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5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 5 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.

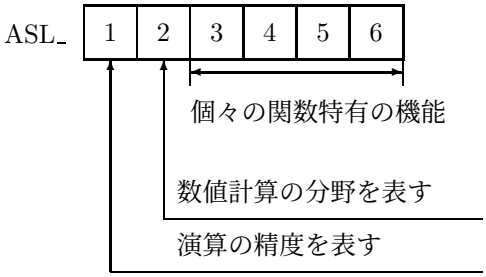


図 1−1　関数名の構成要素

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　特殊関数

2.1　概 要

本章では特殊関数の関数値を求める関数について説明する. 特殊関数の計算方法には,

- テーラー展開や漸近展開による方法
- 近似式による方法
- 連分数による方法
- 漸化式による方法

などがあるが, ここでは区間を分割し最も最適と考えられる計算方法を各区間に対し適用する方法をとった.

2.1.1 使用上の注意

(1) 第 2 種ベッセル関数 $N_\nu(z)$ や第 2 種球ベッセル関数 $n_\nu(z)$ はここでの $Y_\nu(z), y_\nu(z)$ と同じものである.

(2) 実数次や整数次の各種ベッセル関数は, 引数 z や次数 ν が大きいほど, 計算時間が多くかかり, 一般に $|\nu| < 1000.0, |z| < 1000.0$ とすることが望ましい.

(3) 第 1 種のベッセル, 変形ベッセル, 球ベッセル, 変形球ベッセル関数で次数が 1 ずつ変化する連続した値を得たいときは, 高い次数の連続する 2 個の値をこの関数より求め, あとは高い次数から順次次数を減ずる方向に漸化式を用いて計算すると良い.

(4) ディログ関数 $Li_2(x)$・デバイ関数 $F_D(x)$・球面調和関数 $P_n^{*m}(x)$ はベクトル処理・共通な係数・漸化式を用いて値が求められることにより, 個別に計算するよりはまとめて計算する方が効率的である.

(5) $J_{-\nu}(x), I_{-\nu}(x), Y_{-\nu}(x), i_{-n}(x), Y_{-n}(x)$ は漸化式を用いて計算する. 漸化式は各関数の注意事項を参照.

(6) 半奇数次のベッセル関数や変形ベッセル関数は球ベッセル関数を用いる方が効率的である.

$$J_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot j_n(x), Y_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot y_n(x)$$

$$I_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot i_n(x), K_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} \cdot k_n(x)$$

2.1.2 使用しているアルゴリズム

2.1.2.1 ベッセル関数

(1) 第 1 種 0 次, 1 次ベッセル関数 $J_0(x), J_1(x)$

① $x < 0.0$ のとき
$J_0(x) = J_0(-x),\ J_1(x) = -J_1(-x)$ として $J_0(-x),\ J_1(-x)$ について以下に述べる方法を適用する.

② $0.0 \leq x < 4.0$ のとき次の式を最良近似式化して求める.

$$J_0(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(k!)^2} \left(\frac{x}{2}\right)^{2k}$$
$$J_1(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1}$$

③ $4.0 \leq x \leq 8.0$ のとき次の式を最良近似式化して求める.

$$J_0(x) = \sum_{k=0}^{\infty} \frac{J_0^{(k)}(6)}{k!}(x-6)^k$$
$$J_1(x) = \sum_{k=0}^{\infty} \frac{J_1^{(k)}(6)}{k!}(x-6)^k$$

($J_0^{(k)}(6), J_1^{(k)}(6)$ は $x = 6.0$ での k 階微分値)
最良近似式化については 2.1.2.22 に記述されている. また, $J_0^{(k)}(6), J_1^{(k)}(6)$ 計算法は文献 (19) に記述されている.

④ $x > 8.0$ のとき次の漸近展開式で求める.

$$J_0(x) \text{ または } J_1(x) = \frac{P\cos(\phi) - Q\sin(\phi)}{\sqrt{x}}.$$

ここで
$J_0(x)$ のとき

$$P = \frac{\sum_{n=0}^{m} a_n^{(1)} \left(\frac{8}{x}\right)^{2n}}{\sum_{n=0}^{m'} b_n^{(1)} \left(\frac{8}{x}\right)^{2n}}$$
$$\phi = x - \frac{\pi}{4}$$

$J_1(x)$ のとき

$$Q = \frac{\sum_{n=0}^{m} c_n^{(2)} \left(\frac{8}{x}\right)^{2n}}{\sum_{n=0}^{m'} d_n^{(2)} \left(\frac{8}{x}\right)^{2n}}$$
$$\phi = x - \frac{3}{4}\pi$$

である. 係数 $a_n^{(1)}, a_n^{(2)}, b_n^{(1)}, b_n^{(2)}, c_n^{(1)}, c_n^{(2)}, d_n^{(1)}, d_n^{(2)}$ は文献 (2) に記述されている.

(2) 第 2 種 0 次, 1 次ベッセル関数 $Y_0(x), Y_1(x)$ $(x > 0.0)$

① $0.0 < x < 4.0$ のとき次の式を最良近似式化して求める.

$$\begin{aligned} Y_0(x) &= \frac{2}{\pi}\left[J_0(x)\{\log\left(\frac{x}{2}\right)+\gamma\} - \sum_{k=1}^{\infty}\left\{\frac{(-1)^k}{(k!)^2}\left(\frac{x}{2}\right)^{2k}\sum_{m=1}^{k}\frac{1}{m}\right\}\right] \\ Y_1(x) &= \frac{2}{\pi}\left[J_1(x)\{\log(\frac{x}{2})+\gamma\}\right] \\ &+ \frac{2}{\pi}\left[-\frac{1}{x} - \frac{1}{2}\sum_{k=0}^{\infty}\left\{\frac{(-1)^k}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k+1}\left(\sum_{m=1}^{k}\frac{1}{m}+\sum_{m=1}^{k+1}\frac{1}{m}\right)\right\}\right] \end{aligned}$$

ここで γ はオイラーの定数で $0.5772\cdots$ である.

② $4.0 \le x \le 8.0$ のとき次の式を最良近似式化して求める.

$$\begin{aligned} Y_0(x) &= \sum_{k=0}^{\infty}\frac{Y_0^{(k)}(6)}{k!}(x-6)^k \\ Y_1(x) &= \sum_{k=0}^{\infty}\frac{Y_1^{(k)}(6)}{k!}(x-6)^k \end{aligned}$$

最良近似式化については 2.1.2.22 に記述されている. ここで, $Y_0(6), Y_1(6)$ の k 階微分値は, ベッセル関数を $Z_k(x)$ として,

$$\begin{aligned} Z_k'(x) &= Z_{k-1}(x) - \frac{k}{x}Z_k(x) \\ Z_1^{(k)}(x) &= -Z_0^{(k+1)}(x) \end{aligned}$$

より漸化式を作り求める. この方法については文献 (19) に詳細が記述されている.

③ $x > 8.0$ のとき次の漸近展開式で求める.

$$Y_0(x) \text{ または } Y_1(x) = \frac{P\sin(\phi) + Q\cos(\phi)}{\sqrt{x}}.$$

ここで P, Q, ϕ は $J_0(x), J_1(x)$ の場合と同じである. 係数 $a_n^{(1)}, a_n^{(2)}, b_n^{(1)}, b_n^{(2)}$ は文献 (2) に記述されている.

(3) 第 1 種整数次ベッセル関数 $J_n(x)$

$J_n(x)$ は x, n の符号に応じて $J_{|n|}(|x|)$ を用いて表 2−1 のように表せる.

表 2−1　x, n の符号と $J_n(x)$ の計算方法

—	$n < 0$	$n \ge 0$
$x < 0.0$	$J_{\lvert n\rvert}(\lvert x\rvert)$	$(-1)^n J_{\lvert n\rvert}(\lvert x\rvert)$
$x \ge 0.0$	$(-1)^n J_{\lvert n\rvert}(\lvert x\rvert)$	$J_{\lvert n\rvert}(\lvert x\rvert)$

したがって以下に, $J_{|n|}(|x|)$ の計算法について記述する. 表記を簡略化するため, $|n| \to n, |x| \to x$ と表す.

① $n = 0, 1$ のときは $J_0(x), J_1(x)$ を利用する.

② $n \ge 2$ のとき

(a) $0.0 \le x^2 < \left\{\begin{matrix}\text{倍精度}: 0.1\mathrm{n} + 0.4 \\ \text{単精度}: 1.9\mathrm{n} + 7.6\end{matrix}\right\}$ で $n < \left\{\begin{matrix}\text{倍精度}: 170 \\ \text{単精度}: 34\end{matrix}\right\}$ 場合

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{9}\frac{(-1)^k(\frac{x}{2})^{2k}}{k!(n+k)!}$$

のべき級数展開で求める.

(b) $x^2 \ge \left\{\begin{matrix}\text{倍精度}: 0.1\mathrm{n} + 0.4 \\ \text{単精度}: 1.9\mathrm{n} + 7.6\end{matrix}\right\}$ または $n \ge \left\{\begin{matrix}\text{倍精度}: 170 \\ \text{単精度}: 34\end{matrix}\right\}$の場合

i. $x \leq n$ ならば

$T_{n+k+1} = 0.0$ とし, 次の漸化式を用いて連分数近似におけるパラメータ T_i $(i = n, n+1, \cdots, n+k-1, n+k)$ を計算する.

$$T_i = \frac{x^2}{2i - T_{i+1}} \ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n - T_{n+1}}$$

(T_n は $\frac{J_n(x)}{J_{n-1}(x)}$ の値)

k の値は表 2−2 に示すようにとる.

表 2−2　k の値

単精度演算	倍精度演算
$k = \lfloor \frac{(3.0+1.5\sqrt{n})x}{n} + 1.5 \rfloor$	$k = \lfloor \frac{(3.0+2.8\sqrt{n})x}{n} + 5.2 \rfloor$

$B_n = T_n, B_{n-1} = 1.0$ とし, 次の漸化式を用いて B_i $(i = 1, 2, \cdots, n-2)$ を計算する.

$$B_{i-1} = \frac{2i}{x} B_i - B_{i+1} \ (i = 2, \cdots, n-2, n-1)$$

(B_1 は $\frac{J_1(x)}{J_{n-1}(x)}$ の値)

$J_0(x) \simeq 0$ または $J_1(x) \simeq 0$ のとき精度が低下するので次の処理を行う.

$$K = \lfloor 1.27324x + 3 \rfloor \pmod 4$$

として

$K = 0$ または 3 のとき

$$B = \frac{2B_1}{x} - B_2, \ J = J_0(x)$$

$K = 1$ または 2 のとき

$$B = B_1, \ J = J_1(x)$$

得られた T_n と B と J の値より

$$J_n(x) = \frac{T_n J}{B}$$

とする.

ii. $x > n$ ならば

$J_1(x), J_0(x)$ を初期値とし, 漸化式

$$J_{k+1}(x) = \frac{2k}{x} J_k(x) - J_{k-1}(x) \ (k = 1, 2, \cdots, n-1)$$

を用いて $J_n(x)$ を求める.

