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

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

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

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

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

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

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

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

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

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

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

第 2 章　連立 1 次方程式 (直接法)

2.1　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.
また, 行列の三角分解はその行列の性質に最も適した手法で行われているため, おのおのの行列ごとに使用手法が異なっている.
また, 実 3 重対角行列では, 係数行列の性質により, 実 3 重対角行列 (ベクトル型) と定係数型実 3 重対角行列 (スカラ型) の 2 種類に分類され, それぞれ以下の機能をもつ関数が用意されている.

(1) 連立 1 次方程式を解く (リダクション操作と求解またはガウス法を用いた求解を行う).

(2) 解を求める (リダクション操作後の求解のみを行う).

利用者は, (1), (2) の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.

2.1.1 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

(1) $\left\{\begin{matrix}\text{ASL_dbgmsl}\\ \text{ASL_rbgmsl}\end{matrix}\right\}$を利用する方法

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

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

(2) $\left\{\begin{matrix}\text{ASL_dbgmlu}\\ \text{ASL_rbgmlu}\end{matrix}\right\}$と$\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$を利用する方法

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A, \cdots);$$

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

$\left\{\begin{matrix}\text{ASL_dbgmlu}\\ \text{ASL_rbgmlu}\end{matrix}\right\}$で係数行列 A の三角分解を行い, $\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$で $A\boldsymbol{x} = \boldsymbol{b}$ の解を求める.

(3) 条件数も求める方法

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\} (A, \cdots, \&\ \text{cond}, \cdots);$$

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

$\left\{\begin{matrix}\text{ASL_dbgmlc}\\ \text{ASL_rbgmlc}\end{matrix}\right\}$で係数行列 A の三角分解と条件数の算出を行い, $\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$で $A\boldsymbol{x} = \boldsymbol{b}$ の解を求める.

(2) 行列式, 逆行列

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A, \cdots);$$

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

$\left\{\begin{matrix}\text{ASL_dbgmlu}\\ \text{ASL_rbgmlu}\end{matrix}\right\}$で行列 A の三角分解を行い, $\left\{\begin{matrix}\text{ASL_dbgmdi}\\ \text{ASL_rbgmdi}\end{matrix}\right\}$で行列式と逆行列を求める.

(3) 解の改良

(1) $\left\{\begin{matrix}\text{ASL_dbgmsl}\\ \text{ASL_rbgmsl}\end{matrix}\right\}$を利用する方法

$A_2 \leftarrow A$

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

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

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

$\left\{\begin{matrix}\text{ASL_dbgmsl}\\ \text{ASL_rbgmsl}\end{matrix}\right\}$で求められた解を改良する.

(2) $\left\{\begin{matrix}\text{ASL_dbgmlu}\\ \text{ASL_rbgmlu}\end{matrix}\right\}$と$\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$を利用する方法

$A_2 \leftarrow A$

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

$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A_2, \cdots);$

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

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

$\left\{\begin{matrix}\text{ASL_dbgmlu}\\ \text{ASL_rbgmlu}\end{matrix}\right\}$で A を三角分解し, $\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$でその解を求め, $\left\{\begin{matrix}\text{ASL_dbgmlx}\\ \text{ASL_rbgmlx}\end{matrix}\right\}$で解を改良する.

2.1.2 使用上の注意

(1) 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.

(2) 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,

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\} (A, \cdots, \boldsymbol{b}_1, \cdots);$$

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

とするか, または

$$\text{ierr} = \left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\} (A, \cdots);$$

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

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

のようにするとよい. 係数行列 A は $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ または $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ 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.

(3) 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は節約できる.

(4) 複素引数型と記載されている関数の入力データおよび出力データの配列の型は複素数型であるが, その他の関数ではすべて実数型である.

(5) スパース行列を係数行列に持つ連立 1 次方程式を解く際, 反復法を用いることもできるが, 以下の点を考慮して使い分けるとよい.

- スパース行列を係数にもつ連立 1 次方程式を解く場合, 直接法では係数行列の性質に依存せず, 有限回の演算により解が求まるのに対して, 反復法では, 係数行列の性質により速く解が収束する場合や逆に解が求まらない場合がある.
- 係数行列が正値対称である場合や対角優位である場合等, 一般に反復法の関数を用いた方が速く解が求まる.
- 一方反復法で解が求まらない場合でも, 直接法を用いれば解が求まることがある.
- なお, 係数行列が特異に近い場合には, どちらの方法を用いても解を精度良く求めることはできない.

- 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くる.

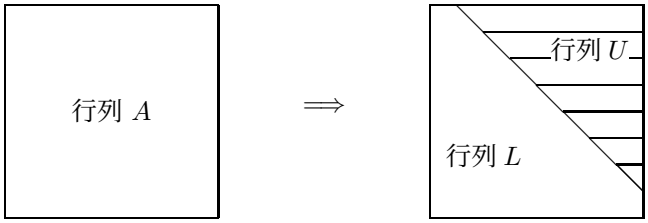
  n 次元の行列の場合, 前者は後者より約 n^2 回乗算が多い. 従って, 条件数を特に必要としない限り, 条件数を求めずに三角分解のみを行った方が実行時間は節約できる.

2.1.3 使用しているアルゴリズム

2.1.3.1 クラウト (Crout) 法

係数行列 A を下三角行列 L と単位上三角行列 U の積に分解する.

$A = LU$



本ライブラリでは, 部分軸選択を行うため, 実際は

$PA = LU$(P は行交換用置換行列)

となる.

$A = (a_{ij}), L = (l_{ij}), U = (u_{ij}) \quad (i, j = 1, 2, \cdots, N)$

とすると, アルゴリズムは以下のとおりである.

$l_{i1} \leftarrow a_{i1} \quad (i = 1, 2, \cdots, N)$

部 分 軸 選 択

$u_{1j} \leftarrow a_{1j}/l_{11} \quad (j = 1, 2, \cdots, N)$

for $k = 2, 3, \cdots, N$

$$l_{kk} \leftarrow a_{kk} - \sum_{m=1}^{k-1} l_{km} \cdot u_{mk}$$

for $i = k+1, k+2, \cdots, N$

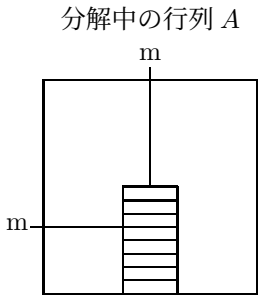
$$l_{ik} \leftarrow a_{ik} - \sum_{m=1}^{k-1} l_{im} \cdot u_{mk}$$

部 分 軸 選 択

for $j = k+1, k+2, \cdots, N$

$$u_{kj} \leftarrow (a_{kj} - \sum_{m=1}^{k-1} l_{km} \cdot u_{mj})/l_{kk}$$

ここで, 部分軸選択とは分解を安定に行うためにピボット (pivot) がその列中最大になるように行を交換する操作である. 第 m 段目 (上記アルゴリズムで k=m の時) の操作は次のとおりである.

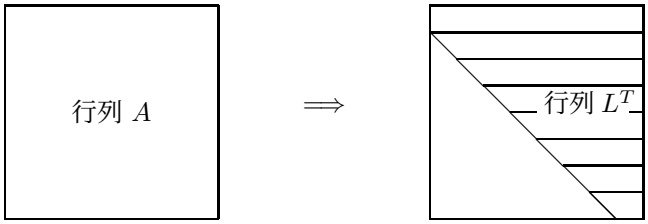


図中斜線の部分で絶対値最大の要素を選び, その要素の含まれている行と第 m 行とを交換する.

2.1.3.2 コレスキー (Cholesky) 法

係数行列 A を下三角行列 L と上三角行列 L^T の積に分解する.

$$A = LL^T$$



$$A = (a_{ij}), L = (l_{ij}), L^T = (l'_{ij}) \quad (i, j = 1, 2, \cdots, N)$$

とする. 行列 A の右上三角部分に列方向のコレスキー法を施すと, アルゴリズムは以下のようになる.

for $k = 1, 2, \cdots, N$

  for $i = 1, 2, \cdots, k-1$

    $l'_{ik} \leftarrow (a_{ik} - \sum_{m=1}^{i-1} l'_{mi} \cdot l'_{mk}) / l'_{ii}$

  $l'_{kk} \leftarrow \sqrt{a_{kk} - \sum_{m=1}^{k-1} {l'_{mk}}^2}$

行列演算は, 一般に内積型演算よりも外積型演算が適しており, さらにメモリアクセス回数を減少させるアンローリング技法を施すことにより演算効率が向上する (参考文献 (9) 参照).

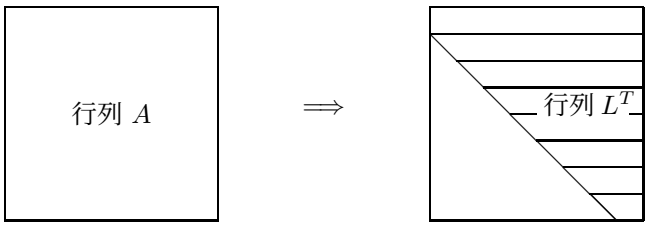
そのため 1 次元圧縮型行列の連立 1 次方程式において外積型演算のコレスキー法を採用した. さらにデータを行方向に格納することにより, データのアクセスが連続となるようにした.

2.1.3.3 修正コレスキー法

係数行列 A を下三角行列 L , 対角行列 D , および上三角行列 L^T の積に分解する.

$$A = LDL^T$$

対角行列 D は, 上三角行列 L^T の対角成分の逆数を成分とする.



$A = (a_{ij}), L = (l_{ij}), D = (d_{ij}), L^T = (l'_{ij}) \quad (i, j = 1, 2, \cdots, N)$

とすると, アルゴリズムは以下のとおりである.

$l'_{1j} \leftarrow a_{1j} \quad (j = 1, 2, 3, \cdots, N)$

for $k = 2, 3, \cdots, N$
  for $i = 1, 2, \cdots, k-1$
    $w_i \leftarrow l'_{ik}/l'_{ii}$

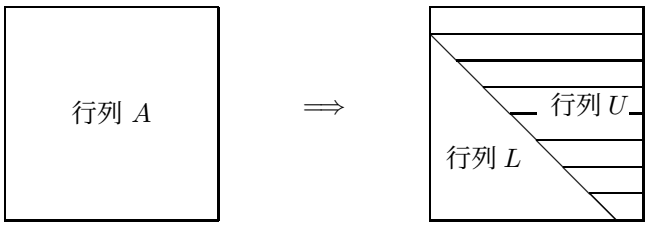
  for $j = k, k+1, \cdots, N$
    $l'_{kj} \leftarrow a_{kj} - \sum_{m=1}^{k-1} w_m \cdot l'_{mj}$

w はワークエリアであり, N 個の領域を必要とする.

2.1.3.4 ガウス (Gauss) 法

係数行列 A を単位下三角行列 L と上三角行列 U の積に分解する.

$A = LU$



本ライブラリでは部分軸選択を行うため, 実際は

$PA = LU$(P は行交換用置換行列)

となる.

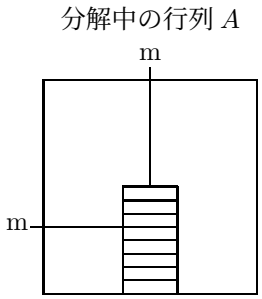
$A = (a_{ij}), L = (l_{ij}), U = (u_{ij}) \quad (i, j = 1, 2, \cdots, N)$

とすると, アルゴリズムは以下のとおりである.

for $k = 1, 2, \cdots, N$
  部 分 軸 選 択
  for $i = k+1, k+2, \cdots, N$
    $l_{ik} \leftarrow a_{ik}/u_{kk}$
    for $j = k+1, k+2, \cdots, N$
      $u_{ij} \leftarrow a_{ij} - l_{ik} \cdot u_{kj}$

ここで, 部分軸選択とは, 分解を安定に行うために, ピボット (pivot) がその列中最大になるように行を交換する操作である.

第 m 段目 (上記アルゴリズムで $k = m$ のとき) の操作は次のとおりである.



図中斜線の部分で絶対値最大の要素を選び, その要素の含まれている行と第 m 行との, 第 m 列目から第 N 列目までを交換する.

2.1.3.5 Levinson の方法

Toeplitz 行列 R

$$R = \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\ r_1 & r_0 & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_{-1} \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

を係数行列とする連立 1 次方程式

$$\sum_{j=1}^{n} r_{i-j} x_j = b_i \quad (i = 1, \cdots, n)$$

は次の様な n 個の連立 1 次方程式

$$\sum_{j=1}^{m} r_{i-j} x_j^{(m)} = b_i \quad (i = 1, \cdots, m;\ m = 1, 2, \cdots, n)$$

の解 $x_j^{(m)}$ $(j = 1, \cdots, m;\ m = 1, 2, \cdots, n)$ を考えることによって次の様に解くことができる.

(1) 初期解 $(m = 1)$

$$x_1^{(1)} = \frac{b_1}{r_0}$$

$$g_1^{(1)} = \frac{r_{-1}}{r_0}$$

$$h_1^{(1)} = \frac{r_1}{r_0}$$

(2) $m = 2, 3, \cdots, n$ について以下を逐次繰り返し計算する.

$$x^{(nu)} = \sum_{j=1}^{m-1} r_{m-j} x_j - b_m$$

$$x^{(de)} = \sum_{j=1}^{m-1} r_{m-j} g_{m-j}^{(m-1)} - r_0$$

$$x_m^{(m)} = \frac{x^{(nu)}}{x^{(de)}}$$

$$x_j^{(m)} = x_j^{(m-1)} - x_m^{(m)} g_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

$$g^{(nu)} = \sum_{j=1}^{m-1} r_{j-m} g_j^{(m-1)} - r_{-m}$$

$$g^{(de)} = \sum_{j=1}^{m-1} r_{j-m} h_{m-j}^{(m-1)} - r_0$$

$$h^{(nu)} = \sum_{j=1}^{m-1} r_{m-j} h_j^{(m-1)} - r_m$$

$$g_m^{(m)} = \frac{g^{(nu)}}{g^{(de)}}$$

$$h_m^{(m)} = \frac{h^{(nu)}}{x^{(de)}}$$

$$g_j^{(m)} = g_j^{(m-1)} - g_m^{(m)} h_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

$$h_j^{(m)} = h_j^{(m-1)} - h_m^{(m)} g_{m-j}^{(m-1)} \quad (j = 1, 2, \cdots, m-1)$$

求める解は $x_j = x_j^{(n)}$ として求められる. なお, 対称 Toeplitz 行列の場合には

$$r_i = r_{-i} \quad (i = 1, 2, \cdots, n)$$

であるので

$$g_j^{(m)} = h_j^{(m)} \quad (j = 1, 2, \cdots, m;\ m = 1, 2, \cdots, n)$$

が成立し, 一般の場合よりも効率良く計算を進めることができる. なお, この方法は行列の性質を十分活用しているので一般のガウス消去法よりもメモリ使用量や計算効率の点で優れているが, 反面, 行列が正則であっても原理的に解を求められない場合がある. 例えば, $r_0 = 0$ の場合には明らかにこの方法で解を得ることはできない.

2.1.3.6 Vandermonde 行列

相異なる n 個の要素 v_k $(k = 1, 2, \cdots, n)$ で構成される n 次の Vandermonde 行列 V

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & \cdots & v_1^{n-2} & v_1^{n-1} \\ 1 & v_2 & v_2^2 & \cdots & v_2^{n-2} & v_2^{n-1} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 1 & v_{n-1} & v_{n-1}^2 & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\ 1 & v_n & v_n^2 & \cdots & v_n^{n-2} & v_n^{n-1} \end{bmatrix}$$

を係数行列とする連立 1 次方程式 $V\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} v_i^{j-1} x_j = b_i \quad (i = 1, \cdots, n)$$

を解くことを考える. いま, $n-1$ 次の多項式 $P_i^{(n)}(x)$ を

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

と定義すると, $P_i^{(n)}(v_k) = \delta_{ik}$ (ここで, δ_{ik} はクロネッカーのデルタ) が成立するので, この多項式の x^{j-1} の項の係数からなる行列を $U = \{u_{i,j}\}$ とすると, $UV^T = E(E$ は単位行列) すなわち $V^{-1} = U^T$ が成立する. したがって, 与えられた連立 1 次方程式 $V\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ は

$$\boldsymbol{x} = U^T\boldsymbol{b}$$

を計算することによって得られる. いま, U の各係数を計算するために次式で定義されるマスター多項式 $P^{(n)}(x)$ を考える.

$$P^{(n)}(x) = \prod_{k=1}^{n}(x-v_k)$$

マスター多項式 $P^{(n)}(x)$ の x^{j-1} の項の係数を $w_{n-j+1}^{(n)}$ とおいて

$$P^{(n)}(x) = x^n + w_1^{(n)}x^{n-1} + \cdots + w_{n-1}^{(n)}x + w_n^{(n)}$$

と表す. $P^{(i)}(x) = (x-v_i)P^{(i-1)}(x)$ $(i = 2, 3, \cdots, n)$ の関係から, x^{j-1} についての係数を比較することによって, 次の関係が得られる.

$$w_1^{(i)} = w_1^{(i-1)} - v_i \quad (i = 2, \cdots, n)$$

$$w_j^{(i)} = w_j^{(i-1)} - v_i w_{j-1}^{(i-1)} \quad (j = i, i-1, \cdots, 2;\ i = 2, 3, \cdots, n)$$

ただし,

$$w_1^{(1)} = -v_1$$

$$w_j^{(j-1)} = 0 \quad (j = 2, 3, \cdots, n)$$

としている. 以上から, マスター多項式の各係数を計算できる. 一方,

$$\frac{dP^{(n)}(x)}{dx}|_{x=v_i} = \prod_{\substack{k=1 \\ (k\neq i)}}^{n}(v_i - v_k)$$

が成立するがこの値は,

$$\frac{dP^{(n)}(x)}{dx}|_{x=v_i} = n{v_i}^{n-1} + (n-1)w_1^{(n)}{v_i}^{n-2} + \cdots + w_{n-1}^{(n)}$$

から計算できる. また,

$$P_i^{(n)}(x) = \frac{P^{(n)}(x)}{(x-v_i)\frac{dP^{(n)}(x)}{dx}|_{x=v_i}}$$

であるので, この多項式の x^{j-1} の項の係数 $u_{i,j}$ は $\dfrac{P^{(n)}(x)}{(x-v_i)}$ の x^{j-1} の項の係数を組み立て除法を用いて計算することによって得ることができる. なお, Vandermonde 行列を係数とする連立 1 次方程式は本質的に悪条件であり, n が極めて小さい場合以外は解を精度良く求めることは難しい.

2.1.3.7 サイクリック・リダクション法

(1) サイクリック・リダクション法

実 3 重対角行列 A を係数行列とする連立 1 次方程式

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.1}$$

を解く. ここで,

$$A = \begin{bmatrix} d_1 & u_1 & & & 0 \\ \ell_2 & d_2 & u_2 & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & u_{n-1} \\ & 0 & & \ell_n & d_n \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

とすると

$$\ell_i x_{i-1} + d_i x_i + u_i x_{i+1} = b_i \tag{2.2}$$

となる.

このアルゴリズムでは, 次数が半分の連立 1 次方程式を作り出すリダクション操作を $\lfloor \log_2(n) \rfloor$ 回繰り返すことで最終的に次数 1 の方程式を作り出せる. よって解が 1 つ求まる.

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

$$\boldsymbol{x} = \boldsymbol{b}/\boldsymbol{d} \tag{2.3}$$

この解をもとに後退代入を繰り返せばすべての解が得られる. ただし, 記号 $\lfloor x \rfloor$ は x を超えない最大の整数を表す.

次に, サイクリック・リダクション法のリダクション操作及び後退代入について述べる.

(a) リダクション操作

まず, はじめに $n = 2^m - 1$ とする.

(2.1) の偶数行を中心に前後の行との 3 行から x_{i-1} と x_{i+1} を消去する. すなわち,

$$\begin{cases} \ell_{i-1}x_{i-2} + d_{i-1}x_{i-1} + u_{i-1}x_i & = b_{i-1} \\ \ell_i x_{i-1} + d_i x_i + u_i x_{i+1} & = b_i \\ \ell_{i+1}x_i + d_{i+1}x_{i+1} + u_{i+1}x_{i+2} & = b_{i+1} \end{cases} , i \text{ は偶数}$$

の 3 行から次式を得る.

$$\ell'_i x_{i-2} + d'_i x_i + u'_i x_{i+2} = b'_i \tag{2.4}$$

$$\begin{cases} \ell'_i &= d_{i+1}\ell_{i-1}\ell_i \\ u'_i &= d_{i-1}u_i u_{i+1} \\ d'_i &= \ell_i d_{i+1} u_{i-1} + \ell_{i+1} d_{i-1} u_i - d_{i-1} d_i d_{i+1} \\ b'_i &= \ell_i d_{i+1} b_{i-1} + d_{i-1} u_i b_{i+1} - d_{i-1} d_{i+1} b_i \end{cases}$$

(2.1) に含まれるすべての偶数行について (2.4) を適用すると $(x_0 = x_{n+1} = 0)$, $\lfloor n/2 \rfloor$ の実 3 重対角行列を係数行列とする連立 1 次方程式が得られる.

次に, $n=2^m$ を考える. $n=2^m-1$ の場合はすべての偶数行において (2.4) が適用できるが, $n=2^m$ の場合は $n-1$ 行が奇数行になるので $n-1$ 行と n 行については (2.4) が適用できない. そこで,

$$\begin{cases} \ell_{n-1}x_{n-2} + d_{n-1}x_{n-1} + u_{n-1}x_n = b_{n-1} \\ \ell_n x_{n-1} + d_n x_n = b_n \end{cases}$$

の 2 行から x_{n-1} を消去した次式を適用する.

$$\ell'_n x_{n-2} + d'_n x_n = b'_n \tag{2.5}$$

$$\begin{cases} \ell'_n = \ell_{n-1}\ell_n \\ d'_n = \ell_n u_{n-1} - d_n d_{n-1} \\ b'_n = \ell_n b_{n-1} - d_{n-1} d_n \end{cases}$$

したがって, n の値によらず $\lfloor n/2 \rfloor$ の実 3 重対角行列を係数行列とする連立 1 次方程式に縮小することができる.

(b) 後退代入

リダクション操作によって求まった解 (2.3) をもとにリダクション操作によってできた各々の連立 1 次方程式からリダクション操作とは逆方向に解を求めていく. もし, 偶数行の解が求まっていれば奇数行の解は,

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

より求められる.

(2) サイクリック・リダクション法の高速化について

サイクリック・リダクション法は, ガウス (Gauss) 法のような漸化式になっていないので計算式に独立性があり, 本質的にベクトル化が可能となるが, さらに以下のようなベクトル化をほどこしている.

(a) 定係数型サイクリック・リダクション法の高速化について

ディリクレ境界値問題やノイマン境界値問題を離散化するときに現れる係数行列に対しては, サイクリック・リダクション法を変形した定係数型サイクリック・リダクション法を採用することにより, さらに高速化が可能となった. 以下に, 定係数型サイクリック・リダクション法について述べる.

第一に, 係数行列が

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & s & d \end{bmatrix}, \quad d \neq 0, \ \ s \neq 0 \tag{2.6}$$

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & 2\cdot s & d \end{bmatrix}, \quad d \neq 0, \ \ s \neq 0 \tag{2.7}$$

の場合, (2.7) の最終行を 2 で正規化した行列と (2.6) と比較すると, 最終行だけ異なり他行は同じであることがわかる. そこで, (2.6) と (2.7) を次の (2.8) に置き換えて考える.

$$\begin{bmatrix} d & s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & d & s & \\ 0 & & & s & e \end{bmatrix}, \quad d \neq 0, \ \ s \neq 0, \ \ e \neq 0 \tag{2.8}$$

まず, はじめに $n = 2^m - 1$ とする.

(2.7) の偶数行を中心に前後の行との 3 行から x_{i-1} と x_{i+1} を消去する. すなわち,

$$\begin{cases} sx_{i-2} + dx_{i-1} + sx_i = b_{i-1} \\ sx_{i-1} + dx_i + sx_{i+1} = b_i \\ sx_i + dx_{i+1} + sx_{i+2} = b_{i+1} \end{cases} \quad , i \text{ は偶数}$$

の 3 行から次式を得る.

$$s'x_{i-2} + d'x_i + s'x_{i+2} = b'_i \tag{2.9}$$

$$\begin{cases} s' = s^2 \\ d' = 2 \cdot s^2 - d^2 \\ b' = s(b_{i-1} + b_{i+1}) - db_i \end{cases}$$

(2.8) に含まれる偶数行について (2.9) を適用すると $(x_0 = x_{n+1} = 0)$, $\lfloor n/2 \rfloor$ の実 3 重対角行列を係数行列とする連立 1 次方程式が得られる. ただし, $n - 1$ 行については

$$s'x_{n-3} + e'x_{n-1} = b'_{n-1} \tag{2.10}$$

$$\begin{cases} s' = e \cdot s^2 \\ e' = e \cdot s^2 - e \cdot d^2 + d \cdot s^2 \\ b'_{n-1} = e \cdot s \cdot b_{n-2} - e \cdot d \cdot b_{n-1} + d \cdot s \cdot b_n \end{cases}$$

となる.

次に, $n = 2^m$ を考える. $n = 2^m$ の場合は $n - 1$ 行が奇数行になるので (2.10) の代わりに

$$\begin{cases} sx_{n-2} + dx_{n-1} + sx_n = b_{n-1} \\ sx_{n-1} + ex_n = b_n \end{cases}$$

の 2 行から x_{n-1} を消去した次式を適用する.

$$s'x_{n-2} + e'x_n = b_{n-1} \tag{2.11}$$

$$\begin{cases} s' = s^2 \\ d' = s^2 - d \cdot e \\ b'_{n-1} = s \cdot b_{n-1} - d \cdot b_n \end{cases}$$

したがって, n の値によらず $\lfloor n/2 \rfloor$ の実 3 重対角行列を係数行列とする連立 1 次方程式に縮小することができる. この操作を $\lfloor \log_2(n) \rfloor$ 回繰り返すことで最終的に次数 1 の方程式を作りだせる. よって解が一つ求まる.

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

この解をもとに後退代入を繰り返すことですべての解を求める. もし偶数行の解が求まっていれば奇数行の解は,

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

より求められる.

第二に, 係数行列が

$$\begin{bmatrix} d & 2 \cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & s & d \end{bmatrix} \quad , \quad d \neq 0, \ \ s \neq 0 \tag{2.12}$$

$$\begin{bmatrix} d & 2\cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & s \\ 0 & & & 2\cdot s & d \end{bmatrix} \quad , \quad d \neq 0, \quad s \neq 0 \tag{2.13}$$

の場合, (2.13) の最終行を 2 で正規化した行列と (2.12) を比較すると, 最終行だけ異なり他行は同じであることがわかる. そこで, (2.12) と (2.13) を次の (2.14) に置き換えて考える.

$$\begin{bmatrix} d & 2\cdot s & & & 0 \\ s & d & s & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & d & s \\ 0 & & & s & e \end{bmatrix} \quad , \quad d \neq 0, \quad s \neq 0, \quad e \neq 0 \tag{2.14}$$

今度は偶数行ではなく奇数行を中心に考える.

はじめに第 1 行と第 2 行から x_2 を消去する. すなわち,

$$\begin{cases} dx_1 + 2\cdot sx_2 = b_1 \\ sx_1 + dx_2 + sx_3 = b_2 \end{cases}$$

の 2 行から次式を得る.

$$(d^2 - 2\cdot s^2)x_1 - 2\cdot s^2 x_3 = db_1 - 2sb_2 \tag{2.15}$$

次に, (2.14) の第 $2i$ 行, 第 $2i+1$ 行, 第 $2i+2$ 行の 3 行から x_{2i}, x_{2i+2} を消去する.

すなわち,

$$\begin{cases} sx_{2i-1} + dx_{2i} + sx_{2i+1} = b_{2i} \\ sx_{2i} + dx_{2i+1} + sx_{2i+2} = b_{2i+1} \\ sd_{2i+1} + sx_{2i+2} + sx_{2i+3} = b_{2i+2} \end{cases}$$

の 3 行から次式を得る.

$$s'x_{2i-1} + d'x_{2i+1} + s'x_{2i+3} = b'_{2i+1} \tag{2.16}$$

$$\begin{cases} s' = s^2 \\ d' = 2\cdot s^2 - d^2 \\ b'_{2i+1} = s\cdot b_{2i} - d\cdot b_{2i+1} + s\cdot b_{2i+2} \end{cases}$$

これを $i = 1, 2, \cdots, m$ (ただし, m は $2i+1 \leq n-2$ を満たす最大の i) の各々について行う.

最後に, $n = 2^m$ の場合には $n-2$ 行, $n-1$ 行, n 行の 3 行から x_{n-2}, x_n を消去して (2.17) を得る. また, $n = 2^m - 1$ の場合には $n-1$ 行が偶数行になるので $n-1$ 行, n 行の 2 行から x_{n-1} を消去して (2.18) を得る. すなわち, $n = 2^m$ のとき

$$\begin{cases} sx_{n-3} + dx_{n-2} + sx_{n-1} = b_{n-2} \\ sx_{n-2} + dx_{n-1} + sx_n = b_{n-1} \\ sx_{n-1} + ex_n = b_n \end{cases}$$

の 3 行から次式を得る.

$$s'x_{n-3} + e'x_{n-1} = b'_{n-1} \tag{2.17}$$

$$\begin{cases} s' = e\cdot s^2 \\ e' = e\cdot s^2 - e\cdot d^2 + d\cdot s^2 \\ b'_{n-1} = s\cdot e\cdot b_{n-2} - d\cdot e\cdot b_{n-1} + d\cdot s\cdot b_n \end{cases}$$

$n = 2^m - 1$ のとき

$$\begin{cases} sx_{n-2} + dx_{n-1} + sx_n = b_{n-1} \\ sx_{n-1} + ex_n = b_n \end{cases}$$

の 2 行から次式を得る.

$$s'x_{n-2} + e'x_n = b'_n \tag{2.18}$$

$$\begin{cases} s' = s^2 \\ e' = s^2 - e \cdot d \\ b'_n = s \cdot b_{n-1} - d \cdot b_n \end{cases}$$

したがって, n の値によらず $\lfloor (n-1)/2 \rfloor + 1$ の実 3 重対角行列を係数行列とする連立 1 次方程式に縮小することができる. この操作を $\lfloor \log_2(n-1) \rfloor$ 回繰り返すことで最終的に

$$\begin{bmatrix} d^{(m)} & 2 \\ 1 & e^{(m)} \end{bmatrix}, \quad m = \lfloor (n-1)/2 \rfloor + 1$$

を係数行列とする方程式が得られる. これを解いて後退代入を繰り返せばすべての解が求められる.

(b) リダクション操作の打ち切り

リダクション操作を繰り返していくと, ある仮定 (十分条件にはなるが必要ではない) のもとで対角要素の値が大きくなり, リダクション操作の途中の段階で対角要素と副対角要素との比が $1/E_{PS}$ (E_{PS}：誤差判定のための単位) より大きくなることがある.

上記仮定の一つとして

$$| l_i^{(k)} |, | u_i^{(k)} | < | d_i^{(k)}/2 |, \quad 1 \le i \le n \tag{2.19}$$

を考える. ここで, $l_i^{(k)}$, $d_i^{(k)}$, $u_i^{(k)}$ は k 番目のリダクション操作後の係数行列の第 i 行における下副対角成分, 対角成分, 上副対角成分とする. これらの仮定が保たれると係数行列を

$$(\cdots, {l_i}^{(k)}/{d_i}^{(k)}, 1, {u_i}^{(k)}/{d_i}^{(k)}, \cdots) \tag{2.20}$$

に正規化する. そうすると, 副対角要素は E_{PS} 程度まで小さくなることがあり, リダクション操作の途中で定数ベクトル $\boldsymbol{b}^{(k)}$ (k：リダクション回数) がいくつかの解に収束していく.

したがって, リダクション操作を行う前にあらかじめ収束時のリダクション回数がわかっていれば最後までリダクション操作を行う必要はなく, リダクション操作を途中でやめて後退代入に移れば計算時間が短縮されることになり効率がよい. これをサイクリック・リダクション操作の打ち切りという.

以下に打ち切りまでのリダクション回数を求めるために (2.19) を満足する場合の収束の下限を調べる.

まず, $\boldsymbol{e} = \max_i({l_i}^{(k)}/{d_i}^{(k)}, {u_i}^{(k)}/{d_i}^{(k)})$ を求め (2.20) のすべての ${l_i}^{(k)}/{d_i}^{(k)}, {u_i}^{(k)}/{d_i}^{(k)}$ を $\boldsymbol{e}$ に置き換えた行列 $(\cdots, \boldsymbol{e}, 1, \boldsymbol{e}, \cdots)$ を考える. これは, $(\cdots, 1, \boldsymbol{d}, 1, \cdots)$ のような係数行列を考えれば十分だろう. 収束の割合を決定するために

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

と定義する. $\boldsymbol{d}^{(k)}$ は k 番目の反復中に計算された対角要素である. ここで, $| \boldsymbol{d}^{(k)} |$ が k の関数として $1/E_{PS}$ の方向へ大きくなっていくか測定してみる. (2.9) から

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

の絶対値をとると,

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

となるので

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

となる.

このことから,

- $\varepsilon^{(k)} < 1$ のとき,
  $\varepsilon^{(k+1)} > 4\varepsilon^{(k)}$ となり, 増大の割合が少なくとも 1 次の速さ
- $\varepsilon^{(k)} > 1$ のとき,
  $\varepsilon^{(k+1)} > \left[\varepsilon^{(k)}\right]^2$ となり, 増大の割合が少なくとも 2 次の速さ

であることがわかる.

したがって, (2.21) から求められる $\varepsilon^{(k)}$ が

$$\varepsilon^{(k)} \geq 1/E_{PS}$$

となる最小の整数 k を打ち切りまでのリダクション回数とする. なお, $k \geq \lfloor \log_2(n) \rfloor$ のときは打ち切りは起こらない.

(3) **補足事項**

- 計算時間への影響

  条件 (2.19) を満たさない (対角要素が強くない) 実 3 重対角行列の連立 1 次方程式においては, 計算の過程で特異性を持つか否かの判定を行う. そのため, 対角要素が強い係数行列を持つ場合に比べ計算時間がかかる.

2.1.3.8 逆行列の算出方法

行列 A の逆行列を算出するには, その三角分解を利用して行う.

$A = LU$ と分解された場合, まず第一段階として L^{-1} または U^{-1} を掃出し法によって求め, 次に第二段階としてその結果を変形して $A^{-1} = U^{-1}L^{-1}$ を計算する.

第二段階の操作は, 三角分解時の変換行列を L^{-1} または, U^{-1} の反対側から施したものとなる.

たとえば, コレスキー法の場合,

$$L^{-1}A = L^T$$

として L^T を求めているので, A^{-1} は $(L^T)^{-1}$ に分解時の変換行列 L^{-1} を右側から掛けることによって求められる.

第一段階で L^{-1} と U^{-1} のどちらを計算するかは, 三角分解の手法によって異なる.

2.1.3.9 行列式の値の算出方法

行列式の値は, 以下のように求められる.

$A = LU$ と分解されたとき,

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

ただし, $L = (l_{ij})$, $U = (u_{ij})$ とする.

2.1.3.10　解の改良

連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解を改良することを考える. 最初求められた解を $\boldsymbol{x}^{(1)}$ とすると, 計算誤差のため,

$$A\boldsymbol{x}^{(1)} \neq \boldsymbol{b}$$

である. そこでこの $\boldsymbol{x}^{(1)}$ を改良するため, 以下のアルゴリズムを使用する.

(1) $\boldsymbol{r}^{(k)} = \boldsymbol{b} - A\boldsymbol{x}^{(k)}$

(2) $A\boldsymbol{y}^{(k)} = \boldsymbol{r}^{(k)}$

(3) $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)} \quad (k = 1, 2, \cdots)$

この反復において, (2) で丸め誤差が発生する. 従って (2) の式は, 正確には

$$(A + E)\boldsymbol{y}^{(k)} = \boldsymbol{r}^{(k)}$$

となる. この式と (1), (3) より,

$$\boldsymbol{x}^{(k+1)} - \boldsymbol{x} = [I - (A + E)^{-1}A]^k(\boldsymbol{x}^{(1)} - \boldsymbol{x})$$

$$\boldsymbol{r}^{(k+1)} = [I - A(A + E)^{-1}]\boldsymbol{r}^{(k)}$$

が導かれる.

従って $\|E\|\ \|A^{-1}\| < \dfrac{1}{2}$ を満たせば,

$$\begin{matrix} \boldsymbol{x}^{(k+1)} \to \boldsymbol{x} \\ \boldsymbol{r}^{(k+1)} \to 0 \end{matrix} \quad (k \to \infty)$$

となる.

なお, $\dfrac{\|\boldsymbol{y}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k+1)}\|_\infty} > \dfrac{1}{2} \cdot \dfrac{\|\boldsymbol{y}^{(k-1)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$ となった場合, 解は収束しない. 詳細は参考文献 (7) を参照されたい.

2.1.3.11　近似解の精度推定

近似解 $\boldsymbol{x}^{(k)}$ について考える.

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

である. ところで解の相対誤差 $\dfrac{\|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$ は, 解が十分収束し, 行列 A の条件のよい場合は $\dfrac{\|\boldsymbol{y}^{(k)}\|_\infty}{\|\boldsymbol{x}^{(k)}\|_\infty}$ で置きかえられる.

2.1.3.12　条件数

(1) **条件数とその使用方法**

行列 A の条件数 $\kappa(A)$ とは, 連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く場合, その係数行列 A または定数ベクトル $\boldsymbol{b}$ に含まれる誤差が解に与える影響の程度を示す数値であり, 次の式で与えられる.

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

係数行列 A に誤差 E が含まれるとき, 求められた解 $\boldsymbol{y}$ の真の解 $\boldsymbol{x}$ に対する相対誤差は, 次の範囲にある.

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

ただし, $\varepsilon = \dfrac{\|E\|}{\|A\|}$ である.