(4) 第 2 種整数次ベッセル関数 $Y_n(x)$ $(x > 0.0)$

① $n < 0$ のとき

$Y_n(x) = (-1)^n Y_{-n}(x)$ として $Y_{-n}(x)$ について以下に述べる方法を適用する.

② $n = 0, 1$ のときは $Y_0(x), Y_1(x)$ を利用する.

③ $n \geq 2$ のときは

$Y_1(x), Y_0(x)$ を初期値とし, 漸化式

$$Y_{k+1}(x) = \frac{2k}{x} Y_k(x) - Y_{k-1}(x) \ (k = 1, 2, \cdots, n-1)$$

を用いて $Y_n(x)$ を求める.

(5) 第 1 種実数次ベッセル関数 $J_\nu(x)$

① ν が整数のとき $n = \nu$ として整数次ベッセル関数の関数を利用する.

② ν が整数でない実数のとき $(x > 0.0, \nu > 0.0)$

(a) $0.0 < x^2 < \left\{\begin{array}{l}\text{倍精度}: 0.1\nu + 0.4\\ \text{単精度}: 1.9\nu + 7.6\end{array}\right\}$ で $\nu < \left\{\begin{array}{l}\text{倍精度}: 170.0\\ \text{単精度}: 34.0\end{array}\right\}$ の場合

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{9} \frac{(-1)^k (\frac{x}{2})^{2k}}{k!\Gamma(\nu + k + 1)}$$

のべき級数展開で求める.

(b) $x^2 \geq \left\{\begin{array}{l}\text{倍精度}: 0.1\nu + 0.4\\ \text{単精度}: 1.9\nu + 7.6\end{array}\right\}$ または $\nu \geq \left\{\begin{array}{l}\text{倍精度}: 170.0\\ \text{単精度}: 34.0\end{array}\right\}$ の場合

i. $x > \left\{\begin{array}{l}\text{倍精度}: 30.0\\ \text{単精度}: 15.0\end{array}\right\}$ かつ $x \geq 0.55\nu^2$ ならば

$$J_\nu(x) = \sqrt{\frac{2}{\pi x}}(P\cos(\phi) - Q\sin(\phi))$$

の漸近展開式で求める. ここで

$$\begin{aligned} P &= 1 + \sum_{k=1}^{m} (-1)^k \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2)\cdots(4\nu^2 - (4k-1)^2)}{(2k)!(8x)^{2k}} \\ Q &= \sum_{k=0}^{m'} (-1)^k \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2)\cdots(4\nu^2 - (4k+1)^2)}{(2k+1)!(8x)^{2k+1}} \\ \phi &= x - (\frac{\nu}{2} + \frac{1}{4})\pi \end{aligned}$$

$(m, m'$ は末項がそれ以前の計算値に影響しなくなるまでの値である. $)$

ii. 上述以外のとき次の漸化式 (逆行法) より求める.

δ を ν の小数部, n をその整数部, M を十分大きい値, a を正の最小値 (浮動小数点データの内部表現であらわすことのできる最も 0.0 に近い正の値) とする.

$F_{\delta+M+1}(x) = 0, F_{\delta+M}(x) = a$ を初期値とし

$$F_{\delta+k-1}(x) = \frac{2(\delta + k)}{x} F_{\delta+k}(x) - F_{\delta+k+1}(x) \ (k = M, M-1, \cdots, 1)$$

として

$$J_\nu(x) = \frac{F_{\delta+n}(x)(\frac{x}{2})^\delta}{\sum_{n=0}^{\lfloor \frac{M}{2} \rfloor} \frac{(\delta + 2m)\Gamma(\delta + m)}{m!} F_{\delta+2m}(x)}$$

M の値は x, ν の値より近似式を使って決める.

(6) 第 2 種実数次ベッセル関数 $Y_\nu(x)$ $(x > 0.0)$

① ν が整数のとき $n = \nu$ として整数次ベッセル関数の関数を利用する.

② ν が整数でない実数のとき $(\nu > 0.0)$

ν を整数部 n と小数部 δ に分解する.

(a) $0.0 < x \leq 4.0$ の場合

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}$$

に $J_\nu(x), J_{-\nu}(x)$ のべき級数展開式を代入し, 項をまとめ, 桁落ちが生ずる部分を最良近似式化する吉田・二宮の方法で計算する. 以下にその内容を示す (参考文献 (5) 参照).

i. $0.0 < \nu \leq 0.5$ では

$$Y_\nu(x) = \sum_{k=0}^{\infty} - \left(-\frac{x^2}{4}\right)^k \frac{\tilde{A}_k(\nu) + \tilde{B}_k(\nu)}{\sin(\nu\pi)}$$

より計算する. $\tilde{A}_k(\nu)$ は

$$\tilde{A}_0(\nu) = (\nu - \nu_0)\sum_{k=0}^{M} P_k^{(1)}\nu^k \ (\nu_0 = 0.221521\cdots)$$

を初期値として, 漸化式

$$\tilde{A}_k(\nu) = \frac{\dfrac{1}{k!}\left\{\dfrac{1}{\Gamma(k-\nu)} + \dfrac{\cos(\nu\pi)}{\Gamma(k+\nu)}\right\} + \tilde{A}_{k-1}(\nu)}{(k+\nu)(k-\nu)}$$

で計算する. $\tilde{B}_k(\nu)$ は

$$\tilde{B}_k(\nu) = \frac{1}{k!}\left\{\frac{\phi_1}{\Gamma(k+1-\nu)} + \frac{\phi_2\cos(\nu\pi)}{\Gamma(k+1+\nu)}\right\}$$

として計算する. ここで, $(\frac{x}{2})^\nu < 0.5$ または $(\frac{x}{2})^\nu > 2.0$ のときは,

$$\phi_1 = \frac{(\frac{x}{2})^{-\nu} - 1}{\nu},\ \phi_2 = \frac{1 - (\frac{x}{2})^\nu}{\nu}$$

とし, また $0.5 \le (\frac{x}{2})^\nu \le 2.0$ のときは,

$$\phi_1 = -f(-\nu\log(\frac{x}{2}))\log(\frac{x}{2}),\ \phi_2 = -f(\nu\log(\frac{x}{2}))\log(\frac{x}{2})$$

とする. $f(t)$ は $\dfrac{e^t - 1}{t}$ の関数の最良近似式を用いて計算する.

ii. $0.5 < \nu \le 1.5$ ならば

A. $\delta \le 0.5$ のとき $\delta = \delta + 1$, $n = n - 1$ として $0.5 < \delta \le 1.5$ の範囲で計算する.

B. $0.5 < \delta \le 1.5$ のとき

$$Y_\delta(x) = -\frac{\dfrac{2^{1+\alpha}}{x^{1+\alpha}\Gamma(1-\alpha)} + \displaystyle\sum_{k=0}^{\infty}\left\{\frac{x}{2}\left(-\frac{x^2}{4}\right)^k(\tilde{C}_k(\alpha) + \tilde{D}_k(\alpha))\right\}}{\sin(\alpha\pi)}$$

とし ($\alpha = \delta - 1$ とする), $\tilde{C}_k(\alpha)$ や $\tilde{D}_k(\alpha)$ を $0.0 < \nu \le 0.5$ のときと同様に最良近似式を利用して求める.

さらに

$$Y_{\delta+1}(x) = \frac{-\dfrac{2^\alpha\{4(\alpha+1) + x^2\}}{x^{\alpha+2}\Gamma(1-\alpha)} + \displaystyle\sum_{k=0}^{\infty}\left\{\left(-\frac{x^2}{4}\right)^{k+1}(\tilde{E}_k(\alpha) + \tilde{F}_k(\alpha))\right\}}{\sin(\alpha\pi)}$$

とし, $\tilde{E}_k(\alpha)$ や, $\tilde{F}_k(\alpha)$ を $0.0 < \nu \le 0.5$ のときと同様に最良近似式を利用して求める.

iii. $\nu > 1.5$ ならば

ii. で得られた $Y_\delta(x)$ と $Y_{\delta+1}(x)$ から, 漸化式

$$Y_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}Y_{k+\delta}(x) - Y_{k+\delta-1}(x)\ (k = 1, 2, \cdots, n-1)$$

を用いて $Y_\nu(x)$ を得る.

(b) $4.0 < x \le \left\{\begin{array}{l}\text{倍精度 : } 30.0\\ \text{単精度 : } 15.0\end{array}\right\}$の場合

i. $0.0 < \nu < 2.0$ ならば

δ または $1 - \delta$ が $\sqrt{\text{誤差判定のための単位}/4}$ より小なる場合

n または $n + 1$ を次数とする整数次ベッセル関数で近似する.

漸化式 (逆行法) により $J_\delta(x)$ と $J_{\delta+1}(x)$ を求める. また $t = 1 - \delta$ とし,

同様に $J_t(x), J_{t+1}(x)$ を求め,

$$J_{-\delta}(x) = \frac{2(1-\delta)}{x}J_t(x) - J_{t+1}(x)$$

として $J_{-\delta}(x)$ を計算する. ((5)$J_\nu(x)$ 参照).

$$Y_\delta(x) \quad = \quad \frac{J_\delta(x)\cos(\delta\pi) - J_{-\delta}(x)}{\sin(\delta\pi)}$$

$$Y_{\delta+1}(x) = \frac{J_{\delta+1}(x)\cos(\delta\pi) - \frac{2\delta J_{-\delta}(x)}{x} - J_t(x)}{\sin(\delta\pi)}$$

として $Y_\delta(x)$ と $Y_{\delta+1}(x)$ を得る.

ii. $\nu \geq 2.0$ ならば

i. で得られた $Y_\delta(x)$ と $Y_{\delta+1}(x)$ より漸化式

$$Y_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}Y_{k+\delta}(x) - Y_{k+\delta-1}(x)\ (k = 1, 2, \cdots, n-1)$$

を用いて $Y_\nu(x)$ を得る.

(c) $x >$ {倍精度 : 30.0 / 単精度 : 15.0} の場合

i. $x \leq 0.55\nu^2$ ならば

$$Y_\nu(x) = \sqrt{\frac{2}{\pi x}}(P\sin(\phi) + Q\cos(\phi))$$

の漸近展開式で求める. ここで P, Q, ϕ は (5)$J_\nu(x)$ の計算時と同様にして求める.

ii. $x > 0.55\nu^2$ ならば

$m = \lfloor\sqrt{\frac{x}{0.55}}\rfloor - 1$ とする.

$Y_{m+\delta}(x)$ と $Y_{m+\delta+1}(x)$ を上の漸近展開式で求め, 漸化式

$$Y_{m+\delta+k+1}(x) = \frac{2(m+\delta+k)}{x}Y_{m+\delta+k}(x) - Y_{m+\delta+k-1}(x)$$
$$(k = 1, 2, \cdots, n-m-1)$$

により $Y_\nu(x)$ を得る.

(7) 複素変数第 1 種整数次ベッセル関数 $J_n(z)$

$$J_n(z) = (-i)^n I_n(iz)\ (i = \sqrt{-1})$$

より求める.

(8) 複素変数第 2 種整数次ベッセル関数 $Y_n(z)$ $(|z| > 0.0)$

① $n < 0$ のとき

$Y_n(z) = (-1)^n Y_{-n}(z)$ として $Y_{-n}(z)$ について以下に述べる方法を適用する.

② $n \geq 0$ のとき

z の虚部が負ならば $Y_n(\bar{z}) = \overline{Y_n(z)}$ として求める.

$$Y_n(z) = i^{n+1}I_n(-iz) - \frac{2}{\pi}(-i)^n K_n(-iz)\ (i = \sqrt{-1})$$

より求める.

2.1.2.2 変形ベッセル関数

(1) 第 1 種 0 次, 1 次変形ベッセル関数 $I_0(x), I_1(x)$

$x < 0.0$ のとき

$I_0(x) = I_0(-x), I_1(x) = -I_1(-x)$ として $I_0(-x), I_1(-x)$ について以下に述べる方法を適用する.

・単精度

① $0.0 \leq x \leq 3.75$ のとき次の式を最良近似式化して求める.

$$I_0(x) = \sum_{k=0}^{\infty}\frac{1}{(k!)^2}\left(\frac{x}{2}\right)^{2k}$$

$$I_1(x) = \sum_{k=0}^{\infty}\frac{1}{k!(k+1)!}\left(\frac{x}{2}\right)^{2k+1}$$

② $x > 3.75$ のとき次の近似式で求める.

$$I_0(x) = \frac{\sum_{n=0}^{8} a_n^{(1)} \left(\frac{3.75}{x}\right)^n}{e^{-x}\sqrt{x}}$$

$$I_1(x) = \frac{\sum_{n=0}^{8} a_n^{(2)} \left(\frac{3.75}{x}\right)^n}{e^{-x}\sqrt{x}}$$

係数 $a_n^{(1)}, a_n^{(2)}$ は文献 (1) に記述されている.

・倍精度

① $0.0 \leq x \leq 8.0$ のとき次の式を最良近似式化して求める.

$$I_0(x) = \sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k}$$

$$I_1(x) = \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1}$$

② $8.0 < x < 24.0$ のとき, 次の近似式で求める.

$$I_0(x) = \frac{\sum_{n=0}^{28} a_n^{(3)} x^{-n}}{e^{-x}\sqrt{x}}$$

$$I_1(x) = \frac{\sum_{n=0}^{28} a_n^{(4)} x^{-n}}{e^{-x}\sqrt{x}}$$

③ $x \geq 24.0$ のとき, 次の式を最良近似化して求める.

$$I_0(x) = \frac{e^x}{\sqrt{2\pi x}} \sum_{k=0}^{\infty} \frac{1^2 \cdot 1^2 \cdot 3^2 \cdots (2k-1)^2}{k!(8x)^k}$$

$$I_1(x) = \frac{e^x}{\sqrt{2\pi x}} \left\{ 1 + \sum_{k=1}^{\infty} \frac{(-3) \cdot 5 \cdots ((2k-1)^2 - 4)}{k!(8x)^k} \right\}$$

係数 $a_n^{(3)}, a_n^{(4)}$ は文献 (21) を参考にテレスコーピング計算をして求めた.
最良近似式化については 2.1.2.22 に記述されている.

(2) 第 2 種 0 次, 1 次変形ベッセル関数 $K_0(x), K_1(x)$ $(x > 0.0)$

・単精度

① $0.0 < x \leq 2.0$ のとき次の式を最良近似式化して求める.

$$K_0(x) = -\gamma + \sum_{k=1}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \{(\sum_{m=1}^{k} \frac{1}{m}) - \gamma\} - \log\left(\frac{x}{2}\right) \{\sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k}\}$$

$$K_1(x) = \frac{1 + \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+2} (2\gamma - \sum_{m=1}^{k} \frac{1}{m} - \sum_{m=1}^{k+1} \frac{1}{m})}{x} + \log\left(\frac{x}{2}\right) \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1}$$

(ただし, $\sum_{m=1}^{0} \frac{1}{m} = 0$)

② $x > 2.0$ のとき次の近似式で求める.

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{6} a_n^{(1)} \left(\frac{2}{x}\right)^n$$

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{6} a_n^{(2)} \left(\frac{2}{x}\right)^n$$

係数 $a_n^{(1)}, a_n^{(2)}$ は文献 (1) に記述されている.

・倍精度

① $0.0 < x \leq 2.0$ のとき次の式を最良近似式化して求める.

$$K_0(x) = -\gamma + \sum_{k=1}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k} \{(\sum_{m=1}^{k} \frac{1}{m}) - \gamma\} - \log\left(\frac{x}{2}\right) \{\sum_{k=0}^{\infty} \frac{1}{(k!)^2} \left(\frac{x}{2}\right)^{2k}\}$$

$$K_1(x) = \frac{1 + \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+2} (2\gamma - \sum_{m=1}^{k} \frac{1}{m} - \sum_{m=1}^{k+1} \frac{1}{m})}{x} + \log\left(\frac{x}{2}\right) \sum_{k=0}^{\infty} \frac{1}{k!(k+1)!} \left(\frac{x}{2}\right)^{2k+1}$$

(ただし, $\sum_{m=1}^{0} \frac{1}{m} = 0$)

ここで γ はオイラーの定数で $0.5772\cdots$ である.

② $2.0 < x \leq 5.0$ のとき次の式を最良近似式化して求める.

$$K_0(x) = \sum_{k=0}^{\infty} \frac{K_0^{(k)}(3.5)}{k!} (x - 3.5)^k$$

$$K_1(x) = \sum_{k=0}^{\infty} \frac{K_1^{(k)}(3.5)}{k!} (x - 3.5)^k$$

ここで $K_0^{(k)}(3.5), K_1^{(k)}(3.5)$ は $x = 3.5$ での k 階微分値である. また $K_0^{(k)}(3.5), K_0^{(k)}(3.5)$ 計算法は文献 (19) を参考に次のように計算できる.

$$t_0 = 1, t_1 = 0, t_2 = 0, u_0 = 0, u_1 = 1$$

$$v_0 = 0, v_1 = 0, w_0 = 0, w_1 = 0, w_2 = -1$$

$$i = 3, x = 3.5$$

反復
$$\left[\begin{array}{l} t'_j = v_j - u_j \ (j = 0, \cdots, i-2) \\ t'_{i-1} = 0, t'_i = 0 \\ u'_0 = w_0 - t_0 \\ u'_j = w_j - t_j - u_{j-1} \ (j = 1, \cdots, i-1) \\ t_j = t'_j \ (j = 0, \cdots, i), u_j = u'_j \ (j = 0, \cdots, i-1) \\ K_0^{(i)}(x) = \sum_{j=0}^{i-2} t_j x^{-j} K_0(x) + \sum_{j=0}^{i-1} u_j x^{-j} K_1(x) \\ v_{j+1} = -j t_j (j = 1, i-2) \\ w_{j+1} = -j u_j (j = 1, i-1) \\ i = i + 1 \end{array}\right.$$

$$K_1^{(i)}(x) = -K_0^{(i+1)}(x), K_0''(x) = K_0(x) + \frac{K_1(x)}{x}, K_0'(x) = -K_1(x)$$

③ $5.0 < x < 24.0$ のとき

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{18} a_n^{(3)} x^{-n}$$

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{n=0}^{18} a_n^{(4)} x^{-n}$$

係数 $a_n^{(3)}, a_n^{(4)}$ は文献 (21) を参考にテレスコーピング計算をして求めた.

④ $x \geq 24.0$ とき, 次の式を最良近似式化して求める.

$$K_0(x) = \sqrt{\frac{\pi}{2x}} e^{-x} \sum_{k=0}^{\infty} (-1)^k \frac{1^2 \cdot 1^2 \cdot 3^2 \cdots (2k-1)^2}{k!(8x)^k}$$

$$K_1(x) = \sqrt{\frac{\pi}{2x}} e^{-x} \{1 + \sum_{k=1}^{\infty} \frac{3 \cdot (-5) \cdots (4-(2k-1)^2)}{k!(8x)^k}\}$$

最良近似式化については, 2.1.2.22 に記述されている.

(3) 第 1 種整数次変形ベッセル関数 $I_n(x)$

$I_n(x)$ は x, n の符号に応じて $I_{|n|}(|x|)$ を用いて表 2–3 のように表せる. したがって以下に, $I_{|n|}(|x|)$ の計算法

表 2–3　x, n の符号と $I_n(x)$ の計算方法

—	$n < 0$	$n \geq 0$
$x < 0.0$	$(-1)^n I_{\|n\|}(\|x\|)$	$(-1)^n I_{\|n\|}(\|x\|)$
$x \geq 0.0$	$I_{\|n\|}(\|x\|)$	$I_{\|n\|}(\|x\|)$

について記述する. 表記を簡略化するために $|n| \to n$, $|x| \to x$ と表す.

① $n = 0, 1$ のときは $I_0(x), I_1(x)$ を利用する.

② $n \geq 2$ のとき

(a) $0.0 \leq x^2 < \begin{Bmatrix} \text{倍精度} : 0.1n + 0.4 \\ \text{単精度} : 1.9n + 7.6 \end{Bmatrix}$ で $n < \begin{Bmatrix} \text{倍精度} : 170 \\ \text{単精度} : 34 \end{Bmatrix}$ の場合

$$I_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{9} \frac{(\frac{x}{2})^{2k}}{k!(n+k)!}$$

のべき級数展開で求める.