また, 定数ベクトル $\boldsymbol{b}$ に誤差 $\boldsymbol{e}$ が含まれるとき, 相対誤差は次の範囲にある.

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

ただし, $\varepsilon = \dfrac{\|\boldsymbol{e}\|}{\|\boldsymbol{b}\|}$ とする.

従って, 条件数が 10^{α} のオーダーのとき求められた解の精度は, 元のデータの精度よりも約 α ケタ減少する可能性がある.

本ライブラリでは, 条件数の逆数を求め変数 cond に格納している. 利用者はこの cond の値が著しく小さいような係数行列をもつ連立 1 次方程式については, 解が求められたとしても精度は悪くなっていることに注意しなければならない. 特に以下の判別式が成り立つときは, その行列は計算機上特異であり解は信用できない.

(特異な行列の判別式) :

$1.0 + \mathrm{cond} \simeq 1.0$

(2) **条件数の算出方法**

条件数 $k(A)$ は,

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

であるが, 本ライブラリでは A^{-1} は求めずに $\|A^{-1}\|$ の概算を行い, それを $\|A\|$ に掛け合わせる方法を採用している.

A の特異値分解を

$$A = U\Sigma V^{T}$$

U, V : 直交行列

$$\Sigma = \begin{bmatrix} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \ddots & \\ & & & \ddots \\ 0 & & & \sigma_n \end{bmatrix}$$

σ_i : 特異値

$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$

とする. 今, $A\boldsymbol{x} = \boldsymbol{y}$ という方程式系を考える. $\boldsymbol{y}$ を U の列ベクトルを基底として

$$\boldsymbol{y} = \|\boldsymbol{y}\|\sum_{i=1}^{n}\alpha_i\boldsymbol{u}_i \quad (\sum_i {\alpha_i}^2 = 1)$$

と表すと,

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

が成り立つ. 右辺は, α_n が特に小さくない限り, $\boldsymbol{y}$ がどのようなベクトルでも $\sigma_n^{-1}(=\|A^{-1}\|)$ 程度の大きさをもつ.

本ライブラリにおいては, この近似値がさらによいものになるように, $\boldsymbol{y}$ を選定している.

上記の不等式で等号が成り立つのは, $\boldsymbol{y} = \boldsymbol{u}_n(\alpha_n = 1, \alpha_i = 0, i = 1, 2, \cdots, n-1)$ のときである. よって, $\boldsymbol{y}$ が $\boldsymbol{u}_n$ を主要な要素としてもつように $\boldsymbol{y}$ を決定すればよい. 実際には,

$$\boldsymbol{z} = \begin{bmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{bmatrix}$$

として, $A^T\boldsymbol{y} = \boldsymbol{z}$ で $\dfrac{\|\boldsymbol{y}\|}{\|\boldsymbol{z}\|}$ が最大になるように $\boldsymbol{z}$ の各要素の符号を定め, $\boldsymbol{y}$ を求める.

この $\boldsymbol{y}$ を使用して $A\boldsymbol{x} = \boldsymbol{y}$ を解き, $\dfrac{\|\boldsymbol{x}\|}{\|\boldsymbol{y}\|}$ を $\|A^{-1}\|$ の近似値とする.
条件数を求める具体的な手順を以下に示す.

(a) $\|A\|$ を求める.

(b) $A = LU$ と三角分解する.

(c) $U^T\boldsymbol{w} = \boldsymbol{z}$ で $\dfrac{\|\boldsymbol{w}\|}{\|\boldsymbol{z}\|}$ が最大になるように $\boldsymbol{z}$ を決定し, $\boldsymbol{w}$ を求める.

(d) $L^T\boldsymbol{y} = \boldsymbol{w}$ を解き, $\boldsymbol{y}$ を求める.

(e) $LU\boldsymbol{x} = \boldsymbol{y}$ を解き, $\boldsymbol{x}$ を求める.

(f) $\dfrac{\|\boldsymbol{y}\|}{\|\boldsymbol{x}\| \cdot \|A\|}$ (条件数の逆数) を求め, 引数 cond に値を格納する.

詳細は参考文献 (3) を参照されたい.

2.1.4 参考文献

(1) Wilkinson, J. H. and Reinsch, C. , “Handbook for Automatic Computation, vol II, Linear Algebra”, Springer-Verlag, (1971).

(2) Dahlquist, G. and Björck, Å. , “Numerical Methods”, translated by N. Anderson, Prentice-Hall, Inc. , (1974).

(3) Cline, A. K. , Moler, C. B. , Stewart, G. W. and Wilkinson, J. H. , “An estimate for the condition number of a matrix”, SIAM Numerical Analysis Vol. 16, pp. 368-375, (1979).

(4) Dongarra, J. J. , Moler, C. B. , Bunch, J. R. and Stewart, G. W. , “LINPACK Users’ Guide”, SIAM, (1979).

(5) Forsythe, G. E. and Moler, C. B. , “線形計算の基礎” 渋谷政昭 他訳, 培風館.

(6) 戸川隼人, “マトリクスの数値計算”, オーム社, (1971).

(7) Wilkinson, J. H. , “丸め誤差解析”, 一松信 他訳, 培風館.

(8) Robert, Y. and Sguazzero, P. “The LU decomposition algorithm and its efficient FORTRAN implementation on IBM3090 Vector Multiprocessor”, IBM Tech. Rep. , ICE-0006(1987).

(9) 高橋幸夫, 怡土好夫, 坂井日出雄, 花村光泰, 萬淳一, 津和義昭, “スーパーコンピュータ SX システムに適した高速化技法”, 情報処理学会第 32 回全国大会講演集, (1986).

(10) STONE, HAROLD. S. , “Parallel Tridiagonal Equation Solvers”, ACM Transactions on Mathematical Software, Vol. 1, No. 4, 289, (1975).

(11) Hockney, R. W. and Jesshope, C. R. , “並列計算機”

2.2 実行列 (2 次元配列型)

2.2.1 ASL_dbgmsm, ASL_rbgmsm
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_dbgmsm (ab, lna, n, m, ipvt);

単精度関数:

ierr = ASL_rbgmsm (ab, lna, n, m, ipvt);

(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	ab	{ D∗ / R∗ }	内容参照	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) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **大きさ:** (lna×(n + 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 (a) 参照)
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) $0 < \mathrm{m}$

(5) **エラーインディケータ (戻り値)**

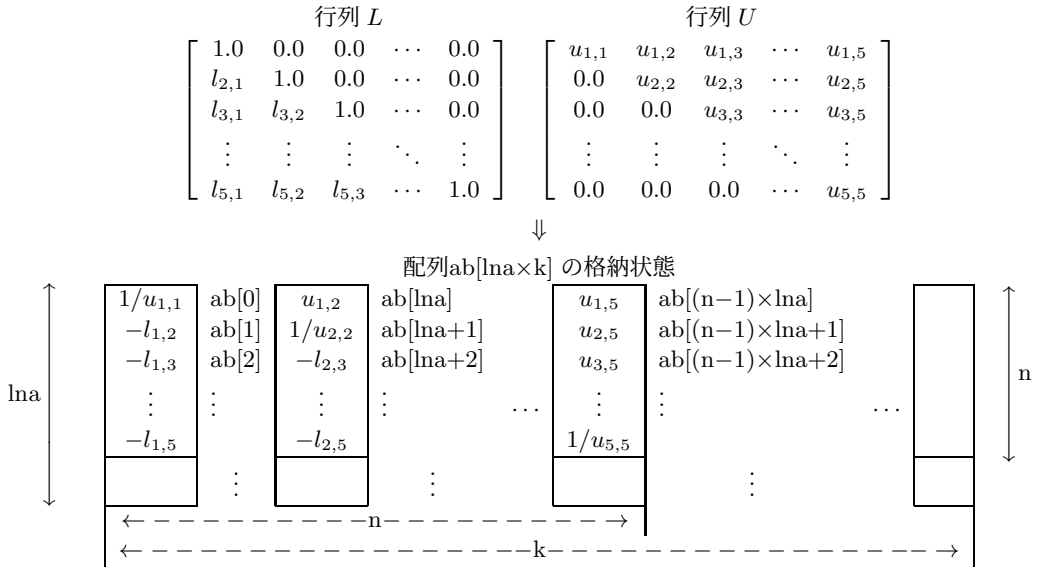
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	ab[lna∗(n+i−1)] ← ab[lna∗(n+i−1)]/ab[0] $(i = 1, 2, \cdots, \mathrm{m})$ とする.
2100	係数行列 A の LU 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3010	制限条件 (b) を満足しなかった.	
$4000 + i$	係数行列 A の LU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

(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

(c) 主プログラム

```
/*      C interface example for ASL_dbgmsm */

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

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

    printf( "    *** ASL_dbgmsm ***\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 * )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_dbgmsm(ab, lna, n, m, ipvt);

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

    free( ab );
    free( ipvt );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbgmsm ***

 ** Input **

 n =      4 m =      2

Coefficient Matrix

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

Constant Vectors

      36        11
      15         0
      22         7
      -6         4

 ** Output **

ierr =      0

Solution

       1         1
       2         1
       4         1
       5         1
```

2.2.2 ASL_dbgmsl, ASL_rbgmsl
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 $A\boldsymbol{x} = \boldsymbol{b}$ を, ガウス法またはクラウト法を用いて解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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∗ / R∗	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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, ピボットが 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 2.2.1 $\left\{\begin{matrix}\text{ASL_dbgmsm}\\ \text{ASL_rbgmsm}\end{matrix}\right\}$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, 2.2.1 $\left\{\begin{matrix}\text{ASL_dbgmsm}\\ \text{ASL_rbgmsm}\end{matrix}\right\}$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 2.2.5 $\left\{\begin{matrix}\text{ASL_dbgmls}\\ \text{ASL_rbgmls}\end{matrix}\right\}$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 ($\text{i} \leq \text{j}$) 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.

図 2−2　行列 L と行列 U の格納状態

行列 L

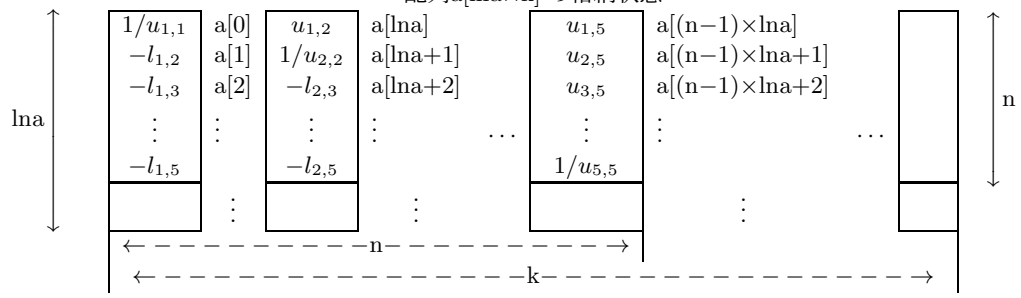
$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & \cdots & 0.0 \\ l_{2,1} & 1.0 & 0.0 & \cdots & 0.0 \\ l_{3,1} & l_{3,2} & 1.0 & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{5,1} & l_{5,2} & l_{5,3} & \cdots & 1.0 \end{bmatrix}$$

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a[lna×k] の格納状態



備　考

a.　$\text{lna} \geq \text{n}$, $\text{n} \leq \text{k}$ 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***b***

(c) 主プログラム

```
/*      C interface example for ASL_dbgmsl */

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

```
#include <asl.h>

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

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

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

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

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

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

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

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

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

    fclose( fp );

    ierr = ASL_dbgmsl(a, lna, n, b, ipvt);

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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_dbgmsl ***

 ** Input **

 n =      4

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
```

2.2.3 ASL_dbgmlu, ASL_rbgmlu
実行列の LU 分解

(1) **機 能**

実行列 A(2 次元配列型) をガウス法またはクラウト法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbgmlu  (a, lna, n, ipvt);
```

単精度関数:

```
ierr = ASL_rbgmlu  (a, lna, n, ipvt);
```

(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	実行列 A(2 次元配列型)
				出 力	$A = LU$ と分解され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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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, ピボットが 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 配列 a には, 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 (2.2.2 図 2−2 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.

2.2.4 ASL_dbgmlc, ASL_rbgmlc
実行列の LU 分解と条件数

(1) **機 能**

実行列 A(2 次元配列型) をガウス法またはクラウト法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	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∗ / R∗}	1	出 力	条件数の逆数
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)

(4) **制限条件**

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

(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, ピボットが 0.0 となった. 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 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 (2.2.2 図 2−2 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のはその概算値である.

2.2.5 ASL_dbgmls, ASL_rbgmls
連立 1 次方程式 (LU 分解後の実行列)

(1) **機 能**

ガウス法またはクラウト法で LU 分解された実行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	LU 分解後の係数行列 A(実行列, 2 次元配列型) (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	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解ベクトル$\boldsymbol{x}$
5	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある. 通常は関数 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ を使用する. また, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる. 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 2.2.6 $\left\{\begin{matrix}\text{ASL\_dbgmms}\\ \text{ASL\_rbgmms}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 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 が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい. また, U の対角成分はその逆数が格納されていなければならない (2.2.2 図 2−2 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LU 分解を行う 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ によって与えられる.

2.2.6 ASL_dbgmms, ASL_rbgmms
多重右辺連立 1 次方程式 (LU 分解後の実行列)

(1) **機 能**

LU 分解された実行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く. 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_dbgmms (a, lna, n, b, lnb, m, ipvt);

単精度関数:

ierr = ASL_rbgmms (a, lna, n, b, lnb, m, ipvt);

(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	LU 分解後の係数行列 A(実行列, 2 次元配列型) (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	b	{D∗ / R∗}	lnb×m	入 力	定数ベクトル$\boldsymbol{b_i}$からなる行列 $[\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	解ベクトル$\boldsymbol{x_i}$からなる行列 $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	入 力	配列 b の整合寸法
6	m	I	1	入 力	右辺ベクトルの数
7	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1]：LU 分解の 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(c) $0 < \mathrm{ipvt}[i-1] \leq \mathrm{n} \quad (i = 1, \ldots, \mathrm{n})$

(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[lna ∗ (i − 1)] ← b[lna ∗ (i − 1)]/a[0] ($i = 1, 2, \cdots,$ m) 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.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある. 通常は関数 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ を使用する. また, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる.

(b) 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 が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい. また, U の対角成分はその逆数が格納されていなければならない.

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LU 分解を行う 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$,2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ によって与えられる.

(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, lna=10, n=4, 定数ベクトル$\boldsymbol{b}$からなる行列 B, lnb=10, m=2

(c) 主プログラム

```
/* C interface example for ASL_dbgmms */

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

int main()
{
    double *a;
    int lna=11;
    int n=4;
    double *b;
    int lnb=11;
    int m=2;
    int *ipvt;
    int ierr_lu,ierr_ms;
    int i,j;
    FILE *fp;

    fp = fopen( "dbgmms.dat", "r" );

    if( fp == NULL )
```

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

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

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

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

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

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", m );

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

    ierr_lu = ASL_dbgmlu(a, lna, n, ipvt);

    if( ierr_lu != 0 ) {
        printf( "\tierr ( ASL_dbgmlu ) = %6d\n", ierr_lu );
        return 0;
    }

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

    fclose( fp );

    ierr_ms = ASL_dbgmms(a, lna, n, b, lnb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr ( ASL_dbgmlu ) = %6d\n\n", ierr_lu );
    printf( "\tierr ( ASL_dbgmms ) = %6d\n",   ierr_ms );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "%8.3g", b[i+lnb*j] );
        }
        printf( "\n" );
    }

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbgmms ***

 ** Input **

n =      4
```

```
m =      2

Coefficient Matrix a

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

Constant Vectors b

      36      11
      15       0
      22       7
      -6       4

 ** Output **

ierr ( ASL_dbgmlu ) =      0

ierr ( ASL_dbgmms ) =      0

Solution

       1       1
       2       1
       4       1
       5       1
```

2.2.7 ASL_dbgmdi, ASL_rbgmdi 実行列の行列式と逆行列

(1) **機 能**

ガウス法またはクラウト法で LU 分解された実行列 A(2 次元配列型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbgmdi  (a, lna, n, ipvt, det, isw, w1);
```

単精度関数:

```
ierr = ASL_rbgmdi  (a, lna, n, ipvt, det, isw, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	LU 分解後の実行列 A(2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列
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	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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)
5	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (d) 参照)
6	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
7	w1	{D∗ / R∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある.
分解は 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$, 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$, 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ のいずれかで行えばよい.

(b) 入力時の配列 a には, 下三角部分に $A = LU$ となる下三角行列 L が上三角部分に上三角行列 U が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい (2.2.2 図 2−2 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LU 分解を行う関数によって与えられる.

(d) 行列式の値は次の式によって与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
この時, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての 連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.2.8 ASL_dbgmlx, ASL_rbgmlx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = b$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbgmlx  (a, lna, n, alu, b, x, & itol, nit, ipvt, w1);
```

単精度関数:

```
ierr = ASL_rbgmlx  (a, lna, n, alu, b, x, & itol, nit, ipvt, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×n	入 力	係数行列 A(実行列, 2 次元配列型)
2	lna	I	1	入 力	配列 a, alu の整合寸法
3	n	I	1	入 力	行列 A の次数
4	alu	{D∗ R∗}	lna×n	入 力	LU 分解後の係数行列 A(注意事項 (a) 参照)
5	b	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
6	x	{D∗ R∗}	n	入 力	近似解 $\boldsymbol{x}$
				出 力	反復改良された解 $\boldsymbol{x}$
7	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似値 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)
10	w1	{D∗ R∗}	n	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ または 2.2.5 $\left\{\begin{matrix}\text{ASL\_dbgmls}\\ \text{ASL\_rbgmls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$, 2.2.3 $\left\{\begin{matrix}\text{ASL\_dbgmlu}\\ \text{ASL\_rbgmlu}\end{matrix}\right\}$ または 2.2.4 $\left\{\begin{matrix}\text{ASL\_dbgmlc}\\ \text{ASL\_rbgmlc}\end{matrix}\right\}$ によって分解された係数行列 A とその時得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$($\varepsilon$: 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1\\ 9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1\\ 8 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1\\ 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1\\ 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1\\ 5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1\\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1\\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1\\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7\\ x_8\\ x_9\\ x_{10} \end{bmatrix} = \begin{bmatrix} 6\\ 5\\ 4\\ 4\\ 4\\ 3\\ 2\\ 2\\ 2\\ 1 \end{bmatrix}$$

を解き, 解の改良を行う.

(b) 入力データ
係数行列 A, lna=11, n=10, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbgmlx */

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

int main()
{
    double *a,*sa;
    int lna;
    int n;
    double *b,*sb;
    int itol=0;
    int nnit=0;
```

```
int *ipvt;
double *wk;
int ierr;
int i,j;
FILE *fp;

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

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

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

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

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

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

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

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

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

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

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

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


fclose( fp );


ierr = ASL_dbgmsl(a, lna, n, b, ipvt);

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

```
    ierr = ASL_dbgmlx(sa, lna, n, a, sb, b, &itol, nnit, ipvt, wk);

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbgmlx ***

 ** Input **

 n =     10

Coefficient Matrix

      10         9         8         7         6         5         4         3         2         1
       9         9         8         7         6         5         4         3         2         1
       8         8         8         7         6         5         4         3         2         1
       7         7         7         7         6         5         4         3         2         1
       6         6         6         6         6         5         4         3         2         1
       5         5         5         5         5         5         4         3         2         1
       4         4         4         4         4         4         4         3         2         1
       3         3         3         3         3         3         3         3         2         1
       2         2         2         2         2         2         2         2         2         1
       1         1         1         1         1         1         1         1         1         1


Constant Vector

       6
       5
       4
       4
       4
       3
       2
       2
       2
       1

Original Solution

  x[     0] =         1
  x[     1] = -1.23e-16
  x[     2] =        -1
  x[     3] = -2.54e-16
  x[     4] =         1
  x[     5] = 7.99e-16
  x[     6] =        -1
  x[     7] = -7.4e-17
  x[     8] =         1
  x[     9] =         0

 ** Output **

ierr =      0

Improved Solution

  x[     0] =         1
  x[     1] = -4.68e-31
  x[     2] =        -1
  x[     3] = -1.33e-30
  x[     4] =         1
  x[     5] = -1.97e-31
  x[     6] =        -1
  x[     7] = -9.86e-32
  x[     8] =         1
  x[     9] =         0
```

2.3 複素行列 (2 次元配列型)(実数引数型)

2.3.1 ASL_zbgmsm, ASL_cbgmsm
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_zbgmsm (abr, abi, lna, n, m, ipvt, w1);

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	abr	{D∗ / R∗}	内容参照	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) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **大きさ:** (lna×(n + 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∗ / R∗}	内容参照	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) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$ **大きさ:** (lna×(n + 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∗ / R∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

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

(5) **エラーインディケータ (戻り値)**

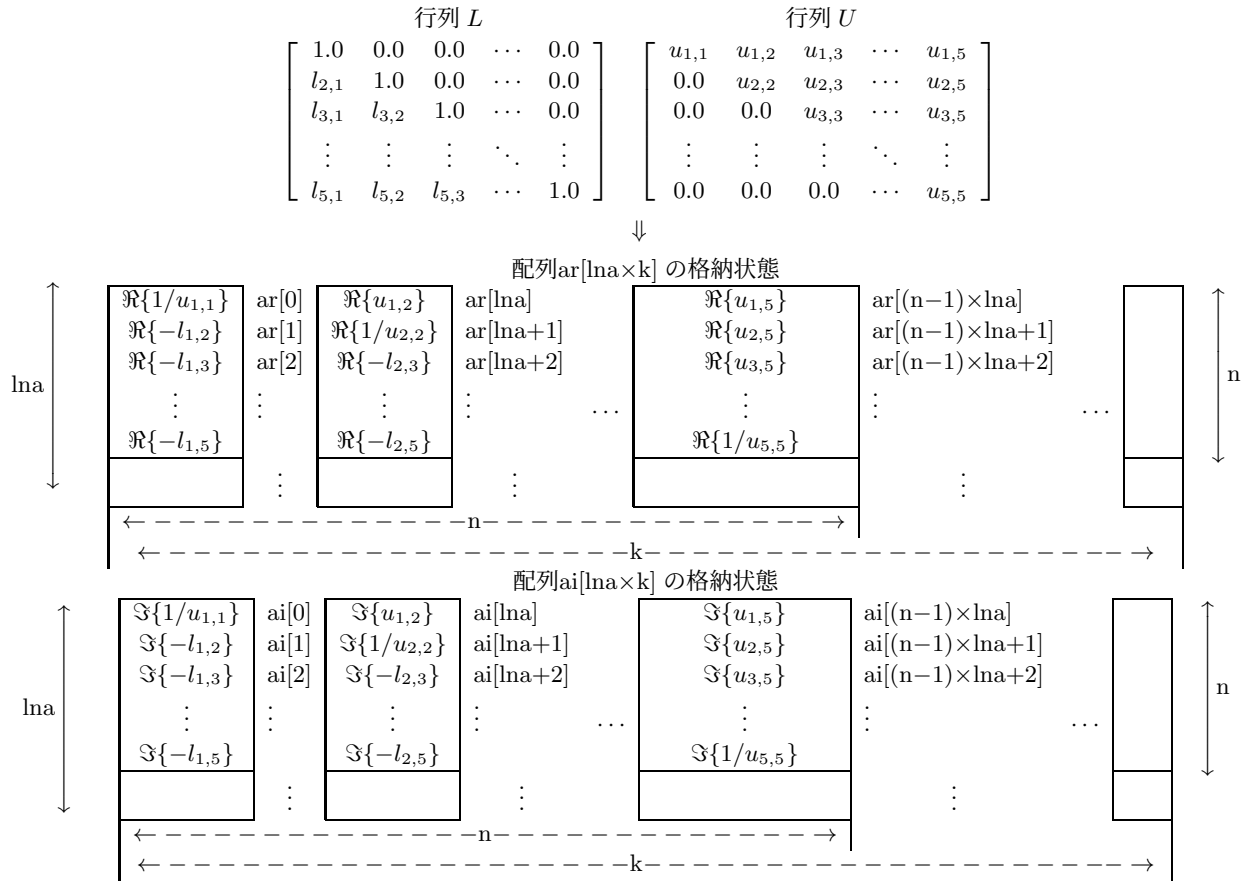
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	$\mathrm{abr}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \leftarrow$ $\{\ \mathrm{abr}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \times \mathrm{abr}[0]$ $+\ \mathrm{abi}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \times \mathrm{abi}[0]\ \}$ $/\ \{\mathrm{abr}[0]^2 + \mathrm{abi}[0]^2\},$ $\mathrm{abi}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \leftarrow$ $\{\ \mathrm{abi}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \times \mathrm{abr}[0]$ $-\ \mathrm{abr}[\mathrm{lna} \times (\mathrm{n} + i - 1)] \times \mathrm{abi}[0]\ \}$ $/\ \{\mathrm{abr}[0]^2 + \mathrm{abi}[0]^2\}$ $(i = 1, 2, \ldots, \mathrm{m})$ 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.	
$4000 + i$	係数行列 A の LU 分解の i 段目の処理において, ピボットが 0.0 となった. 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 ($\mathrm{i} \leq \mathrm{j}$) 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 2–3 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.

図 2−3　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.

(7) 使用例

(a) 問 題

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

を解く.

(b) 入力データ

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

(c) 主プログラム

```
/*      C interface example for ASL_zbgmsm */

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

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

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

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

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

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

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

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

    w1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( w1 == NULL )
    {
        printf( "no enough memory for array w1\n" );
        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_zbgmsm(abr, abi, lna, n, m, ipvt, w1);

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_zbgmsm ***

 ** 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)
```

2.3.2 ASL_zbgmsl, ASL_cbgmsl
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 $A\boldsymbol{x} = \boldsymbol{b}$ をガウス法またはクラウト法を用いて解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmsl  (ar, ai, lna, n, br, bi, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbgmsl  (ar, ai, lna, n, br, bi, ipvt, w1);
```

(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 (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 の実部 (注意事項 (b), (c) 参照)
2	ai	{D∗ / R∗}	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 の虚部 (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∗ / 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 i と交換した行の番号 (注意事項 (b) 参照)
8	w1	{D∗ / R∗}	n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	br[0] ← {br[0] × ar[0] + bi[0] × ai[0]}/{ar[0]^2 + ai[0]^2} bi[1] ← {bi[0] × ar[0] − br[0] × ai[0]}/{ar[0]^2 + ai[0]^2} 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, ピボットが 0.0 となった. A は特異である.	

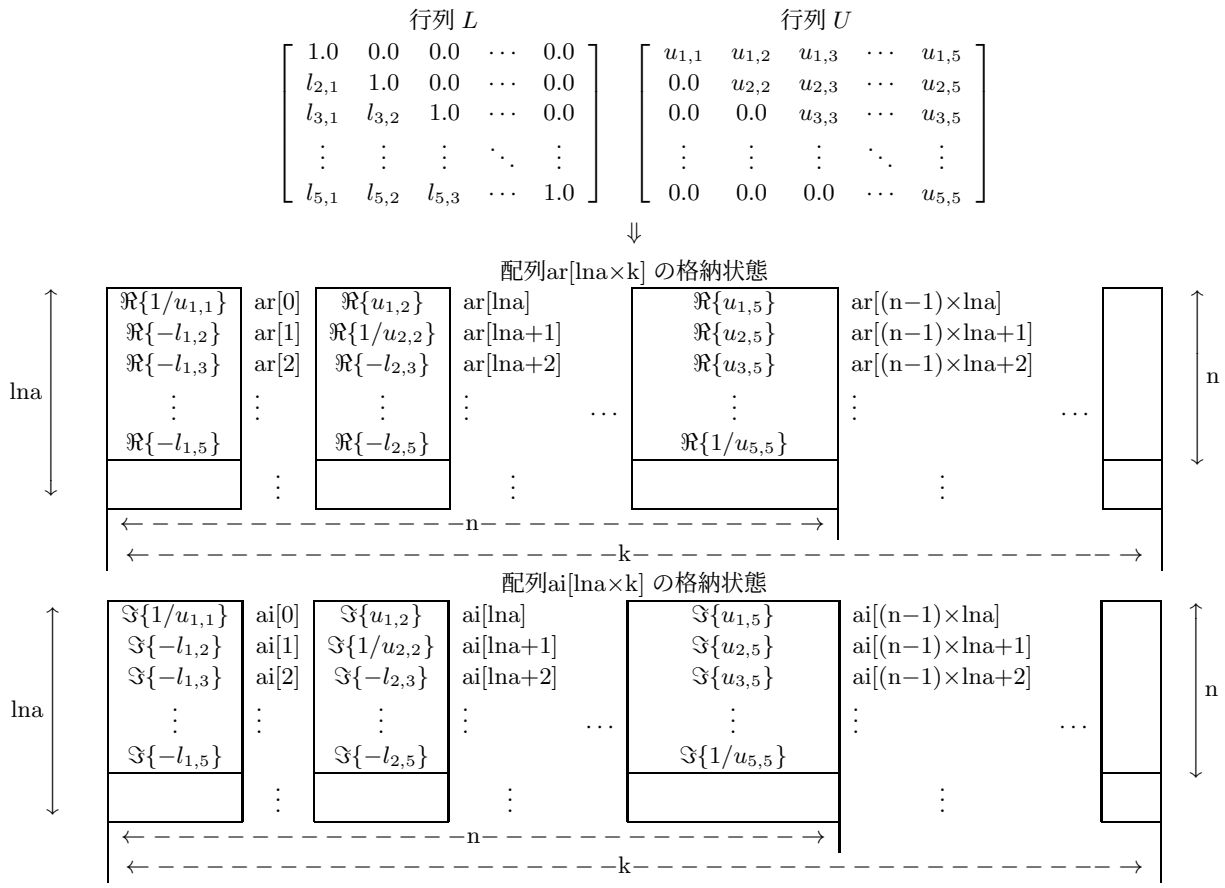
(6) **注意事項**

(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 2.3.1 $\left\{\begin{matrix}\text{ASL_zbgmsm}\\ \text{ASL_cbgmsm}\end{matrix}\right\}$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, 2.3.1 $\left\{\begin{matrix}\text{ASL_zbgmsm}\\ \text{ASL_cbgmsm}\end{matrix}\right\}$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 2.3.5 $\left\{\begin{matrix}\text{ASL_zbgmls}\\ \text{ASL_cbgmls}\end{matrix}\right\}$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 (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 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. 図 2−4 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.

図 2−4　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

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

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_zbgmsl */

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

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    int *kpvt;
    double *w;
    int ierr;
    int i,j;
    FILE *fp;

    fp = fopen( "zbgmsl.dat", "r" );
```

```
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

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

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

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

ai = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
if( ai == NULL )
{
    printf( "no enough memory for array ai\n" );
    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( "\n\tCoefficient Matrix (Real, Imaginary)\n\n" );
for( i=0 ; i<n ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &ar[i+na*j] );
    }
}

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

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

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

```
    fclose( fp );

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

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

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_zbgmsl ***

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

2.3.3 ASL_zbgmlu, ASL_cbgmlu
複素行列の LU 分解

(1) **機 能**

複素行列 A=(ar, ai) (2 次元配列型) をガウス法またはクラウト法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmlu  (ar, ai, lna, n, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbgmlu  (ar, ai, lna, n, ipvt, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	ar	{D∗ / R∗}	lna×n	入 力	複素行列 A の実部 (2 次元配列型)
				出 力	$A = LU$ と分角したときの, 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∗ / R∗}	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	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)

(4) **制限条件**

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

(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) 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. 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 (2.3.2 図 2−4 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.

2.3.4 ASL_zbgmlc, ASL_cbgmlc
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) をガウス法またはクラウト法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmlc  (ar, ai, lna, n, ipvt, & cond, w1);
```

単精度関数:

```
ierr = ASL_cbgmlc  (ar, ai, lna, n, ipvt, & cond, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	ar	{D∗ / R∗}	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∗ / R∗}	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∗ / R∗}	1	出 力	条件数の逆数
7	w1	{D∗ / R∗}	2 × n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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) 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. 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 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 (2.3.2 図 2−4 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値はその概算値である.

2.3.5 ASL_zbgmls, ASL_cbgmls
連立 1 次方程式 (LU 分解後の複素行列)

(1) **機 能**

ガウス法またはクラウト法で LU 分解された複素行列 A=(ar, ai) (2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmls  (ar, ai, lna, n, br, bi, ipvt);
```

単精度関数:

```
ierr = ASL_cbgmls  (ar, ai, lna, n, br, bi, ipvt);
```

(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	LU 分解後の係数行列 A の実部 (複素行列, 2 次元配列型) (注意事項 (a), (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	LU 分解後の係数行列 A の虚部 (複素行列, 2 次元配列型) (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	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	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	br[0] ← {br[0] × ar[0] + bi[0] × ai[0]}/{ar[0]^2 + ai[0]^2} bi[0] ← {bi[0] × ar[0] − br[0] × ai[0]}/{ar[0]^2 + ai[0]^2}
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A=(ar, ai) を LU 分解しておく必要がある. 通常は 2.3.3 {ASL_zbgmlu / ASL_cbgmlu} を使用して分解するが, 条件数も求めたい場合は 2.3.4 {ASL_zbgmlc / ASL_cbgmlc} を使用する. また, 2.3.2 {ASL_zbgmsl / ASL_cbgmsl} を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる. 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 2.3.6 {ASL_zbgmms / ASL_cbgmms} を用いて計算する方が効率良く解が求まる.

(b) 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 が格納されていなければならない. 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 には格納されていなくてよい. また, U の対角成分は, その逆数が格納されていなければならない (2.3.2 図 2−4 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は, 2.3.3 {ASL_zbgmlu / ASL_cbgmlu}, 2.3.4 {ASL_zbgmlc / ASL_cbgmlc}, 2.3.2 {ASL_zbgmsl / ASL_cbgmsl} によって与えられる.

2.3.6 ASL_zbgmms, ASL_cbgmms
多重右辺連立 1 次方程式 (LU 分解後の複素行列)

(1) **機 能**

ガウス法またはクラウト法で LU 分解された複素行列 A=(ar, ai) (2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x}_i = \boldsymbol{b}_i (i = 1, 2, \cdots, m)$ を解く．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_zbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt);

単精度関数:

ierr = ASL_cbgmms (ar, ai, lna, n, br, bi, lnb, m, ipvt);

(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	LU 分解後の係数行列 A の実部 (複素行列, 2 次元配列型) (注意事項 (a), (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	LU 分解後の係数行列 A の虚部 (複素行列, 2 次元配列型) (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	br	{D∗ / R∗}	lnb×m	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∗}	lnb×m	入 力	定数ベクトル $\boldsymbol{b}$ の虚部
				出 力	解$\boldsymbol{x}$の虚部
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	m	I	1	入 力	行列 B の次数
9	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(c) $0 < \mathrm{ipvt}[i-1] \leq \mathrm{n} \quad (i = 1, \ldots, \mathrm{n})$

(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.	br [lnb×i] ← { br [lnb×i] ×ar [0] +bi [lnb×i] ×ai [0] } / { ar [0] ^{2}+ ai [0] ^{2} } bi [lnb×i] ← { bi [lnb×i] ×ar [0] −br [lnb×i] ×ai [0] } / { ar [0] ^{2}+ ai [0] ^{2} } (i =0,1,$\cdots$,m-1) 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.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A=(ar, ai) を LU 分解しておく必要がある. 通常は 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$ を使用する. また, 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$ を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる.

(b) 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 が格納されていなければならない. 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 には格納されていなくてよい. また, U の対角成分は, その逆数が格納されていなければならない (2.3.2 図 2−4 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は, 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL\_zbgmsl}\\ \text{ASL\_cbgmsl}\end{matrix}\right\}$ によって与えられる.

(7) **使用例**

(a) 問 題

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

を解く.

(b) 入力データ
係数行列 A の実部 ar および虚部 ai と, 定数ベクトル$\boldsymbol{b_1}$, $\cdots$,$\boldsymbol{b_4}$からなる行列の実部 br および虚部 bi, lna=11, lnb=11, n=4, m=4

(c) 主プログラム

```
/*      C interface example for ASL_zbgmms */

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

```
#include <asl.h>

int main()
{
    double *ar;
    double *ai;
    int lna=11;
    int lnb=11;
    int n;
    int m;
    double *br;
    double *bi;
    int *ipvt;
    double *w;
    int ierr,kerr;
    int i,j;
    FILE *fp;

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

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

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

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

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

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

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

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

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

    printf( "\tn = %6d\tm = %6d\n", n,m );

    printf( "\n\tCoefficient Matrix (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &ar[i+lna*j] );
            fscanf( fp, "%lf", &ai[i+lna*j] );
            printf( "(%6.3g,%6.3g) ", ar[i+lna*j],ai[i+lna*j] );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            fscanf( fp, "%lf", &br[i+lnb*j] );
            fscanf( fp, "%lf", &bi[i+lnb*j] );
            printf( "(%6.3g,%6.3g) ", br[i+lnb*j],bi[i+lnb*j] );
```

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

    fclose( fp );

    ierr = ASL_zbgmlu(ar, ai, lna, n, ipvt, w);
    kerr = ASL_zbgmms(ar, ai, lna, n, br, bi, lnb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tzbgmlu ierr = %6d\n", ierr );
    printf( "\tzbgmms ierr = %6d\n", kerr );

    printf( "\n\tSolution (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g) ", br[i+lnb*j],bi[i+lnb*j] );
        }
        printf( "\n" );
    }

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

    return 0;
}
```

(d) 出力結果

```
    *** ASL_zbgmms ***

     ** Input **

    n =      4  m =      4

    Coefficient Matrix (Real,Imaginary)

    (     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 (Real,Imaginary)

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

     ** Output **

    zbgmlu ierr =      0
    zbgmms ierr =      0

    Solution (Real,Imaginary)

(  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)
```

2.3.7 ASL_zbgmdi, ASL_cbgmdi 複素行列の行列式と逆行列

(1) **機 能**

ガウス法またはクラウト法で LU 分解された複素行列 A = (ar, ai) (2 次元配列型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmdi  (ar, ai, lna, n, ipvt, det, isw, w1);
```

単精度関数:

```
ierr = ASL_cbgmdi  (ar, ai, lna, n, ipvt, det, isw, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	ar	{D∗ / R∗}	lna×n	入 力	LU 分解後の複素行列 A の実部 (2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の実部
2	ai	{D∗ / R∗}	lna×n	入 力	LU 分解後の複素行列 A の虚部 (2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の虚部
3	lna	I	1	入 力	配列 ar, ai の整合寸法
4	n	I	1	入 力	行列 A の次数
5	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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	det	{D∗ / R∗}	3	出 力	行列 A の行列式の値 (注意事項 (d) 参照)
7	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
8	w1	{D∗ / R∗}	2 × n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← ar[0] det[1] ← ai[0] det[2] ← 0.0 ar[0] ← ar[0]/{ar[0]^2 + ai[0]^2} ai[0] ← −ai[0]/{ar[0]^2 + ai[0]^2} とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある.
分解は 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL_zbgmsl}\\ \text{ASL_cbgmsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 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 が格納されていなければならない. 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 には格納されていなくてよい. また, U の対角成分は, その逆数が格納されていなければならない (2.3.2 図 2−4 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は, 2.3.3 $\left\{\begin{matrix}\text{ASL\_zbgmlu}\\ \text{ASL\_cbgmlu}\end{matrix}\right\}$, 2.3.4 $\left\{\begin{matrix}\text{ASL\_zbgmlc}\\ \text{ASL\_cbgmlc}\end{matrix}\right\}$, 2.3.2 $\left\{\begin{matrix}\text{ASL_zbgmsl}\\ \text{ASL_cbgmsl}\end{matrix}\right\}$によって与えられる.