(b) $x^2 \geq \begin{Bmatrix} \text{倍精度} : 0.1n + 0.4 \\ \text{単精度} : 1.9n + 7.6 \end{Bmatrix}$ または $n \geq \begin{Bmatrix} \text{倍精度} : 170 \\ \text{単精度} : 34 \end{Bmatrix}$ の場合

i. $x \leq n$ ならば

$T_{n+k+1} = 0.0$ とし, 次の漸化式を用いて連分数近似におけるパラメータ T_i $(i = n, n+1, \cdots, n+k-1, n+k)$ を計算する.

$$T_i = \frac{x^2}{2i + T_{i+1}} \ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n + T_{n+1}}$$

(T_n は $\dfrac{I_n(x)}{I_{n-1}(x)}$ の値)

k の値は表 2–4 に示すようにとる. $B_n = T_n, B_{n-1} = 1.0$ とし, 次の漸化式を用いて B_i $(i = 1, 2, \cdots, n-2)$ を計算する.

$$B_{i-1} = \frac{2i}{x} B_i + B_{i+1} \ (i = 2, \cdots, n-2, n-1)$$

表 2−4　k の値

単精度演算	倍精度演算
$k=\lfloor\frac{(4.6+0.5\sqrt{n})x}{n}+2.0\rfloor$	$k=\lfloor\frac{(6.0+1.2\sqrt{n})x}{n}+6.0\rfloor$

(B_1 は $\dfrac{I_1(x)}{I_{n-1}(x)}$ の値)
得られた T_n と B_1 と $I_1(x)$ の値より

$$I_n(x)=\frac{T_nI_1(x)}{B_1}$$

とする.

ii. $x>n$ ならば
a を正の最小値 (浮動小数点データの内部表現であらわすことのできる最も 0.0 に近い正の値) ,
M を n より十分大きい値とする.
$G_{M+1}(x)=0.0, G_M(x)=a$ を初期値とし

$$G_{k-1}(x)=\frac{2k}{x}G_k(x)+G_{k+1}(x)\ (k=M,M-1,\cdots,1)$$

として

$$I_n(x)=\frac{G_n(x)e^x}{\sum_{m=0}^{M}\varepsilon_mG_m(x)}\ (\varepsilon_0=1,\varepsilon_m=2\ (m\geq 1))$$

とする.

(4) 第 2 種整数次変形ベッセル関数 $K_n(x)$ $(x>0.0)$

① $n<0$ のとき
$K_n(x)=K_{-n}(x)$ として $K_{-n}(x)$ について以下に述べる方法を適用する.

② $n=0,1$ のときは $K_0(x), K_1(x)$ を利用する.

③ $n\geq 2$ のときは
$K_1(x), K_0(x)$ を初期値とし, 漸化式

$$K_{k+1}(x)=\frac{2k}{x}K_k(x)+K_{k-1}(x)\ (k=1,2,\cdots,n-1)$$

を用いて $K_n(x)$ を求める.

(5) 第 1 種実数次変形ベッセル関数 $I_\nu(x)$

① ν が整数のとき $n=\nu$ として整数次変形ベッセル関数の関数を利用する.

② ν が整数でない実数のとき $(x>0.0, \nu>0.0)$

(a) $0.0\leq x^2<\left\{\begin{array}{l}\text{倍精度}:0.1\nu+0.4\\ \text{単精度}:1.9\nu+7.6\end{array}\right\}$ で $\nu<\left\{\begin{array}{l}\text{倍精度}:170.0\\ \text{単精度}:34.0\end{array}\right\}$ の場合

$$I_\nu(x)=\left(\frac{x}{2}\right)^\nu\sum_{k=0}^{9}\frac{(\frac{x}{2})^{2k}}{k!\Gamma(\nu+k+1)}$$

のべき級数展開で求める.

(b) $x^2\geq\left\{\begin{array}{l}\text{倍精度}:0.1\nu+0.4\\ \text{単精度}:1.9\nu+7.6\end{array}\right\}$ または $\nu\geq\left\{\begin{array}{l}\text{倍精度}:170.0\\ \text{単精度}:34.0\end{array}\right\}$ の場合

i. $x>\left\{\begin{array}{l}\text{倍精度}:30.0\\ \text{単精度}:15.0\end{array}\right\}$ かつ $x\geq 0.55\nu^2$ ならば

$$I_\nu(x)=\frac{e^x}{\sqrt{2\pi x}}\sum_{k=0}^{m}(-1)^k\frac{(4\nu^2-1^2)(4\nu^2-3^2)\cdots(4\nu^2-(2k-1)^2)}{k!(8x)^k}$$

の漸近展開式で求める.

(m は末項がそれ以前の計算値に影響しなくなるまでの値である.)

ii. x が前項以外のとき次の漸化式 (逆行法) より求める.

δ を ν の小数部, n をその整数部, M を十分大きい値, a を正の最小値 (浮動小数点データの内部表現であらわすことのできる最も 0.0 に近い正の値) とする.

$G_{\delta+M+1}(x)=0.0, G_{\delta+M}(x)=a$ を初期値とし,

$$G_{\delta+k-1}(x)=\frac{2(\delta+k)}{x}G_{\delta+k}(x)+G_{\delta+k+1}(x)\ (k=M,M-1,\cdots,1)$$

として

$$I_\nu(x)=\frac{1}{2}(\frac{x}{2})^\delta\frac{\Gamma(2\delta+1)}{\Gamma(\delta+1)}e^x\frac{G_{n+\delta}(x)}{\sum_{k=0}^{M}\frac{(\delta+k)\Gamma(2\delta+k)}{k!}G_{\delta+k}(x)}$$

M の値は x, ν の値より近似式を使って決める.

(6) 第 2 種実数次変形ベッセル関数 $K_\nu(x)$ $(x>0.0)$

① $\nu<0.0$ のとき

$K_\nu(x)=K_{-\nu}(x)$ として $K_{-\nu}(x)$ について以下に述べる方法を適用する.

② ν が整数のとき $n=\nu$ として整数次変形ベッセル関数の関数を利用する.

③ ν が整数でない実数のとき

x が小さい場合は

$$K_\nu(x)=\frac{\pi}{2}\frac{I_{-\nu}(x)-I_\nu(x)}{\sin(\nu\pi)}$$

に $I_\nu(x), I_{-\nu}(x)$ の級数展開を代入し, 項をまとめ, 桁落ちが生ずる部分を最良近似化する吉田・二宮の方法で計算する.

x が大きい場合は $\tau-$method による $K_n(x)$ の計算法を $K_\nu(x)$ 計算に拡張した方法を利用する. こうして $0.0\le\nu\le2.5$ の区間の $K_\nu(x)$ を計算し, $\nu>2.5$ については, 漸化式

$$K_{\nu+1}(x)=\frac{2\nu}{x}K_\nu(x)+K_{\nu-1}(x)$$

により求める. 以下にその内容を示す (文献 (8) 参照)

(a) $0.0\le\nu\le0.5$ の場合

i. $x<-0.75\nu^2+0.0235\nu+0.778$ のとき

$$K_\nu(x)=\frac{\sum_{k=0}^{\infty}\{\left(\frac{x}{2}\right)^{2k}(\tilde{A}_k(\nu)+\tilde{B}_k(\nu))\}}{\frac{\pi}{2}\sin(\nu\pi)}$$

により計算する. ただし, $\tilde{A}_k(\nu)$ は $\tilde{A}_0(\nu)=\nu\sum_{k=0}^{M}p_k^{(1)}\nu^{2k}$, $\tilde{A}_1(\nu)=\nu\sum_{k=0}^{M}q_k^{(1)}\nu^{2k}$ とし,

$\tilde{A}_0(\nu), \tilde{A}_1(\nu)$ を初期値として, $k\ge2$ について, 漸化式

$$\tilde{A}_k(\nu)=\frac{\frac{\nu}{k!}\left\{\frac{1}{\Gamma(k-\nu)}+\frac{1}{\Gamma(k+\nu)}\right\}+\tilde{A}_{k-1}(\nu)}{(k+\nu)(k-\nu)}$$

から求める.

$\tilde{B}_k(\nu)$ は

$$\phi_1 = (\frac{x}{2})^{-\nu}-1,\ \phi_2=1-(\frac{x}{2})^\nu$$

$((\frac{x}{2})^\nu<0.5$ または $(\frac{x}{2})^\nu>2.0$ のとき)

$$\phi_1 \quad = \quad f(-\nu\log(\frac{x}{2})), \phi_2 = -f(\nu\log(\frac{x}{2}))$$
$$(0.5 \leq (\frac{x}{2})^{\nu} \leq 2.0 \text{ のとき})$$

とし, $f(t)$ を $e^t - 1$ の関数の最良近似式を用いて計算し,

$$\tilde{B}_k(\nu) = \frac{1}{k!}\left(\frac{\phi_1}{\Gamma(k+1-\nu)} + \frac{\phi_2}{\Gamma(k+1+\nu)}\right)$$

として求める. p_k, q_k の係数は文献 (8) を参照して計算できる.

ii. $x \geq -0.75\nu^2 + 0.0235\nu + 0.778$ のとき

$$K_\nu(x) = \sqrt{\frac{1}{x}}e^{-x}\frac{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i\{\sum_{j=0}^{i}b_{ij}(\nu^2)^j\}}{\sum_{i=0}^{m}\left(\frac{1}{x}\right)^i e_i\Psi_i}$$

の τ − method の式により計算する. ただし, b_{ij}, e_i の係数は文献 (20) を参照して計算できる. また, Ψ_i は,

$$\Psi_0 = 1,\ \Psi i = \prod_{l=0}^{i-1}\{\nu^2 - (m-l+\frac{1}{2})^2\}\ (i \geq 1)$$

として求める. ($e_i = \sqrt{\frac{2}{\pi}}\frac{(m-i)!}{(m+1)!}\frac{p^*_{m,m-i}}{2^i}$ で $p^*_{m,m-i}$ は, ずらしルジャンドル多項式の係数)

(b) $0.5 < \nu \leq 2.5$ の場合

ν を整数部 n と小数部 δ に分解する. $\alpha = \delta - 1$ とする.

$\delta \leq 0.5$ のときは $\delta = \delta + 1$, $n = n - 1$, $\alpha = \alpha + 1$ とする.

i. K_δ の計算を以下の A. または B. の方法で行う.

A. $x < -0.675\delta^2 + 1.973\delta - 0.12$ のとき

$$K_\delta(x) = -\frac{\frac{2^{1+\alpha}}{x^{1+\alpha}\Gamma(1-\alpha)} + \sum_{k=0}^{\infty}\{(\frac{x}{2})^{2k+1}(\tilde{C}_k(\alpha) + \tilde{D}_k(\alpha))\}}{\frac{\pi}{2}\sin(\alpha\pi)}$$

とし, $\tilde{C}_k(\alpha)$ や $\tilde{D}_k(\alpha)$ を $\nu \leq 0.5$ の場合と同様に最良近似式を利用して求める.

B. $x \geq -0.675\delta^2 + 1.973\delta - 0.12$ のとき

$\nu \leq 0.5$ で x が大きい場合と同様に τ − method により $K_\delta(x)$ を計算する.

ii. $K_{\delta+1}$ の計算を以下の A. または B. の方法で行う.

A. $x < -0.277(\delta+1)^2 + 1.817(\delta+1) - 0.94$ のとき

$$K_{\delta+1}(x) = \frac{\frac{2^\alpha\alpha(4\alpha+5)}{x^\alpha\Gamma(1-\alpha)} + \sum_{k=0}^{\infty}\{(\frac{x}{2})^{2k+2}(\tilde{E}_k(\alpha) + \tilde{F}_k(\alpha))\}}{\frac{\pi}{2}\sin(\alpha\pi)}$$

とし, $\tilde{E}_k(\alpha)$ や $\tilde{F}_k(\alpha)$ を $v \leq 0.5$ の場合と同様に最良近似式を利用して求める.

B. $x \geq -0.277(\delta+1)^2 + 1.817(\delta+1) - 0.94$ のとき

$\nu \leq 0.5$ で x が大きい場合と同様に τ − method により $K_{\delta+1}(x)$ を計算する.

(c) $\nu > 2.5$ の場合

上記で得られた $K_\delta(x)$ と $K_{\delta+1}(x)$ から漸化式

$$K_{k+\delta+1}(x) = \frac{2(k+\delta)}{x}K_{k+\delta}(x) + K_{k+\delta-1}(x)\ (k = 1, 2, \cdots, n-1)$$

を用いて $K_\nu(x)$ を計算する.

(7) 複素変数第 1 種整数次変形ベッセル関数 $I_n(z)$

① $n < 0$ のとき

$I_n(z) = I_{-n}(z)$ として $I_{-n}(z)$ について以下に述べる方法を適用する.

② $n \geq 0$ のとき

(a) z の実部が負ならば $I_n(z) = (-1)^n I_n(-z)$ として $I_n(-z)$ について以下に述べる方法を適用する.

(b) z の実部が正であり,

i. $|z|^2 < \left\{\begin{array}{l}\text{倍精度}: 0.1n + 0.4 \\ \text{単精度}: 1.9n + 7.6\end{array}\right\}$ ならば

$$I_n(z) = \left(\frac{z}{2}\right)^n \sum_{k=0}^{9} \frac{(\frac{z}{2})^{2k}}{k!(n+k)!}$$

のべき級数展開で求める.

ii. $|z|^2 \geq \left\{\begin{array}{l}\text{倍精度}: 0.1n + 0.4 \\ \text{単精度}: 1.9n + 7.6\end{array}\right\}$の場合

A. $\Re(z) > 100.0$ または $|\Im(z)| > 100.0$ で $n \leq 15$ ならば漸近展開式

$$\begin{aligned} I_n(z) &= \frac{e^z}{\sqrt{2\pi z}} \sum_{k=0}^{m} (-1)^k \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2n-1)^2)}{k!(8z)^k} \\ &+ \frac{i(-1)^n e^{-z}}{\sqrt{2\pi z}} \sum_{k=0}^{m'} \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2n-1)^2)}{k!(8z)^k} \end{aligned}$$

(m, m' は末項がそれ以前の計算値に影響しなくなるまでの値である.)

B. それ以外のとき次の漸化式 (逆行法) より求める.

十分大きい M をとり, a を正の最小値 (浮動小数点データの内部表現であらわすことのできる最も 0.0 に近い正の値) とする.

$G_{M+1}(z) = 0.0, G_M(z) = a$ を初期値とし,

$$G_{k-1}(z) = \frac{2k}{z} G_k(z) + G_{k+1}(z) \ (k = M, M-1, \cdots, 1)$$

として

$$I_n(z) = \frac{G_n(z) e^z}{\sum_{m=0}^{M} \varepsilon_m G_m(z)} \ (\varepsilon_0 = 1, \varepsilon_m = 2 \ (m \geq 1))$$

M の値は z, n の値より近似式を使って決める (文献 (6), (7) 参照).

(8) 複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$ ($|z| > 0.0$)

① $\Re(z) < 0$ 負のとき

$$K_n(z) = (-1)^n K_n(-z) + \pi i I_n(-z) \cdot (\Im(-z) \text{ の符号})$$

として $K_n(-z)$ について以下に述べる方法を適用する.

② $\Re(z) > 0$ のとき

(a) $n < 0$ ならば

$K_n(z) = K_{-n}(z)$ として $K_{-n}(z)$ について以下に述べる方法を適用する.

(b) $0 \leq n < 2$ であり

i. $|\Im(z)| < \left\{\begin{array}{l}\text{倍精度}: -4.0\Re(z) + 8.0 \\ \text{単精度}: -2.25\Re(z) + 4.5\end{array}\right\}$のとき

$$\begin{aligned} K_0(z) &= -\{\gamma + \log(\frac{z}{2})\} I_0(z) + \sum_{k=1}^{n} \frac{(\frac{z}{2})^{2k}}{(k!)^2} (\sum_{m=1}^{k} \frac{1}{m}) \\ K_1(z) &= \frac{\frac{1}{z} - I_1(z) K_0(z)}{I_0(z)} \end{aligned}$$

(ここで, γ はオイラーの定数) により $K_0(z)$, $K_1(z)$ を計算する.

ii. $|\Im(z)| \geq \left\{\begin{array}{l}\text{倍精度}: -4.0\Re(z)+8.0\\ \text{単精度}: -2.25\Re(z)+4.5\end{array}\right\}$のとき

$$K_n(z) = \sqrt{\frac{1}{z}}e^{-z}\frac{\sum_{k=0}^{m} c_k z^{k-m}}{\sum_{k=0}^{m} d_k z^{k-m}}$$

の $\tau-$method により $K_0(z)$ と $K_1(z)$ を計算する. $K_0(z)$ および $K_1(z)$ に対応する係数 c_k, d_k は文献 (9) 参照.

(c) $n \geq 2$ の場合, $K_0(z)$ と $K_1(z)$ より漸化式

$$K_{k+1}(z) = \frac{2k}{z}K_k(z) + K_{k-1}(z)\ (k = 1, 2, \cdots, n-1)$$

にて求める.

2.1.2.3 球ベッセル関数

(1) 第 1 種整数次球ベッセル関数 $j_n(x)$ $(x \geq 0.0)$

① $n < 0$ のとき

$$j_n(x) = (-1)^n y_{-n-1}(x).$$

より求める.

② $n \geq 0$ のとき

(a) $x^2 < 0.1n + 0.47$ で $n < 35$ のとき

$$j_n(x) = x^n \sum_{k=0}^{9} \frac{(-\frac{x^2}{2})^k}{k!(2n+2k+1)!!}$$

のべき級数展開で求める.

(b) $x^2 \geq 0.1n + 0.47$ または $n \geq 35$ のとき

i. $n = 0, 1$ の場合

$j_0(0) = 1$, $j_1(0) = 0$,

$$j_0(x) = \frac{\sin(x)}{x},\ j_1(x) = \frac{\sin(x) - x\cos(x)}{x^2}$$

により計算する.

ii. $n \geq 2$ の場合

A. $x \leq n$ ならば

$T_{n+k+1} = 0.0$ とし, 次の漸化式を用いて連分数近似におけるパラメータ T_i $(i = n, n+1, \cdots, n+k-1, n+k)$ を計算する.

$$T_i = \frac{x^2}{2i - T_{i+1} + 1}\ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n - T_{n+1} + 1}$$

(T_n は $\dfrac{j_n(x)}{j_{n-1}(x)}$ の値)

k の値は表 2–5 に示すようにとる.

表 2–5　k の値

単精度演算	倍精度演算
$k = \lfloor \frac{(3.0+1.5\sqrt{n})x}{n} + 1.5 \rfloor$	$k = \lfloor \frac{(3.0+2.8\sqrt{n})x}{n} + 5.2 \rfloor$

$B_n = T_n, B_{n-1} = 1.0$ とし, 次の漸化式を用いて B_i $(i = 1, 2, \cdots, n-2)$ を計算する.

$$B_{i-1} = \frac{2i+1}{x} B_i - B_{i+1} \ (i = 2, \cdots, n-2, n-1)$$

$j_0(x) \simeq 0$ または $j_1(x) \simeq 0$ のとき精度が低下するので次の処理を行う.

$$K = \lfloor 1.27324x \rfloor \pmod 4$$

として

K が 1 または 2 のとき

$$B = \frac{3B_1}{x} - B_2, \ j = j_0(x)$$

K が 0 または 3 のとき

$$B = B_1, \ j = j_1(x)$$

得られた T_n と B と j の値より

$$j_n(x) = \frac{T_n j}{B}$$

とする.

B. $x > n$ ならば

$j_1(x), j_0(x)$ を初期値とし, 漸化式

$$j_{k+1}(x) = \frac{2k+1}{x} j_k(x) - j_{k-1}(x) \ (k = 1, 2, \cdots, n-1)$$

を用いて $j_n(x)$ を求める.

(2) 第 2 種整数次球ベッセル関数 $y_n(x)$ $(x > 0.0)$

① $n < 0$ のとき

$y_n(x) = (-1)^{n+1} j_{-n-1}(x)$ より求める.

② $n \geq 0$ のとき

(a) $x \leq 0.41$ ならば

$$y_n(x) = -\frac{(2n-1)!!}{x^{n+1}} \left\{ 1 + \sum_{k=1}^{9} \frac{(-\frac{x^2}{2})^k}{k!(1-2n)(3-2n)\cdots(2k-1-2n)} \right\}$$

のべき級数展開で求める.

(b) $x > 0.41$ ならば

i. $n = 0, 1$ の場合

$$y_0(x) = -\frac{\cos(x)}{x}, \ y_1(x) = -\frac{\cos(x) + x\sin(x)}{x^2}$$

により計算する.

ii. $n \geq 2$ の場合

$y_1(x), y_0(x)$ を初期値とし, 漸化式

$$y_{k+1}(x) = \frac{2k+1}{x} y_k(x) - y_{k-1}(x) \ (k = 1, 2, \cdots n-1)$$

を用いて $y_n(x)$ を求める.

(3) 第 1 種整数次変形球ベッセル関数 $i_n(x)$ $(n \geq 0, x \geq 0.0)$

① $x^2 < 0.1n + 0.47$ で $n \leq 30$ のとき

$$i_n(x) = x^n \sum_{k=0}^{9} \frac{(\frac{x^2}{2})^k}{k!(2n+2k+1)!!}$$

のべき級数展開で求める.

② $x^2 \geq 0.1n + 0.47$ または $n > 30$ のとき

(a) $n = 0, 1$ の場合

$$i_0(x) = \frac{\sinh(x)}{x}$$

$$i_1(x) = \frac{x\cosh(x) - \sinh(x)}{x^2} = \frac{e^x(x-1) + e^{-x}(x+1)}{2x^2}$$

により計算する.

(b) $n \geq 2$ の場合

i. $x \leq n$ ならば

$T_{n+k+1} = 0.0$ とし, 次の漸化式を用いて連分数近似におけるパラメータ T_i $(i = n, n+1, \cdots, n+k-1, n+k)$ を計算する.

$$T_i = \frac{x^2}{2i + T_{i+1} + 1} \ (i = n+1, \cdots, n+k-1, n+k)$$

$$T_n = \frac{x}{2n + T_{n+1} + 1}$$

(T_n は $\dfrac{i_n(x)}{i_{n-1}(x)}$ の値)

k の値は表 2−6 に示すようにとる.