(d) 行列式の値は次の式によって与えられる.
$\Re\{det(A)\} = \text{det}[0] \times 10^{\text{det}[2]}$
$\Im\{det(A)\} = \text{det}[1] \times 10^{\text{det}[2]}$
この時, $1.0 \leq |\text{det}[0]| + |\text{det}[1]| < 10.0$ となるようにスケーリングされている. ここで $\Re, \Im$ はそれぞれ複素数の実部, 虚部を取り出すことを意味している.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.3.8 ASL_zbgmlx, ASL_cbgmlx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgmlx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbgmlx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, ipvt, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	ar	{D∗ R∗}	lna×n	入 力	係数行列 A の実部 (複素行列, 2 次元配列型)
2	ai	{D∗ R∗}	lna×n	入 力	係数行列 A の虚部 (複素行列, 2 次元配列型)
3	lna	I	1	入 力	配列 ar, ai, alr, ali の整合寸法
4	n	I	1	入 力	行列 A の次数
5	alr	{D∗ R∗}	lna×n	入 力	LU 分解後の係数行列 A の実部 (注意事項 (a) 参照)
6	ali	{D∗ R∗}	lna×n	入 力	LU 分解後の係数行列 A の虚部 (注意事項 (a) 参照)
7	br	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$の実部
8	bi	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$の虚部
9	xr	{D∗ R∗}	n	入 力	近似解$\boldsymbol{x}$の実部
				出 力	反復改良された解$\boldsymbol{x}$の実部
10	xi	{D∗ R∗}	n	入 力	近似解$\boldsymbol{x}$の虚部
				出 力	反復改良された解$\boldsymbol{x}$の虚部
11	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似数 (注意事項 (c) 参照)
12	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
13	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)
14	w1	{D∗ R∗}	3 × n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.3.2 $\left\{\begin{matrix}\mathrm{ASL\_zbgmsl}\\ \mathrm{ASL\_cbgmsl}\end{matrix}\right\}$ または 2.3.5 $\left\{\begin{matrix}\mathrm{ASL\_zbgmls}\\ \mathrm{ASL\_cbgmls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.3.2 $\left\{\begin{matrix}\mathrm{ASL\_zbgmsl}\\ \mathrm{ASL\_cbgmsl}\end{matrix}\right\}$, 2.3.3 $\left\{\begin{matrix}\mathrm{ASL\_zbgmlu}\\ \mathrm{ASL\_cbgmlu}\end{matrix}\right\}$ または 2.3.4 $\left\{\begin{matrix}\mathrm{ASL\_zbgmlc}\\ \mathrm{ASL\_cbgmlc}\end{matrix}\right\}$ によって分解された係数行列 A と, その時のピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
$\mathrm{itol} \leq 0$ または $\mathrm{itol} \geq -\log_{10}(2 \times \varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.4 複素行列 (2 次元配列型) (複素引数型)

2.4.1 ASL_zbgnsm, ASL_cbgnsm
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_zbgnsm (ab, lna, n, m, ipvt);

単精度関数:

ierr = ASL_cbgnsm (ab, lna, n, m, ipvt);

(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* / C* }	内容参照	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) $[A, \boldsymbol{b_1}, \boldsymbol{b_2}, \ldots, \boldsymbol{b_m}]$ **大きさ:** lna×(n + 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) $[A', \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}]$ (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	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) $0 < \mathrm{m}$

(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.	
$4000 + i$	係数行列 A の LU 分解の i 段目の処理において, ピボットが 0.0 となった. 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 (2.2.1 図 2−1 参照).

(7) **使用例**

(a) 問 題

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

を解く. ただし, $i = \sqrt{-1}$.

(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

(c) 主プログラム

```
/*      C interface example for ASL_zbgnsm */

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

    fp = fopen( "zbgnsm.dat", "r" );
```

```
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zbgnsm ***\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_zbgnsm(ab, lna, n, m, ipvt);

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

 ** 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)
```

2.4.2 ASL_zbgnsl, ASL_cbgnsl
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 $A\boldsymbol{x} = \boldsymbol{b}$ をガウス法またはクラウト法を用いて解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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(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∗ / 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 i と交換した行の番号 (注意事項 (b) 参照)
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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, ピボットが 0.0 となった. A は特異にである.	

(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 2.4.1 $\left\{\begin{matrix}\text{ASL\_zbgnsm}\\ \text{ASL\_cbgnsm}\end{matrix}\right\}$ 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, 2.4.1 $\left\{\begin{matrix}\text{ASL_zbgnsm}\\ \text{ASL_cbgnsm}\end{matrix}\right\}$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 2.4.5 $\left\{\begin{matrix}\text{ASL\_zbgnls}\\ \text{ASL\_cbgnls}\end{matrix}\right\}$ 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 (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 (2.2.1 図 2−1 参照).

(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$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_zbgnsl */

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

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

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

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

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

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (lna*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        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_zbgnsl(a, lna, n, b, ipvt);

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

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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_zbgnsl ***

 ** 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)
```

2.4.3 ASL_zbgnlu, ASL_cbgnlu
複素行列の LU 分解

(1) **機 能**

複素行列 A (2 次元配列型) をガウス法またはクラウト法を用いて LU 分解する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgnlu  (a, lna, n, ipvt);
```

単精度関数:

```
ierr = ASL_cbgnlu  (a, lna, n, ipvt);
```

(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	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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, ピボットが 0.0 となった. A は特異である.	

(6) **注意事項**

(a) 配列 a には, 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 (2.2.2 図 2−2 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.

2.4.4 ASL_zbgnlc, ASL_cbgnlc
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) をガウス法またはクラウト法を用いて LU 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{ Z∗ / C∗ }	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∗ / R∗ }	1	出 力	条件数の逆数
6	w1	{ Z∗ / C∗ }	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)

(4) **制限条件**

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

(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, ピボットが 0.0 となった. 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 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 (2.2.2 図 2−2 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.

2.4.5 ASL_zbgnls, ASL_cbgnls
連立 1 次方程式 (LU 分解後の複素行列)

(1) **機 能**

ガウス法またはクラウト法で LU 分解された複素行列 A (2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	LU 分解後の係数行列 A(複素行列, 2 次元配列型) (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	b	{Z∗ / C∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
5	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある. 通常は関数 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$ を使用する. また, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる. 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 2.4.6 $\left\{\begin{matrix}\text{ASL\_zbgnms}\\ \text{ASL\_cbgnms}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 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 が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい. また, U の対角成分はその逆数が格納されていなければならない (2.2.2 図 2−2 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ によって与えられる.

2.4.6 ASL_zbgnms, ASL_cbgnms
多重右辺連立 1 次方程式 (LU 分解後の複素行列)

(1) **機 能**

ガウス法またはクラウト法で LU 分解された複素行列 A (2 次元配列型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x}_i = \boldsymbol{b}_i (i = 1, 2, \cdots, m)$ を解く. 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_zbgnms (a, lna, n, b, lnb, m, ipvt);

単精度関数:

ierr = ASL_cbgnms (a, lna, n, b, lnb, m, ipvt);

(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	LU 分解後の係数行列 A(複素行列, 2 次元配列型) (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	b	{Z∗ / C∗}	lnb×m	入 力	定数ベクトル $\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
5	lnb	I	1	入 力	配列 b の整合寸法
6	m	I	1	入 力	行列 B の次数
7	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(c) $0 < \mathrm{ipvt}[i-1] \leq \mathrm{n} \quad (i = 1, \ldots, \mathrm{n})$

(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[lnb × i] ← b[lnb × i]/a[0] ($i = 0, 1, \cdots,$ m − 1) 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.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある. 通常は関数 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$ を使用する. また, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LU 分解を利用することもできる.

(b) 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 が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい. また, U の対角成分はその逆数が格納されていなければならない (2.2.2 図 2−2 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ によって与えられる.

(7) **使用例**

(a) 問 題

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

を解く.

(b) 入力データ
係数行列 A の a と, 定数ベクトル$\boldsymbol{b_1}$, ⋯,$\boldsymbol{b_4}$からなる行列 b, lna=11, lnb=11, n=4, m=4

(c) 主プログラム

```
/*      C interface example for ASL_zbgnms */

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

int main()
{
    double _Complex *a;
    int lna=11;
    int lnb=11;
    int n;
    int m;
    double _Complex *b;
    int *ipvt;
    int ierr,kerr;
    int i,j;
    FILE *fp;
```

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

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

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

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

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

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

    printf( "\tn = %6d\tm = %6d\n", n,m );

    printf( "\n\tCoefficient Matrix (Real,Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            a[i+lna*j] = tmp_re + tmp_im * _Complex_I;
            printf( "(%6.3g,%6.3g) ", creal(a[i+lna*j]),cimag(a[i+lna*j]) );
        }
        printf( "\n" );
    }

    printf( "\n\tConstant Vectors (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            double tmp_re, tmp_im;
            fscanf( fp, "%lf", &tmp_re );
            fscanf( fp, "%lf", &tmp_im );
            b[i+lnb*j] = tmp_re + tmp_im * _Complex_I;
            printf( "(%6.3g,%6.3g) ", creal(b[i+lnb*j]),cimag(b[i+lnb*j]) );
        }
        printf( "\n" );
    }

    fclose( fp );

    ierr = ASL_zbgnlu(a, lna, n, ipvt);
    kerr = ASL_zbgnms(a, lna, n, b, lnb, m, ipvt);

    printf( "\n     ** Output **\n\n" );
    printf( "\tzbgnlu ierr = %6d\n", ierr );
    printf( "\tzbgnms ierr = %6d\n", kerr );

    printf( "\n\tSolution (Real,Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%8.3g,%8.3g) ", creal(b[i+lnb*j]),cimag(b[i+lnb*j]) );
        }
        printf( "\n" );
    }

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_zbgnms ***

 ** Input **

n =      4  m =      4

Coefficient Matrix (Real,Imaginary)

(     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 (Real,Imaginary)

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

 ** Output **

zbgnlu ierr =      0
zbgnms ierr =      0

Solution (Real,Imaginary)

(  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)
```

2.4.7 ASL_zbgndi, ASL_cbgndi 複素行列の行列式と逆行列

(1) 機 能

ガウス法またはクラウト法で LU 分解された複素行列 A (2 次元配列型) の行列式と逆行列を求める.

(2) 使用法

倍精度関数:

```
ierr = ASL_zbgndi  (a, lna, n, ipvt, & cdet, & det, isw, w1);
```

単精度関数:

```
ierr = ASL_cbgndi  (a, lna, n, ipvt, & cdet, & det, isw, w1);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{Z∗ / C∗}	lna×n	入 力	LU 分解後の複素行列 A(2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列
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	入 力	ピボッティング情報 ipvt[i − 1] : LU 分解の 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)
5	cdet	{Z∗ / C∗}	1	出 力	行列 A の行列式の値 (注意事項 (d) 参照)
6	det	{D∗ / R∗}	1	出 力	行列 A の行列式の値 (注意事項 (d) 参照)
7	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
8	w1	{Z∗ / C∗}	n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(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.	cdet ← a[0] det ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある.
分解は 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL_zbgnsl}\\ \text{ASL_cbgnsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 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 が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい. また, U の対角成分はその逆数が格納されていなければならない (2.2.2 図 2−2 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$, 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ によって与えられる.

(d) 行列式の値は次の式によって与えられる.
$det(A) = \text{cdet} \times 10^{\text{det}}$
この時, $1.0 \leq |\Re\{\text{cdet}\}| + |\Im\{\text{cdet}\}| < 10.0$ となるようにスケーリングされている. ここで $\Re, \Im$ はそれぞれ複素数の実部, 虚部を取り出すことを意味している.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.4.8 ASL_zbgnlx, ASL_cbgnlx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbgnlx  (a, lna, n, alu, b, x, & itol, nit, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbgnlx  (a, lna, n, alu, b, x, & itol, nit, ipvt, w1);
```

(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(複素行列, 2 次元配列型)
2	lna	I	1	入 力	配列 a, alu の整合寸法
3	n	I	1	入 力	行列 A の次数
4	alu	{Z∗ / C∗}	lna×n	入 力	LU 分解後の係数行列 A(注意事項 (a) 参照)
5	b	{Z∗ / C∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
6	x	{Z∗ / C∗}	n	入 力	近似解$\boldsymbol{x}$
				出 力	反復改良された解$\boldsymbol{x}$
7	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似数 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)
10	w1	{Z∗ / C∗}	n	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$ または 2.4.5 $\left\{\begin{matrix}\text{ASL\_zbgnls}\\ \text{ASL\_cbgnls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.4.2 $\left\{\begin{matrix}\text{ASL\_zbgnsl}\\ \text{ASL\_cbgnsl}\end{matrix}\right\}$, 2.4.3 $\left\{\begin{matrix}\text{ASL\_zbgnlu}\\ \text{ASL\_cbgnlu}\end{matrix}\right\}$ または 2.4.4 $\left\{\begin{matrix}\text{ASL\_zbgnlc}\\ \text{ASL\_cbgnlc}\end{matrix}\right\}$ によって分解された係数行列 A とその時得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.5 正値対称行列 (2 次元配列型) (上三角型)

2.5.1 ASL_dbpdsl, ASL_rbpdsl
連立 1 次方程式 (正値対称行列)

(1) **機 能**

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

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbpdsl  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbpdsl  (a, lna, n, b);
```

(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(正値対称行列, 2 次元配列型, 上三角型)
				出 力	$A = LL^T$ 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	a[0] ← $\sqrt{\mathrm{a[0]}}$ b[0] ← b[0]/a[0] とする.
2100	係数行列 A の $\mathrm{LL^T}$ 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.

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.
$4000+i$	係数行列 A の $\mathrm{LL^T}$ 分解の i 段目の処理において, 対角要素が 0.0 以下となった. 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, この関数を一度呼び出した後, 続けて 2.5.4 $\left\{\begin{matrix}\text{ASL_dbpdls}\\ \text{ASL_rbpdls}\end{matrix}\right\}$を配列 b の内容のみを変えて呼び出せばよい. このようにすれば, 行列 A の $\mathrm{LL^T}$ 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 の上三角部分に上三角行列 L^T が格納される. 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. この関数は, 配列 a の上三角部分のみを使用する.

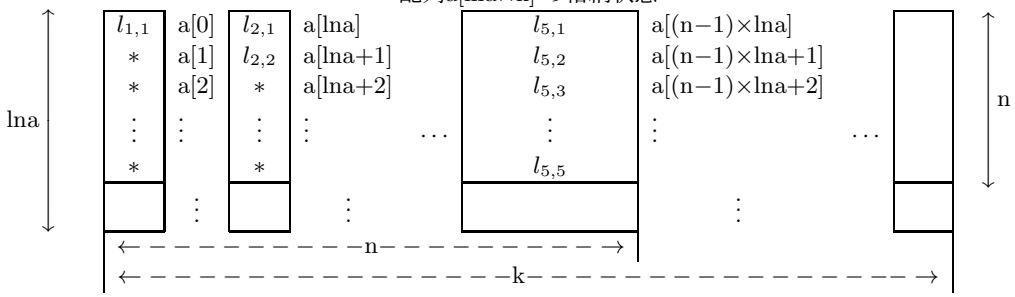
図 2−5　行列 L^T の格納状態

行列 L^T

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

⇓

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

$$\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

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_dbpdsl */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
    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", n );
    printf( "\tCoefficient Matrix \n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &a[i+na*j] );
            printf( "%8.3g  ", a[i+na*j] );
        }
        printf( "\n" );
    }

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

    fclose( fp );

    ierr = ASL_dbpdsl(a, na, n, b);

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

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
```

```
            printf( "\t  x[%6d] = %8.3g\n",i, b[i] );
        }


        free( a );
        free( b );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dbpdsl ***

         ** Input **

         n =      4

        Coefficient Matrix

               5         7         6         5
               7        10         8         7
               6         8        10         9
               5         7         9        10

        Constant Vector

              23
              32
              33
              31

         ** Output **

        ierr =      0

        Solution

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

2.5.2 ASL_dbpduu, ASL_rbpduu 正値対称行列の $\mathrm{LL^T}$ 分解

(1) **機 能**

正値対称行列 A(2 次元配列型)(上三角型) をコレスキー法を用いて $\mathrm{LL^T}$ 分解する.

(2) **使用法**

倍精度関数:

ierr = ASL_dbpduu (a, lna, n);

単精度関数:

ierr = ASL_rbpduu (a, lna, n);

(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	正値対称行列 A(2 次元配列型)(上三角型)
				出 力	$A = LL^T$ 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	$\mathrm{a[0]} \leftarrow \sqrt{\mathrm{a[0]}}$ とする.
2100	係数行列 A の $\mathrm{LL^T}$ 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.
$4000 + i$	i 段目の処理において, 対角要素が 0.0 以下となった. 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^T が格納される. 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. この関数は配列 a の上三角部分のみを使用する (2.5.1 図 2−5 参照).

2.5.3 ASL_dbpduc, ASL_rbpduc 正値対称行列の LL^{T} 分解と条件数

(1) **機 能**

正値対称行列 A(2 次元配列型)(上三角型) をコレスキー法を用いて LL^{T} 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	正値対称行列 A(2 次元配列型)(上三角型)
				出 力	$A = LL^T$ 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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
5	w1	{D∗ / R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	$\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}$ $\mathrm{cond} \leftarrow 1.0$ とする.
2100	係数行列 A の LL^{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.
$4000 + i$	i 段目の処理において, 対角要素が 0.0 以下となった. 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, 上三角部分に上三角行列 L^T が格納される. 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. この関数は配列 a の上三角部分のみを使用する (2.5.1 図 2−5 参照).

(b) 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.

2.5.4 ASL_dbpdls, ASL_rbpdls
連立 1 次方程式 ($\mathrm{LL^T}$ 分解後の正値対称行列)

(1) **機 能**

コレスキー法で $\mathrm{LL^T}$ 分解された正値対称行列 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 $LL^T\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbpdls  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbpdls  (a, lna, n, b);
```

(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	$\mathrm{LL^T}$ 分解後の係数行列 A(正値対称行列, 2 次元配列型, 上三角型) (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	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	$\mathrm{b[0]} \leftarrow \mathrm{b[0]/a[0]^2}$ とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LL^T}$ 分解しておく必要がある. 通常は 2.5.2 $\left\{\begin{matrix}\mathrm{ASL_dbpduu}\\ \mathrm{ASL_rbpduu}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.5.3 $\left\{\begin{matrix}\mathrm{ASL_dbpduc}\\ \mathrm{ASL_rbpduc}\end{matrix}\right\}$を使用する. また, 2.5.1 $\left\{\begin{matrix}\mathrm{ASL_dbpdsl}\\ \mathrm{ASL_rbpdsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる $\mathrm{LL^T}$ 分解を利用することもできる.

(b) 配列 a には, 上三角部分に上三角行列 L^T が格納されていなければならない. 下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい.
この関数は配列 a の上三角部分のみを使用する (2.5.1 図 2−5 参照).

2.5.5 ASL_dbpddi, ASL_rbpddi
正値対称行列の行列式と逆行列

(1) **機 能**

コレスキー法で LL^T 分解された正値対称行列 A(2 次元配列型)(上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbpddi  (a, lna, n, det, isw);
```

単精度関数:

```
ierr = ASL_rbpddi  (a, lna, n, det, isw);
```

(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	LL^T 分解後の正値対称行列 A(2 次元配列型)(上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列 (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	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
5	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	$\det[0] \leftarrow \mathrm{a}[0]^2$, $\det[1] \leftarrow 0.0$, $\mathrm{a}[0] \leftarrow 1.0/\mathrm{a}[0]^2$ とする.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LL^T}$ 分解しておく必要がある.
分解は 2.5.2 $\left\{\begin{matrix}\mathrm{ASL\_dbpduu}\\ \mathrm{ASL\_rbpduu}\end{matrix}\right\}$, 2.5.3 $\left\{\begin{matrix}\mathrm{ASL\_dbpduc}\\ \mathrm{ASL\_rbpduc}\end{matrix}\right\}$, 2.5.1 $\left\{\begin{matrix}\mathrm{ASL_dbpdsl}\\ \mathrm{ASL_rbpdsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 入力時の配列 a には上三角部分に上三角行列 L^T が格納されていなければならない. 下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい. 逆行列 A^{-1} はやはり対称行列であるので, 上三角部分のみが配列 a に格納される. この関数は配列 a の上三角部分のみを使用する (2.5.1 図 2−5 参照).

(c) 行列式の値は次の式によって与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
この時, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(d) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.5.6 ASL_dbpdlx, ASL_rbpdlx
連立 1 次方程式の解の改良 (正値対称行列)

(1) **機 能**

正値対称行列 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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbpdlx  (a, lna, n, all, b, x, & itol, nit, w1);
```

単精度関数:

```
ierr = ASL_rbpdlx  (a, lna, n, all, b, x, & itol, nit, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lna×n	入 力	係数行列 A(正値対称行列, 2 次元配列型, 上三角型)
2	lna	I	1	入 力	配列 a, all の整合寸法
3	n	I	1	入 力	行列 A の次数
4	all	{D∗ R∗}	lna×n	入 力	LL^T 分解後の係数行列 A(注意事項 (a) 参照)
5	b	{D∗ R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
6	x	{D∗ R∗}	n	入 力	近似解 $\boldsymbol{x}$
				出 力	反復改良された解 $\boldsymbol{x}$
7	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似数 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	w1	{D∗ R∗}	n	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.5.1 $\left\{\begin{matrix}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{matrix}\right\}$ または 2.5.4 $\left\{\begin{matrix}\text{ASL\_dbpdls}\\ \text{ASL\_rbpdls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.5.1 $\left\{\begin{matrix}\text{ASL\_dbpdsl}\\ \text{ASL\_rbpdsl}\end{matrix}\right\}$, 2.5.2 $\left\{\begin{matrix}\text{ASL\_dbpduu}\\ \text{ASL\_rbpduu}\end{matrix}\right\}$ または 2.5.3 $\left\{\begin{matrix}\text{ASL\_dbpduc}\\ \text{ASL\_rbpduc}\end{matrix}\right\}$ によって分解された係数行列 A を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.6 実対称行列 (2 次元配列型) (上三角型)

2.6.1 ASL_dbspsl, ASL_rbspsl
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_dbspsl (a, lna, n, b, ipvt, wk);

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000+i$	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 0.0 となった. A は特異にである.	

(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 2.6.4 $\left\{\begin{matrix}\mathrm{ASL_dbspls}\\ \mathrm{ASL_rbspls}\end{matrix}\right\}$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 2−6　行列 L^T の格納状態と行列 D の内容

行列 L^T

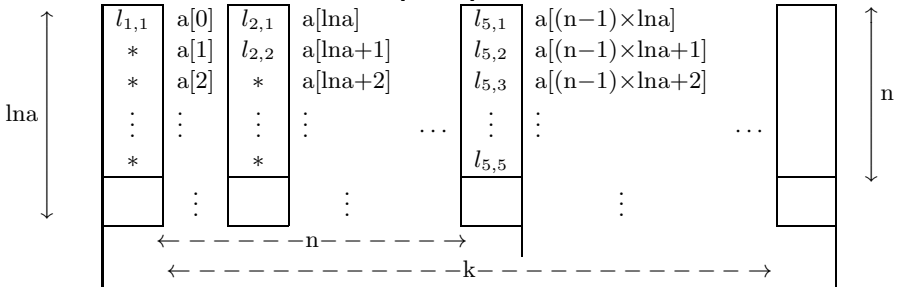
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

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_dbspsl */

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

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

    int i,j;

    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    a = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        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_dbspsl(a, na, n, b, ipvt, wk);

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


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

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


        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dbspsl ***

         ** 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
```

2.6.2 ASL_dbspud, ASL_rbspud
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_dbspud  (a, lna, n, ipvt, wk);
```

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000 + i$	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 (2.6.1 図 2−6 参照).

(b) この関数では, 係数行列 A の LDL$^{\mathrm{T}}$ 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.

2.6.3 ASL_dbspuc, ASL_rbspuc
実対称行列の $\mathrm{LDL^T}$ 分解と条件数

(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 $\mathrm{LDL^T}$ 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbspuc  (a, lna, n, ipvt, & cond, wk);
```

単精度関数:

```
ierr = ASL_rbspuc  (a, lna, n, ipvt, & cond, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
6	wk	{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)

(4) **制限条件**

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

(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 $\leftarrow$ 1.0 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.
$4000+i$	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.	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, 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 (2.6.1 図 2−6 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)(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.

(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.

2.6.4 ASL_dbspls, ASL_rbspls
連立 1 次方程式 (LDL$^{\mathrm{T}}$ 分解後の実対称行列)

(1) **機 能**

修正コレスキー法で LDL$^{\mathrm{T}}$ 分解された実対称行列 A (2 次元配列型) (上三角型) を係数行列とする連立 1 次方程式を解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	LDL^T 分解後の係数行列 A(実対称行列, 2 次元配列型, 上三角型) (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	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL$^{\mathrm{T}}$ 分解しておく必要がある. 通常は 2.6.2 $\left\{\begin{matrix}\mathrm{ASL_dbspud}\\ \mathrm{ASL_rbspud}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.6.3 $\left\{\begin{matrix}\mathrm{ASL_dbspuc}\\ \mathrm{ASL_rbspuc}\end{matrix}\right\}$を使用する. また, 2.6.1 $\left\{\begin{matrix}\mathrm{ASL_dbspsl}\\ \mathrm{ASL_rbspsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL$^{\mathrm{T}}$ 分解を利用することもできる. 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 2.6.5 $\left\{\begin{matrix}\mathrm{ASL\_dbspms}\\ \mathrm{ASL\_rbspms}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 配列 a には, 上三角行列 L^T が格納されていなければならない. 対角行列 D と下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい. この関数は配列 a の上三角部分のみを使用する (2.6.1 図 2−6 参照).

(c) この関数では, 係数行列 A の LDL$^{\mathrm{T}}$ 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 $\leq$ 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.

2.6.5 ASL_dbspms, ASL_rbspms
多重右辺連立 1 次方程式 (LDL$^{\mathrm{T}}$ 分解後の実対称行列)

(1) **機 能**

LDL^T 分解された実行列 A(上三角型) を係数行列とする連立 1 次方程式 $LDL^T\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く. 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_dbspms (a, lna, n, b, lnb, m, ipvt);

単精度関数:

ierr = ASL_rbspms (a, lna, n, b, lnb, m, ipvt);

(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	LDL$^{\mathrm{T}}$ 分解後の係数行列 A(実対称行列, 上三角型) (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	b	{D∗ / R∗}	lnb×m	入 力	定数ベクトル$\boldsymbol{b_i}$からなる行列 $[\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	解ベクトル$\boldsymbol{x_i}$からなる行列 $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	入 力	配列 b の整合寸法
6	m	I	1	入 力	右辺ベクトルの数
7	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1]：LDL$^{\mathrm{T}}$ 分解の 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(c) $0 < \mathrm{ipvt}[i-1] \leq \mathrm{n} \quad (i = 1, \ldots, \mathrm{n})$

(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[lna × (i − 1)] ← b[lna × (i − 1)]/a[0] $(i = 1, 2, \cdots, \mathrm{m})$ 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.	
3020	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL^T 分解しておく必要がある. 通常は関数 2.6.2 $\left\{\begin{matrix}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.6.3 $\left\{\begin{matrix}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{matrix}\right\}$ を使用する. また,2.6.1 $\left\{\begin{matrix}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 方程式をすでに解いている場合は, その出力として得られる LDL^T 分解を利用することもできる.

(b) 入力時の配列 a には, 上三角行列 L^T が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはり対称行列であるので, 上三角部分のみが配列 a に格納される. この関数は配列 a の上三角部分のみを使用する.

(c) ipvt には, LDL^T 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL^T 分解を行う 2.6.2 $\left\{\begin{matrix}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{matrix}\right\}$ によって与えられる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13 \end{bmatrix}$$

を解く.

(b) 入力データ
係数行列 A, lna=10, n=4, 定数ベクトル***b***からなる行列 B, lnb=10, m=2

(c) 主プログラム

```
/* C interface example for ASL_dbspms */

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

int main()
{
    double *a;
    int lna=11;
    int n=4;
    double *b;
    int lnb=11;
    int m=2;
    double *wk;
    int *ipvt;
    int ierr_ud,ierr_ms;
    int i,j;
    FILE *fp;
```

```
fp = fopen( "dbspms.dat", "r" );

if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

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

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

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

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

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

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", m );

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

ierr_ud = ASL_dbspud(a, lna, n, ipvt, wk);

if( ierr_ud != 0 ) {
    printf( "\tierr ( ASL_dbspud ) = %6d\n", ierr_ud );
    return 0;
}

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

fclose( fp );

ierr_ms = ASL_dbspms(a, lna, n, b, lnb, m, ipvt);

printf( "\n     ** Output **\n\n" );
printf( "\tierr ( ASL_dbspud ) = %6d\n\n", ierr_ud );
printf( "\tierr ( ASL_dbspms ) = %6d\n",   ierr_ms );

printf( "\n\tSolution\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "%8.3g", b[i+lnb*j] );
    }
    printf( "\n" );
}

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

```
    return 0;
}
```

(d) 出力結果

```
*** ASL_dbspms ***

 ** Input **

n =      4
m =      2

Coefficient Matrix a

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

Constant Vectors b

       1      -2
      -1       1
       4       9
      -4      13

 ** Output **

ierr ( ASL_dbspud ) =      0

ierr ( ASL_dbspms ) =      0

Solution

       1      -2
      -1       1
       2       1
      -2       3
```

2.6.6 ASL_dbspdi, ASL_rbspdi 実対称行列の行列式と逆行列

(1) **機 能**

修正コレスキー法で LDL^T 分解された実対称行列 A (2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbspdi  (a, lna, n, ipvt, det, isw, wk);
```

単精度関数:

```
ierr = ASL_rbspdi  (a, lna, n, ipvt, det, isw, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	LDL^T 分解後の実対称行列 A(2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列 (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	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)
5	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
6	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
7	wk	{D∗ / R∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LDL^T}$ 分解しておく必要がある.
分解は 2.6.2 $\left\{\begin{matrix}\mathrm{ASL\_dbspud}\\ \mathrm{ASL\_rbspud}\end{matrix}\right\}$, 2.6.3 $\left\{\begin{matrix}\mathrm{ASL\_dbspuc}\\ \mathrm{ASL\_rbspuc}\end{matrix}\right\}$, 2.6.1 $\left\{\begin{matrix}\mathrm{ASL_dbspsl}\\ \mathrm{ASL_rbspsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 入力時の配列 a には, 上三角行列 L^T が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはり対称行列であるので, 上三角部分のみが配列 a に格納される. この関数は配列 a の上三角部分のみを使用する (2.6.1 図 2−6 参照).

(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.

(d) 行列式の値は次の式によって与えられる.
$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$
このとき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.6.7 ASL_dbsplx, ASL_rbsplx
連立 1 次方程式の解の改良 (実対称行列)

(1) **機 能**

実対称行列 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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbsplx  (a, lna, n, ald, b, x, & itol, nit, ipvt, wk);
```

単精度関数:

```
ierr = ASL_rbsplx  (a, lna, n, ald, b, x, & itol, nit, ipvt, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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)
2	lna	I	1	入 力	配列 a, ald の整合寸法
3	n	I	1	入 力	行列 A の次数
4	ald	{D∗ R∗}	lna×n	入 力	LDL^T 分解後の係数行列 A(注意事項 (a) 参照)
5	b	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
6	x	{D∗ R∗}	n	入 力	近似解$\boldsymbol{x}$
				出 力	反復改良された解$\boldsymbol{x}$
7	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似値 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	ipvt	I*	n	出 力	ピボッティング情報 (注意事項 (a) 参照)
10	wk	{D∗ R∗}	n	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.6.1 $\left\{\begin{matrix}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{matrix}\right\}$ または 2.6.4 $\left\{\begin{matrix}\text{ASL\_dbspls}\\ \text{ASL\_rbspls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.6.1 $\left\{\begin{matrix}\text{ASL\_dbspsl}\\ \text{ASL\_rbspsl}\end{matrix}\right\}$, 2.6.2 $\left\{\begin{matrix}\text{ASL\_dbspud}\\ \text{ASL\_rbspud}\end{matrix}\right\}$ または 2.6.3 $\left\{\begin{matrix}\text{ASL\_dbspuc}\\ \text{ASL\_rbspuc}\end{matrix}\right\}$ によって分解された係数行列 A とその時得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.7 実対称行列 (2 次元配列型) (上三角型) (軸選択なし)

2.7.1 ASL_dbsmsl, ASL_rbsmsl
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) 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_dbsmsl  (a, lna, n, b, w1);
```

単精度関数:

```
ierr = ASL_rbsmsl  (a, lna, n, b, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	w1	{D∗ R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(5) エラーインディケータ (戻り値)

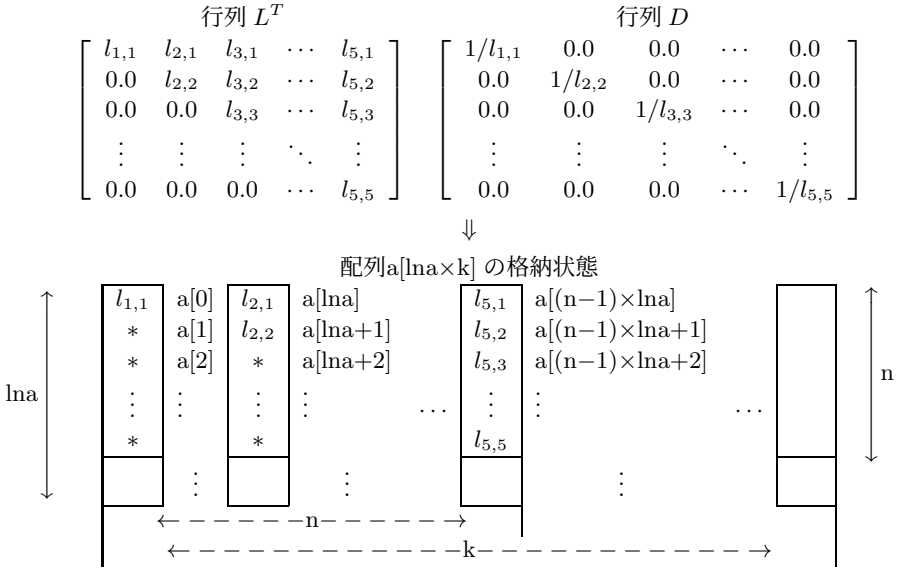
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	b[0] ← b[0]/a[0] とする.
2100	係数行列 A の $\mathrm{LDL^T}$ 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
$4000+i$	係数行列 A の $\mathrm{LDL^T}$ 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(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 2.7.4 $\left\{\begin{matrix}\text{ASL_dbsmls}\\ \text{ASL_rbsmls}\end{matrix}\right\}$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 2–7　行列 L^T 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

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

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_dbsmsl */

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

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

    int i,j;

    FILE *fp;

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

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

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

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

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

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

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

```
    fclose( fp );

    ierr = ASL_dbsmsl(a, na, n, b, wk);

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


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

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


    return 0;
}
```

(d) 出力結果

```
*** ASL_dbsmsl ***

 ** 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
```

2.7.2 ASL_dbsmud, ASL_rbsmud
実対称行列の $\mathrm{LDL^T}$ 分解 (軸選択なし)

(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_dbsmud  (a, lna, n, w1);
```

単精度関数:

```
ierr = ASL_rbsmud  (a, lna, n, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	w1	{D∗ / R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.
$4000 + i$	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 (2.7.1 図 2−7 参照).

2.7.3 ASL_dbsmuc, ASL_rbsmuc
実対称行列の $\mathrm{LDL^T}$ 分解と条件数 (軸選択なし)

(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 $\mathrm{LDL^T}$ 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
5	w1	{D∗ / R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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 $\mathrm{LDL^T}$ 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.
$4000 + i$	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.	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, 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 (2.7.1 図 2−7 参照).

(b) 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.

2.7.4 ASL_dbsmls, ASL_rbsmls
連立 1 次方程式 (LDL$^{\mathrm{T}}$ 分解後の実対称行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL$^{\mathrm{T}}$ 分解された実対称行列 A(2 次元配列型) (上三角型) を係数行列とする連立 1 次方程式を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbsmls  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbsmls  (a, lna, n, b);
```

(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	LDL^T 分解後の係数行列 A(実対称行列, 2 次元配列型, 上三角型) (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	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL$^{\mathrm{T}}$ 分解しておく必要がある. 通常は 2.7.2 $\left\{\begin{matrix}\mathrm{ASL_dbsmud}\\ \mathrm{ASL_rbsmud}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.7.3 $\left\{\begin{matrix}\mathrm{ASL_dbsmuc}\\ \mathrm{ASL_rbsmuc}\end{matrix}\right\}$を使用する. また, 2.7.1 $\left\{\begin{matrix}\mathrm{ASL_dbsmsl}\\ \mathrm{ASL_rbsmsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL$^{\mathrm{T}}$ 分解を利用することもできる. 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 2.7.5 $\left\{\begin{matrix}\mathrm{ASL\_dbsmms}\\ \mathrm{ASL\_rbsmms}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 配列 a には, 上三角行列 L^T が格納されていなければならない. 対角行列 D と下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい. この関数は配列 a の上三角部分のみを使用する (2.7.1 図 2−7 参照).