表 2−6　k の値

単精度演算	倍精度演算
$k = \lfloor \frac{(4.6+0.5\sqrt{n})x}{n} + 2.0 \rfloor$	$k = \lfloor \frac{(6.0+1.2\sqrt{n})x}{n} + 6.0 \rfloor$

$B_n = T_n, B_{n-1} = 1.0$ とし, 次の漸化式を用いて B_k $(k = 1, 2, \cdots, n-2)$ を計算する.

$$B_{k-1} = \frac{2k+1}{x} B_k + B_{k+1} \ (k = 2, \cdots, n-2, n-1)$$

(B_1 は $\dfrac{i_1(x)}{i_{n-1}(x)}$ の値)

得られた T_n と B_1 と $i_1(x)$ の値より

$$i_n(x) = \frac{T_n i_1(x)}{B_1}$$

とする.

ii. $x > n$ ならば

a を正の最小値 (浮動小数点データの内部表現であらわすことのできる最も 0.0 に近い正の値) , M を n より十分大きな値とする.

$G_{M+1}(x) = 0.0, G_M(x) = a$ を初期値とし,

$$G_{k-1}(x) = \frac{2k+1}{x} G_k(x) + G_{k+1}(x) \ (k = M, M-1, \cdots, 1)$$

として

$$i_n(x) = \frac{G_n(x) e^x}{2\sum_{m=0}^{M} (m + 0.5) G_m(x)}$$

とする.

(4) 第 2 種整数次変形球ベッセル関数 $k_n(x)$ $(x > 0.0)$

① $n < 0$ のとき

$k_n(x) = k_{-n}(x)$ として $k_{-n}(x)$ について以下に述べる方法を適用する.

② $n \geq 0$ のとき

(a) $n = 0, 1$ の場合

$$k_0(x) = \frac{\pi}{2}\frac{e^{-x}}{x}, \ k_1(x) = \frac{\pi}{2}\frac{e^{-x}}{x}\left(1 + \frac{1}{x}\right)$$

により計算する.

(b) $n \geq 2$ の場合

$k_1(x), k_0(x)$ を初期値とし, 漸化式

$$k_{i+1}(x) = \frac{2i+1}{x} k_i(x) + k_{i-1}(x) \ (i = 1, 2, \cdots, n-1)$$

を用いて $k_n(x)$ を求める.

2.1.2.4 ベッセル関数に関連した関数

(1) 第 1 種, 第 2 種整数次ハンケル関数 $H_n^{(1)}(z), H_n^{(2)}(z)$

$$\begin{aligned} H_n^{(1)}(z) &= J_n(z) + iY_n(z) \\ H_n^{(2)}(z) &= J_n(z) - iY_n(z) \end{aligned}$$

より求める.

(2) ケルビン関数 $\mathrm{ber}_n(x), \mathrm{bei}_n(x)$

① $n < 0$ または, $x < 0.0$ のとき

まず, $\mathrm{ber}_{|n|}(|x|), \mathrm{bei}_{|n|}(|x|)$ を②$n \geq 0$ かつ $x \geq 0.0$ のときで述べる方法によって計算し, 次に以下の関係を用いて $\mathrm{ber}_n(x), \mathrm{bei}_n(x)$ を求める.

(a) n か x のどちらかが負のとき

$$\mathrm{ber}_n(x) = (-1)^n \, \mathrm{ber}_{|n|}(|x|), \ \ \mathrm{bei}_n(x) = (-1)^n \, \mathrm{bei}_{|n|}(|x|)$$

(b) n も x も負のとき

$$\mathrm{ber}_n(x) = \mathrm{ber}_{|n|}(|x|), \ \ \mathrm{bei}_n(x) = \mathrm{bei}_{|n|}(|x|)$$

② $n \geq 0$ かつ $x \geq 0.0$ のとき

(a) $x \leq \left\{ \begin{matrix} \mathrm{ber_n(x)} : 5.65 + 0.25\mathrm{n} \\ \mathrm{bei_n(x)} : 6.92 + 0.25\mathrm{n} \end{matrix} \right\}$ とき, 次の近似式で求める.

$$\begin{aligned} \mathrm{ber}_n(x) &\simeq \sum_{k=0}^{m} \frac{\cos\{\frac{1}{4}(3n+2k)\pi\}}{k!(n+k)!} \left(\frac{x}{2}\right)^{n+2k} \\ \mathrm{bei}_n(x) &\simeq \sum_{k=0}^{m} \frac{\sin\{\frac{1}{4}(3n+2k)\pi\}}{k!(n+k)!} \left(\frac{x}{2}\right)^{n+2k} \end{aligned}$$

ただし, $n = 0$ のときは $\mathrm{ber}(x), \mathrm{bei}(x)$ ともに上の式を最良近似式化して求める. 最良近似式化については 2.1.2.22 に記述されている.

(b) $x \geq \left\{ \begin{matrix} \text{倍精度} : 20.0 \\ \text{単精度} : 10.0 \end{matrix} \right\}$ かつ $x \geq 0.5n^2$ のとき

$$\begin{aligned} \mathrm{ber}_n(x) &\simeq \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}} \{P\cos(\Psi) + Q\sin(\Psi)\} \\ \mathrm{bei}_n(x) &\simeq \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}} \{P\sin(\Psi) - Q\cos(\Psi)\} \end{aligned}$$

の漸近展開式で求める.

ここで

$$\begin{aligned} P &= 1 + \sum_{k=1}^{m} (-1)^k \cos\left(\frac{k\pi}{4}\right) \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k} \\ Q &= \sum_{k=1}^{m} (-1)^k \sin\left(\frac{k\pi}{4}\right) \frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k} \\ \Psi &= \frac{x}{\sqrt{2}} + (\frac{n}{2} - \frac{1}{8})\pi \end{aligned}$$

ただし, $n=0$ のときは, P,Q ともに最良近似式化して求める. 最良近似式化については 2.1.2.22 に記述されている. (m は末項がそれ以前の計算値に影響しなくなるところの値である.)

(c) (a), (b) 以外は, 複素変数第 1 種整数次ベッセル関数 $J_n(z)$ より

$$\begin{aligned}\mathrm{ber}_n(x) &= \Re\left(J_n(-\frac{x}{\sqrt{2}},\frac{x}{\sqrt{2}})\right)\\ \mathrm{bei}_n(x) &= \Im\left(J_n(-\frac{x}{\sqrt{2}},\frac{x}{\sqrt{2}})\right)\end{aligned}$$

とする.

(3) ケルビン関数 $\mathrm{ker}_n(x),\mathrm{kei}_n(x)$ $(x>0.0)$

① $n<0$ のとき

まず, $\mathrm{ker}_{|n|}(x),\mathrm{kei}_{|n|}(x)$ を③ $n>0$ のときで述べる方法によって計算し, 次に以下の関係を用いて $\mathrm{ker}_n(x),\mathrm{kei}_n(x)$ を求める.

$$\mathrm{ker}_n(x)=(-1)^n\,\mathrm{ker}_{|n|}(x),\ \ \mathrm{kei}_n(x)=(-1)^n\,\mathrm{kei}_{|n|}(x)$$

② $n=0$ で $x\le\left\{\begin{matrix}\mathrm{ker(x)}:5.77\\ \mathrm{kei(x)}:6.56\end{matrix}\right\}$のとき

$$\begin{aligned}\mathrm{ker}(x) &= -\log(\frac{x}{2})\,\mathrm{ber}(x)+\frac{\pi}{4}\,\mathrm{bei}(x)+\sum_{n=0}^{\infty}\left\{\frac{(-1)^n}{((2n)!)^2}\left(\sum_{s=1}^{2n}\frac{1}{s}-\gamma\right)(\frac{x}{2})^{4n}\right\}\\ \mathrm{kei}(x) &= -\log(\frac{x}{2})\,\mathrm{bei}(x)-\frac{\pi}{4}\,\mathrm{ber}(x)+\sum_{n=0}^{\infty}\left\{\frac{(-1)^n}{((2n+1)!)^2}\left(\sum_{s=1}^{2n}\frac{1}{s}-\gamma\right)(\frac{x}{2})^{4n+2}\right\}\end{aligned}$$

を最良近似式化して求める (γ : オイラーの定数 : $0.57721566\cdots$). 最良近似式化については 2.1.2.22 に記述されている.

③ $n>0$ のとき

(a) $x\ge\left\{\begin{matrix}\text{倍精度}:20.0\\ \text{単精度}:10.0\end{matrix}\right\}$ かつ $x\ge 0.5n^2$ のとき

$$\begin{aligned}\mathrm{ker}_n(x) &\simeq \sqrt{\frac{\pi}{2x}}e^{-x\sqrt{2}}\{P\cos(\Psi)-Q\sin(\Psi)\}\\ \mathrm{kei}_n(x) &\simeq \sqrt{\frac{\pi}{2x}}e^{-x\sqrt{2}}\{-P\sin(\Psi)-Q\cos(\Psi)\}\end{aligned}$$

の漸近展開式で求める.

ここで

$$\begin{aligned}P &= 1+\sum_{k=1}^{m}\cos\left(\frac{k\pi}{4}\right)\frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k}\\ Q &= \sum_{k=1}^{m}\sin\left(\frac{k\pi}{4}\right)\frac{(4n^2-1^2)(4n^2-3^2)\cdots(4n^2-(2k-1)^2)}{k!(8x)^k}\\ \Psi &= \frac{x}{\sqrt{2}}+\left(\frac{n}{2}-\frac{1}{8}\right)\pi\end{aligned}$$

ただし, $n=0$ のときは, P, Q ともに最良近似式化して求める. 最良近似式化については 2.1.2.22 に記述されている. (m は末項がそれ以前の計算値に影響しなくなるところの値である.)

(b) (a) 以外は, 複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$ より, $K_n(\frac{x}{\sqrt{2}},\frac{x}{\sqrt{2}})$ の実部を A, 虚部を B とし, $\frac{n}{4}$ の余りが

i. 0 ならば

$\mathrm{ker}_n(x)=A,\ \ \mathrm{kei}_n(x)=B$

ii. 1 ならば
$\ker_n(x) = B,\ \ \mathrm{kei}_n(x) = -A$

iii. 2 ならば
$\ker_n(x) = -A,\ \ \mathrm{kei}_n(x) = -B$

iv. 3 ならば
$\ker_n(x) = -B,\ \ \mathrm{kei}_n(x) = A$

とする.

(4) ストルーブ関数 $\mathbf{H}_0(x), \mathbf{H}_1(x), \mathbf{H}_0(x) - Y_0(x), \mathbf{H}_1(x) - Y_1(x)$

① $x < 0.0$ のとき
$\mathbf{H}_0(x) = -\mathbf{H}_0(-x), \mathbf{H}_1(x) = \mathbf{H}_1(-x)$ として $\mathbf{H}_0(-x), \mathbf{H}_1(-x)$ について以下に述べる方法を適用する.
ただし, ベッセル関数との差 $\mathbf{H}_0(x) - Y_0(x), \mathbf{H}_1(x) - Y_1(x)$ はエラーとする.