2.7.5 ASL_dbsmms, ASL_rbsmms
多重右辺連立 1 次方程式 (LDL$^{\mathrm{T}}$ 分解後の実対称行列) (軸選択なし)

(1) **機 能**

LDL^T 分解された実行列 A(上三角型) を係数行列とする連立 1 次方程式 $LDL^T\boldsymbol{x_i} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く. 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_dbsmms (a, lna, n, b, lnb, m);

単精度関数:

ierr = ASL_rbsmms (a, lna, n, b, lnb, m);

(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	LDL$^{\mathrm{T}}$ 分解後の係数行列 A(実対称行列, 上三角型) (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	b	{D∗ / R∗}	lnb×m	入 力	定数ベクトル$\boldsymbol{b_i}$からなる行列 $[\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_m}]$
				出 力	解ベクトル$\boldsymbol{x_i}$からなる行列 $[\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_m}]$
5	lnb	I	1	入 力	配列 b の整合寸法
6	m	I	1	入 力	右辺ベクトルの数
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了	
1000	n = 1 であった.	b[lna × $(i-1)$] ← b[lna × $(i-1)$]/a[0] $(i = 1, 2, \cdots, \mathrm{m})$ 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.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL^T 分解しておく必要がある. 通常は関数 2.7.2 $\left\{\begin{matrix}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.7.3 $\left\{\begin{matrix}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{matrix}\right\}$ を使用する.

また,2.7.1 $\left\{\begin{matrix}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{matrix}\right\}$ を使用して, 同一の係数行列 A を持つ連立 1 方程式をすでに解いている場合は, その出力として得られる LDL^T 分解を利用することもできる.

(b) 入力時の配列 a には, 上三角行列 L^T が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはり対称行列であるので, 上三角部分のみが配列 a に格納される. この関数は配列 a の上三角部分のみを使用する.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ 4 & 9 \\ -4 & 13 \end{bmatrix}$$

を解く.

(b) 入力データ

係数行列 A, lna=10, n=4, 定数ベクトル$\boldsymbol{b}$からなる行列 B, lnb=10, m=2

(c) 主プログラム

```
/* C interface example for ASL_dbsmms */

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

int main()
{
    double *a;
    int lna=11;
    int n=4;
    double *b;
    int lnb=11;
    int m=2;
    double *wk;
    int ierr_ud,ierr_ms;
    int i,j;
    FILE *fp;

    fp = fopen( "dbsmms.dat", "r" );

    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

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

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

    wk = ( double * )malloc((size_t)( sizeof(double) * (n) ));
    if( b == NULL )
```

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

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", m );

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

    ierr_ud = ASL_dbsmud(a, lna, n, wk);

    if( ierr_ud != 0 ) {
        printf( "\tierr ( ASL_dbsmud )= %6d\n", ierr_ud );
        return 0;
    }

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

    fclose( fp );

    ierr_ms = ASL_dbsmms(a, lna, n, b, lnb, m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr ( ASL_dbsmud )= %6d\n\n", ierr_ud );
    printf( "\tierr ( ASL_dbsmms )= %6d\n",   ierr_ms );

    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "%8.3g", b[i+lnb*j] );
        }
        printf( "\n" );
    }

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

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbsmms ***

 ** Input **

n =      4
m =      2

Coefficient Matrix a

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

Constant Vectors b

       1      -2
      -1       1
       4       9
      -4      13

 ** Output **

ierr ( ASL_dbsmud )=      0

ierr ( ASL_dbsmms )=      0

Solution

       1      -2
      -1       1
       2       1
      -2       3
```

2.7.6 ASL_dbsmdi, ASL_rbsmdi
実対称行列の行列式と逆行列 (軸選択なし)

(1) **機 能**

修正コレスキー法で $\mathrm{LDL}^{\mathrm{T}}$ 分解された実対称行列 A(2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbsmdi  (a, lna, n, det, isw, w1);
```

単精度関数:

```
ierr = ASL_rbsmdi  (a, lna, n, det, isw, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	$\mathrm{LDL}^{\mathrm{T}}$ 分解後の実対称行列 A(2 次元配列型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列 (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	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
5	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
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)

(4) **制限条件**

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

(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.	det[0] ← a[0], det[1] ← 0.0, a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LDL^T}$ 分解しておく必要がある.
分解は 2.7.2 $\left\{\begin{matrix}\mathrm{ASL\_dbsmud}\\ \mathrm{ASL\_rbsmud}\end{matrix}\right\}$, 2.7.3 $\left\{\begin{matrix}\mathrm{ASL\_dbsmuc}\\ \mathrm{ASL\_rbsmuc}\end{matrix}\right\}$, 2.7.1 $\left\{\begin{matrix}\mathrm{ASL_dbsmsl}\\ \mathrm{ASL_rbsmsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 入力時の配列 a には, 上三角行列 L^T が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはり対称行列であるので, 上三角部分のみが配列 a に格納される. この関数は配列 a の上三角部分のみを使用する (2.7.1 図 2−7 参照).

(c) 行列式の値は次の式によって与えられる.
$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$
このとき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている.

(d) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.7.7 ASL_dbsmlx, ASL_rbsmlx
連立 1 次方程式の解の改良 (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)(上三角型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbsmlx  (a, lna, n, ald, b, x, & itol, nit, w1);
```

単精度関数:

```
ierr = ASL_rbsmlx  (a, lna, n, ald, b, x, & itol, nit, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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)
2	lna	I	1	入 力	配列 a, ald の整合寸法
3	n	I	1	入 力	行列 A の次数
4	ald	{D∗ / R∗}	lna×n	入 力	$\mathrm{LDL^T}$ 分解後の係数行列 A(注意事項 (a) 参照)
5	b	{D∗ / R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
6	x	{D∗ / R∗}	n	入 力	近似解$\boldsymbol{x}$
				出 力	反復改良された解$\boldsymbol{x}$
7	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似値 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	w1	{D∗ / R∗}	n	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.7.1 $\left\{\begin{matrix}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{matrix}\right\}$ または 2.7.4 $\left\{\begin{matrix}\text{ASL\_dbsmls}\\ \text{ASL\_rbsmls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.7.1 $\left\{\begin{matrix}\text{ASL\_dbsmsl}\\ \text{ASL\_rbsmsl}\end{matrix}\right\}$, 2.7.2 $\left\{\begin{matrix}\text{ASL\_dbsmud}\\ \text{ASL\_rbsmud}\end{matrix}\right\}$ または 2.7.3 $\left\{\begin{matrix}\text{ASL\_dbsmuc}\\ \text{ASL\_rbsmuc}\end{matrix}\right\}$ によって分解された係数行列 A を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.8 実対称行列 (2 次元配列型) (下三角型)(軸選択なし)

2.8.1 ASL_dbsnsl, ASL_rbsnsl
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_dbsnsl (a, lna, n, b);

単精度関数:

ierr = ASL_rbsnsl (a, lna, n, b);

(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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000 + i$	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 0.0 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 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 2–8　行列 U^T の格納状態と行列 D の内容

行列 U^T

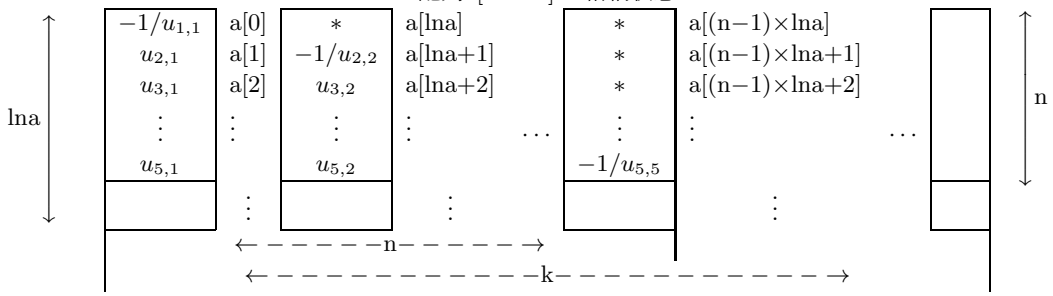
$$\begin{bmatrix} u_{1,1} & 0.0 & 0.0 & \cdots & 0.0 \\ u_{2,1} & u_{2,2} & 0.0 & \cdots & 0.0 \\ u_{3,1} & u_{3,2} & u_{3,3} & \cdots & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{5,1} & u_{5,2} & u_{5,3} & \cdots & u_{5,5} \end{bmatrix}$$

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 $\geq$ n, n $\leq$ 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_dbsnsl */

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

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

```
    int i,j;

    FILE *fp;

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

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

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

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


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

    free( a );
    free( b );


    return 0;
}
```

(d) 出力結果

```
   *** ASL_dbsnsl ***

  ** 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
```

2.8.2 ASL_dbsnud, ASL_rbsnud
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_dbsnud (a, lna, n);

単精度関数:

ierr = ASL_rbsnud (a, lna, n);

(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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $0 < n \leq lna$

(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.
$4000 + i$	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, 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 (2.8.1 図 2–8 参照).

2.8.3 ASL_dbsnls, ASL_rbsnls
連立 1 次方程式 (U^TDU 分解後の実対称行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で U^TDU 分解された実対称行列 A(2 次元配列型) (下三角型) を係数行列とする連立 1 次方程式を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbsnls  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbsnls  (a, lna, n, b);
```

(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	U^TDU 分解後の係数行列 A(実対称行列, 2 次元配列型, 下三角型) (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	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{U^TDU}$ 分解しておく必要がある. 通常は 2.8.2 $\left\{\begin{matrix}\mathrm{ASL_dbsnud}\\ \mathrm{ASL_rbsnud}\end{matrix}\right\}$を使用して分解する. また, 2.8.1 $\left\{\begin{matrix}\mathrm{ASL_dbsnsl}\\ \mathrm{ASL_rbsnsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる $\mathrm{U^TDU}$ 分解を利用することもできる.

(b) 配列 a には, 下三角行列 U^T が格納されていなければならない. 対角行列 D と上三角行列 U は U^T より算出されるので, 配列 a には格納されていなくてよい. この関数は配列 a の下三角部分のみを使用する (2.8.1 図 2−8 参照).

2.9 エルミート行列 (2 次元配列型) (上三角型) (実数引数型)

2.9.1 ASL_zbhpsl, ASL_cbhpsl
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_zbhpsl (ar, ai, lna, n, br, bi, ipvt, w1);

単精度関数:

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

(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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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. br[0] ← br[0]/ar[0], bi[0] ← bi[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.
$4000+i$	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.	

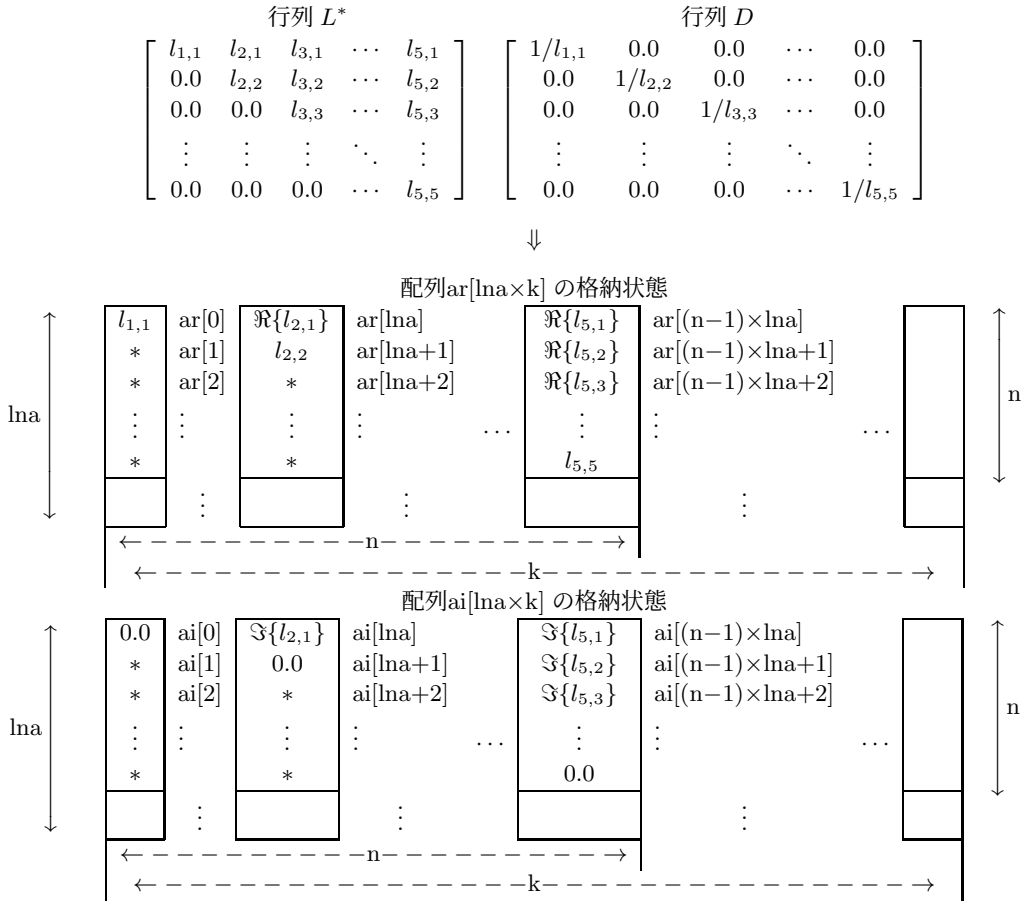
(6) **注意事項**

(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 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 2−9 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 ≤ 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 2−9　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.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_zbhpsl */

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

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

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

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

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

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

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

br = ( double * )malloc((size_t)( sizeof(double) * n ));
if( br == NULL )
{
    printf( "no enough memory for array br\n" );
    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_zbhpsl(ar, ai, na, n, br, bi, ipvt, w1);

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

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

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( ipvt );
    free( w1 );


    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhpsl ***

 ** 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 , 8.88e-17)
  x[     2] = (-4.97e-17 ,        1)
  x[     3] = (-4.17e-17 ,        1)
```

2.9.2 ASL_zbhpud, ASL_cbhpud
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_zbhpud  (ar, ai, lna, n, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbhpud  (ar, ai, lna, n, ipvt, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000 + i$	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 (2.9.1 図 2−9 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.

2.9.3 ASL_zbhpuc, ASL_cbhpuc
エルミート行列の LDL* 分解と条件数

(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* 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhpuc  (ar, ai, lna, n, ipvt, & cond, w1);
```

単精度関数:

```
ierr = ASL_cbhpuc  (ar, ai, lna, n, ipvt, & cond, w1);
```

(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	$A = LDL^*$ 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	cond	{D*/R*}	1	出 力	条件数の逆数
7	w1	{D*/R*}	2 × n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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 $\leftarrow$ 1.0 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.
$4000+i$	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.	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, 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 (2.9.1 図 2−9 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 $\leq$ 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.

(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.

2.9.4 ASL_zbhpls, ASL_cbhpls
連立 1 次方程式 (LDL* 分解後のエルミート行列)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhpls  (ar, ai, lna, n, br, bi, ipvt);
```

単精度関数:

```
ierr = ASL_cbhpls  (ar, ai, lna, n, br, bi, ipvt);
```

(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	LDL* 分解後の係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型) (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	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	入 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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]/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.9.2 {ASL_zbhpud / ASL_cbhpud} を使用して分解するが, 条件数も求めたい場合は 2.9.3 {ASL_zbhpuc / ASL_cbhpuc} を使用する. また, 2.9.1 {ASL_zbhpsl / ASL_cbhpsl} を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる. 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 2.9.5 {ASL_zbhpms / ASL_cbhpms} を用いて計算する方が効率良く解が求まる.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されていなくてよい.
この関数は配列 ar, ai の上三角部分のみを使用する.

(c) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL^* 分解を行う関数によって与えられる.

2.9.5 ASL_zbhpms, ASL_cbhpms
多重右辺連立 1 次方程式 (LDL* 分解後のエルミート行列)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhpms  (ar, ai, lna, n, br, bi, lnb, m, ipvt);
```

単精度関数:

```
ierr = ASL_cbhpms  (ar, ai, lna, n, br, bi, lnb, m, ipvt);
```

(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	LDL* 分解後の係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	ai	{D*, R*}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型) (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	br	{D*, R*}	lnb×m	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*}	lnb×m	入 力	定数ベクトル$\boldsymbol{b}$の虚部
				出 力	解$\boldsymbol{x}$の虚部
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	m	I	1	入 力	右辺ベクトルの数 m
9	ipvt	I*	n	入 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)
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	br[lnb × (i − 1)] ← br[lnb × (i − 1)]/ar[0], bi[lnb × (i − 1)] ← bi[lnb × (i − 1)]/ar[0] ($i = 1, 2, \cdots, m$) 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.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\ \text{ASL\_cbhpud}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\ \text{ASL\_cbhpuc}\end{matrix}\right\}$ を使用する. また, 2.9.1 $\left\{\begin{matrix}\text{ASL\_zbhpsl}\\ \text{ASL\_cbhpsl}\end{matrix}\right\}$ を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されていなくてよい.
この関数は配列 ar, ai の上三角部分のみを使用する.

(c) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL* 分解を行う関数によって与えられる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

を解く. ただし, $i = \sqrt{-1}$.

(b) 入力データ
LDL* 後の係数行列の A, lna = 11, n = 4, m, 定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, \mathrm{M})$

(c) 主プログラム

```
/*      C interface example for ASL_zbhpms */

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

int main()
{
    double *ar, *ai, *br, *bi;
    double *wk;
    int *ipvt;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

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

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

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

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

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

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

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

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

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

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", n );

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

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

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( j=0 ; j<m ; j++ )
{
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf", &br[i+nb*j], &bi[i+nb*j]);
    }
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", br[i+nb*j],bi[i+nb*j] );
    }
    printf( "\n" );
}
fclose( fp );

ierr = ASL_zbhpud(ar, ai, na, n, ipvt,wk);
```

```
    ierr = ASL_zbhpms(ar, ai, na, n, br, bi, nb, m, ipvt);

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

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", br[i+nb*j],bi[i+nb*j] );
        }
        printf( "\n" );
    }

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( wk );
    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhpms ***

 ** Input **

n =      4
m =      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) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-4.28e-16 ,         1) ( 5.35e-17 ,         1) (        1 ,         0)
(        1 ,  8.88e-17) (        1 , -1.78e-16) (-1.78e-16 ,         1) (        0 ,         1)
(-4.97e-17 ,         1) (        1 , -1.33e-16) (        1 ,         0) (        0 ,         1)
(-4.17e-17 ,         1) ( 8.34e-17 ,         1) (        1 , -8.34e-17) (        1 , -8.34e-17)
```

2.9.6 ASL_zbhpdi, ASL_cbhpdi
エルミート行列の行列式と逆行列

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 A (2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhpdi  (ar, ai, lna, n, ipvt, det, isw, w1);
```

単精度関数:

```
ierr = ASL_cbhpdi  (ar, ai, lna, n, ipvt, det, isw, w1);
```

(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	LDL* 分解後のエルミート行列 A の実部 (2 次元配列型)(上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の実部 (注意事項 (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	LDL* 分解後のエルミート行列 A の虚部 (2 次元配列型)(上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の虚部 (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	ipvt	I*	n	入 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 (d) 参照)
6	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
7	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
8	w1	{D∗ / R∗}	n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 ar[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある.
分解は, 2.9.2 $\left\{\begin{matrix}\text{ASL\_zbhpud}\\ \text{ASL\_cbhpud}\end{matrix}\right\}$, 2.9.3 $\left\{\begin{matrix}\text{ASL\_zbhpuc}\\ \text{ASL\_cbhpuc}\end{matrix}\right\}$, 2.9.1 $\left\{\begin{matrix}\text{ASL_zbhpsl}\\ \text{ASL_cbhpsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはりエルミート行列であるので, 上三角部分のみが A に格納される.
この関数は配列 ar, ai の上三角部分のみを使用する (2.9.1 図 2−9 参照).

(c) 行列式の値は次の式で与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
このとき, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(d) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL^* 分解を行う関数によって与えられる.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.9.7 ASL_zbhplx, ASL_cbhplx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhplx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbhplx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, ipvt, w1);
```

(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)
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)
3	lna	I	1	入 力	配列 ar, ai, alr, ali の整合寸法
4	n	I	1	入 力	行列 A の次数
5	alr	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の実部 (注意事項 (a) 参照)
6	ali	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (注意事項 (a) 参照)
7	br	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の実部
8	bi	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の虚部
9	xr	{D∗ / R∗}	n	入 力	近似解 $\boldsymbol{x}$ の実部
				出 力	反復改良された解 $\boldsymbol{x}$ の実部
10	xi	{D∗ / R∗}	n	入 力	近似解 $\boldsymbol{x}$ の虚部
				出 力	反復改良された解 $\boldsymbol{x}$ の虚部
11	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似値 (注意事項 (c) 参照)
12	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
13	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)

項番	引数と戻り値	型	大きさ	入出力	内　　容
14	w1	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	$3 \times \mathrm{n}$	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.9.1 $\left\{\begin{matrix}\mathrm{ASL\_zbhpsl} \\ \mathrm{ASL\_cbhpsl}\end{matrix}\right\}$ または 2.9.4 $\left\{\begin{matrix}\mathrm{ASL\_zbhpls} \\ \mathrm{ASL\_cbhpls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.9.1 $\left\{\begin{matrix}\mathrm{ASL\_zbhpsl} \\ \mathrm{ASL\_cbhpsl}\end{matrix}\right\}$, 2.9.2 $\left\{\begin{matrix}\mathrm{ASL\_zbhpud} \\ \mathrm{ASL\_cbhpud}\end{matrix}\right\}$ または 2.9.3 $\left\{\begin{matrix}\mathrm{ASL\_zbhpuc} \\ \mathrm{ASL\_cbhpuc}\end{matrix}\right\}$ によって分解された係数行列 A とその時得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2 \times \varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.10 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)

2.10.1 ASL_zbhrsl, ASL_cbhrsl
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_zbhrsl (ar, ai, lna, n, br, bi, w1);

単精度関数:

ierr = ASL_cbhrsl (ar, ai, lna, n, br, bi, w1);

(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	w1	{D∗ / R∗}	2 × n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(5) **エラーインディケータ (戻り値)**

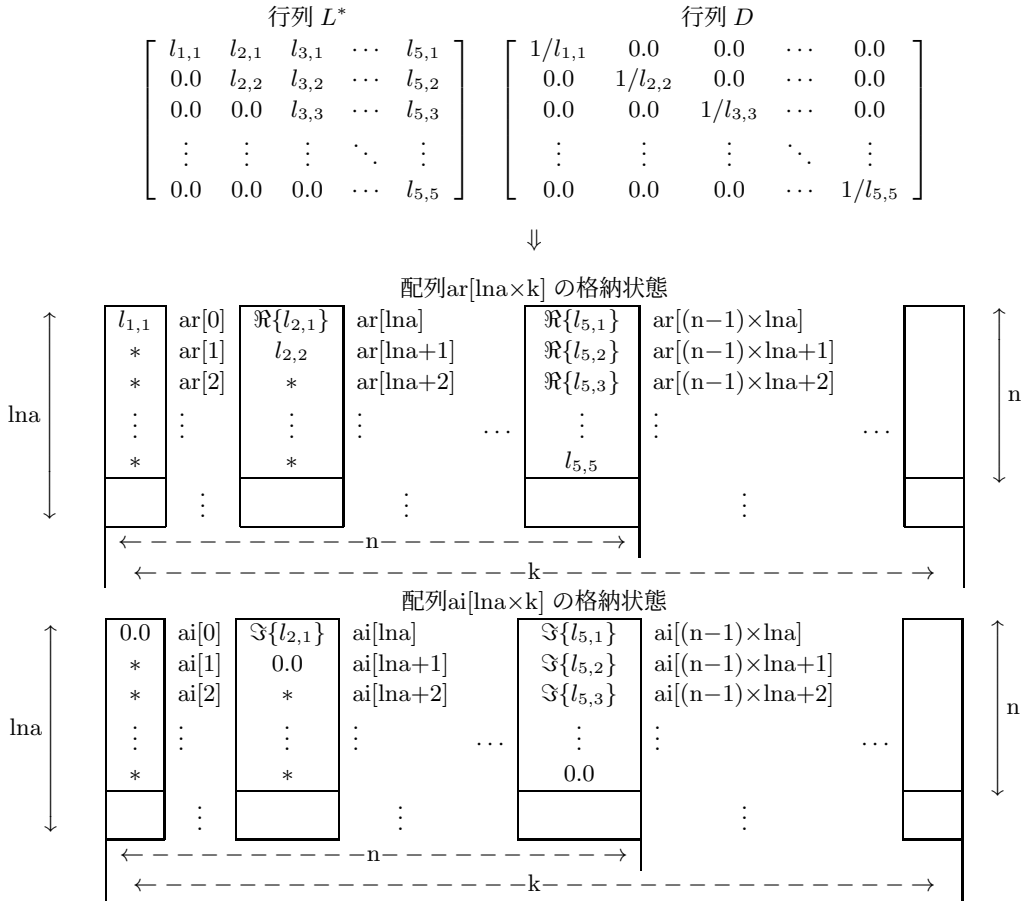
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	配列 ar, ai の内容は変更されない. br[0] ← br[0]/ar[0], bi[0] ← bi[0]/ar[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
$4000+i$	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0.0 となった. 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 2.10.4 $\left\{\begin{matrix}\text{ASL\_zbhrls}\\ \text{ASL\_cbhrls}\end{matrix}\right\}$ 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 2−10 参照)

図 2−10　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_zbhrsl */

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

int main()
{
    double *ar;
    double *ai;
    int na;
    int n;
    double *br;
    double *bi;
    double *w1;
    int ierr;
    int i,j;
```

```
FILE *fp;

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

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

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

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

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

bi = ( double * )malloc((size_t)( sizeof(double) * n ));
if( bi == NULL )
{
    printf( "no enough memory for array bi\n" );
    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_zbhrsl(ar, ai, na, n, br, bi, w1);

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

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

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( w1 );


    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhrsl ***

 ** 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 , 5.46e-17)
  x[     2] = (-1.02e-16 ,        1)
  x[     3] = (-4.17e-17 ,        1)
```

2.10.2 ASL_zbhrud, ASL_cbhrud
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_zbhrud  (ar, ai, lna, n, w1);
```

単精度関数:

```
ierr = ASL_cbhrud  (ar, ai, lna, n, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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*}	2 × n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.
$4000 + i$	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 (2.10.1 図 2−10 参照).

2.10.3 ASL_zbhruc, ASL_cbhruc
エルミート行列の LDL* 分解と条件数 (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* 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhruc  (ar, ai, lna, n, & cond, w1);
```

単精度関数:

```
ierr = ASL_cbhruc  (ar, ai, lna, n, & cond, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	cond	{D*/R*}	1	出 力	条件数の逆数
6	w1	{D*/R*}	2 × 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)

(4) **制限条件**

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

(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 $\leftarrow$ 1.0 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.
$4000 + i$	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.	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, 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 (2.10.1 図 2−10 参照).

(b) 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.

2.10.4 ASL_zbhrls, ASL_cbhrls
連立 1 次方程式 (LDL* 分解後のエルミート行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhrls  (ar, ai, lna, n, br, bi);
```

単精度関数:

```
ierr = ASL_cbhrls  (ar, ai, lna, n, br, bi);
```

(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	LDL* 分解後の係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	ai	{D*, R*}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型) (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	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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]/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.10.2 {ASL_zbhrud / ASL_cbhrud} を使用して分解するが, 条件数も求めたい場合は 2.10.3 {ASL_zbhruc / ASL_cbhruc} を使用する. また, 2.10.1 {ASL_zbhrsl / ASL_cbhrsl} を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる. 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 2.10.5 {ASL_zbhrms / ASL_cbhrms} を用いて計算する方が効率良く解が求まる.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されていなくてよい.
この関数は配列 ar, ai の上三角部分のみを使用する (2.10.1 図 2−10 参照).

2.10.5 ASL_zbhrms, ASL_cbhrms
多重右辺連立 1 次方程式 (LDL* 分解後のエルミート行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_zbhrms (ar, ai, lna, n, br, bi, lnb, m);

単精度関数:

ierr = ASL_cbhrms (ar, ai, lna, n, br, bi, lnb, m);

(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	LDL* 分解後の係数行列 A の実部 (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	ai	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (エルミート行列, 2 次元配列型, 上三角型) (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	br	{D∗ / R∗}	lnb×m	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∗}	lnb×m	入 力	定数ベクトル$\boldsymbol{b}$の虚部
				出 力	解$\boldsymbol{x}$の虚部
7	lnb	I	1	入 力	配列 br, bi の整合寸法
8	m	I	1	入 力	右辺ベクトルの数 m
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	br[lnb × (i − 1)] ← br[lnb × (i − 1)]/ar[0], bi[lnb × (i − 1)] ← bi[lnb × (i − 1)]/ar[0] ($i = 1, 2, \cdots, m$) 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.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.10.2 {ASL_zbhrud / ASL_cbhrud} を使用して分解するが, 条件数も求めたい場合は 2.10.3 {ASL_zbhruc / ASL_cbhruc} を使用する. また, 2.10.1 {ASL_zbhrsl / ASL_cbhrsl} を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 ar, ai には格納されていなくてよい.
この関数は配列 ar, ai の上三角部分のみを使用する (2.10.1 図 2−10 参照).

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

を解く. ただし, $i = \sqrt{-1}$.

(b) 入力データ
LDL* 後の係数行列の A, lna = 11, n = 4, m, 定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, \mathrm{M})$

(c) 主プログラム

```
/*      C interface example for ASL_zbhrms */

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

int main()
{
    double *ar, *ai, *br, *bi;
    double *wk;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

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

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

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

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

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

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

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

bi = ( double * )malloc((size_t)( sizeof(double) * (nb*m) ));
if( bi == NULL )
{
    printf( "no enough memory for array bi\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;
}

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", n );

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

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

printf( "\n\tConstant Vector (Real, Imaginary)\n\n");
for( j=0 ; j<m ; j++ )
{
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf %lf", &br[i+nb*j], &bi[i+nb*j]);
    }
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "(%8.3g , %8.3g) ", br[i+nb*j],bi[i+nb*j] );
    }
    printf( "\n" );
}
fclose( fp );

ierr = ASL_zbhrud(ar, ai, na, n, wk);
ierr = ASL_zbhrms(ar, ai, na, n, br, bi, nb, m);

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

printf( "\n\tSolution  (Real, Imaginary)\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t" );
    for( j=0 ; j<m ; j++ )
    {
        printf( "(%9.3g , %9.3g) ", br[i+nb*j],bi[i+nb*j] );
    }
```

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

    free( ar );
    free( ai );
    free( br );
    free( bi );
    free( wk );
    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhrms ***

 ** Input **

n =      4
m =      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) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-9.87e-16 ,         1) ( 9.87e-17 ,         1) (        1 ,  3.95e-16)
(        1 ,  6.25e-17) (        1 , -5.62e-16) (-4.68e-16 ,         1) (  2.5e-16 ,         1)
(-5.11e-17 ,         1) (        1 , -6.13e-16) (        1 ,  1.02e-16) (-2.04e-16 ,         1)
(-4.17e-17 ,         1) ( 5.01e-16 ,         1) (        1 , -2.09e-16) (        1 ,         0)
```

2.10.6 ASL_zbhrdi, ASL_cbhrdi
エルミート行列の行列式と逆行列 (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 A (2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zbhrdi (ar, ai, lna, n, det, isw, w1);

単精度関数:

ierr = ASL_cbhrdi (ar, ai, lna, n, det, isw, w1);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	ar	{D*, R*}	lna×n	入 力	LDL* 分解後のエルミート行列 A の実部 (2 次元配列型)(上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の実部 (注意事項 (b) 参照)
2	ai	{D*, R*}	lna×n	入 力	LDL* 分解後のエルミート行列 A の虚部 (2 次元配列型)(上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列の虚部 (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	det	{D*, R*}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
6	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
7	w1	{D*, R*}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 ar[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある.
分解は, 2.10.2 $\left\{\begin{matrix}\text{ASL\_zbhrud}\\ \text{ASL\_cbhrud}\end{matrix}\right\}$, 2.10.3 $\left\{\begin{matrix}\text{ASL\_zbhruc}\\ \text{ASL\_cbhruc}\end{matrix}\right\}$, 2.10.1 $\left\{\begin{matrix}\text{ASL_zbhrsl}\\ \text{ASL_cbhrsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 配列 ar, ai には, 上三角行列 L^* が格納されていなければならない. 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 には格納されなくてよい. 逆行列 A^{-1} はやはりエルミート行列であるので, 上三角部分のみが A に格納される.
この関数は配列 ar, ai の上三角部分のみを使用する (2.10.1 図 2−10 参照).

(c) 行列式の値は次の式で与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
このとき, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(d) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.10.7 ASL_zbhrlx, ASL_cbhrlx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhrlx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, w1);
```

単精度関数:

```
ierr = ASL_cbhrlx  (ar, ai, lna, n, alr, ali, br, bi, xr, xi, & itol, nit, w1);
```

(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)
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)
3	lna	I	1	入 力	配列 ar, ai, alr, ali の整合寸法
4	n	I	1	入 力	行列 A の次数
5	alr	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の実部 (注意事項 (a) 参照)
6	ali	{D∗ / R∗}	lna×n	入 力	LDL* 分解後の係数行列 A の虚部 (注意事項 (a) 参照)
7	br	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の実部
8	bi	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$の虚部
9	xr	{D∗ / R∗}	n	入 力	近似解 $\boldsymbol{x}$ の実部
				出 力	反復改良された解 $\boldsymbol{x}$ の実部
10	xi	{D∗ / R∗}	n	入 力	近似解 $\boldsymbol{x}$ の虚部
				出 力	反復改良された解 $\boldsymbol{x}$ の虚部
11	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似値 (注意事項 (c) 参照)
12	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
13	w1	{D∗ / R∗}	3 × n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.10.1 $\left\{\begin{matrix}\mathrm{ASL\_zbhrsl}\\ \mathrm{ASL\_cbhrsl}\end{matrix}\right\}$ または 2.10.4 $\left\{\begin{matrix}\mathrm{ASL\_zbhrls}\\ \mathrm{ASL\_cbhrls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.10.1 $\left\{\begin{matrix}\mathrm{ASL\_zbhrsl}\\ \mathrm{ASL\_cbhrsl}\end{matrix}\right\}$, 2.10.2 $\left\{\begin{matrix}\mathrm{ASL\_zbhrud}\\ \mathrm{ASL\_cbhrud}\end{matrix}\right\}$ または 2.10.3 $\left\{\begin{matrix}\mathrm{ASL\_zbhruc}\\ \mathrm{ASL\_cbhruc}\end{matrix}\right\}$ によって分解された係数行列 A を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2 \times \varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.11 エルミート行列 (2 次元配列型) (上三角型) (複素引数型)

2.11.1 ASL_zbhfsl, ASL_cbhfsl
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_zbhfsl (a, lna, n, b, ipvt, w1);

単精度関数:

ierr = ASL_cbhfsl (a, lna, n, b, ipvt, w1);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(5) **エラーインディケータ (戻り値)**

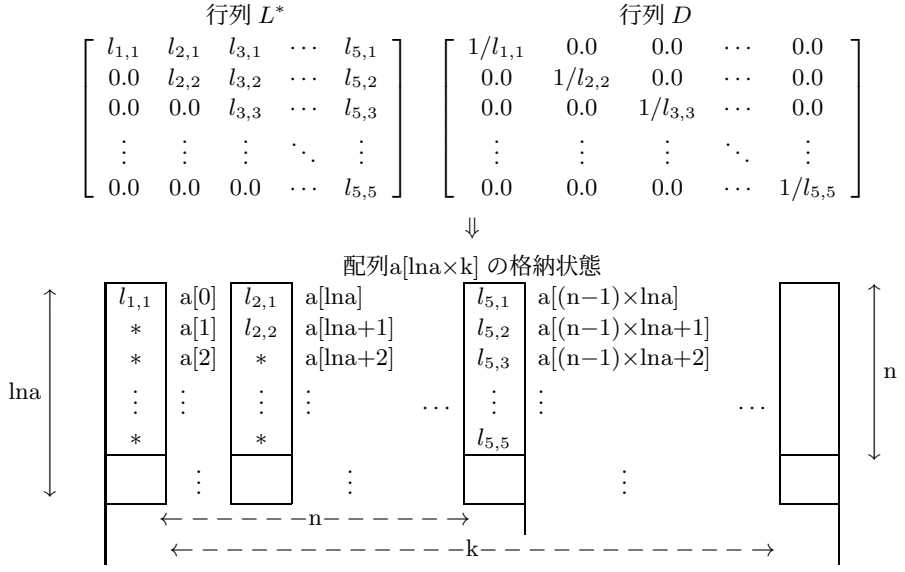
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	配列 a の内容は変更されない. b[0] ← b[0]/a[0] とする.
2100	係数行列 A の LDL* 分解の処理において, 対角要素が 0 に近いものがあった. 分解行列を使って求解もしくは逆行列を計算する場合, 精度の良い結果が得られない場合がある.	処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
$4000+i$	係数行列 A の LDL* 分解の i 段目の処理において, 対角要素が 0.0 となった. A は特異である.	