② $0.0 \leq x \leq 8.0$ のとき次の式を最良近似式化して求める.

$$\mathbf{H}_0(x) = \frac{2}{\pi}\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k+1}}{((2k+1)!!)^2}$$

$$\mathbf{H}_1(x) = \frac{2}{\pi}\sum_{k=1}^{\infty}\frac{(-1)^k (2k+1) x^{2k}}{((2k+1)!!)^2}$$

最良近似式化については 2.1.2.22 に記述されている.
$\mathbf{H}_0(x) - Y_0(x), \mathbf{H}_1(x) - Y_1(x)$ は, これにより得られた $\mathbf{H}_0(x), \mathbf{H}_1(x)$ より第 2 種ベッセル関数 $Y_0(x), Y_1(x)$ を減算して求める.

③ $x > 8.0$ のとき次の近似式で求める.

$$\mathbf{H}_0(x) - Y_0(x) = \frac{1}{x}\sum_{k=0}^{n} a_k^{(1)} \left(\frac{1}{x}\right)^{2n} \qquad n = \begin{cases} \text{倍精度} : 28 \\ \text{単精度} : 6 \end{cases}$$

$$\mathbf{H}_1(x) - Y_1(x) = \sum_{k=0}^{n} a_k^{(2)} \left(\frac{1}{x}\right)^{2n} \qquad n = \begin{cases} \text{倍精度} : 25 \\ \text{単精度} : 5 \end{cases}$$

係数 $a_n^{(1)}, a_n^{(2)}$ は文献 (21) を参考に近似式をテレスコーピング計算して求めた.
$\mathbf{H}_0(x), \mathbf{H}_1(x)$ は, これより得られたベッセル関数との差の値に $Y_0(x), Y_1(x)$ を加算して求める.

(5) エアリ関数とその導関数 $\mathrm{Ai}(x), \mathrm{Bi}(x), \mathrm{Ai}'(x), \mathrm{Bi}'(x)$
$\zeta = \frac{2}{3}|x|^{\frac{3}{2}}$ とする.

① $x < \left\{ \begin{array}{l} -3.8315472\ (\mathrm{Ai}(x), \mathrm{Bi}(x)\ \text{のとき}) \\ -5.2414828\ (\mathrm{Ai}'(x), \mathrm{Bi}'(x)\ \text{のとき}) \end{array} \right\}$:

$$\mathrm{Ai}(x) = \frac{\sqrt{-x}}{3}\{\frac{4}{3\zeta}J_{\frac{2}{3}}(\zeta) - J_{\frac{5}{3}}(\zeta) + J_{\frac{1}{3}}(\zeta)\}$$

$$\mathrm{Bi}(x) = \sqrt{\frac{-x}{3}}\{\frac{4}{3\zeta}J_{\frac{2}{3}}(\zeta) - J_{\frac{5}{3}}(\zeta) - J_{\frac{1}{3}}(\zeta)\}$$

$$\mathrm{Ai}'(x) = \frac{x}{3}\{\frac{2}{3\zeta}J_{\frac{1}{3}}(\zeta) - J_{\frac{4}{3}}(\zeta) - J_{\frac{2}{3}}(\zeta)\}$$

$$\mathrm{Bi}'(x) = \frac{-x}{\sqrt{3}}\{\frac{2}{3\zeta}J_{\frac{1}{3}}(\zeta) - J_{\frac{4}{3}}(\zeta) + J_{\frac{2}{3}}(\zeta)\}$$

② $\left\{ \begin{array}{l} -3.8315472\ (\mathrm{Ai}(x), \mathrm{Bi}(x)\ \text{のとき}) \\ -5.2414828\ (\mathrm{Ai}'(x), \mathrm{Bi}'(x)\ \text{のとき}) \end{array} \right\} \leq x < \left\{ \begin{array}{l} 3.8315472\ (\mathrm{Ai}(x), \mathrm{Bi}(x)\ \text{のとき}) \\ 5.2414828\ (\mathrm{Ai}'(x), \mathrm{Bi}'(x)\ \text{のとき}) \end{array} \right\}$

$$C_1 = \frac{3^{-\frac{2}{3}}}{\Gamma(\frac{2}{3})}$$

$$
\begin{aligned}
C_2 &= \frac{3^{-\frac{1}{3}}}{\Gamma(\frac{1}{3})} \\
f(x) &= 1+\frac{1}{3!}x^3+\frac{1\cdot 4}{6!}x^6+\frac{1\cdot 4\cdot 7}{9!}x^9+\cdots \\
g(x) &= x+\frac{2}{4!}x^4+\frac{2\cdot 5}{7!}x^7+\frac{2\cdot 5\cdot 8}{10!}x^{10}+\cdots
\end{aligned}
$$

$x<0$ と $x\geq 0$ の区間に分割して,

$$
\begin{aligned}
\mathrm{Ai}(x) &= C_1 f(x)-C_2 g(x) \\
\mathrm{Bi}(x) &= \sqrt{3}(C_1 f(x)+C_2 g(x))
\end{aligned}
$$

を最良近似式化して求める.

$\mathrm{Ai}'(x)$ は, $\mathrm{Ai}(x)$ の近似式を微分した式を作り, これを最良近似式化して求める. $\mathrm{Bi}'(x)$ は, $\mathrm{Bi}(x)$ の近似式を微分した式を作り, これを最良近似式化して求める.

最良近似式化については, 2.1.2.22 に記述されている.

③ $x\geq \left\{ \begin{array}{l} 3.8315472\ (\mathrm{Ai}(x),\mathrm{Bi}(x)\ \text{のとき}) \\ 5.2414828\ (\mathrm{Ai}'(x),\mathrm{Bi}'(x)\ \text{のとき}) \end{array} \right\}$

$$
\begin{aligned}
\mathrm{Ai}(x) &= e^{-\zeta}x^{-\frac{1}{4}}\sum_{k=0}^{m1}a_k^{(1)}\left(\frac{1}{\zeta}\right)^k \\
\mathrm{Bi}(x) &= e^{\zeta}x^{-\frac{1}{4}}\sum_{k=0}^{m2}a_k^{(2)}\left(\frac{1}{\zeta}\right)^k+\sqrt{\frac{x}{3}}I_{\frac{5}{3}}(\zeta) \\
\mathrm{Ai}'(x) &= e^{-\zeta}x^{\frac{1}{4}}\sum_{k=0}^{m3}a_k^{(3)}\left(\frac{1}{\zeta}\right)^k \\
\mathrm{Bi}'(x) &= e^{\zeta}x^{\frac{1}{4}}\sum_{k=0}^{m4}a_k^{(4)}\left(\frac{1}{\zeta}\right)^k+\frac{x}{\sqrt{3}}I_{\frac{4}{3}}(\zeta)
\end{aligned}
$$

係数 $a_k^{(1)}\sim a_k^{(4)}$ は文献 (21) を参考に, 新しく求めなおした.

2.1.2.5 ガンマ関数

(1) 実変数ガンマ関数 $\Gamma(x)$ および実変数対数ガンマ関数 $\log_e(\Gamma(x))$

① $x\geq 11.0$ のとき以下の対数ガンマ関数の漸近展開より求める.

$$
\log_e(\Gamma(x)) \sim (x-\frac{1}{2})\log_e x-x+\frac{1}{2}\log_e(2\pi)+\sum_{n=1}^{\infty}\frac{B_{2n}}{(2n-1)(2n)x^{2n-1}}
$$

$$
\Gamma(x)=\exp\left(\log_e(\Gamma(x))\right)
$$

② $0<x<11.0$ のとき, 最良有理式近似を用いる. z を x の小数部分として

$$
\Gamma(z)=(\sum_{k=0}^{6}a_k z^k)/(\sum_{k=0}^{7}b_k z^k)
$$

これを, 関数等式により, $\Gamma(x)$ に移す.

③ $x<0.0$ のとき

$$
\begin{aligned}
\Gamma(x) &= \frac{\pi}{-x\sin(\pi x)\Gamma(-x)} \\
\log_e(\Gamma(x)) &= \log_e\pi-\log_e((-x)\sin(\pi x))-\log_e(\Gamma(-x))
\end{aligned}
$$

を用いる.

(2) 複素変数ガンマ関数 $\Gamma(z)$ および複素変数対数ガンマ関数 $\log(\Gamma(z))$

① $\Im(z) = 0.0$ のときは実変数対数ガンマ関数を利用する.

② $\Re(z) = 0.0$ で $|\Im(z)| > 12.0$ のとき以下の漸近展開より求める.

$$\begin{aligned}\Re(\log_e(\Gamma(z))) &\sim \frac{1}{2}(\log_e(2\pi) - \pi\Im(z) - \log_e(\Im(z)))\\ \Im(\log_e(\Gamma(z))) &\sim \Im(z)\log_e(\Im(z)) - \Im(z) - \frac{1}{4}\pi - \sum_{n=1}^{\infty}\frac{(-1)^{n-1}B_{2n}}{(2n-1)(2n)(\Im(z))^{2n-1}}\end{aligned}$$

ここで, B_n はベルヌーイ数である.

③ $\Im(z) < 0.0$ のとき

$$\log_e(\Gamma(z)) = \log_e \pi - \log_e(-z)\sin(\pi z) - \log_e(\Gamma(-z))$$

であるので, 以下の方法を $\log_e(\Gamma(-z))$ に適用して計算する.

④ $\Im(z) > 0.0$ のとき

(a) $|\Re(z)| < 11.0$ のとき

$$\log_e(\Gamma(z)) = \log_e(\Gamma(n+z)) - \log_e((n-1+z)(n-2+z)\cdots(z))$$

であるので, 以下の方法を $\log_e(\Gamma(n+z))$ に適用して計算する. ただし, $n = \lfloor 12.0 - \Re(z) \rfloor$ である.

(b) $|\Re(z)| \geq 11.0$ のとき以下の漸近展開より求める.

$$\log_e(\Gamma(z)) \sim (z-\frac{1}{2})\log_e z - z + \frac{1}{2}\log_e(2\pi) + \sum_{n=1}^{\infty}\frac{B_{2n}}{(2n-1)(2n)z^{2n-1}}$$

⑤ 複素変数ガンマ関数 $\Gamma(z)$ は, $\exp(\log_e(\Gamma(z)))$ として計算する.

(3) 第 1 種不完全ガンマ関数 $\gamma(\nu, x)$, $(\nu \geq 0.0, x \geq 0.0)$

① $x = 0.0$ のとき $\gamma(\nu, x) = 0.0$ とする.

② $x \leq \nu - 0.5$ または $x \leq 3.5$ のとき

$\nu \leq \dfrac{e^{3.5}}{(\text{最大値})}$ のとき

$$\gamma(\nu, x) = \frac{1}{\nu}$$

その他のとき

$$\gamma(\nu, x) = \frac{e^{\nu\log(x)-x}P}{\nu}$$

ここで,

$$P = 1 + \sum_{k=1}^{m}\frac{x^k}{(\nu+1)(\nu+2)\cdots(\nu+k)}$$

(m は末項がそれ以前の計算値に影響しなくなるまでの項である)

③ ①, ②以外のとき,

$$\gamma(\nu, x) = \Gamma(\nu) - \Gamma(\nu, x)$$

とする.