(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 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 2−11　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.

(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 ≤ 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

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

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

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

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

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

    a = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (na*n) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        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_zbhfsl(a, na, n, b, ipvt, w1);

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

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

    free( a );
    free( b );
    free( ipvt );
    free( w1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhfsl ***

 ** 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 , 8.88e-17)
  x[     2] = (-4.97e-17 ,         1)
  x[     3] = (-4.17e-17 ,         1)
```

2.11.2 ASL_zbhfud, ASL_cbhfud
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_zbhfud  (a, lna, n, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbhfud  (a, lna, n, ipvt, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000 + i$	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 (2.11.1 図 2−11 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.

2.11.3 ASL_zbhfuc, ASL_cbhfuc
エルミート行列の LDL* 分解と条件数

(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* 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	cond	{D*, R*}	1	出 力	条件数の逆数
6	w1	{Z*, C*}	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)

(4) **制限条件**

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

(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 $\leftarrow$ 1.0 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.
$4000 + i$	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.	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, 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 (2.11.1 図 2−11 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 $\leq$ 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.

(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.

2.11.4 ASL_zbhfls, ASL_cbhfls
連立 1 次方程式 (LDL* 分解後のエルミート行列)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(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	LDL* 分解後の係数行列 A (エルミート行列, 2 次元配列型, 上三角型)(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	b	{Z*, C*}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
5	ipvt	I*	n	入 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.11.2 $\left\{\begin{matrix}\text{ASL_zbhfud}\\ \text{ASL_cbhfud}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.11.3 $\left\{\begin{matrix}\text{ASL_zbhfuc}\\ \text{ASL_cbhfuc}\end{matrix}\right\}$を使用する. また, 2.11.1 $\left\{\begin{matrix}\text{ASL_zbhfsl}\\ \text{ASL_cbhfsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる. 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 2.11.5 $\left\{\begin{matrix}\text{ASL\_zbhfms}\\ \text{ASL\_cbhfms}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 a には格納されていなくてよい (2.11.1 図 2−11 参照).

(c) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL^* 分解を行う 2.11.2 $\left\{\begin{matrix}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{matrix}\right\}$, 2.11.3 $\left\{\begin{matrix}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{matrix}\right\}$, 2.11.1 $\left\{\begin{matrix}\text{ASL_zbhfsl}\\ \text{ASL_cbhfsl}\end{matrix}\right\}$によって与えられる.

2.11.5 ASL_zbhfms, ASL_cbhfms
多重右辺連立 1 次方程式 (LDL* 分解後のエルミート行列)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhfms  (a, lna, n, b, lnb, m, ipvt);
```

単精度関数:

```
ierr = ASL_cbhfms  (a, lna, n, b, lnb, m, ipvt);
```

(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	LDL* 分解後の係数行列 A (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	lna	I	1	入 力	配列 ar, ai の整合寸法
3	n	I	1	入 力	行列 A の次数
4	b	{Z*, C*}	lnb×m	入 力	定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, m)$
				出 力	解$\boldsymbol{x_i}(i = 1, 2, \cdots, m)$
5	lnb	I	1	入 力	配列 br, bi の整合寸法
6	m	I	1	入 力	右辺ベクトルの数 m
7	ipvt	I*	n	入 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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	b[lnb × (i − 1)] ← b[lnb × (i − 1)])/a[0] $(i = 1, 2, \cdots, m)$ 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.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.11.2 {ASL_zbhfud / ASL_cbhfud} を使用して分解するが, 条件数も求めたい場合は 2.11.3 {ASL_zbhfuc / ASL_cbhfuc} を使用する. また, 2.11.1 {ASL_zbhfsl / ASL_cbhfsl} を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 a には格納されていなくてよい.
この関数は配列 a の上三角部分のみを使用する (2.11.1 図 2−11 参照).

(c) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL* 分解を行う関数 2.11.2 {ASL_zbhfud / ASL_cbhfud},2.11.3 {ASL_zbhfuc / ASL_cbhfuc},2.11.1 {ASL_zbhfsl / ASL_cbhfsl} によって与えられる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

を解く. ただし, $i = \sqrt{-1}$.

(b) 入力データ
LDL* 後の係数行列の A, lna = 11, n = 4, m, 定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, \mathrm{M})$

(c) 主プログラム

```
/*      C interface example for ASL_zbhfms */

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

int main()
{
    double _Complex *a, *b;
    double *wk;
    int *ipvt;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

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

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

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

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

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

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

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

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

printf( "\tn = %6d\n", n );
printf( "\tm = %6d\n", n );

printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
for( i=0 ; i<n ; i++ )
{
    for( j=i ; j<n ; j++ )
    {
        double tmp_re, tmp_im;
        fscanf( fp, "%lf %lf", &tmp_re, &tmp_im );
        a[i+na*j] = tmp_re + tmp_im * _Complex_I;
    }
}

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

ierr = ASL_zbhfud(a, na, n, ipvt,wk);
ierr = ASL_zbhfms(a, na, n, b, nb, m, ipvt);

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

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

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    free( wk );
    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhfms ***

 ** Input **

n =      4
m =      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) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,         0) (-4.28e-16 ,         1) ( 5.35e-17 ,         1) (        1 ,         0)
(        1 ,  8.88e-17) (        1 , -1.78e-16) (-1.78e-16 ,         1) (        0 ,         1)
(-4.97e-17 ,         1) (        1 , -1.33e-16) (        1 ,         0) (        0 ,         1)
(-4.17e-17 ,         1) ( 8.34e-17 ,         1) (        1 , -8.34e-17) (        1 , -8.34e-17)
```

2.11.6 ASL_zbhfdi, ASL_cbhfdi
エルミート行列の行列式と逆行列

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 A (2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zbhfdi (a, lna, n, ipvt, det, isw, w1);

単精度関数:

ierr = ASL_cbhfdi (a, lna, n, ipvt, det, isw, w1);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{Z∗ / C∗}	lna×n	入 力	LDL* 分解後のエルミート行列 A(2 次元配列型) (上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列 (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	ipvt	I*	n	入 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 (d) 参照)
5	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
6	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
7	w1	{Z∗ / C∗}	n	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある.
分解は, 2.11.2 $\left\{\begin{matrix}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{matrix}\right\}$, 2.11.3 $\left\{\begin{matrix}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{matrix}\right\}$, 2.11.1 $\left\{\begin{matrix}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{matrix}\right\}$ のいずれかで行えばよい.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 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 には格納されていなくてよい. 逆行列 A^{-1} はやはりエルミート行列であるので, 上三角部分のみが A に格納される (2.11.1 図 2−11 参照).

(c) 行列式の値は次の式で与えられる.
$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$
このとき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている.

(d) ipvt には, LDL* 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LDL^* 分解を行う関数によって与えられる.

(e) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.11.7 ASL_zbhflx, ASL_cbhflx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhflx  (a, lna, n, al, b, x, & itol, nit, ipvt, w1);
```

単精度関数:

```
ierr = ASL_cbhflx  (a, lna, n, al, b, x, & itol, nit, ipvt, w1);
```

(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)
2	lna	I	1	入 力	配列 a, al の整合寸法
3	n	I	1	入 力	行列 A の次数
4	al	{Z∗ / C∗}	lna×n	入 力	LDL* 分解後の係数行列 A (注意事項 (a) 参照)
5	b	{Z∗ / C∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
6	x	{Z∗ / C∗}	n	入 力	近似解 $\boldsymbol{x}$
				出 力	反復改良された解 $\boldsymbol{x}$
7	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似値 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)
10	w1	{Z∗ / C∗}	n	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.11.1 $\left\{\begin{matrix}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{matrix}\right\}$ または 2.11.4 $\left\{\begin{matrix}\text{ASL\_zbhfls}\\ \text{ASL\_cbhfls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.11.3 $\left\{\begin{matrix}\text{ASL\_zbhfuc}\\ \text{ASL\_cbhfuc}\end{matrix}\right\}$, 2.11.1 $\left\{\begin{matrix}\text{ASL\_zbhfsl}\\ \text{ASL\_cbhfsl}\end{matrix}\right\}$ または 2.11.2 $\left\{\begin{matrix}\text{ASL\_zbhfud}\\ \text{ASL\_cbhfud}\end{matrix}\right\}$ によって分解された係数行列 A とその時得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.12 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)

2.12.1 ASL_zbhesl, ASL_cbhesl
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_zbhesl (a, lna, n, b);

単精度関数:

ierr = ASL_cbhesl (a, lna, n, b);

(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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000+i$	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 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 2−12　行列 L^* の格納状態と行列 D の内容

行列 L^*

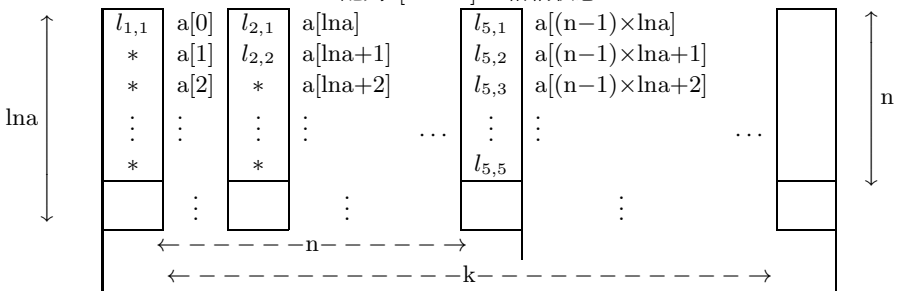
$$\begin{bmatrix} l_{1,1} & l_{2,1} & l_{3,1} & \cdots & l_{5,1} \\ 0.0 & l_{2,2} & l_{3,2} & \cdots & l_{5,2} \\ 0.0 & 0.0 & l_{3,3} & \cdots & l_{5,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & l_{5,5} \end{bmatrix}$$

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_zbhesl */

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

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

    printf( "    *** ASL_zbhesl ***\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_zbhesl(a, na, n, b);
    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_zbhesl ***

 ** 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 , 9.37e-17)
  x[     2] = (-1.02e-16 ,        1)
  x[     3] = (       0 ,        1)
```

2.12.2 ASL_zbheud, ASL_cbheud
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_zbheud  (a, lna, n);
```

単精度関数:

```
ierr = ASL_cbheud  (a, lna, n);
```

(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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.
$4000 + i$	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 (2.12.1 図 2−12 参照).

2.12.3 ASL_zbheuc, ASL_cbheuc
エルミート行列の LDL* 分解と条件数 (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* 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
5	w1	{Z∗ / C∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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 $\leftarrow$ 1.0 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.
$4000 + i$	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.	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, 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 (2.12.1 図 2−12 参照).

(b) 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.

2.12.4 ASL_zbhels, ASL_cbhels
連立 1 次方程式 (LDL* 分解後のエルミート行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhels  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_cbhels  (a, lna, n, b);
```

(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	LDL* 分解後の係数行列 A (エルミート行列, 2 次元配列型, 上三角型)(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	b	{Z*, C*}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.12.2 $\left\{\begin{matrix}\text{ASL_zbheud}\\ \text{ASL_cbheud}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.12.3 $\left\{\begin{matrix}\text{ASL_zbheuc}\\ \text{ASL_cbheuc}\end{matrix}\right\}$を使用する. また, 2.12.1 $\left\{\begin{matrix}\text{ASL_zbhesl}\\ \text{ASL_cbhesl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる. 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 2.12.5 $\left\{\begin{matrix}\text{ASL\_zbhems}\\ \text{ASL\_cbhems}\end{matrix}\right\}$ を用いて計算する方が効率良く解が求まる.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 a には格納されていなくてよい (2.12.1 図 2−12 参照).

2.12.5 ASL_zbhems, ASL_cbhems
多重右辺連立 1 次方程式 (LDL* 分解後のエルミート行列) (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 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 $LDL^*\boldsymbol{x} = \boldsymbol{b_i}(i = 1, 2, \cdots, m)$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhems  (a, lna, n, b, lnb, m);
```

単精度関数:

```
ierr = ASL_cbhems  (a, lna, n, b, lnb, m);
```

(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	LDL* 分解後の係数行列 A (エルミート行列, 2 次元配列型, 上三角型) (注意事項 (a), (b) 参照)
2	lna	I	1	入 力	配列 ar, ai の整合寸法
3	n	I	1	入 力	行列 A の次数
4	b	{Z∗ / C∗}	lnb×m	入 力	定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, m)$
				出 力	解$\boldsymbol{x_i}(i = 1, 2, \cdots, m)$
5	lnb	I	1	入 力	配列 br, bi の整合寸法
6	m	I	1	入 力	右辺ベクトルの数 m
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) $0 < \mathrm{n} \leq \mathrm{lna}, \mathrm{lnb}$

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

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	b[lnb × (i − 1)] ← b[lnb × (i − 1)])/a[0] $(i = 1, 2, \cdots, m)$ 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.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある. 通常は 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$ を使用して分解するが, 条件数も求めたい場合は 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$ を使用する. また, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ を使用して同一の係数行列 A を持つ連立 1 次方程式をすでに解いている場合は, その出力として得られる LDL* 分解を利用することもできる.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 対角行列 D と下三角行列 L は L^* より算出されるので, 配列 a には格納されていなくてよい.
この関数は配列 a の上三角部分のみを使用する (2.12.1 図 2−12 参照).

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 9 & 7+3i & 2+5i & 1+1i \\ 7-3i & 10 & 3+2i & 2+4i \\ 2-5i & 3-2i & 8 & 5+1i \\ 1-1i & 2-4i & 5-1i & 6 \end{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix} = \begin{bmatrix} 10+6i & 8+18i & 22i & 2+10i \\ 11+2i & 12+11i & 8+23i & 7+14i \\ 4+6i & 15+5i & 20+6i & 9+7i \\ 4+6i & 8+2i & 16+2i & 12+6i \end{bmatrix}$$

を解く. ただし, $i = \sqrt{-1}$.

(b) 入力データ
LDL* 後の係数行列の A, lna = 11, n = 4, m, 定数ベクトル$\boldsymbol{b_i}(i = 1, 2, \cdots, \mathrm{M})$

(c) 主プログラム

```
/*      C interface example for ASL_zbhems */

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

int main()
{
    double _Complex *a, *b;
    int na, nb, n, m, ierr, i, j;
    FILE *fp;

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

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

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

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

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

    printf( "\tn = %6d\n", n );
    printf( "\tm = %6d\n", n );

    printf( "\n\tCoefficient Matrix  (Real, Imaginary)\n\n");
    for( i=0 ; i<n ; i++ )
```

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

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

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

    ierr = ASL_zbheud(a, na, n);
    ierr = ASL_zbhems(a, na, n, b, nb, m);

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

    printf( "\n\tSolution  (Real, Imaginary)\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<m ; j++ )
        {
            printf( "(%9.3g , %9.3g) ", creal(b[i+nb*j]),cimag(b[i+nb*j]) );
        }
        printf( "\n" );
    }

    free( a );
    free( b );
    return 0;
}
```

(d) 出力結果

```
*** ASL_zbhems ***

 ** Input **

n =      4
m =      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) (       8 ,       18) (       0 ,       22) (       2 ,       10)
(      11 ,        2) (      12 ,       11) (       8 ,       23) (       7 ,       14)
(       4 ,        6) (      15 ,        5) (      20 ,        6) (       9 ,        7)
(       4 ,        6) (       8 ,        2) (      16 ,        2) (      12 ,        6)

 ** Output **

ierr =      0

Solution  (Real, Imaginary)

(        1 ,  2.96e-16) (        0 ,         1) ( -7.4e-17 ,         1) (        1 ,         0)
(        1 ,  6.25e-17) (        1 , -4.06e-16) (-6.25e-17 ,         1) (        0 ,         1)
(-1.02e-16 ,         1) (        1 , -2.04e-16) (        1 ,  1.02e-16) (        0 ,         1)
( 4.17e-17 ,         1) ( 6.67e-16 ,         1) (        1 , -4.17e-17) (        1 ,         0)
```

2.12.6 ASL_zbhedi, ASL_cbhedi
エルミート行列の行列式と逆行列 (軸選択なし)

(1) **機 能**

修正コレスキー法で LDL* 分解されたエルミート行列 A (2 次元配列型) (上三角型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhedi  (a, lna, n, det, isw, w1);
```

単精度関数:

```
ierr = ASL_cbhedi  (a, lna, n, det, isw, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{Z∗ / C∗}	lna×n	入 力	LDL* 分解後のエルミート行列 A(2 次元配列型) (上三角型) (注意事項 (a), (b) 参照)
				出 力	行列 A の逆行列 (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	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
5	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
6	w1	{Z∗ / C∗}	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)

(4) **制限条件**

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

(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.	det[0] ← a[0] det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LDL* 分解しておく必要がある.
分解は, 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$, 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ のいずれかで行えばよい.

(b) 配列 a には, 上三角行列 L^* が格納されていなければならない. 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 には格納されていなくてよい. 逆行列 A^{-1} はやはりエルミート行列であるので, 上三角部分のみが A に格納される (2.12.1 図 2−12 参照).

(c) 行列式の値は次の式で与えられる.
$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$
このとき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている.

(d) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.12.7 ASL_zbhelx, ASL_cbhelx
連立 1 次方程式の解の改良 (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 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zbhelx  (a, lna, n, al, b, x, & itol, nit, w1);
```

単精度関数:

```
ierr = ASL_cbhelx  (a, lna, n, al, b, x, & itol, nit, w1);
```

(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)
2	lna	I	1	入 力	配列 a, al の整合寸法
3	n	I	1	入 力	行列 A の次数
4	al	{Z∗ C∗}	lna×n	入 力	LDL* 分解後の係数行列 A (注意事項 (a) 参照)
5	b	{Z∗ C∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
6	x	{Z∗ C∗}	n	入 力	近似解 $\boldsymbol{x}$
				出 力	反復改良された解 $\boldsymbol{x}$
7	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似値 (注意事項 (c) 参照)
8	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
9	w1	{Z∗ C∗}	n	ワーク	作業領域
10	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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.	解は改良されない.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ または 2.12.4 $\left\{\begin{matrix}\text{ASL\_zbhels}\\ \text{ASL\_cbhels}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.12.3 $\left\{\begin{matrix}\text{ASL\_zbheuc}\\ \text{ASL\_cbheuc}\end{matrix}\right\}$, 2.12.1 $\left\{\begin{matrix}\text{ASL\_zbhesl}\\ \text{ASL\_cbhesl}\end{matrix}\right\}$ または 2.12.2 $\left\{\begin{matrix}\text{ASL\_zbheud}\\ \text{ASL\_cbheud}\end{matrix}\right\}$ によって分解された係数行列 A を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2\times\varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.13 実バンド行列 (バンド型)

2.13.1 ASL_dbbdsl, ASL_rbbdsl
連立 1 次方程式 (実バンド行列)

(1) **機 能**

実バンド行列 A(バンド型) 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_dbbdsl  (a, lma, n, mu, ml, b, ipvt);
```

単精度関数:

```
ierr = ASL_rbbdsl  (a, lma, n, mu, ml, b, ipvt);
```

(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×n	入 力	係数行列 A(実バンド行列, バンド型) (付録 B 参照)
				出 力	$A = LU$ と分解されたときの, 上三角行列 U および単位下三角行列 L(注意事項 (b) 参照)
2	lma	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	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
6	b	{D∗ / R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
				出 力	解 $\boldsymbol{x}$
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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \le \mathrm{mu} \le \mathrm{n} - 1$
$0 \le \mathrm{ml} \le \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \le \mathrm{lma}$

(5) **エラーインディケータ (戻り値)**

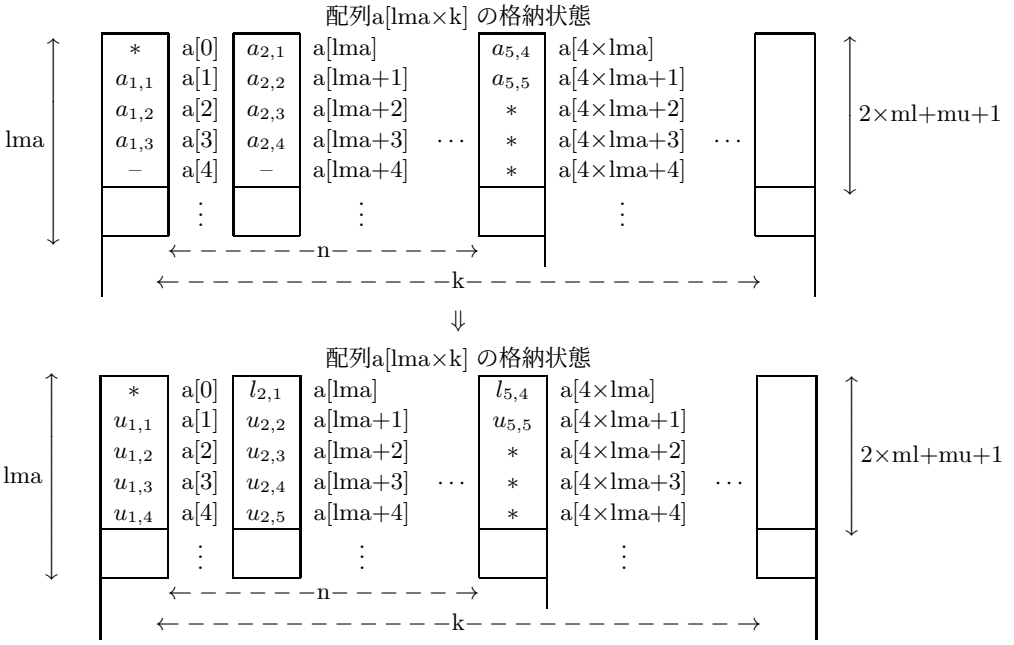
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	配列 a の内容は変更されない. b[0] ← b[0]/a[0]
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
$4000+i$	係数行列 A の LU 分解の i 段目の処理において, ピボットが 0.0 となった. A は特異に近い.	

(6) **注意事項**

(a) 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には, この関数を一度使用した後, 続けて 2.13.4 $\left\{\begin{matrix}\text{ASL_dbbdls}\\ \text{ASL_rbbdls}\end{matrix}\right\}$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 i 行 (i ≤ j) となった場合, ipvt[i − 1] に j が格納される. またこの時, 行列 A の第 i 行と第 j 行の第 i 列目以降が実際に交換されるため, 配列 a の格納領域は ml × n のサイズだけ増える. 従って, n < 2 × ml + mu + 1 の場合は, 実行列用関数を使用した方がメモリが少なくてすむ.

図 2−13　LU 分解前後の配列 a の格納状態



備　考

a. ∗ に対応する入力時の値は保証される.
b. $u_{1,4}$, $u_{2,5}$ は部分軸選択で対応する行が実際に交換された場合に設定される.
c. mu は上バンド幅, ml は下バンド幅である.
d. lma ≥ 2 × ml+mu+1, k ≥ n を満たさなければならない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 1 & -2 & 0 & 0 \\ -1 & 3 & 2 & 0 \\ 1 & -1 & 4 & -2 \\ 0 & 1 & -1 & 7 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 3 \\ -7 \\ 1 \\ 13 \end{bmatrix} \quad \text{を解く.}$$

(b) 入力データ

係数行列 A, lma=11, n=4, mu=1, ml=2, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbbdsl */

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

int main()
{
    double *a,wa;
    int ma;
    int n;
    int mu;
    int ml;
    double *b;
    int *kpvt;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

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

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

    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &mu );
    fscanf( fp, "%d", &ml );

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

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

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

    printf( "\tn = %6d\n\n", n );
    printf( "\tUpper Band Width = %6d \n\n",mu);
    printf( "\tLower Band Width = %6d \n\n",ml);
    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &wa );
            if(j-i<=mu && i-j<=ml){
            if(i-ml>=0){
            a[j-i+ml+ma*i]=wa;
            }
            else {
            a[j+ml-i+ma*i]=wa;
            }
            }
        }
    }
    printf( "\n\tCoefficient Matrix\n\n");
    printf( "\t%8c %8c %8.3g %8.3g\n",   SP,SP,  a[  2*ma],a[  3*ma] );
```

```
        printf( "\t%8c %8.3g %8.3g %8.3g\n", SP,a[1+ma],a[1+2*ma],a[1+3*ma] );
        printf( "\t%8.3g %8.3g %8.3g %8.3g\n", a[2],a[2+ma],a[2+2*ma],a[2+3*ma] );
        printf( "\t%8.3g %8.3g %8.3g\n",       a[3],a[3+ma],a[3+2*ma]           );

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

        fclose( fp );

        ierr = ASL_dbbdsl(a, ma, n, mu, ml, b, kpvt);

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

        free( a );
        free( b );
        free( kpvt );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dbbdsl ***

         ** Input **

        n =      4

        Upper Band Width =      1

        Lower Band Width =      2


        Coefficient Matrix

                                 1        1
                       -1       -1       -1
               1        3        4        7
              -2        2       -2

        Constant Vector

               3
              -7
               1
              13

         ** Output **

        ierr =      0

        Solution

          x[     0] =      -29
          x[     1] =      -16
          x[     2] =        6
          x[     3] =        5
```

2.13.2 ASL_dbbdlu, ASL_rbbdlu 実バンド行列の LU 分解

(1) 機 能

実バンド行列 A(バンド型) 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_dbbdlu  (a, lma, n, mu, ml, ipvt);
```

単精度関数:

```
ierr = ASL_rbbdlu  (a, lma, n, mu, ml, ipvt);
```

(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×n	入 力	実バンド行列 A(バンド型) (付録 B 参照)
				出 力	$A = LU$ と分解されたときの, 上三角行列 U および単位下三角行列 L(注意事項 (a), (b) 参照)
2	lma	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	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
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 (b) 参照)
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(b) $0 \le \mathrm{mu} \le \mathrm{n} - 1$
$0 \le \mathrm{ml} \le \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \le \mathrm{lma}$

(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.
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.
$4000 + i$	i 段目の処理において, ピボットが 0.0 となった. A は特異に近い.	

(6) **注意事項**

(a) 配列 a には, 単位下三角行列 L と上三角行列 U がバンド型で格納される. ただし, L の対角要素は常に 1.0 なので, 配列 a には格納されない (2.13.1 図 2−13 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. この時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] には j が格納される. またこのとき, 行列 A の第 i 行と第 j 行の第 i 列目以降が実際に交換されるため, 配列 a 中の格納領域は $ml \times n$ のサイズだけ増える. 従って, $n < 2 \times ml + mu + 1$ の場合は, 実行列用関数を使用した方がメモリが少なくてすむ (2.13.1 図 2−13 参照).

2.13.3 ASL_dbbdlc, ASL_rbbdlc
実バンド行列の LU 分解と条件数

(1) **機 能**

実バンド行列 A(バンド型) 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_dbbdlc  (a, lma, n, mu, ml, ipvt, & cond, w1);
```

単精度関数:

```
ierr = ASL_rbbdlc  (a, lma, n, mu, ml, ipvt, & cond, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lma×n	入 力	実バンド行列 A(バンド型) (付録 B 参照)
				出 力	$A = LU$ と分解したときの, 上三角行列 U および単位下三角行列 L(注意事項 (a), (b) 参照)
2	lma	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	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
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 (b) 参照)
7	cond	{D∗ / R∗}	1	出 力	条件数の逆数
8	w1	{D∗ / R∗}	n	ワーク	作業領域
9	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n > 0$

(b) $0 \le mu \le n - 1$
$0 \le ml \le n - 1$

(c) $\min(2 \times ml + mu + 1, n + ml) \le lma$

(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.
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.
$4000 + i$	i 段目の処理において, ピボットが 0.0 となった. A は特異に近い.	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, 単位下三角行列 L と上三角行列 U がバンド型で格納される. ただし, L の対角要素は常に 1.0 なので, 配列 a には格納されない (2.13.1 図 2−13 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. この時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] には j が格納される. またこのとき, 行列 A の第 i 行と第 j 行の第 i 列目以降が実際に交換されるため, 配列 a 中の格納領域は ml × n のサイズだけ増える. 従って, $n < 2 \times ml + mu + 1$ の場合は, 実行列用関数を使用した方がメモリが少なくてすむ (2.13.1 図 2−13 参照).

(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.

2.13.4 ASL_dbbdls, ASL_rbbdls
連立 1 次方程式 (LU 分解後の実バンド行列)

(1) **機 能**

ガウス法で LU 分解された実バンド行列 A(バンド型) を係数行列とする連立 1 次方程式 $LU\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbdls  (a, lma, n, mu, ml, b, ipvt);
```

単精度関数:

```
ierr = ASL_rbbdls  (a, lma, n, mu, ml, b, ipvt);
```

(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×n	入 力	LU 分解後の係数行列 A(実バンド行列, バンド型) (付録 B 参照) (注意事項 (a), (b) 参照)
2	lma	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	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
6	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
7	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1]: LU 分解の 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)
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \le \mathrm{mu} \le \mathrm{n} - 1$
$0 \le \mathrm{ml} \le \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \le \mathrm{lma}$

(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.
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.
$4000 + i$	L が 0.0 の対角要素を持つ. i は 0.0 である最初の対角要素の番号である.	

(6) **注意事項**

(a) 2.13.2 $\left\{\begin{matrix}\text{ASL_dbbdlu}\\ \text{ASL_rbbdlu}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.13.3 $\left\{\begin{matrix}\text{ASL_dbbdlc}\\ \text{ASL_rbbdlc}\end{matrix}\right\}$を使用する. また, 2.13.1 $\left\{\begin{matrix}\text{ASL_dbbdsl}\\ \text{ASL_rbbdsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立方程式を, すでに解いている場合は, その出力として得られる LU 分解を利用することもできる.

(b) 配列 a には, 単位下三角行列 L と上三角行列 U がバンド型で格納されていなければならない. ただし, 行列 L の対角要素は常に 1.0 であるので, 配列 a には格納されていなくてよい (2.13.1 図 2−13 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LU 分解を行う 2.13.2 $\left\{\begin{matrix}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{matrix}\right\}$, 2.13.3 $\left\{\begin{matrix}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{matrix}\right\}$, 2.13.1 $\left\{\begin{matrix}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{matrix}\right\}$ によって与えられる.

2.13.5 ASL_dbbddi, ASL_rbbddi 実バンド行列の行列式

(1) **機 能**

ガウス法で LU 分解された実バンド行列 A(バンド型) の行列式を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbddi  (a, lma, n, mu, ml, ipvt, det);
```

単精度関数:

```
ierr = ASL_rbbddi  (a, lma, n, mu, ml, ipvt, det);
```

(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×n	入 力	LU 分解後の実バンド行列 A(バンド型) (付録 B 参照) (注意事項 (a), (b) 参照)
2	lma	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	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
6	ipvt	I*	n	入 力	ピボッティング情報 ipvt[i − 1]: LU 分解の 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)
7	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (d) 参照)
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n > 0$

(b) $0 \le mu \le n - 1$
$0 \le ml \le n - 1$

(c) $\min(2 \times ml + mu + 1, n + ml) \le lma$

(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.	det[0] ← a[0] det[1] ← 0.0
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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を LU 分解しておく必要がある.
分解は 2.13.2 $\left\{\begin{matrix}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{matrix}\right\}$, 2.13.3 $\left\{\begin{matrix}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{matrix}\right\}$, 2.13.1 $\left\{\begin{matrix}\text{ASL_dbbdsl}\\ \text{ASL_rbbdsl}\end{matrix}\right\}$のいずれかで行えばよい.

(b) 入力時の配列 a には, 単位下三角行列 L, および上三角行列 U が格納されていなければならない. ただし, 行列 L の対角成分は常に 1.0 であるので, 配列 a には格納されていなくてよい (2.13.1 図 2−13 参照).

(c) ipvt には, LU 分解時に行った部分軸選択 (partial pivoting) についての情報が格納されていなければならない. この情報は行列 A の LU 分解を行う関数によって与えられる.

(d) 行列式の値は次の式によって与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
このとき, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(e) バンド行列の逆行列は一般に密行列であるため, この関数においては求められない.

2.13.6 ASL_dbbdlx, ASL_rbbdlx
連立 1 次方程式の解の改良 (実バンド行列)

(1) **機 能**

実バンド行列 A(バンド型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) **使用法**

倍精度関数:

ierr = ASL_dbbdlx (a, lma, n, mu, ml, alu, b, x, & itol, nit, ipvt, w1);

単精度関数:

ierr = ASL_rbbdlx (a, lma, n, mu, ml, alu, b, x, & itol, nit, ipvt, w1);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ R∗}	lma×n	入 力	係数行列 A(実バンド行列, バンド型) (付録 B 参照)
2	lma	I	1	入 力	配列 a, alu の整合寸法
3	n	I	1	入 力	行列 A の次数
4	mu	I	1	入 力	行列 A の上バンド幅
5	ml	I	1	入 力	行列 A の下バンド幅
6	alu	{D∗ R∗}	lma×n	入 力	LU 分解後の係数行列 A(注意事項 (a) 参照)
7	b	{D∗ R∗}	n	入 力	定数ベクトル $\boldsymbol{b}$
8	x	{D∗ R∗}	n	入 力	近似解 $\boldsymbol{x}$
				出 力	反復改良された解 $\boldsymbol{x}$
9	itol	I*	1	入 力	反復改良したい桁数 (注意事項 (b) 参照)
				出 力	反復改良された桁数の近似値 (注意事項 (c) 参照)
10	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
11	ipvt	I*	n	入 力	ピボッティング情報 (注意事項 (a) 参照)
12	w1	{D∗ R∗}	n	ワーク	作業領域
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \le \mathrm{mu} \le \mathrm{n} - 1$
$0 \le \mathrm{ml} \le \mathrm{n} - 1$

(c) $\min(2 \times \mathrm{ml} + \mathrm{mu} + 1, \mathrm{n} + \mathrm{ml}) \le \mathrm{lma}$

(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.	解は改良されない.
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.
$4000 + i$	alu の i 番目の対角要素が 0.0 であった.	
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.13.1 $\left\{\begin{matrix}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{matrix}\right\}$ または 2.13.4 $\left\{\begin{matrix}\text{ASL\_dbbdls}\\ \text{ASL\_rbbdls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.13.1 $\left\{\begin{matrix}\text{ASL\_dbbdsl}\\ \text{ASL\_rbbdsl}\end{matrix}\right\}$, 2.13.2 $\left\{\begin{matrix}\text{ASL\_dbbdlu}\\ \text{ASL\_rbbdlu}\end{matrix}\right\}$ または 2.13.3 $\left\{\begin{matrix}\text{ASL\_dbbdlc}\\ \text{ASL\_rbbdlc}\end{matrix}\right\}$ によって分解された係数行列 A とそのとき得られたピボッティング情報を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2 \times \varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 10 & 9 & 8 & 7 & 6 & 0 & 0 & 0 & 0 & 0 \\ 9 & 9 & 8 & 7 & 6 & 5 & 0 & 0 & 0 & 0 \\ 8 & 8 & 8 & 7 & 6 & 5 & 4 & 0 & 0 & 0 \\ 7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 0 & 0 \\ 6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 0 \\ 0 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\ 0 & 0 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 0 & 0 & 0 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 2 \\ 2 \\ 4 \\ -2 \\ -2 \\ 2 \\ 2 \\ 0 \end{bmatrix}$$

を解き, 解の改良を行う.

(b) 入力データ
係数行列 A, lna=21, n=10, mu=4, ml=4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbbdlx */

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

```
int main()
{
    double *a,*sa,wa;
    int ma;
    int n;
    int mu;
    int ml;
    double *b,*sb;
    int itol=0;
    int nnit=0;
    int *kpvt;
    double *wk;


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

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

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

    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &mu );
    fscanf( fp, "%d", &ml );

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

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

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

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

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

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

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

    for( i=0 ; i<ma ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            a[i+ma*j] = 0.0;
            sa[i+ma*j] = 0.0;
        }
    }

    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
            fscanf( fp, "%lf", &wa );
            if(j-i<=mu && i-j<=ml){
            if(i-ml>=0){
            a[j-i+ml+ma*i]=wa;
            sa[j-i+ml+ma*i]=wa;
            }
            else {
```

```
                a[j+ml-i+ma*i]=wa;
                sa[j+ml-i+ma*i]=wa;
                }
                }
            }
        }
        printf( "\n\tCoefficient Matrix a\n\n" );
        for( j=0 ; j<mu+ml+1 ; j++ ){
            printf( "\t" );
            for( i=0 ; i<n ; i++ )
            printf( "%8.3g", a[j+ma*i] );
            printf( "\n");
        }

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

        fclose( fp );


        ierr = ASL_dbbdsl(a, ma, n, mu, ml, b, kpvt);


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


        ierr = ASL_dbbdlx(sa, ma, n, mu, ml, a, sb, b, &itol, nnit, kpvt, wk);

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

        free( a );
        free( sa );
        free( b );
        free( sb );
        free( wk );
        free( kpvt );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_dbbdlx ***

 ** Input **

 n  =     10
 mu =      4
 ml =      4

Coefficient Matrix a

       0       0       0       0       6       5       4       3       2       1
       0       0       0       7       6       5       4       3       2       1
       0       0       8       7       6       5       4       3       2       1
       0       9       8       7       6       5       4       3       2       1
      10       9       8       7       6       5       4       3       2       1
       9       8       7       6       5       4       3       2       1       0
       8       7       6       5       4       3       2       1       0       0
       7       6       5       4       3       2       1       0       0       0
       6       5       4       3       2       1       0       0       0       0

Constant Vector

       8
       7
       2
       2
       4
      -2
      -2
       2
       2
       0

Original Solution

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

```
  x[     5] = -3.78e-17
  x[     6] =       -1
  x[     7] = -6.29e-16
  x[     8] =        1
  x[     9] = 4.88e-16

 ** Output **

ierr =      0

Improved Solution

  x[     0] =        1
  x[     1] = 7.89e-32
  x[     2] =       -1
  x[     3] = -6.57e-32
  x[     4] =        1
  x[     5] = -4.93e-32
  x[     6] =       -1
  x[     7] = 9.86e-32
  x[     8] =        1
  x[     9] = 2.96e-31
```

2.14 正値対称バンド行列 (対称バンド型)

2.14.1 ASL_dbbpsl, ASL_rbbpsl
連立 1 次方程式 (正値対称バンド行列)

(1) **機 能**

正値対称バンド行列 A(対称バンド型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ をコレスキー法を用いて解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbpsl  (a, lma, n, mb, b);
```

単精度関数:

```
ierr = ASL_rbbpsl  (a, lma, n, mb, b);
```

(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×n	入 力	正値対称バンド行列 A (対称バンド型)(付録 B 参照)
				出 力	$A = LL^T$ と分解したときの, 上三角行列 L^T (注意事項 (b) 参照)
2	lma	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	mb	I	1	入 力	行列 A のバンド幅
5	b	{D∗ R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(5) **エラーインディケータ (戻り値)**

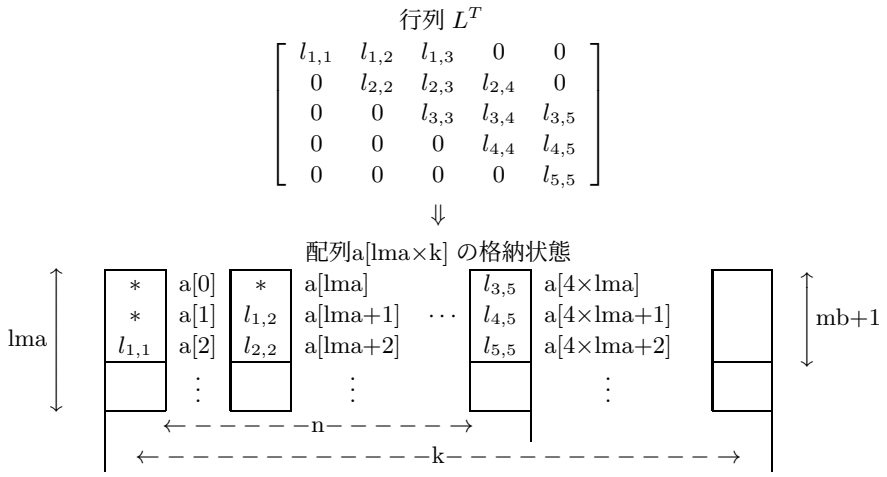
戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	a[0] $\leftarrow \sqrt{\text{a[0]}}$ b[0] $\leftarrow$ b[0]/a[0] とする.
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
$4000+i$	係数行列 A の $\mathrm{LL^T}$ 分解の i 段目の処理において, 対角要素が 0.0 以下となった. A は特異に近い.	

(6) **注意事項**

(a) 右辺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には, この関数を一度使用した後, 続けて 2.14.4 $\left\{\begin{matrix}\text{ASL_dbbpls}\\ \text{ASL_rbbpls}\end{matrix}\right\}$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 $\mathrm{LL^T}$ 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 には, 上三角行列 L^T のみが格納される. 下三角行列 L は L^T より算出されるため, 配列 a には格納されない.

図 2−14　行列 L^T の格納状態



備　考

a.　∗ に対応する入力時の値は保証される.

b.　mb は, バンド幅である.

c.　lma $\geq$ mb+1, k $\geq$ n を満たさなければならない.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 10 & -2 & 1 & 0 \\ -2 & 9 & -1 & 2 \\ 1 & -1 & 8 & -3 \\ 0 & 2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 72 \\ 9 \\ 62 \\ -4 \end{bmatrix} \quad \text{を解く.}$$

(b) 入力データ

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

(c) 主プログラム

```
/*      C interface example for ASL_dbbpsl */

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


int main()
{
    double *a,*wa;
    int ma;
    int n;
    int m;
    double *b;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

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

    printf( "    *** ASL_dbbpsl ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );

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

    wa = ( double * )malloc((size_t)( sizeof(double) * (ma*n) ));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\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\t Band Width = %6d\n", n,m );

    for( i=0 ; i<n ; i++ )
        for( j=0 ; j<n ; j++ )
            fscanf( fp, "%lf", &wa[i+ma*j]);

    for( j=0 ; j<n; j++ ){
        if(i-m<=0){
        for( i=0 ; i<=j ; i++ )
            a[i+(m-j)+ma*j] = wa[i+ma*j];
        } else {
        for( i=j-m ; i<=j ; i++ ){
        if(i>=0)a[i+(m-j)+ma*j] = wa[i+ma*j];
        }
        }
    }

    printf( "\n\tCoefficient Matrix\n\n" );
    printf( "\t%8c %8c %8.3g %8.3g\n",  SP,SP, a[  ma*2],a[  ma*3] );
    printf( "\t%8c %8.3g %8.3g %8.3g\n",SP,a[1+ma],a[1+ma*2],a[1+ma*3] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n",a[2],a[2+ma],a[2+ma*2],a[2+ma*3] );

    printf( "\n\tConstant Vector\n\n" );
```

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

    fclose( fp );

    ierr = ASL_dbbpsl(a, ma, n, m, b);

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

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

    free( a );
    free( wa );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbbpsl ***

 ** Input **

 n =      4
 Band Width =      2

Coefficient Matrix

                          1        2
               -2        -1       -3
      10        9         8        7

Constant Vector

      72
       9
      62
      -4

 ** Output **

ierr =      0

Solution

  x[     0] =        7
  x[     1] =        3
  x[     2] =        8
  x[     3] =        2
```

2.14.2 ASL_dbbpuu, ASL_rbbpuu 正値対称バンド行列の $\mathrm{LL^T}$ 分解

(1) 機 能

正値対称バンド行列 A(対称バンド型) をコレスキー法を用いて $\mathrm{LL^T}$ 分解する.

(2) 使用法

倍精度関数:

```
ierr = ASL_dbbpuu  (a, lma, n, mb);
```

単精度関数:

```
ierr = ASL_rbbpuu  (a, lma, n, mb);
```

(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×n	入 力	正値対称バンド行列 A (対称バンド型) (付録 B 参照)
				出 力	$A = LL^T$ と分解されたときの, 上三角行列 L^T (注意事項 (a) 参照)
2	lma	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	mb	I	1	入 力	行列 A のバンド幅
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) $\mathrm{n} > 0$

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(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.	$\mathrm{a}[0] \leftarrow \sqrt{\mathrm{a}[0]}$ とする.
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.
$4000 + i$	i 段目の処理において, 対角要素が 0.0 以下となった.	

(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. 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 (2.14.1 図 2−14 参照).

2.14.3 ASL_dbbpuc, ASL_rbbpuc
正値対称バンド行列の LL^{T} 分解と条件数

(1) **機 能**

正値対称バンド行列 A(対称バンド型) をコレスキー法を用いて LL^{T} 分解し, 条件数を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbpuc  (a, lma, n, mb, & cond, w1);
```

単精度関数:

```
ierr = ASL_rbbpuc  (a, lma, n, mb, & cond, w1);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lma×n	入 力	正値対称バンド行列 A (対称バンド型) (付録 B 参照)
				出 力	$A = LL^T$ と分解したときの, 上三角行列 L^T (注意事項 (a) 参照)
2	lma	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	mb	I	1	入 力	行列 A のバンド幅
5	cond	{D∗ / R∗}	1	出 力	条件数の逆数
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)

(4) **制限条件**

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

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(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.	a[0] $\leftarrow \sqrt{\mathrm{a[0]}}$ cond $\leftarrow$ 1.0 とする.
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.
$4000+i$	i 段目の処理において, 対角要素が 0.0 以下となった.	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, 上三角行列 L^T のみが格納される. 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 (2.14.1 図 2−14 参照).

(b) 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.

2.14.4 ASL_dbbpls, ASL_rbbpls
連立 1 次方程式 (LL^T 分解後の正値対称バンド行列)

(1) **機 能**

コレスキー法で LL^T 分解された正値対称バンド行列 (対称バンド型) を係数行列とする連立 1 次方程式 $LL^T\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbpls  (a, lma, n, mb, b);
```

単精度関数:

```
ierr = ASL_rbbpls  (a, lma, n, mb, b);
```

(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∗}	lma×n	入 力	LL^T 分解後の係数行列 A(正値対称バンド行列, 対称バンド型) (付録 B 参照) (注意事項 (a), (b) 参照)
2	lma	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	mb	I	1	入 力	行列 A のバンド幅
5	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(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.	$\mathrm{b}[0] \leftarrow \mathrm{b}[0]/\mathrm{a}[0]^2$ とする.
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.
$4000 + i$	L^T が 0.0 以下の対角要素を持つ. i は, 0.0 以下である最初の対角要素の番号である.	

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LL^T}$ 分解しておく必要がある. 通常は 2.14.2 $\left\{\begin{matrix}\mathrm{ASL_dbbpuu}\\ \mathrm{ASL_rbbpuu}\end{matrix}\right\}$を使用して分解するが, 条件数も求めたい場合は 2.14.3 $\left\{\begin{matrix}\mathrm{ASL_dbbpuc}\\ \mathrm{ASL_rbbpuc}\end{matrix}\right\}$を使用する. また, 2.14.1 $\left\{\begin{matrix}\mathrm{ASL_dbbpsl}\\ \mathrm{ASL_rbbpsl}\end{matrix}\right\}$を使用して同一の係数行列 A を持つ連立 1 次方程式を, すでに解いている場合は, その出力として得られる $\mathrm{LL^T}$ 分解を利用することもできる.

(b) 配列 a には, 上三角行列 L^T が対称バンド型で格納されていなければならない. 下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい (2.14.1 図 2−14 参照).

2.14.5 ASL_dbbpdi, ASL_rbbpdi 正値対称バンド行列の行列式

(1) **機 能**

コレスキー法で LL^{T} 分解された正値対称バンド行列 A(対称バンド型) の行列式を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbbpdi  (a, lma, n, mb, det);
```

単精度関数:

```
ierr = ASL_rbbpdi  (a, lma, n, mb, det);
```

(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×n	入 力	LL^{T} 分解後の上三角行列 L^T (注意事項 (a), (b) 参照)
2	lma	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	mb	I	1	入 力	行列 A のバンド幅
5	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (c) 参照)
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(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.	det[0] ← a[0] det[1] ← 0.0 とする.
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.

(6) **注意事項**

(a) この関数を使用するには, 係数行列 A を $\mathrm{LL^T}$ 分解しておく必要がある.
分解は 2.14.2 $\left\{\begin{matrix}\mathrm{ASL\_dbbpuu}\\ \mathrm{ASL\_rbbpuu}\end{matrix}\right\}$, 2.14.3 $\left\{\begin{matrix}\mathrm{ASL\_dbbpuc}\\ \mathrm{ASL\_rbbpuc}\end{matrix}\right\}$, 2.14.1 $\left\{\begin{matrix}\mathrm{ASL\_dbbpsl}\\ \mathrm{ASL\_rbbpsl}\end{matrix}\right\}$ のいずれかで行う.

(b) 配列 a には, 上三角行列 L^T が対称バンド型で格納されていなければならない. 下三角行列 L は L^T より算出されるので, 配列 a には格納されていなくてよい (2.14.1 図 2−14 参照).

(c) 行列 A の行列式の値は, 次の式で与えられる.
$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$
このき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている.

(d) 正値対称バンド行列の逆行列は一般には対称な密行列であるため, この関数では求められない.

2.14.6 ASL_dbbplx, ASL_rbbplx
連立 1 次方程式の解の改良 (正値対称バンド行列)

(1) 機 能

正値対称バンド行列 A(対称バンド型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解を反復法により改良する.

(2) 使用法

倍精度関数:

```
ierr = ASL_dbbplx  (a, lma, n, mb, all, b, x, & itol, nit, w1);
```

単精度関数:

```
ierr = ASL_rbbplx  (a, lma, n, mb, all, b, x, & itol, nit, w1);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lma×n	入 力	係数行列 A(正値対称バンド行列, 対称バンド型) (付録 B 参照)
2	lma	I	1	入 力	配列 a, all の整合寸法
3	n	I	1	入 力	行列 A の次数
4	mb	I	1	入 力	行列 A バンド幅
5	all	{D∗ / R∗}	lma×n	入 力	LL^T 分解後の係数行列 A(注意事項 (a) 参照)
6	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
7	x	{D∗ / R∗}	n	入 力	近似解$\boldsymbol{x}$
				出 力	反復改良された解$\boldsymbol{x}$
8	itol	I*	1	入 力	改良したい桁数 (注意事項 (b) 参照)
				出 力	改良された桁数の近似値 (注意事項 (c) 参照)
9	nit	I	1	入 力	最大反復回数 (注意事項 (d) 参照)
10	w1	{D∗ / R∗}	n	ワーク	作業領域
11	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(b) $0 \leq \mathrm{mb} \leq \mathrm{n} - 1$

(c) $\mathrm{mb} + 1 \leq \mathrm{lma}$

(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.	解は改良されない.
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.
$4000 + i$	all の i 番目の対角要素が 0.0 以下であった.	
5000	最大反復回数以内で収束しなかった.	itol の出力値を計算し, 処理を打ち切る.
6000	解を改良できなかった.	

(6) **注意事項**

(a) この関数は, 2.14.1 $\left\{\begin{matrix}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{matrix}\right\}$ または 2.14.4 $\left\{\begin{matrix}\text{ASL\_dbbpls}\\ \text{ASL\_rbbpls}\end{matrix}\right\}$ によって得られた解を, さらに改良するものである. 従って, 入力として 2.14.1 $\left\{\begin{matrix}\text{ASL\_dbbpsl}\\ \text{ASL\_rbbpsl}\end{matrix}\right\}$, 2.14.2 $\left\{\begin{matrix}\text{ASL\_dbbpuu}\\ \text{ASL\_rbbpuu}\end{matrix}\right\}$ または 2.14.3 $\left\{\begin{matrix}\text{ASL\_dbbpuc}\\ \text{ASL\_rbbpuc}\end{matrix}\right\}$ によって分解された係数行列 A を与えなければならない.

(b) 解の改良は, 解の上位 itol 桁が修正されなくなるまで反復される. ただし, 以下の条件を満たす場合は, 解の修正が下位 1 ビット以下になるまで反復される.
itol ≤ 0 または itol $\geq -\log_{10}(2 \times \varepsilon)$ (ε : 誤差判定のための単位)

(c) 反復回数以内で, 要求された桁数が収束しなかった場合, 修正されなくなった桁数の近似値が itol に返される.

(d) nit の入力値が 0 以下の場合, 既定値として 40 がとられる.

2.15 実 3 重対角行列 (ベクトル型)

2.15.1 ASL_dbtdsl, ASL_rbtdsl
連立 1 次方程式 (実 3 重対角行列)

(1) **機 能**

実 3 重対角行列 A(ベクトル型) 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_dbtdsl  (sdl, d, sdu, n, b);
```

単精度関数:

```
ierr = ASL_rbtdsl  (sdl, d, sdu, n, b);
```

(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	sdl	{D∗ / R∗}	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の下副対角成分
				出 力	入力時の内容は保存されない.
2	d	{D∗ / R∗}	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の対角成分
				出 力	入力時の内容は保存されない.
3	sdu	{D∗ / R∗}	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の上副対角成分
				出 力	入力時の内容は保存されない.
4	n	I	1	入 力	行列 A の次数
5	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解$\boldsymbol{x}$
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) $n > 0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	b[0] ← b[0]/d[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.
$4000 + i$	i 段目の処理で, ピボットが 0.0 となった. A は特異に近い.	

(6) **注意事項**

(a) この関数では, 部分軸選択 (partial pivoting) を行っている.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 2 & 3 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 14 \\ 20 \\ 11 \end{bmatrix}$$ を解く.

(b) 入力データ

下副対角成分 sdl, 対角成分 d, 上副対角成分 sdu, n = 4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbtdsl */

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

int main()
{
    double *a1,*wa;
    double *a2;
    double *a3;
    int nn;
    double *b;
    int ierr;
    int i,j;
    char SP=' ';
    FILE *fp;

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

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

    fscanf( fp, "%d", &nn );

    a1 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    wa = ( double * )malloc((size_t)( sizeof(double) * nn * nn ));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }

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

    printf( "\tn = %6d ", nn );
    for( i=0 ; i<nn ; i++ )
    {
        for( j=0 ; j<nn ; j++ )
        {
            fscanf( fp, "%lf", &wa[i+nn*j] );
        }
    }
```

```
    for( i=0 ; i<nn ; i++ )
    {
        a2[i]=wa[i+(nn)*i];
    }
    for( i=0 ; i<nn-1 ; i++ )
    {
        a1[i+1]=wa[i+1+(nn)*i];
        a3[i]=wa[i+(nn)*(i+1)];
    }
    printf( "\n\n\tCoefficient Matrix\n\n" );
    printf( "\t%8c %8.3g %8.3g %8.3g\n",  SP,a1[1],a1[2],a1[3] );
    printf( "\t%8.3g %8.3g %8.3g %8.3g\n",a2[0],a2[1],a2[2],a2[3] );
    printf( "\t%8.3g %8.3g %8.3g\n",      a3[0],a3[1],a3[2]       );

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

    fclose( fp );

    ierr = ASL_dbtdsl(a1, a2, a3, nn, b);

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

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

    free( a1 );
    free( wa );
    free( a2 );
    free( a3 );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtdsl ***

 ** Input **

n =      4

Coefficient Matrix

                1        1        1
       2        2        2        2
       3        3        3

Constant Vector

       8
      14
      20
      11

 ** Output **

ierr =      0

Solution

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

2.15.2 ASL_dbtpsl, ASL_rbtpsl
連立 1 次方程式 (正値対称 3 重対角行列)

(1) **機 能**

正値対称 3 重対角行列 A(ベクトル型) 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_dbtpsl  (d, sd, n, b);
```

単精度関数:

```
ierr = ASL_rbtpsl  (d, sd, n, b);
```

(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	d	{D∗ R∗}	n	入 力	係数行列 A(正値対称 3 重対角行列, ベクトル型) (付録 B 参照) の対角成分
				出 力	入力時の内容は保存されない.
2	sd	{D∗ R∗}	n	入 力	係数行列 A(正値対称 3 重対角行列, ベクトル型) (付録 B 参照) の副対角成分
				出 力	入力時の内容は保存されない.
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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n > 0$

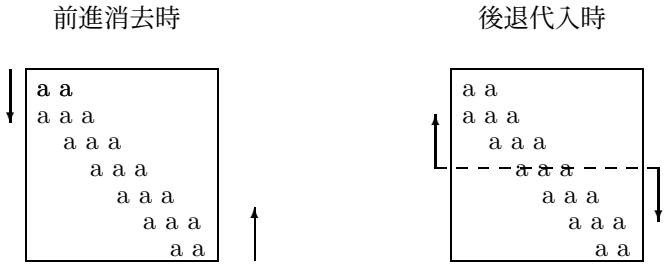
(5) **エラーインディケータ (戻り値)**

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]/d[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.
4000	処理途中で, 対角成分が 0.0 となった. A は特異に近い.	

(6) **注意事項**

(a) この関数では, ガウスの消去法を, 行列 A の対角の両端より同時に行っている. 従って, 前進消去, 後退代入は共に対角の中央で折り返して行われる.

図 2−15　正値対称 3 重対角行列に対する操作



(7) **使用例**

(a) 問 題

$$\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ を解く.

(b) 入力データ

対角成分 d, 副対角成分 sd, n = 4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbtpsl */

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

int main()
{
    double *wa;
    double *a2;
    double *a3;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nn );

    wa = ( double * )malloc((size_t)( sizeof(double) * nn * nn));
    if( wa == NULL )
    {
        printf( "no enough memory for array wa\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
```

```
    }

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

    printf( "\tn = %6d\n", nn );
    for( i=0 ; i<nn ; i++ )
    {
        for( j=0 ; j<nn ; j++ )
        {
            fscanf( fp, "%lf", &wa[i+nn*j] );
        }
    }

    for( i=0 ; i<nn ; i++ )
    {
            a2[i]=wa[i+(nn)*i];
    }
    for( i=0 ; i<nn-1 ; i++ )
    {
            a3[i]=wa[i+(nn)*(i+1)];
    }
    printf( "\n\tCoefficient Matrix\n" );
    for( i=0 ; i<nn ; i++ )
            printf( "\t%8.3g",a2[i]);
            printf( "\n");
    for( i=0 ; i<(nn-1) ; i++ )
            printf( "\t%8.3g",a3[i]);
            printf( "\n");

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

    fclose( fp );

    ierr = ASL_dbtpsl(a2, a3, nn, b);

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

    free( wa );
    free( a2 );
    free( a3 );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtpsl ***

 ** Input **

n =      4

Coefficient Matrix
      -2          -2          -2          -2
       1           1           1

Constant Vector
      -1
       0
       0
       0

 ** Output **

ierr =      0

Solution

  x[     0] =      0.8
  x[     1] =      0.6
  x[     2] =      0.4
  x[     3] =      0.2
```

2.16 実 3 重対角行列 (ベクトル型)

2.16.1 ASL_wbtdsl
連立 1 次方程式 (実 3 重対角行列)

(1) **機 能**

実 3 重対角行列 A(ベクトル型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ をサイクリック・リダクション法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_wbtdsl (sdl, d, sdu, n, b, iw, w1);

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	sdl	D*	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の下副対角成分
				出 力	入力時の内容は保存されない.
2	d	D*	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の対角成分
				出 力	入力時の内容は保存されない.
3	sdu	D*	n	入 力	係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の上副対角成分
				出 力	入力時の内容は保存されない.
4	n	I	1	入 力	行列 A の次数
5	b	D*	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解ベクトル$\boldsymbol{x}$
6	iw	I*	内容参照	ワーク	作業領域 (注意事項 (a) 参照) **大きさ:** $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	$4 \times \mathrm{n}$	ワーク	作業領域
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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]/d[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.
4000	A は特異に近い.	

(6) **注意事項**

(a) $\lfloor \log_2(n) \rfloor$ は $\log_2(n)$ を超えない最大の整数を表す.

(b) この関数は, 倍精度のみサポートされる.

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 6 & 2 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 6 & 2 \\ 0 & 0 & 1 & 6 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 19 \\ 28 \\ 27 \end{bmatrix}$$ を解く.

(b) 入力データ

下副対角成分 sdl, 対角成分 d, 上副対角成分 sdu, n = 4, 定数ベクトル$\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_wbtdsl */

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

int main()
{
    double *a1;
    double *a2;
    double *a3;
    int n;
    double *b;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,niwk;
    char SP=' ';
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
        log2n++;
    }

    a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
```

```
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }
    b = ( 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) * (4*n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

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

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

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

    fclose( fp );

    ierr = ASL_wbtdsl(a1, a2, a3, n, b, iw, wk);

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

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

    free( a1 );
    free( a2 );
    free( a3 );
    free( b );
    free( iw );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_wbtdsl ***

 ** Input **

n=     4

Coefficient Matrix

                1         1         1
       6        6         6         6
       2        2         2

Constant Vector

      10
      19
      28
      27

 ** Output **

ierr =      0

Solution

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

2.16.2 ASL_wbtdls
連立 1 次方程式 (リダクション操作後の実 3 重対角行列)

(1) **機 能**

サイクリック・リダクション法でリダクション操作された実 3 重対角行列 A (ベクトル型) を係数とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_wbtdls  (sdl, d, sdu, n, b, iw, w1);
```

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　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	sdl	D*	n	入 力	リダクション操作後の係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の下副対角成分 (注意事項 (a) 参照)
2	d	D*	n	入 力	リダクション操作後の係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の対角成分 (注意事項 (a) 参照)
3	sdu	D*	n	入 力	リダクション操作後の係数行列 A(実 3 重対角行列, ベクトル型) (付録 B 参照) の上副対角成分 (注意事項 (a) 参照)
4	n	I	1	入 力	行列 A の次数
5	b	D*	n	入 力	定数ベクトル$\boldsymbol{b}$
				出 力	解ベクトル$\boldsymbol{x}$
6	iw	I*	内容参照	入 力	リダクション操作の情報 (注意事項 (a), (b) 参照) **大きさ:** $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	$4 \times$ n	入 力	リダクション操作の情報 (注意事項 (a) 参照)
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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])/d[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.
4000	A は特異に近い (n = 1 のときのみ)	

(6) **注意事項**

(a) この関数は, 係数行列が同じで定数ベクトルの異なる複数の連立 1 次方程式を解く場合に利用できる. 最初, 2.16.1 ASL_wbtdsl を用いて係数行列のリダクション操作と求解を行った後に, この関数を用いて求解だけを繰り返し行えばよい. そのとき, 2.16.1 ASL_wbtdsl の sdl, d, sdu, iw, w1 の内容は, この関数の入力となるので保存しておかなくてはならない.

(b) $\lfloor \log_2(\mathrm{n}) \rfloor$ は $\log_2(\mathrm{n})$ を超えない最大の整数を表す.

(c) この関数は, 倍精度のみサポートされる.

(7) **使用例**

(a) 問 題

$$A = \begin{bmatrix} 6 & 2 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 6 & 2 \\ 0 & 0 & 1 & 6 \end{bmatrix}, \ \boldsymbol{b}_1 = \begin{bmatrix} 10 \\ 19 \\ 28 \\ 27 \end{bmatrix}, \ \boldsymbol{b}_2 = \begin{bmatrix} 30 \\ 26 \\ 17 \\ 8 \end{bmatrix}$$

として$\boldsymbol{x}$および$\boldsymbol{y}$についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}_1$, $A\boldsymbol{y} = \boldsymbol{b}_2$ を解く.

(b) 入力データ
下副対角成分 sdl, 対角成分 d, 上副対角成分 sdu, n = 4, 定数ベクトル $\boldsymbol{b}_1$, $\boldsymbol{b}_2$

(c) 主プログラム

```
/*      C interface example for ASL_wbtdls */

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

int main()
{
    double *a1;
    double *a2;
    double *a3;
    int n;
    double *b1;
    double *b2;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,niwk;
    char SP=' ';
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
```

```
    nn=(nn>>1);
    log2n++;
}

a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a1 == NULL )
{
    printf( "no enough memory for array a1\n" );
    return -1;
}
a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a2 == NULL )
{
    printf( "no enough memory for array a2\n" );
    return -1;
}
a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( a3 == NULL )
{
    printf( "no enough memory for array a3\n" );
    return -1;
}
b1 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( b1 == NULL )
{
    printf( "no enough memory for array b1\n" );
    return -1;
}
b2 = ( double * )malloc((size_t)( sizeof(double) * n ));
if( b2 == NULL )
{
    printf( "no enough memory for array b2\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * (4*n) ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
niwk=3*log2n+1;
iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
if( iw == NULL )
{
    printf( "no enough memory for array iw\n" );
    return -1;
}

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

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

printf( "\n\tConstant Vector\n\n" );
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &b1[i] );
}
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &b2[i] );
}
for( i=0 ; i<n ; i++ )
{
    printf( "\t%8.3g %8.3g\n", b1[i],b2[i] );
}

fclose( fp );


ierr = ASL_wbtdsl(a1, a2, a3, n, b1, iw, wk);


ierr = ASL_wbtdls(a1, a2, a3, n, b2, iw, wk);


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

printf( "\n\tSolution x\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t  x[%6d  ] = %8.3g\n", i,b1[i] );
}
```

```
    printf( "\n\tSolution y\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  y[%6d  ] = %8.3g\n", i,b2[i] );
    }

    free( a1 );
    free( a2 );
    free( a3 );
    free( b1 );
    free( b2 );
    free( iw );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_wbtdls ***

 ** Input **

n=     4

Coefficient Matrix

                1        1        1
       6        6        6        6
       2        2        2

Constant Vector

      10       30
      19       26
      28       17
      27        8

 ** Output **

ierr =      0

Solution x

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

Solution y

  y[     0  ] =        4
  y[     1  ] =        3
  y[     2  ] =        2
  y[     3  ] =        1
```

2.17 定係数型実 3 重対角行列 (スカラ型)

2.17.1 ASL_wbtcsl
連立 1 次方程式 (定係数型実 3 重対角行列)

(1) **機 能**

定係数型実 3 重対角行列 A(スカラ型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ をサイクリック・リダクション法を用いて解く.

(2) **使用法**

倍精度関数:

ierr = ASL_wbtcsl (& d, & sd, n, b, isw, iw, w1);

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	d	D*	1	入 力	係数行列 A (定係数型実 3 重対角行列, スカラ型) (注意事項 (a) 参照) の対角成分
				出 力	入力時の内容は保存されない.
2	sd	D*	1	入 力	係数行列 A (定係数型実 3 重対角行列, スカラ型) (注意事項 (a) 参照) の副対角成分
				出 力	入力時の内容は保存されない.
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	isw	I	1	入 力	係数行列 A の型を指定する (注意事項 (a) 参照) isw=1, 2, 3 or 4
6	iw	I*	内容参照	ワーク	作業領域 (注意事項 (b) 参照) **大きさ:** $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	内容参照	ワーク	作業領域 **大きさ:** $\mathrm{n} + 3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 2$
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $\mathrm{isw} \in \{1, 2, 3, 4\}$

(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]/d[0] とする.
3000	制限条件 (a), (b) を満足しなかった.	処理を打ち切る.
4000	A は特異に近い.	

(6) **注意事項**

(a) 係数行列 A は isw = 1, 2, 3 または 4 のような定係数型実 3 重対角行列である.

・isw=1 の場合

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \ddots & \ddots & \ddots \\ 0 & & & \mathrm{d} & \mathrm{sd} \\ & & & \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=2 の場合

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \ddots & \ddots & \ddots \\ 0 & & & \mathrm{d} & \mathrm{sd} \\ & & & 2 \times \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=3 の場合

$$\begin{bmatrix} \mathrm{d} & 2 \times \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \ddots & \ddots & \ddots \\ 0 & & & \mathrm{d} & \mathrm{sd} \\ & & & \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=4 の場合

$$\begin{bmatrix} \mathrm{d} & 2 \times \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \ddots & \ddots & \ddots \\ 0 & & & \mathrm{d} & \mathrm{sd} \\ & & & 2 \times \mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

上記のような係数行列は, ディリクレ型境界値問題またはノイマン型境界値問題を離散化するときに現れる.

(b) $\lfloor \log_2(\mathrm{n}) \rfloor$ は $\log_2(\mathrm{n})$ を超えない最大の整数を表す.

(c) この関数は, 倍精度のみサポートされる.

(7) 使用例

(a) 問 題

$$\begin{bmatrix} 6 & 2 & 0 & 0 \\ 2 & 6 & 2 & 0 \\ 0 & 2 & 6 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 10 \\ 10 \\ 8 \end{bmatrix}$$ を解く.

(b) 入力データ

対角成分 d, 副対角成分 sd, n = 4, isw = 1, 定数ベクトル $\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_wbtcsl */

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

int main()
{
    double *a1;
    double *a2;
    double *a3;
    double d;
    double sd;
    int n;
    double *b;
    int isw;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,nwk,niwk;
    char SP=' ';
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &isw );
    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
        log2n++;
    }

    a1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a1 == NULL )
    {
        printf( "no enough memory for array a1\n" );
        return -1;
    }
    a2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a2 == NULL )
    {
        printf( "no enough memory for array a2\n" );
        return -1;
    }
    a3 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( a3 == NULL )
    {
        printf( "no enough memory for array a3\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
```

```
        return -1;
    }
    nwk=n+3*log2n+2;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

    printf( "\n\tn  =%6d\n", n );
    printf( "\tisw=%6d\n", isw );

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

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

    d=a2[0];
    sd=a1[1];

    fclose( fp );

    ierr = ASL_wbtcsl(&d, &sd, n, b, isw, iw, wk);

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

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

    free( a1 );
    free( a2 );
    free( a3 );
    free( b );
    free( iw );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_wbtcsl ***

 ** Input **

n  =     4
isw=     1

Coefficient Matrix

                2        2        2
       6        6        6        6
       2        2        2

Constant Vector

       8
      10
      10
       8

 ** Output **

ierr =      0

Solution
```

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

2.17.2 ASL_wbtcls
連立 1 次方程式 (リダクション操作後の定係数型実 3 重対角行列)

(1) **機 能**

サイクリック・リダクション法でリダクション操作された定係数型実3 重対角行列 A(スカラ型) を係数とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_wbtcls  (d, sd, n, b, isw, iw, w1);
```

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　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	d	D	1	入 力	リダクション操作後の係数行列 A (定係数型実 3 重対角行列, スカラ型) の対角成分 (注意事項 (a), (b) 参照)
2	sd	D	1	入 力	リダクション操作後の係数行列 A (定係数型実 3 重対角行列, スカラ型) の副対角成分 (注意事項 (a), (b) 参照)
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	isw	I	1	入 力	係数行列 A の型を指定する (注意事項 (b) 参照) isw=1, 2, 3 or 4
6	iw	I*	内容参照	入 力	リダクション操作の情報 (注意事項 (a), (c) 参照) **大きさ:** $3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 1$
7	w1	D*	内容参照	入 力	リダクション操作の情報 (注意事項 (a), (d) 参照) **大きさ:** $\mathrm{n} + 3 \times \lfloor \log_2(\mathrm{n}) \rfloor + 2$
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $\mathrm{isw} \in \{1, 2, 3, 4\}$

(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]/d[0] とする.
3000	制限条件 (a), (b) を満足しなかった.	処理を打ち切る.
4000	A は特異に近い (n = 1 のときのみ)	

(6) **注意事項**

(a) この関数は, 係数行列が同じで定数ベクトルの異なる複数の連立 1 次方程式を解く場合に利用できる. 最初, 2.17.1 ASL_wbtcsl を用いて係数行列のリダクション操作と求解を行った後に, この関数を用いて求解だけを繰り返し行えばよい. そのとき, 2.17.1 ASL_wbtcsl の d, sd, iw, w1 の内容は, この関数の入力となるので保存しておかなくてはならない.

(b) 係数行列 A は isw = 1, 2, 3 または 4 の場合, 次のような定係数型実 3 重対角行列である.

・isw=1 の場合

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \\ 0 & & & \cdot & \mathrm{d} \\ & & & & \mathrm{sd} \end{bmatrix} \quad \begin{bmatrix} \cdot & \cdot & \\ \cdot & \cdot & \mathrm{sd} \\ \mathrm{sd} & \mathrm{d} & \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=2 の場合

$$\begin{bmatrix} \mathrm{d} & \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \\ 0 & & & \cdot & \mathrm{d} \\ & & & & 2 \times \mathrm{sd} \end{bmatrix} \quad \begin{bmatrix} \cdot & \cdot & \\ \cdot & \cdot & \mathrm{sd} \\ \mathrm{sd} & \mathrm{d} & \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=3 の場合

$$\begin{bmatrix} \mathrm{d} & 2 \times \mathrm{sd} & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \\ & 0 & & \cdot & \mathrm{d} \\ & & & & \mathrm{sd} \end{bmatrix} \quad \begin{bmatrix} \cdot & \cdot & \\ \cdot & \cdot & \mathrm{sd} \\ \mathrm{sd} & \mathrm{d} & \end{bmatrix}, \mathrm{d} \neq 0, \mathrm{sd} \neq 0$$

・isw=4 の場合

$$\begin{bmatrix} \mathrm{d} & 2\times\mathrm{sd} & & & & 0 \\ \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & & \\ & \mathrm{sd} & \mathrm{d} & \mathrm{sd} & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & 0 & & \cdot & \mathrm{d} & \mathrm{sd} \\ & & & & 2\times\mathrm{sd} & \mathrm{d} \end{bmatrix}, \mathrm{d}\neq 0, \mathrm{sd}\neq 0$$

上記のような係数行列は, ディリクレ型境界値問題またはノイマン型境界値問題を離散化するときに現れる.

(c) $\lfloor \log_2(\mathrm{n}) \rfloor$ は $\log_2(\mathrm{n})$ を超えない最大の整数を表す.

(d) この関数は, 倍精度のみサポートされる.

(7) 使用例

(a) 問 題

$$A = \begin{bmatrix} 6 & 2 & 0 & 0 \\ 2 & 6 & 2 & 0 \\ 0 & 2 & 6 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix},\ \boldsymbol{b}_1 = \begin{bmatrix} 8 \\ 10 \\ 10 \\ 8 \end{bmatrix},\ \boldsymbol{b}_2 = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 30 \end{bmatrix}$$

として$\boldsymbol{x}$および$\boldsymbol{y}$についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}_1$, $A\boldsymbol{y} = \boldsymbol{b}_2$ を解く.

(b) 入力データ

対角成分 d, 副対角成分 sd, n = 4, isw = 1, 定数ベクトル $\boldsymbol{b}_1$, $\boldsymbol{b}_2$

(c) 主プログラム

```
/*      C interface example for ASL_wbtcls */

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

int main()
{
    double d;
    double sd;
    int n;
    double *b1;
    double *b2;
    int isw;
    int *iw;
    double *wk;
    int ierr;
    int i,nn,log2n,nwk,niwk;
    FILE *fp;

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

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

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

    /* get floor(log2(n)) */
    nn=n;
    log2n=0;
    while (nn>1)
    {
        nn=(nn>>1);
        log2n++;
    }
```

```
    b1 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b1 == NULL )
    {
        printf( "no enough memory for array b1\n" );
        return -1;
    }
    b2 = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( b2 == NULL )
    {
        printf( "no enough memory for array b2\n" );
        return -1;
    }
    nwk=n+3*log2n+2;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    niwk=3*log2n+1;
    iw = ( int * )malloc((size_t)( sizeof(int) * niwk ));
    if( iw == NULL )
    {
        printf( "no enough memory for array iw\n" );
        return -1;
    }

    printf( "\n\tn  =%6d\n", n );
    printf( "\tisw=%6d\n", isw );
    fscanf( fp, "%lf", &d );
    fscanf( fp, "%lf", &sd );
    printf( "\n\tCoefficient Matrix\n\n" );
    printf( "\t%8.3g\n", d );
    printf( "\t%8.3g\n", sd );
    printf( "\n\tConstant Vector\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b1[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b2[i] );
    }
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g %8.3g\n", b1[i],b2[i] );
    }

    fclose( fp );


    ierr = ASL_wbtcsl(&d, &sd, n, b1, isw, iw, wk);
    ierr = ASL_wbtcls(d, sd, n, b2, isw, iw, wk);


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

    printf( "\n\tSolution x\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  x[%6d  ] = %8.3g\n", i,b1[i] );
    }

    printf( "\n\tSolution y\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t  y[%6d  ] = %8.3g\n", i,b2[i] );
    }

    free( b1 );
    free( b2 );
    free( iw );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_wbtcls ***

 ** Input **

n  =     4
isw=     1

Coefficient Matrix

       6
       2

Constant Vector

       8       10
      10       20
      10       30
```

```
       8       30

 ** Output **

ierr =      0

Solution x

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

Solution y

  y[     0  ] =        1
  y[     1  ] =        2
  y[     2  ] =        3
  y[     3  ] =        4
```

2.18 Vandermonde 行列と Toeplitz 行列

2.18.1 ASL_dbtosl, ASL_rbtosl
連立 1 次方程式 (Toeplitz 行列)

(1) **機 能**

$2 \times n - 1$ 個の要素 r_k $(k = -n+1, -n+2, \cdots, n-1)$ で構成される n 次の Toeplitz 行列 R

$$R = \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \cdots & r_{-n+2} & r_{-n+1} \\ r_1 & r_0 & r_{-1} & \cdots & r_{-n+3} & r_{-n+2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_{-1} \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

を係数行列とする連立 1 次方程式 $R\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} r_{i-j} x_j = b_i \quad (i = 1, \cdots, n)$$

または R^T を係数行列とする連立 1 次方程式 $R^T\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} r_{j-i} x_j = b_i \quad (i = 1, \cdots, n)$$

を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dbtosl (r, n, b, x, w, isw);

単精度関数:

ierr = ASL_rbtosl (r, n, b, x, w, isw);

(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	r	{D∗ / R∗}	$2 \times \mathrm{n} - 1$	入 力	Toeplitz 行列 R の構成要素 r_k $(k = -n+1, -n+2, \cdots, n-1)$
2	n	I	1	入 力	行列 R の次数 n
3	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
4	x	{D∗ / R∗}	n	出 力	解ベクトル$\boldsymbol{x}$
5	w	{D∗ / R∗}	$2 \times \mathrm{n}$	ワーク	作業領域
6	isw	I	1	入 力	処理スイッチ 1: $R\boldsymbol{x} = \boldsymbol{b}$ を解く 2: $R^T\boldsymbol{x} = \boldsymbol{b}$ を解く
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) $\mathrm{isw} \in \{1, 2\}$

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

(c) $\mathrm{r}[\mathrm{n}-1] \neq 0$

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	$\mathrm{n} = 1$ であった.	$\mathrm{x}[0] \leftarrow \mathrm{b}[0]/\mathrm{r}[\mathrm{n}-1]$ 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.	
3020	制限条件 (c) を満足しなかった.	
4000	除数 $x^{(de)}$ がゼロとなった.	
4010	除数 $g^{(de)}$ がゼロとなった.	

(6) **注意事項**

(a) この関数は行列の性質を十分活用しているので 2.2.2 $\left\{\begin{matrix}\text{ASL_dbgmsl}\\ \text{ASL_rbgmsl}\end{matrix}\right\}$を用いるよりもメモリ使用量や計算効率の点で優れているが, 反面, 行列が正則であっても原理的に解を求められない場合がある. 特に計算過程で, 除数となる $x^{(de)}$ や $g^{(de)}$ が 0 に近くなった場合, 得られる解の信頼性は保証されない (2.1.3 使用しているアルゴリズム参照).

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} r_0 & r_{-1} & r_{-2} & r_{-3} \\ r_1 & r_0 & r_{-1} & r_{-2} \\ r_2 & r_1 & r_0 & r_{-1} \\ r_3 & r_2 & r_1 & r_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

を解く.

(b) 入力データ

行列 R の成分を格納する配列 $\mathrm{r} = \{\mathrm{r}_{-3}, \mathrm{r}_{-2}, \mathrm{r}_{-1}, \mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3\}$

n=4, isw=1, 定数ベクトル $\boldsymbol{b}$

補足: $\mathrm{r} = \{\mathrm{r}_3, \mathrm{r}_2, \mathrm{r}_1, \mathrm{r}_0, \mathrm{r}_{-1}, \mathrm{r}_{-2}, \mathrm{r}_{-3}\}$, $\mathrm{isw} = 2$ とすることによって同じ問題を解くことができる.

(c) 主プログラム

```
/*      C interface example for ASL_dbtosl */

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

int main()
{
    double *r;
    int n;
    double *b;
    double *x;
    double *w;
    int isw;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

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

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

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

    r = ( double * )malloc((size_t)( sizeof(double) * (2*lna-1) ));
    if( r == NULL )
    {
        printf( "no enough memory for array r\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    w = ( double * )malloc((size_t)( sizeof(double) * (2*lna) ));
    if( w == NULL )
    {
```

```
        printf( "no enough memory for array w\n" );
        return -1;
    }
    for( i=0 ; i<2*n-1 ; i++ )
    {
        fscanf( fp, "%lf", &r[i] );
    }

    printf( "\n\tCoefficient Matrix\n\n");
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t" );
        for( j=0 ; j<n ; j++ )
        {
            printf( "%8.3g  ", r[n+i-j-1] );
        }
        printf( "\n" );
    }
    printf( "\n\tConstant Vector\n\n");
    printf( "\t" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "%8.3g  ", b[i] );
    }
    fclose( fp );

    ierr = ASL_dbtosl(r, n, b, x, w, isw);

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

    free( r );
    free( b );
    free( x );
    free( w );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtosl ***

 ** Input **

 isw =      1 n =      4

Coefficient Matrix

       1        -2        -3        -4
       2         1        -2        -3
       3         2         1        -2
       4         3         2         1

Constant Vector

      -8        -2         4        10
 ** Output **

ierr =      0

Solution

       1         1         1         1
```

2.18.2 ASL_dbtssl, ASL_rbtssl
連立 1 次方程式 (対称 Toeplitz 行列)

(1) **機 能**

n 個の要素 r_k $(k = 0, 1, \cdots, n-1)$ で構成される n 次の対称 Toeplitz 行列 R

$$R = \begin{bmatrix} r_0 & r_1 & r_2 & \cdots & r_{n-2} & r_{n-1} \\ r_1 & r_0 & r_1 & \cdots & r_{n-3} & r_{n-2} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ r_{n-2} & r_{n-3} & r_{n-4} & \cdots & r_0 & r_1 \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_1 & r_0 \end{bmatrix}$$

を係数行列とする連立 1 次方程式 $R\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} r_{|i-j|} x_j = b_i \quad (i = 1, \cdots, n)$$

を解く.

(2) **使用法**

倍精度関数:

ierr = ASL_dbtssl (r, n, b, x, w);

単精度関数:

ierr = ASL_rbtssl (r, n, b, x, w);

(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	r	{ D∗ / R∗ }	n	入 力	Toeplitz 行列 R の構成要素 r_k $(k = 0, 1, \cdots, n-1)$
2	n	I	1	入 力	行列 R の次数 n
3	b	{ D∗ / R∗ }	n	入 力	定数ベクトル$\boldsymbol{b}$
4	x	{ D∗ / R∗ }	n	出 力	解ベクトル$\boldsymbol{x}$
5	w	{ D∗ / R∗ }	n	ワーク	作業領域
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) n > 0

(b) r[0] ≠ 0

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n = 1 であった.	x[0] ← b[0]/r[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.	
4000	除数 $x^{(de)}$ がゼロとなった.	

(6) **注意事項**

(a) この関数は行列の性質を十分活用しているので 2.2.2 $\left\{\begin{matrix}\text{ASL_dbgmsl}\\ \text{ASL_rbgmsl}\end{matrix}\right\}$を用いるよりもメモリ使用量や計算効率の点で優れているが, 反面, 行列が正則であっても原理的に解を求められない場合がある. 特に計算過程で, 除数となる $x^{(de)}$ が 0 に近くなった場合, 得られる解の信頼性は保証されない (2.1.3 使用しているアルゴリズム参照).

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} r_0 & r_1 & r_2 & r_3 \\ r_1 & r_0 & r_1 & r_2 \\ r_2 & r_1 & r_0 & r_1 \\ r_3 & r_2 & r_1 & r_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

を解く.

(b) 入力データ
行列 R の成分を格納する配列 $\mathrm{r} = \{\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3\}$
n=4, 定数ベクトル $\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbtssl */

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

int main()
{
    double *r;
    int n;
    double *b;
    double *x;
    double *w;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

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

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

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

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

    ierr = ASL_dbtssl(r, n, b, x, w);

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

    free( r );
    free( b );
    free( x );
    free( w );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtssl ***

 ** Input **

 n =      4

Coefficient Matrix

       1         2         3         4
       2         1         2         3
       3         2         1         2
       4         3         2         1

Constant Vector

      10         8         8        10
 ** Output **

ierr =      0

Solution

       1         1         1         1
```

2.18.3 ASL_dbvmsl, ASL_rbvmsl
連立 1 次方程式 (Vandermonde 行列)

(1) **機 能**

相異なる n 個の要素 v_k $(k = 1, 2, \cdots, n)$ で構成される n 次の Vandermonde 行列 V

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & \cdots & v_1^{n-2} & v_1^{n-1} \\ 1 & v_2 & v_2^2 & \cdots & v_2^{n-2} & v_2^{n-1} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ 1 & v_{n-1} & v_{n-1}^2 & \cdots & v_{n-1}^{n-2} & v_{n-1}^{n-1} \\ 1 & v_n & v_n^2 & \cdots & v_n^{n-2} & v_n^{n-1} \end{bmatrix}$$

を係数行列とする連立 1 次方程式 $V\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} v_i^{j-1} x_j = b_i \quad (i = 1, \cdots, n)$$

または V^T を係数行列とする連立 1 次方程式 $V^T\boldsymbol{x} = \boldsymbol{b}$:

$$\sum_{j=1}^{n} v_j^{i-1} x_j = b_i \quad (i = 1, \cdots, n)$$

を解く. なお, Vandermonde 行列を係数とする連立 1 次方程式は本質的に悪条件であり, n が極めて小さい場合以外は解を精度良く求めることは難しい (注意事項 (a) 参照).

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbvmsl  (v, n, b, x, w, isw);
```

単精度関数:

```
ierr = ASL_rbvmsl  (v, n, b, x, w, isw);
```

(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	v	{D∗ / R∗}	n	入 力	Vandermonde 行列 V の構成要素 v_k $(k = 1, 2, \cdots, n)$
2	n	I	1	入 力	行列 V の次数 n
3	b	{D∗ / R∗}	n	入 力	定数ベクトル$\boldsymbol{b}$
4	x	{D∗ / R∗}	n	出 力	解ベクトル$\boldsymbol{x}$
5	w	{D∗ / R∗}	n	ワーク	作業領域 (注意事項 (b) 参照)
6	isw	I	1	入 力	処理スイッチ 1: $V\boldsymbol{x} = \boldsymbol{b}$ を解く 2: $V^T\boldsymbol{x} = \boldsymbol{b}$ を解く
7	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) isw $\in \{1, 2\}$

(b) n > 0

(c) v$[i-1] \neq 0$ $(i = 1, \ldots, \mathrm{n})$

(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.	x[0] ← 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.
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	演算途中でゼロ除算が発生した.	

(6) 注意事項

(a) この関数は行列の性質を活用しているので 2.2.2 {ASL_dbgmsl / ASL_rbgmsl} を用いるよりもメモリ使用量の点で優れているが, ピボッティングを行わずに逆行列を経由して解を求める分, 計算精度の点で劣る場合がある. なお, いずれにせよ, Vandermonde 行列を係数とする連立 1 次方程式は本質的に悪条件であり, n が極めて小さい場合以外は解を精度良く求めることは難しい. 倍精度の関数を用いた場合でも, 解き得る問題の大きさ n の最大値は 15 程度である. また, V^T を係数とする連立 1 次方程式の方が V を係数とする連立 1 次方程式よりも通常は性質が良い.

(b) 作業領域 w には次式で定義されるマスター多項式 $P(x)$ の各項の係数 w_j が格納される.

$$P(x) = \prod_{k=1}^{n}(x - v_k) = x^n + w_1 x^{n-1} + \cdots + w_{n-1}x + w_n$$

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 1 & v_1 & v_1^2 & v_1^3 \\ 1 & v_2 & v_2^2 & v_2^3 \\ 1 & v_3 & v_3^2 & v_3^3 \\ 1 & v_4 & v_4^2 & v_4^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

を解く.

(b) 入力データ

行列 V の成分を格納する配列 $\mathrm{v} = \{\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3, \mathrm{v}_4\}$

n=4, isw=1, 定数ベクトル $\boldsymbol{b}$

(c) 主プログラム

```
/*      C interface example for ASL_dbvmsl */

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

int main()
{
    double *v;
    int n;
    double *b;
    double *x;
    double *w;
    double t;
    int isw;
    int ierr;
    int i,j;
    int lna=4;
    FILE *fp;

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

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

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

    v = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( v == NULL )
    {
        printf( "no enough memory for array v\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    w = ( double * )malloc((size_t)( sizeof(double) * lna ));
    if( w == NULL )
    {
        printf( "no enough memory for array w\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
```

```
        fscanf( fp, "%lf", &v[i] );
    }

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

    ierr = ASL_dbvmsl(v, n, b, x, w, isw);

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

    free( v );
    free( b );
    free( x );
    free( w );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbvmsl ***

 ** Input **

 isw =      1 n =      4

Coefficient Matrix

       1         2         4         8
       1         3         9        27
       1         4        16        64
       1         5        25       125

Constant Vector

      15        40        85       156
 ** Output **

ierr =      0

Solution

       1         1         1         1
```

2.19 実上三角行列 (2 次元配列型)

2.19.1 ASL_dbtusl, ASL_rbtusl
連立 1 次方程式 (実上三角行列)

(1) **機 能**

実上三角行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbtusl  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbtusl  (a, lna, n, b);
```

(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(実上三角行列, 2 次元配列型)
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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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 の対角要素の中に, 0 に近いものがあった. 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 が 0.0 の対角要素を持つ. i は, 0.0 である最初の対角要素の番号である. 行列 A は特異である.	

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 1 & 2 & -3 & 4 \\ 0 & 4 & -1 & 1 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -10 \\ -9 \\ -3 \\ -16 \end{bmatrix}$$ 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_dbtusl */

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

int main()
{
    double *a;
    int na;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

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

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

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

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

    fclose( fp );

    ierr = ASL_dbtusl(a, na, nn, b);

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

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

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

    free( a );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtusl ***

 ** Input **

 n =      4

Coefficient Matrix
       1           2          -3           4
       0           4          -1           1
       0           0           5          -1
       0           0           0           8

Constant Vector
     -10
      -9
      -3
     -16

 ** Output **

ierr =      0

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

2.19.2 ASL_dbtuco, ASL_rbtuco 実上三角行列の条件数

(1) 機 能

実上三角行列 A(2 次元配列型) の条件数を求める.

(2) 使用法

倍精度関数:

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

単精度関数:

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

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	実上三角行列 A(2 次元配列型)
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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
5	w1	{D∗ / R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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.	cond ← 1.0 とする.
2100	係数行列 A の対角要素の中に, 0 に近いものがあった. 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.
$4000 + i$	行列 A が 0.0 の対角要素を持つ. i は, 0.0 である最初の対角要素の番号である.	

(6) **注意事項**

(a) 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.

2.19.3 ASL_dbtudi, ASL_rbtudi
実上三角行列の行列式と逆行列

(1) **機 能**

実上三角行列 A(2 次元配列型) の行列式と逆行列を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbtudi  (a, lna, n, det, isw);
```

単精度関数:

```
ierr = ASL_rbtudi  (a, lna, n, det, isw);
```

(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	実上三角行列 A(2 次元配列型)
				出 力	行列 A の逆行列 (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	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (b) 参照)
5	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める. isw=0:行列式の値と逆行列を求める. isw<0:逆行列を求める.
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

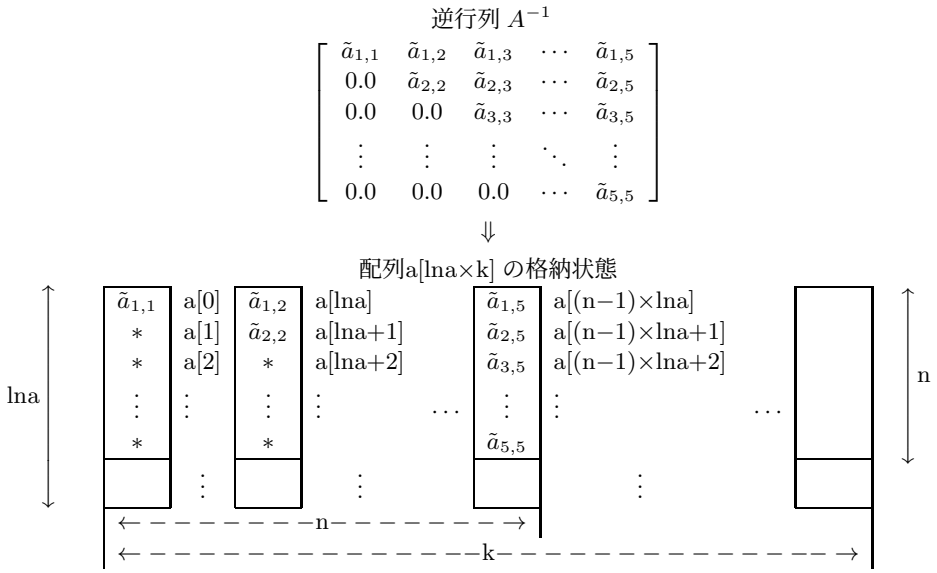
(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	det[0] ← a[0], det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) 上三角行列の逆行列はやはり上三角行列であるので, 逆行列 A^{-1} は配列 a の上三角部分のみに格納される.

図 2−16　逆行列 (上三角行列) 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.

(b) 行列式の値は次の式で与えられる.
$det(A) = \det[0] \times 10^{\det[1]}$
このとき, $1.0 \leq |\det[0]| < 10.0$ となるようにスケーリングされている.

(c) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

2.20 実下三角行列 (2 次元配列型)

2.20.1 ASL_dbtlsl, ASL_rbtlsl
連立 1 次方程式 (実下三角行列)

(1) **機 能**

実下三角行列 A(2 次元配列型) を係数行列とする連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbtlsl  (a, lna, n, b);
```

単精度関数:

```
ierr = ASL_rbtlsl  (a, lna, n, b);
```

(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(実下三角行列, 2 次元配列型)
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	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(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 の対角要素の中に, 0 に近いものがあった. 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 が 0.0 の対角要素を持つ. i は 0.0 である最初の対角要素の番号である. 行列 A は特異である.	

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

$$\begin{bmatrix} 5 & 0 & 0 & 0 \\ -1 & 4 & 0 & 0 \\ 2 & 1 & 2 & 0 \\ 3 & 2 & 7 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 5 \\ 22 \end{bmatrix} \quad \text{を解く}$$

(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_dbtlsl */

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

int main()
{
    double *a;
    int na;
    int nn;
    double *b;
    int ierr;
    int i,j;
    FILE *fp;

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

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

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

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

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


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

    fclose( fp );

    ierr = ASL_dbtlsl(a, na, nn, b);

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

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

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

    free( a );
    free( b );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dbtlsl ***

 ** Input **

 n =      4

Coefficient Matrix

       5           0           0           0
      -1           4           0           0
       2           1           2           0
       3           2           7          10

Constant Vector

       5
       3
       5
      22

 ** Output **

ierr =      0

Solution

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

.

2.20.2 ASL_dbtlco, ASL_rbtlco 実下三角行列の条件数

(1) **機 能**

実下三角行列 A(2 次元配列型) の条件数を求める

(2) **使用法**

倍精度関数:

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

単精度関数:

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

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	lna×n	入 力	実下三角行列 A(2 次元配列型)
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	cond	{D∗ / R∗}	1	出 力	条件数の逆数
5	w1	{D∗ / R∗}	n	ワーク	作業領域
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) $0 < \mathrm{n} \leq \mathrm{lna}$

(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	cond ← 1.0 とする
2100	係数行列 A の対角要素の中に, 0 に近いものがあった. 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.
$4000 + i$	行列 A が 0.0 の対角要素を持つ. i は 0.0 である最初の対角要素の番号である.	

(6) **注意事項**

(a) 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

2.20.3 ASL_dbtldi, ASL_rbtldi 実下三角行列の行列式と逆行列

(1) **機 能**

実下三角行列 A(2 次元配列型) の行列式と逆行列を求める

(2) **使用法**

倍精度関数:

```
ierr = ASL_dbtldi  (a, lna, n, det, isw);
```

単精度関数:

```
ierr = ASL_rbtldi  (a, lna, n, det, isw);
```

(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	実下三角行列 A(2 次元配列型)
				出 力	行列 A の逆行列 (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	det	{D∗ / R∗}	2	出 力	行列 A の行列式の値 (注意事項 (b) 参照)
5	isw	I	1	入 力	処理スイッチ isw>0:行列式の値を求める isw=0:行列式の値と逆行列を求める isw<0:逆行列を求める
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

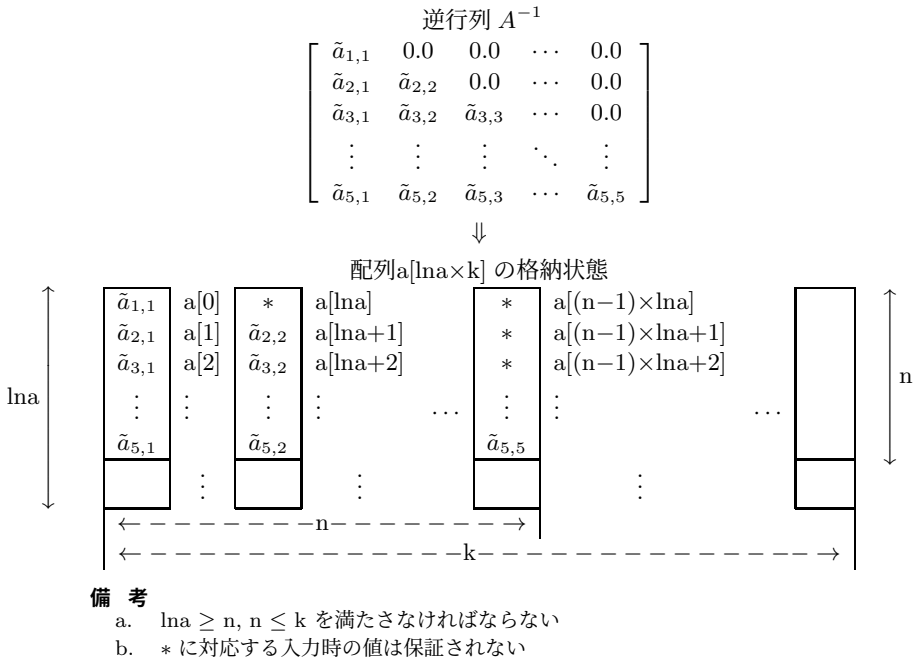
(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了	
1000	n=1 であった	det[0] ← a[0], det[1] ← 0.0 a[0] ← 1.0/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.

(6) **注意事項**

(a) 下三角行列の逆行列はやはり下三角行列であるので, 逆行列 A^{-1} は配列 a の下三角部分のみに格納される

図 2−17　逆行列 (下三角行列) 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

(b) 行列式の値は次の式で与えられる

$det(A) = \mathrm{det}[0] \times 10^{\mathrm{det}[1]}$

このとき, $1.0 \leq |\mathrm{det}[0]| < 10.0$ となるようにスケーリングされている

(c) **行列の次数が 100 以下など十分に小さい場合や, 逆行列そのものが必要である場合を除いて, 逆行列を計算すべきではない.** 数値計算では多くの場合, 逆行列は, $A^{-1}\boldsymbol{b}$ や $A^{-1}B$ といった形式で現れるが, これらはそれぞれ, ベクトル $\boldsymbol{x}$ についての連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$, 行列 X についての多重右辺連立 1 次方程式 $AX = B$ として連立 1 次方程式を解いて計算すべきである. 数学的には, 逆行列を求めて逆行列とベクトルの積や逆行列と行列の積を計算することと前述のような連立 1 次方程式を解くことは同じであるが, 数値計算上は一般に, 逆行列による求解は計算効率も悪く, 計算精度も劣る.

付 録 A　用語説明

(1) **行列**

$m \times n$ の行列 A とは, $m \times n$ 個の元 $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ の矩形の配置をいう.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

元 $a_{i,j}$ を行列 A の (i, j) 要素と呼ぶ. 行列の元としては, 複素数または実数を考える. 特に, 複素数を元とする行列を複素行列, 実数を元とする行列を実行列と呼ぶ. また, 特に $m = n$ である場合, 行列 A を正方行列と呼ぶ. 行列 A は, (a_{ij}) と略記する場合がある. なお, 本書では必要に応じて行の添字 i と列の添字 j を区別するために $(a_{i,j})$ を用いる.

(2) **(数) ベクトル**

$1 \times n$ の行列を n 次の行ベクトル, $m \times 1$ の行列を m 次の列ベクトルと呼ぶ. 両者を特に区別する必要がない場合には単にベクトルと呼ぶ. なお, 数学的にはさらに抽象化した概念としてベクトルを定義し, ここで述べた「ベクトル」は数ベクトルと呼ばれる. 抽象化したベクトルの定義については「ベクトル空間」についての説明を参照されたい.

(3) **行列積**

2 つの行列 $A = (a_{i,j})$, $B = (b_{k,l})$ において, 行列 A の列数と行列 B の行数が等しい場合にのみ行列積 $A \cdot B = (c_{i,l})$ が定義されて,

$$c_{i,l} = \sum_j a_{i,j} \cdot b_{j,l}$$

となる.

(4) **行列ベクトル積**

行列積 $A \cdot B$ において, 特に行列 B が列ベクトル$\boldsymbol{x}$である場合, 積 $A\boldsymbol{x}$ を行列ベクトル積という.

(5) **行列の転置**

$m \times n$ の行列 $A = (a_{i,j})$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ において行と列を入れ換えてできる行列 $A' = (a_{j,i})$ を行列 A の転置行列と呼び, A^T で表す. なお, 転置行列は ${}^tA$ と表す場合もある.

(6) **行列の (主) 対角**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, $a_{i,i}$ $(i = 1, 2, \cdots, n)$ の並びを (主) 対角と呼び, その要素を (主) 対角要素と呼ぶ. また特に, 対角にのみ非零要素 $a_{i,i}$ を持つ行列を対角行列 $(\mathrm{diag}(a_{i,i}))$ と呼ぶ.

(7) **単位行列**

$n \times n$ の対角行列 A=diag$(a_{i,i})$ において, 対角要素 $a_{i,i}$ $(i = 1, 2, \cdots, n)$ がすべて 1 の行列を単位行列と呼び, 記号 E または I を用いて表す.

(8) **逆行列**

正方行列 A に対して, $A \cdot B = B \cdot A = E$ (E は単位行列) を満たす正方行列 B が存在する場合に, 行列 B を行列 A の逆行列と呼び, 記号 A^{-1} で表す.

(9) **一般逆行列**

$m \times n$ の行列 A に対して, 以下の関係を満たすような $n \times m$ 行列 X が一意的に存在し, この行列 X を行列 A の (Moore-Penrose の) 一般逆行列と呼び, 記号 $A^{\dagger}$ で表す.

- $AXA = A$
- $XAX = X$
- $(AX)^T = AX$
- $(XA)^T = XA$

(10) **行列の下三角ならびに上三角**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, $a_{i,j}$ $(i > j)$ の要素をまとめて下三角, $a_{i,j}$ $(i < j)$ の要素をまとめて上三角と呼ぶ. なお, 上三角ならびに下三角の定義に対角を含める場合もある. 対角を含めた下三角にのみ非零要素を持つ行列を下三角行列, 対角を含めた上三角にのみ非零要素を持つ行列を上三角行列とそれぞれ呼ぶ.

(11) **共役転置行列**

複素行列 A の各要素の共役な複素数を要素とする行列の転置行列を共役転置行列と呼び, 記号 A^* で表す. 行列の要素が実数の場合には $A^* = A^T$ である.

(12) **対称行列**

$A = A^T$ が成立する正方行列を対称行列と呼ぶ. 対称行列では, $a_{i,j} = a_{j,i}$ である.

(13) **エルミート行列**

$A = A^*$ が成立する正方行列をエルミート行列と呼ぶ. エルミート行列では, $a_{i,j}$ と $a_{j,i}$ は複素共役である.

(14) **ユニタリ行列**

$UU^* = I$(I は単位行列) が成立する正方行列 U をユニタリ行列と呼ぶ.

(15) **直交行列**

$AA^T = I$(I は単位行列) が成立する実正方行列 A を直交行列と呼ぶ.

(16) **行列の副対角**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, $a_{i,i+p}$ $(i = 1, 2, \cdots, n-p)$ の並びを第 p 上副対角, $a_{i+q,i}$ $(i = 1, 2, \cdots, n-q)$ の並びを第 q 下副対角と呼び, その要素を, それぞれ第 p 上副対角要素, 第 q 下副対角要素と呼ぶ. また, 両者をまとめて単に副対角要素と呼ぶこともある.

(17) **バンド行列**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, 主対角とそれに隣接するいくつかの上副対角ならびに下副対角にのみ 0 でない要素がある行列をバンド行列と呼ぶ. 対角からもっともはなれた 0 でない要素を含む副対角が第 u 上副対角と第 l 下副対角である場合に, 値 u と l をそれぞれ上バンド幅, 下バンド幅と呼ぶ. 特に, $u = l$ の場合これを単にバンド幅と呼ぶ.

(18) **3 重対角行列**

特に, 上バンド幅ならびに下バンド幅が 1 の行列を 3 重対角行列と呼ぶ.

(19) **ヘッセンベルグ行列**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, 第 1 下副対角を除いた下三角のすべての要素が 0 である行列をヘッセンベルグ行列と呼ぶ. 行列の固有値を求める場合に, 一般の行列をこの行列に変換する.

(20) **準上三角行列**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において, 第 1 下副対角の連続する 2 つの副対角要素の少なくとも一方が必ず 0 であり, 第 1 下副対角を除いた下三角のすべての要素が 0 である行列を準上三角行列行列と呼ぶ. ヘッセンベルグ行列の特殊な場合である.

(21) **スパース行列**

一般に, 非零要素の数が全要素数に比べて少ない行列をスパース行列と呼ぶ. スパース行列のうちで要素の並びに規則性があり, この規則性を活用することによって, 問題を解く有効な解法が作成されている場合に, この行列を特に, 規則スパース行列と呼んでいる. 規則スパース行列でない行列は, 不規則スパースと呼ばれている. たとえば, バンド幅が小さいバンド行列は規則スパース行列の一種である.

(22) **正則行列, 特異行列**

正方行列 A が逆行列を持つとき, 行列 A は正則 (*regular*) であるという. 正則でない行列は特異 (*singular*) であるという. 正則な行列を係数に持つ連立 1 次方程式の解は, 一意に定まる. ただし現実には, 有限桁で計算を行うので, 丸め誤差の影響が避けられず, 正則な行列と特異な行列の区別は曖昧になる. たとえば, 数学的に特異な行列を用いて数値的に連立 1 次方程式を解いた場合でも, 見かけ上解が得られる場合がある. したがって, 特にほとんど特異な行列を係数に持つ連立 1 次方程式を解く場合, 見かけ上得られる解については, その妥当性について十分な吟味が必要である.

(23) **LU 分解**

直接法で, 連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く場合には, まず係数行列 A を下三角行列 L と上三角行列 U の積に $A = LU$ と分解する. この分解のことを LU 分解と呼ぶ. このような分解を行えば, 連立 1 次方程式の解$\boldsymbol{x}$は

$$\begin{aligned} L\boldsymbol{y} &= \boldsymbol{b} \\ U\boldsymbol{x} &= \boldsymbol{y} \end{aligned}$$

を逐次解くことによって得られる. この 2 つの連立 1 次方程式は係数行列が三角行列であるので前進代入ならびに後退代入を用いて容易に解くことができる. なお, 行列 A の LU 分解は A が正則であれば, たとえば行列 L の対角要素を 1 に固定することによって一意に定まる. また, 連立 1 次方程式を解く場合には, 一般に部分軸選択を行いながら LU 分解を行うので, 軸選択による行交換行列を P として $PA = LU$ を満たす三角行列 L, U をそれぞれ求める.

(24) $\mathrm{U^TDU}$ **分解**

連立 1 次方程式の係数行列が対称行列である場合には, 軸選択を行わないで LU 分解を行って得られる下三角行列 L と上三角行列 U の間には, $L = U^T D$ の関係がある. ここで D は対角行列である. したがって, L または U の片方と D のみを陽に求めれば, 連立 1 次方程式を解くことができる. 係数行列から U と D を陽に求める分解を $\mathrm{U^TDU}$ 分解と呼ぶ.

(25) $\mathrm{U^*DU}$ **分解**

連立 1 次方程式の係数行列がエルミート行列である場合には, 軸選択を行わないで LU 分解を行って得られる下三角行列と上三角行列 U の間には, $L = U^* D$ の関係がある. ここで D は対角行列である. したがって, L または U の片方と D のみを陽に求めれば, 連立 1 次方程式を解くことができる. 係数行列から U と D を陽に求める分解を $\mathrm{U^*DU}$ 分解と呼ぶ.

(26) **正定値 (Positive definite)**

実対称行列またはエルミート行列 A は, 任意のベクトル$\boldsymbol{x}$ $(\boldsymbol{x} \neq \boldsymbol{0})$ に対して, $\boldsymbol{x}^* A\boldsymbol{x} > 0$ を満たす場合, 正 (定) 値, $\boldsymbol{x}^* A\boldsymbol{x} < 0$ を満たす場合, 負値とそれぞれ呼ぶ. 行列 A が正定値行列であることは, 次の 2 つの条件と同値である.

(a) 行列 A の固有値がすべて正である.

(b) 行列 A の主小行列式がすべて正である.

数学的には, 正定値行列は軸選択を行わなくても LU 分解することができるが, 現実には軸選択を行わなければ, 数値的に安定して LU 分解を行えない場合がある.

(27) **実数固有値 (Real eigenvalue)**

実数成分の正方行列の固有値が全て実数であることの必要十分条件は, 2 つの実対称行列の積であることである. また, 複素数成分の正方行列の固有値が全て実数であることの必要十分条件は, 2 つのエルミート行列の積であることである.

(28) **対角優位 (Diagonally dominant)**

$n \times n$ の正方行列 $A = (a_{i,j})$ $(i, j = 1, 2, \cdots, n)$ において,

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

が成立する場合, 行列 A を対角優位な行列と呼ぶ. 数学的には, 対角優位な行列は軸選択を行わなくても LU 分解することができるが, 現実には軸選択を行わなければ, 数値的に安定して LU 分解を行えない場合がある.

(29) **フィルイン (Fill-in)**

スパース行列を LU 分解する場合に, 元々0 であった要素が演算によって 0 でなくなることを, フィルインと呼ぶ.

(30) **エンベロプ法**

$n \times n$ の対称スパース行列 A を $\mathrm{U^TDU}$ 分解する場合に, 行列 A の各行の最初の非零要素と対角要素をエンベロプ (包絡線) と見立てて, 包絡線内の要素のみに着目して分解を実行する一方法をエンベロプ法と呼ぶ. 行列を $\mathrm{U^TDU}$ 分解する時, フィルインはエンベロプ内部でのみ発生することを用いた手法である. なお, エンベロプ法では, 対称行列の下三角部分に着目して分解を行うが, 上三角部分に着目して同様な分解を行う手法はスカイライン法として知られている.

(31) **ベクトル空間**

集合 V が次の条件 (a), (b) を満足するとき, V をベクトル空間とよびその要素をベクトルと呼ぶ.

(a) V の 2 つの要素$\boldsymbol{a}$, $\boldsymbol{b}$に対して和 $\boldsymbol{a} + \boldsymbol{b}$ が V の要素として一意に定まり, 次の性質を満たす.

i. $(\boldsymbol{a} + \boldsymbol{b}) + \boldsymbol{c} = \boldsymbol{a} + (\boldsymbol{b} + \boldsymbol{c})$ (結合則)
ただし, $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$は, V の任意の要素.

ii. $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{b} + \boldsymbol{a}$ (交換則)
ただし, $\boldsymbol{a}$, $\boldsymbol{b}$は, V の任意の要素.

iii. 零ベクトルと呼ばれる V の要素$\boldsymbol{0}$が存在し, V の任意の要素$\boldsymbol{a}$に対して, $\boldsymbol{a} + \boldsymbol{0} = \boldsymbol{a}$

iv. V の任意の要素$\boldsymbol{a}$に対して, $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{0}$ となる V の要素$\boldsymbol{b}$がただ一つ存在する. なお, このとき, $\boldsymbol{b}$は $-\boldsymbol{a}$ と表される.

(b) V の任意の要素$\boldsymbol{a}$と複素数 c に対して, $\boldsymbol{a}$の c 倍 $c\boldsymbol{a}$ が V の要素として一意に定まり, 次の性質を満たす (スカラ倍).

i. $c(\boldsymbol{a} + \boldsymbol{b}) = c\boldsymbol{a} + c\boldsymbol{b}$ (ベクトル分配則)

ii. $(c + d)\boldsymbol{a} = c\boldsymbol{a} + d\boldsymbol{a}$ (スカラ分配則)

iii. $(cd)\boldsymbol{a} = c(d\boldsymbol{a})$

iv. $1\boldsymbol{a} = \boldsymbol{a}$

(32) **一次結合, 一次独立, 一次従属**

ベクトル空間 V の k 個のベクトル $\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k$ と複素数 $c_1,\ \cdots,\ c_k$ とによって作られるベクトル

$$c_1\boldsymbol{a}_1 + \cdots + c_k\boldsymbol{a}_k$$

を $\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k$ の一次結合といい, $c_1,\ \cdots,\ c_k$ をその係数という. すべてが 0 ではないある係数 $c_1,\ \cdots,\ c_k$ に対して

$$c_1\boldsymbol{a}_1 + \cdots + c_k\boldsymbol{a}_k = \boldsymbol{0}$$

となるとき, ベクトルの集合 $\{\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k\}$ は一次従属であるといい, そうでないときは, 一次独立であるという.

(33) **基底**

S をベクトル空間 V の任意の部分集合とし, S に含まれる一次独立なベクトルの組を $\{\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k\}$ とする. S の任意のベクトル$\boldsymbol{b}$に対して $\{\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k,\ \boldsymbol{b}\}$ が一次従属であるとき, $\{\boldsymbol{a}_1,\ \cdots,\ \boldsymbol{a}_k\}$ は S において極大であると呼ばれ, S としてベクトル空間 V そのものをとった場合, この一次独立なベクトルの組をベクトル空間 V の基底と呼ぶ. なお, V の基底を構成するベクトルの個数を V の次元と呼ぶ. また, n 次元ベクトル空間 V_n の任意の基底を $\{\boldsymbol{u}_1,\ \cdots,\ \boldsymbol{u}_n\}$ とすると, V_n の任意のベクトル$\boldsymbol{a}$ は, $\{\boldsymbol{u}_1,\ \cdots,\ \boldsymbol{u}_n\}$ の一次結合として一意に表される.

(34) **部分 (ベクトル) 空間**

ベクトル空間 V の部分集合 L は, 次の条件 (a), (b) を満足するとき, V の部分 (ベクトル) 空間と呼ばれる.

(a) $\boldsymbol{a}, \boldsymbol{b} \in L$ ならば $\boldsymbol{a} + \boldsymbol{b} \in L$

(b) $\boldsymbol{a} \in L$, c が複素数ならば $c\boldsymbol{a} \in L$

(35) **一次変換**

V_n, V_m をそれぞれ n 次元, m 次元のベクトル空間とする. V_n の各要素$\boldsymbol{x}$を V_m の要素 $\boldsymbol{A}(\boldsymbol{x})$ に対応させる写像 $\boldsymbol{A} : V_n \to V_m$ が次の二つの条件を満たすとき, $\boldsymbol{A}$を V_n から V_m への一次変換とよぶ.

(a) $\boldsymbol{A}(\boldsymbol{x}_1 + \boldsymbol{x}_2) = \boldsymbol{A}(\boldsymbol{x}_1) + \boldsymbol{A}(\boldsymbol{x}_1)\ \ \boldsymbol{x}_1,\ \boldsymbol{x}_2 \in V_n$

(b) $\boldsymbol{A}(c\boldsymbol{x}) = c\boldsymbol{A}(\boldsymbol{x})\ \ \boldsymbol{x} \in V_n,\ c$ は複素数

V_n, V_m それぞれのひとつの基底を $\{\boldsymbol{u}_1,\ \cdots,\ \boldsymbol{u}_n\}$, $\{\boldsymbol{v}_1,\ \cdots,\ \boldsymbol{v}_m\}$ とすると

$$\boldsymbol{A}(\boldsymbol{u}_j) = \sum_{i=1}^{m} a_{i,j}\boldsymbol{v}_i\ \ (j = 1, \cdots, n)$$

の係数行列 $A = (a_{i,j})$ によって, 任意の $\boldsymbol{x} \in V_n$ に対する $\boldsymbol{A}(\boldsymbol{x})$ が定められる. 行列 A をこの基底に関する一次変換$\boldsymbol{A}$ の表現行列という. また, $\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{x},\ \boldsymbol{x} \in V_n$ なる一次変換 $\boldsymbol{E} : V_n \to V_n$ を定義して, 恒等変換とよぶ. 恒等変換の表現行列は基底の取りかたによらず常に単位行列 E になる.

(36) **固有値・固有ベクトル**

n 次元ベクトル空間 V_n の中の一次変換$\boldsymbol{A}$に対して

$$\boldsymbol{A}(\boldsymbol{x}) = \lambda\boldsymbol{x}\text{すなわち } (\boldsymbol{A} - \lambda\boldsymbol{E})(\boldsymbol{x}) = \boldsymbol{0}$$

を満たすような数 λ とベクトル $\boldsymbol{x}(\boldsymbol{x} \neq \boldsymbol{0})$ が存在するとき, λ を$\boldsymbol{A}$の固有値といい, $\boldsymbol{x}$を固有値 λ に属する固有ベクトルという. ここで, $\boldsymbol{E}$は恒等変換である. V_n の中で基底を一つ定めて, 一次変換$\boldsymbol{A}$の表現行列を A, 固有ベクトル$\boldsymbol{x}$に対応する数ベクトルを $\hat{\boldsymbol{x}}$ とすると, 固有値 λ と $\hat{\boldsymbol{x}}$ は次式を満たす.

$$A\hat{\boldsymbol{x}} = \lambda\hat{\boldsymbol{x}}$$

ここで $\hat{\boldsymbol{x}}$ は, $\boldsymbol{x}$の成分 $x_1,\ \cdots,\ x_n$ を用いて

$$\hat{\boldsymbol{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

表される. なお, 普通, λ と $\hat{\boldsymbol{x}}$ は行列 A の固有値, 固有ベクトルとそれぞれ呼ばれ, 本書においてもこの呼称を採用する. また, 数ベクトルとベクトルを区別せず, $\boldsymbol{x}$と表す. 一次変換 $\boldsymbol{A} : V_n \to V_n$ の固有値 λ に属するベクトル全体と零ベクトル$\mathbf{0}$をあわせたものは一つのベクトル空間をなすので, これを$\boldsymbol{A}$の固有値 λ に属する固有ベクトル空間とよぶ.

(37) **不変部分空間**

ベクトル空間 V_n の中の一次変換$\boldsymbol{A}$に対して V_n の部分空間 U が

$$\boldsymbol{A}(U) \subseteq U$$

という性質をもつとき, すなわち, 任意のベクトル$\boldsymbol{x}$に対して $\boldsymbol{Ax} \in U$ であるとき, U は$\boldsymbol{A}$に関して不変 (*invariant*) であるといわれる. 特に, $\boldsymbol{A}$の固有ベクトル空間は$\boldsymbol{A}$に関して不変である. 不変な部分ベクトル空間を不変部分空間 (*invariant subspace*) と呼ぶ.

(38) **平面回転 (Plane rotation)**

次の様な行列 $S_{k:l}(\theta)$ で規定される直交変換を平面回転と呼ぶ.

$$S_{kl}(\theta) = \begin{bmatrix} E_{1:k-1} & O_{1:k-1,k:l} & O_{1:k-1,l:n} \\ O_{k:l,1:k-1} & T_{k:l}(\theta) & O_{k:l,l:n} \\ O_{l:n,1:k-1} & O_{l:n,k:l} & E_{l:n} \end{bmatrix}$$

ここで, $T_{k:l}(\theta)$ は

$$T_{k:l}(\theta) = \begin{bmatrix} \cos\theta & O_{k:k,k+1:l-1} & -\sin\theta \\ O_{k+1:l-1,k:k} & E_{k+1:l-1} & O_{k+1:l-1,l:l} \\ \sin\theta & O_{l:l,k+1:l-1} & \cos\theta \end{bmatrix}$$

$E_{p:q}$ は $q-p+1$ 次の単位行列で

$$E_{p:q} = \begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \end{bmatrix} \begin{matrix} (p \\ (p+1 \\ \vdots \\ (q \end{matrix}$$

$O_{p:r,q:s}$ は $r-p+1 \times s-q+1$ 次のゼロ行列

$$O_{p:r,q:s} = \begin{matrix} \overbrace{\ \ } & \overbrace{\ \ } & & \overbrace{\ \ } \\ q & q+1 & \cdots & s \end{matrix} \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{matrix} (p \\ (p+1 \\ \vdots \\ (r \end{matrix}$$

いま $A=(a_{i,j})$ $(i=1,2,\cdots,n;j=1,2,\cdots,n)$ の部分行列 $A_{p:r,q:s}$ を

$$A_{p:r,q:s}=\begin{bmatrix} a_{p,q} & a_{p,q+1} & \cdots & a_{p,s} \\ a_{p+1,q} & a_{p+1,q+1} & \cdots & a_{p+1,s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r,q} & a_{r,q+1} & \cdots & a_{r,s} \end{bmatrix}$$

と定義し, 行列 A を

$$A=\begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l} & A_{1:k-1,l+1:n} \\ A_{k:l,1:k-1} & A_{k:l,k:l} & A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l} & A_{l+1:n,l+1:n} \end{bmatrix}$$

と表す. この時

$$S_{k:l}(\theta)A=\begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l} & A_{1:k-1,l+1:n} \\ T_{k:l}(\theta)A_{k:l,1:k-1} & T_{k:l}(\theta)A_{k:l,k:l} & T_{k:l}(\theta)A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l} & A_{l+1:n,l+1:n} \end{bmatrix}$$

$$T_{k:l}(\theta)A_{k:l,q:s}=\begin{bmatrix} \cos\theta a_{k,q}-\sin\theta a_{l,q} & \cdots & \cos\theta a_{k,r}-\sin\theta a_{l,s} \\ a_{k+1,q} & \cdots & a_{k+1,r} \\ \vdots & \cdots & \vdots \\ a_{l-1,q} & \cdots & a_{l-1,r} \\ \sin\theta a_{k,q}+\cos\theta a_{l,q} & \cdots & \sin\theta a_{k,r}+\cos\theta a_{l,s} \end{bmatrix}$$

したがって, $\tan\theta=\frac{a_{l,i}}{a_{k,i}}$ または $\tan\theta=-\frac{a_{l,i}}{a_{k,i}}(i=q,\cdots,s)$ を満たす様に θ を決定すれば, $S_{k:l}(\theta)A$ の第 k 行と第 l 行の要素のうち任意の 1 つの要素を 0 とすることができる. 同様に

$$AS_{k:l}(-\theta)=\begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l}T_{k:l}(-\theta) & A_{1:k-1,l+1:n} \\ A_{k:l,1:k-1} & A_{k:l,k:l}T_{k:l}(-\theta) & A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l}T_{k:l}(-\theta) & A_{l+1:n,l+1:n} \end{bmatrix}$$

$$A_{p:r,k:l}T_{k:l}(-\theta)=\begin{bmatrix} \cos\theta a_{p,k}-\sin\theta a_{p,l} & a_{p,k+1} & \cdots & a_{p,l-1} & \sin\theta a_{p,k}+\cos\theta a_{p,l} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cos\theta a_{r,k}-\sin\theta a_{r,l} & a_{r,k+1} & \cdots & a_{r,l-1} & \sin\theta a_{r,k}+\cos\theta a_{r,l} \end{bmatrix}$$

したがって, $\tan\theta=\frac{a_{i,l}}{a_{i,k}}$ または $\tan\theta=-\frac{a_{i,l}}{a_{i,k}}(i=p,\cdots,r)$ を満たす様に θ を決定すれば, $AS_{k:l}(-\theta)$ の第 k 列と第 l 列の要素のうち任意の 1 つの要素を 0 とすることができる. なお,

$$S_{k:l}(-\theta)=S_{k:l}(\theta)^T$$

である. また

$$\tilde{A}=S_{k:l}(\theta)AS_{k:l}(-\theta)=\begin{bmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k:l}T_{k:l}(-\theta) & A_{1:k-1,l+1:n} \\ T_{k:l}(\theta)A_{k:l,1:k-1} & T_{k:l}(\theta)A_{k:l,k:l}T_{k:l}(-\theta) & T_{k:l}(\theta)A_{k:l,l+1:n} \\ A_{l+1:n,1:k-1} & A_{l+1:n,k:l}T_{k:l}(-\theta) & A_{l+1:n,l+1:n} \end{bmatrix}$$

となるので, 行列 A が対称行列であれば, θ 調整することによって $\tilde{A} = S_{k:l}(\theta)AS_{k:l}(-\theta)$ の要素を $(\tilde{a}_{i,j})$ としてある $j(j \neq k, j \neq l)$ について

$$\tilde{a}_{k,j} = \tilde{a}_{j,k} = 0$$

または

$$\tilde{a}_{l,j} = \tilde{a}_{j,l} = 0$$

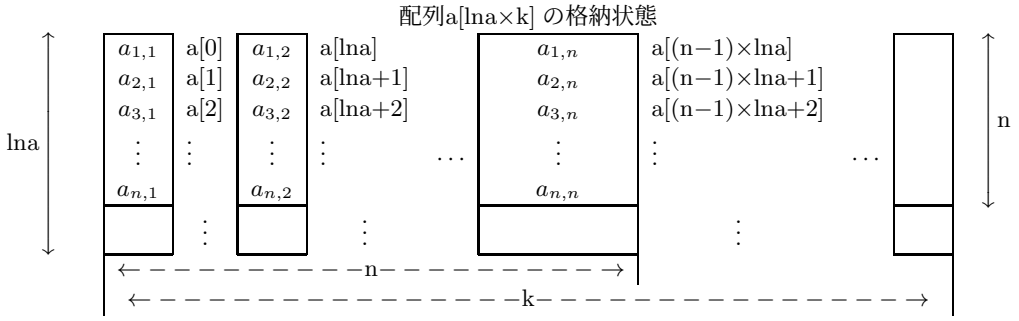
とすることができる.

付 録 B　配列データの取扱い方法

B.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 B−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 [5×4] 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;
```

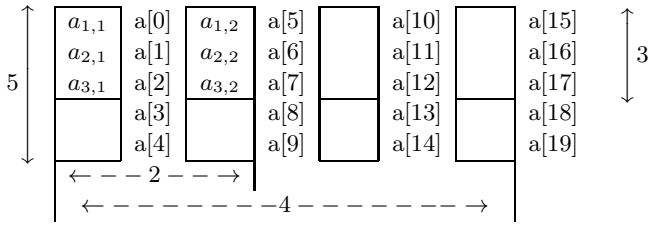
```
    }
    b = (double *)malloc((size_t) sizeof(double) * lmb*k);
    if(b == NULL)
    {
        printf("no enough memory for array b\n");
        return -1;
    }
    c = (double *)malloc((size_t) sizeof(double) * lmc*k);
    if(c == NULL)
    {
        printf("no enough memory for array c\n");
        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 B−2　配列 a 中の格納形式



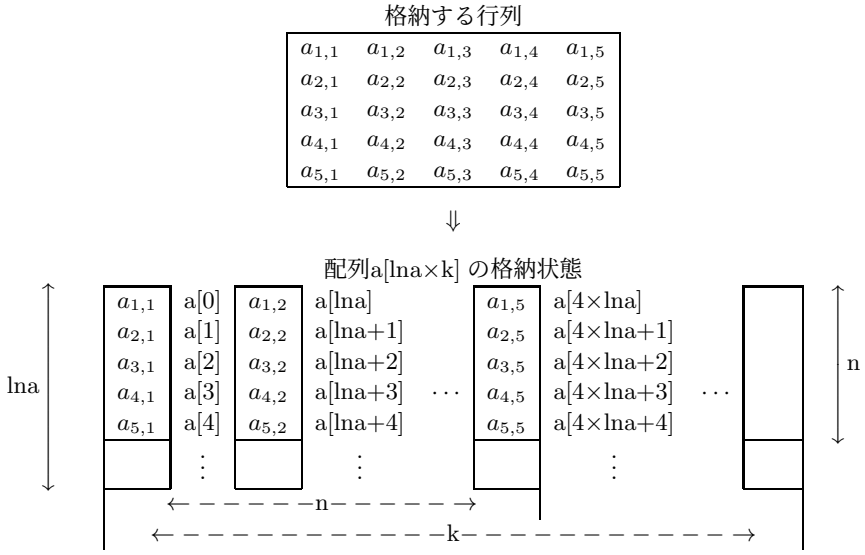
次数の異なるいくつかの配列をデータとして取り扱う場合には, そのうち最も大きな次数を lna とするような配列を一つ用意しておけば, この配列を逐次利用することができる. ただし, この時, 整合寸法として常に lna の値を与える必要がある.

B.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.

B.2.1　実行列 (2 次元配列型)

図 B−3　実行列 (2 次元配列型) の格納形式



備　考

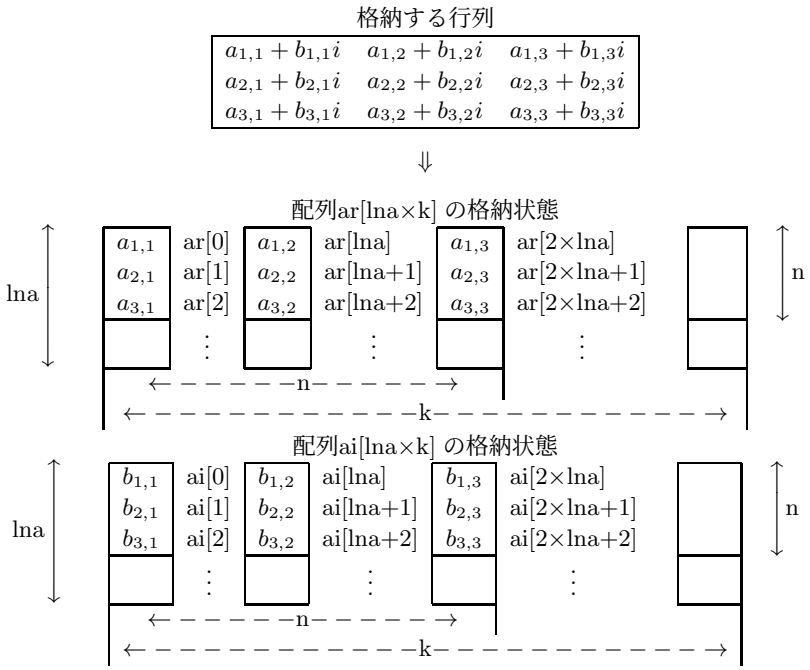
a.　lna ≥ n, k ≥ n を満たさなければならない.

B.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 B−4 複素行列 (2 次元配列型)(実数引数型) の格納形式

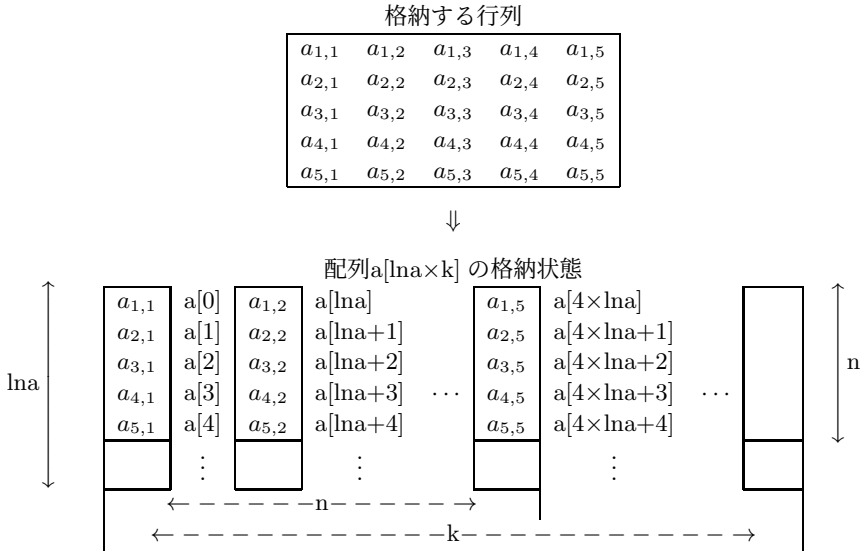


備 考

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 B−5 複素行列 (2 次元配列型)(複素引数型) の格納形式



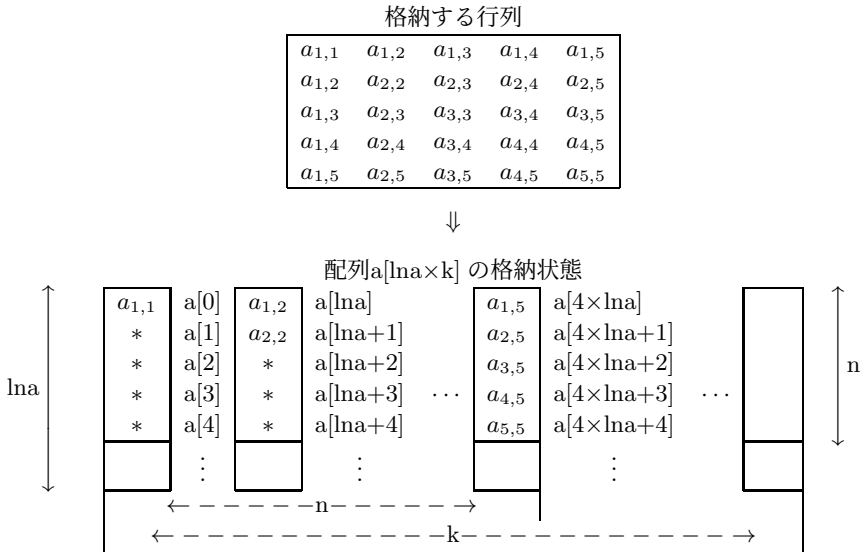
備 考

a. lna ≥ n, k ≥ n を満たさなければならない.

B.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 B−6　実対称行列 (2 次元配列型)(上三角型) の格納形式



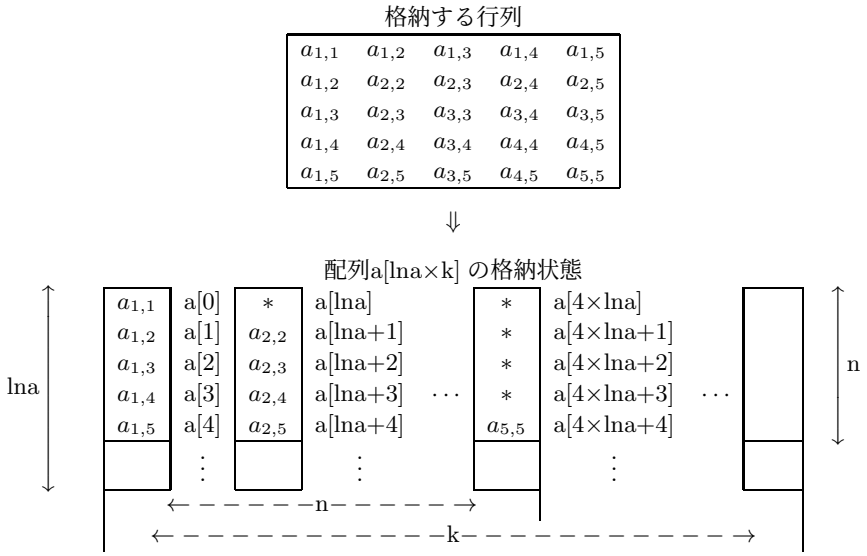
備　考

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 B−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.

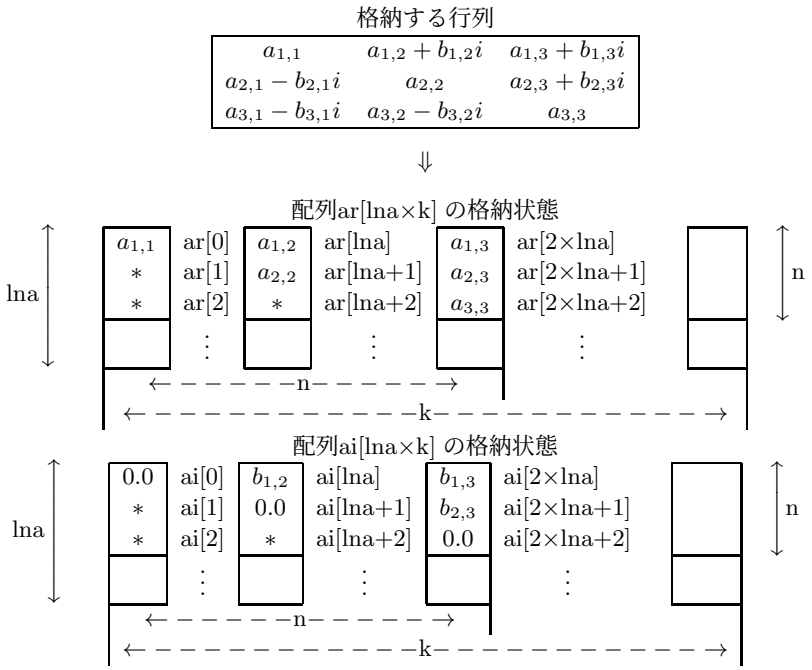
b.　lna ≥ n, k ≥ n を満たさなければならない.

B.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 B–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



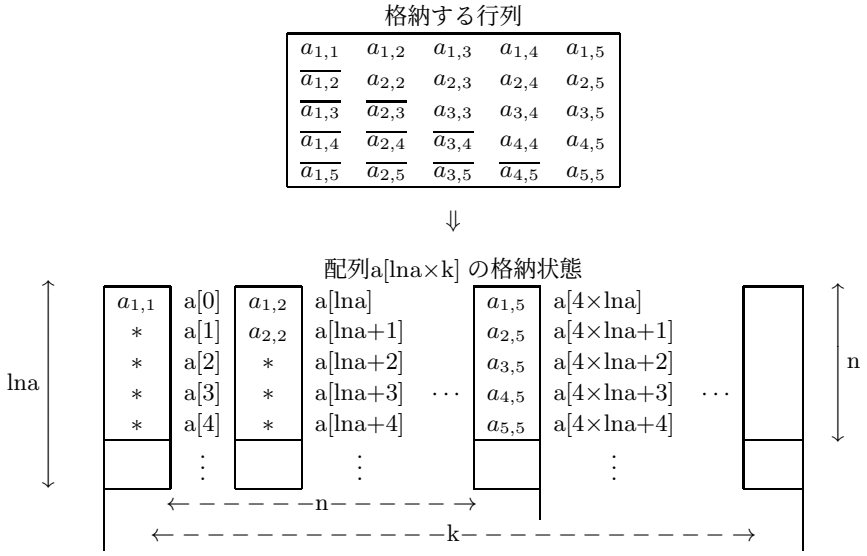
備 考

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 B–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

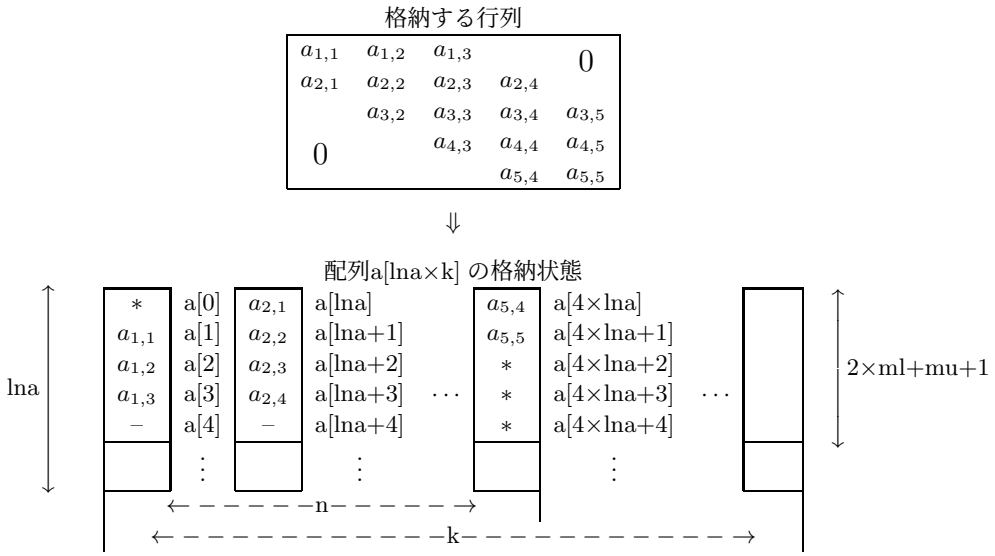
a. x の複素共役を $\overline{x}$ で表している.

b. ∗ は, 任意の値であることを示す.

c. lna ≥ n, k ≥ n を満たさなければならない.

B.2.5 実バンド行列 (バンド型)

図 B−10 実バンド行列 (バンド型) の格納形式



備 考

a. ∗ は, 任意の値であることを示す.

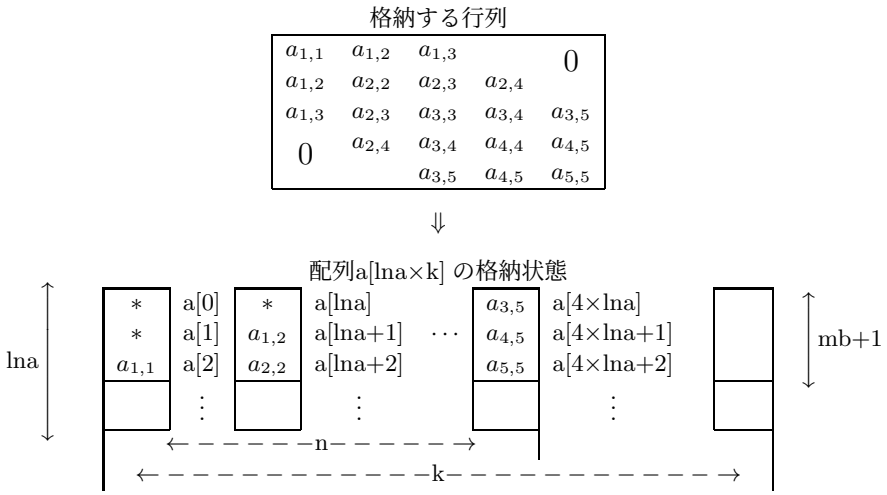
b. –は, 行列の LU 分解時に必要となる領域である.

c. mu は上バンド幅, ml は下バンド幅である.

d. lna $\geq$ 2 × ml + mu + 1, k $\geq$ n を満たさなければならない. (ただし, LU 分解を伴わない場合には, lna $\geq$ ml + mu + 1, k $\geq$ n でよい).

B.2.6 実対称バンド行列, 正値対称バンド行列 (対称バンド型)

図 B−11 実対称バンド行列 (対称バンド型) の格納形式



備 考

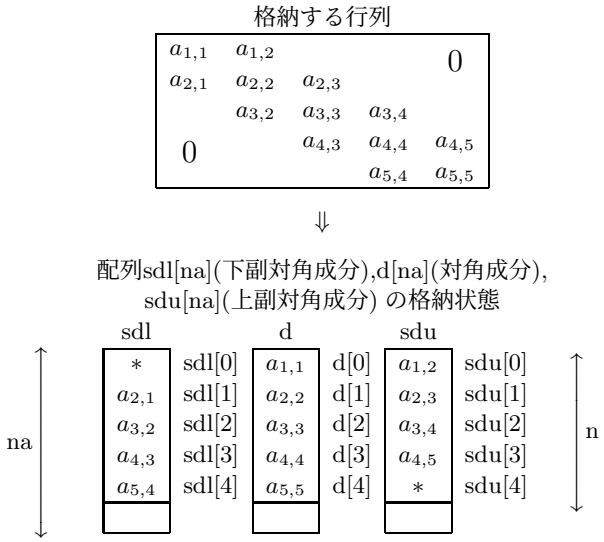
a. ∗ は, 任意の値であることを示す.

b. mb は, バンド幅である.

c. lna ≥ mb + 1, k ≥ n を満たさなければならない.

B.2.7 実 3 重対角行列 (ベクトル型)

図 B−12 実 3 重対角行列 (ベクトル型) の格納形式



備 考

a. ∗ は, 任意の値であることを示す.

b. na ≥ n を満たさなければならない.

B.2.8 実対称 3 重対角行列, 正値対称 3 重対角行列 (ベクトル型)

図 B−13 実対称 3 重対角行列 (ベクトル型) の格納形式

格納する行列

$$\begin{array}{|ccccc|}\hline a_{1,1} & a_{1,2} & & & 0 \\ a_{1,2} & a_{2,2} & a_{2,3} & & \\ & a_{2,3} & a_{3,3} & a_{3,4} & \\ 0 & & a_{3,4} & a_{4,4} & a_{4,5} \\ & & & a_{4,5} & a_{5,5} \\ \hline\end{array}$$

⇓

配列d[na](対角成分),sd[na](副対角成分) の格納状態

	d		sd		
na ↕	$a_{1,1}$	d[0]	$a_{1,2}$	sdu[0]	↕ n
	$a_{2,2}$	d[1]	$a_{2,3}$	sdu[1]	
	$a_{3,3}$	d[2]	$a_{3,4}$	sdu[2]	
	$a_{4,4}$	d[3]	$a_{4,5}$	sdu[3]	
	$a_{5,5}$	d[4]	$*$	sdu[4]	

備 考

a. $*$ は, 任意の値であることを示す.

b. na $\geq$ n を満たさなければならない.

B.2.9 定係数型実 3 重対角行列 (スカラ型)

図 B−14 定係数型実 3 重対角行列の格納形式

格納する行列 (a)

$$\begin{array}{|ccccc|}\hline d & s & & & 0 \\ s & d & s & & \\ & s & d & s & \\ 0 & & s & d & s \\ & & & s & d \\ \hline\end{array}$$

格納する行列 (b)

$$\begin{array}{|ccccc|}\hline d & s & & & 0 \\ s & d & s & & \\ & s & d & s & \\ 0 & & s & d & s \\ & & & 2\times s & d \\ \hline\end{array}$$

格納する行列 (c)

$$\begin{array}{|ccccc|}\hline d & 2\times s & & & 0 \\ s & d & s & & \\ & s & d & s & \\ 0 & & s & d & s \\ & & & s & d \\ \hline\end{array}$$

格納する行列 (d)

$$\begin{array}{|ccccc|}\hline d & 2\times s & & & 0 \\ s & d & s & & \\ & s & d & s & \\ 0 & & s & d & s \\ & & & 2\times s & d \\ \hline\end{array}$$

⇓

変数d(対角成分),sd(副対角成分) の格納状態

d	sd
d	s

B.2.10 三角行列

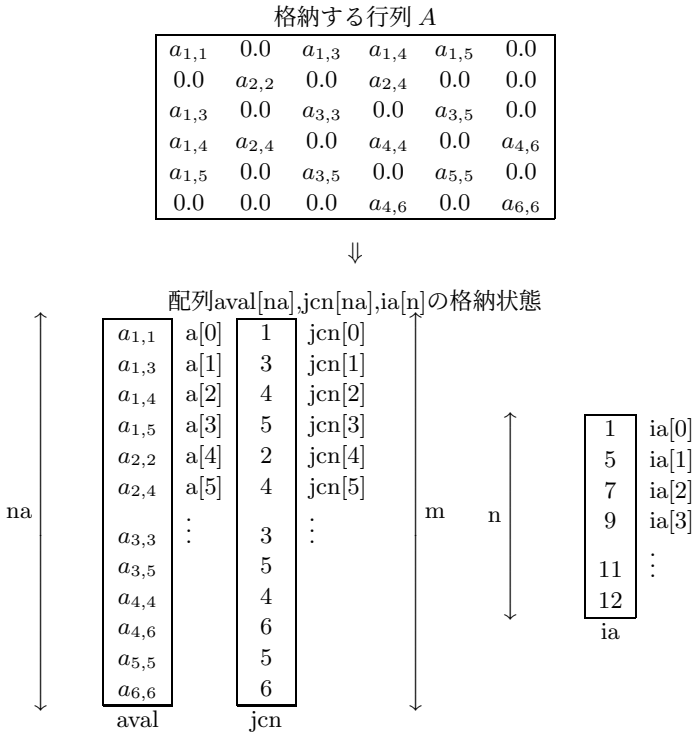
(1) **2 次元配列型**

実対称行列 (2 次元配列型)(上三角型) と格納方法は同じである.

B.2.11 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 B−15 実対称不規則スパース行列 (スパース型) の格納形式



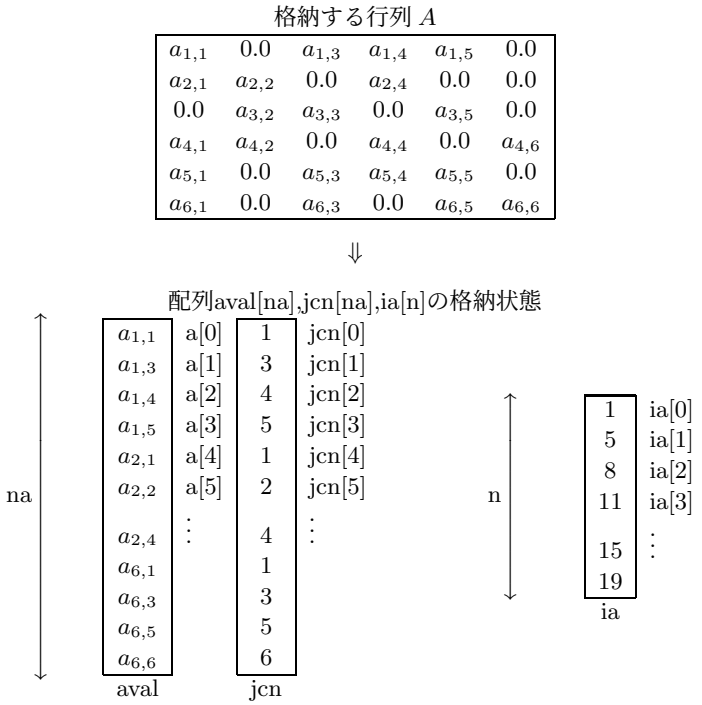
備 考

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.

B.2.12　不規則スパース行列

(1) スパース型

図 B−16　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.

付 録 C　　ASL で使用している計算機依存定数

C.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 C–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.

C.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 C–2　浮動小数点データの値の最大値・最小値

	単精度演算	倍精度演算
最大値	$2^{127}(2-2^{-23})$ $(\simeq 3.40\times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80\times 10^{308})$
正の最小値	2^{-126} $(\simeq 1.17\times 10^{-38})$	2^{-1022} $(\simeq 2.23\times 10^{-308})$
負の最大値	-2^{-126} $(\simeq -1.17\times 10^{-38})$	-2^{-1022} $(\simeq -2.23\times 10^{-308})$
最小値	$-2^{127}(2-2^{-23})$ $(\simeq -3.40\times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80\times 10^{308})$

索 引

アプリケーションシステム

科学技術計算ライブラリ
ASL C 言語インタフェース
ユーザーズガイド

〈 基本機能編 第 2 分冊 〉

2023 年 3 月 ASL (1.1)
付属説明書 3.0.0-230301

日 本 電 気 株 式 会 社