ここで, $\Gamma(\nu, x)$ は, 第 2 種不完全ガンマ関数である.

(4) 第 2 種不完全ガンマ関数 $\Gamma(\nu, x)$ $(\nu \geq 0.0, x \geq 0.0)$

① $\nu \leq \dfrac{1.0}{(\text{最大値})}$ のとき, 指数積分の関数を利用して, $-\mathrm{Ei}(-x)$ を計算する.

② $x = 0.0$ のとき, $\Gamma(\nu, x) = \Gamma(\nu)$ とする.

③ ν が整数で $x > 0.015\nu^2$ のとき,

$$\Gamma(\nu, x) = e^{(\nu-1)\log(x)-x}\{1 + \sum_{k=1}^{m} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\}$$

(m は末項がそれ以前の計算値に影響しなくなるまでか $m \leq \nu + 2$ までの項である)

④ $x \leq \nu - 0.5$ のとき

$\Gamma(\nu, x) = \Gamma(\nu) - \gamma(\nu, x)$ とする.

⑤ $x \geq \begin{cases} \text{倍精度} : 49.0 \\ \text{単精度} : 23.0 \end{cases}$ のとき,

ν が整数で $x \geq 0.015\nu^2$ のときと同じ処理を行う.

⑥ $\nu < 1.0$ のとき

(a) $x < 0.36\nu + 0.85$ のとき,

$$\Gamma(\nu, x) = (\nu-1)\frac{\sum_{k=0}^{M} p_k^{(2)}\nu^k}{\sum_{k=0}^{N} q_k^{(2)}\nu^k} - \frac{e^{\nu\log(x)}-1}{\nu} - x^\nu \sum_{k=1}^{M'} \frac{(-1)^k x^k}{k!(k+\nu)} \qquad \left(M' : \begin{cases} \text{倍精度} : 21 \\ \text{単精度} : 12 \end{cases}\right)$$

(b) $x \leq 1.5\nu + 2.1$ のとき,

$$\Gamma(\nu, x) = \Gamma(1+\nu)e^{-x}\sum_{k=0}^{N'}(x^k\tilde{A}_k(\nu) + x^k\frac{\phi(\nu, x)}{\Gamma(k+1+\nu)} \qquad \left(N' : \begin{cases} \text{倍精度} : 23 \\ \text{単精度} : 14 \end{cases}\right)$$

ここで

$$\begin{aligned} \tilde{A}_0(\nu) &= (1+\nu)(\tilde{A}_1(\nu) - 1) \\ \tilde{A}_1(\nu) &\simeq \sum_{k=0}^{M} p_k^{(1)}\nu^k \end{aligned}$$

として

$$\begin{aligned} \tilde{A}_k(\nu) &= \frac{1}{k+\nu}\{\tilde{A}_{k-1}(\nu) + \frac{1}{k!}\} \\ \phi(\nu, x) &= -\frac{e^{\nu\log(x)}-1}{\nu} \end{aligned}$$

(c) 上記以外のとき,

$$\Gamma(\nu, x) = e^{\nu\log(x)-x} \cdot \cfrac{1}{\left|x\right.} + \mathop{\Phi}_{n=1}^{\infty} \cfrac{n-\nu}{\left|1\right.} + \cfrac{n}{\left|x\right.}$$

ここで, 連分数の項数は, $\left\{\begin{array}{l} \text{倍精度} : \lfloor\frac{120}{x} + 5\rfloor \\ \text{単精度} : \lfloor\frac{25}{x-0.25} + 2\rfloor \end{array}\right\}$とする.

⑦ ① から ⑥以外のとき,

(a) $x \leq 5.6$ のとき, $\alpha = \nu - \lfloor\nu\rfloor$, $\mu = \alpha + 1.0$ として,

$$\Gamma(\mu, x) = \Gamma(\mu)e^{-x}[1 + \alpha\sum_{k=0}^{M''}(x^{k+1}\tilde{C}_k(\alpha) + x^{k+1}\frac{\phi(\alpha, x)}{\Gamma(k+1+\mu)})] \qquad \left(M'' : \begin{cases} \text{倍精度} : 20 \\ \text{単精度} : 12 \end{cases}\right)$$

$$\tilde{C}_0(\alpha) = \tilde{A}_1(\alpha)$$

として,

$$\begin{aligned} \tilde{C}_k(\alpha) &= \frac{1}{k+\nu}\{\tilde{C}_{k-1}(\alpha) + \frac{1}{(k+1)!}\} \\ \phi(\alpha, x) &= -\frac{e^{\alpha\log(x)}-1}{\alpha} \end{aligned}$$

- $\nu > 2.0$ のとき,
  $m = \nu - \mu$ として,

$$\begin{aligned}\Gamma(\nu, x) &= e^{(\nu-1)\log(x)-x}\left\{1 + \sum_{k=1}^{m-1} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\right\} \\ &\quad +(\nu-1)(\nu-2)\cdots(\nu-k)\Gamma(\mu, x)\end{aligned}$$

- $\nu \leq 2.0$ のとき,

$$\Gamma(\nu, x) = \Gamma(\mu, x)$$

(b) (a) 以外のとき,

$$\begin{aligned}\Gamma(\nu, x) &= e^{(\nu-1)\log(x)-x}\{1 + \sum_{k=1}^{m-1} \frac{(\nu-1)(\nu-2)\cdots(\nu-k)}{x^k}\} \\ &\quad +(\nu-1)(\nu-2)\cdots(\nu-m)e^{(\nu-m)\log(x)-x} \\ &\quad \cdot \left[\frac{1}{\left\lceil x \right\rceil} + \underset{n-1}{\overset{\infty}{\Phi}}\left(\frac{n-(\nu-m)}{\left\lceil 1 \right\rceil} + \frac{n}{\left\lceil x \right\rceil}\right)\right]\end{aligned}$$

(m は ν の整数部)

係数 $p_k^{(1)}, p_k^{(2)}, q_k^{(2)}$ は文献 (22) に記述されている.

2.1.2.6 ガンマ関数に関連した関数

(1) ディガンマ関数 $\Psi(x)$ ($x < 0.0$ のとき $x \neq$ 整数)

① $x < -2.0$ のとき

$$\Psi(x) = \log(1-x) - \frac{1}{2(1-x)} + \frac{\sum_{k=0}^{m} a_k^{(2)}(1-x)^{-2k}}{\sum_{k=0}^{m} b_k^{(2)}(1-x)^{-2k}} - \pi\cot(\pi x)$$

を計算する.

② $-2.0 < x < 0.5$ のとき

$$\Psi(x) = (1-c-x)\frac{\sum_{k=0}^{m} a_k^{(1)}(1-x)^k}{\sum_{k=0}^{m} b_k^{(1)}(1-x)^k} - \pi\cot(\pi x)$$

を計算する.

③ $0.5 \leq x \leq 3.0$ のとき

$$\Psi(x) = (x-c)\frac{\sum_{k=0}^{m} a_k^{(1)}x^k}{\sum_{k=0}^{m} b_k^{(1)}x^k}$$

を計算する.

④ $x > 3.0$ のとき

$$\Psi(x) = \log(x) - \frac{1}{2x} + \frac{\sum_{k=0}^{m} a_k^{(2)}x^{-2k}}{\sum_{k=0}^{m} b_k^{(2)}x^{-2k}}$$

を計算する.
なお, $c = 1.46163214496836234126$ とし, 係数 $a_k^{(1)}, b_k^{(1)}, a_k^{(2)}, b_k^{(2)}$ は文献 (13) に記述されている.

(2) ベータ関数 $B(p,q)$ $(p > 0.0, q > 0.0)$
ベータ関数は, つぎの式により定義される.

$$B(p,q) = \int_0^1 x^{p-1}(1-x)^{q-1}dx = B(q,p) \ (p,q > 0)$$

p と q の値により, 以下のように計算する.

① $p < 12.0$ かつ $q < 12.0$ のときガンマ関数を使い, つぎの式で計算する.

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$$

② $p < 12.0$ かつ $q \geq 12.0$ のとき, ガンマ関数とスターリングの公式から計算する.

$$B(p,q) = \Gamma(p)e^{(q-\frac{1}{2})\log(q)+\Phi(q)-(p+q-\frac{1}{2})\log(p+q)+p-\Phi(p+q)}$$

ここで $\Phi(x)$ は,

$$\Phi(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}B_n}{(2n)(2n-1)x^{2n-1}}$$

を最良近似式化して求める (B_n はベルヌーイ数).
最良近似式化については 2.1.2.22 に記述されている.

③ $p \geq 12.0$ かつ $q < 12.0$ のときは $B(p,q) = B(q,p)$ より $p < 12.0$ かつ $q \geq 12.0$ のときと同じ.

④ $p \geq 12.0$ かつ $q \geq 12.0$ のときはスターリングの公式から計算する.

$$B(p,q) = e^{(p-\frac{1}{2})\log(p)+(q-\frac{1}{2})\log(q)-(p+q-\frac{1}{2})\log(p+q)+\log(\sqrt{2\pi})+\Phi(p)+\Phi(q)-\Phi(p+q)}$$

2.1.2.7 楕円関数と楕円積分

(1) 第 1 種, 第 2 種完全楕円積分 $K(m), E(m)$ $(0.0 \leq m \leq 1.0)$

① $0.0 \leq m \leq 0.25$ のとき

$$\begin{aligned} K(m) &= \frac{\pi}{2}\sum_{k=0}^{\infty}\left(\frac{(2k-1)!!}{(2k)!!}\right)^2 m^k \\ E(m) &= \frac{\pi}{2}\left\{1 - \sum_{k=1}^{\infty}\left(\frac{(2k-1)!!}{(2k)!!}\right)^2 \frac{m^k}{2k-1}\right\} \end{aligned}$$

を最良近似式化して求める.
最良近似式化については 2.1.2.22 に記述されている.

② $0.25 < m < 1.0$ のとき

$$\begin{aligned} K(m) &= \sum_{k=0}^{l}\left\{a_k^{(1)}(1-m)^k\right\} - \log(1-m)\sum_{k=0}^{l}\left\{b_k^{(1)}(1-m)^k\right\} \\ E(m) &= \sum_{k=0}^{l}\left\{a_k^{(2)}(1-m)^k\right\} - \log(1-m)\sum_{k=0}^{l}\left\{b_k^{(2)}(1-m)^k\right\} \end{aligned}$$

を計算する.

③ $m = 1.0$ のとき, $K(m) = +\infty, E(m) = 1.0$ とする.
係数 $a_k^{(1)}, b_k^{(1)}, a_k(2), b_k^{(2)}$ は文献 (2) に記述されている.

(2) 第 1 種, 第 2 種不完全楕円積分 $F(x,m), E(x,m)$ $(0.0 \le x \le 1.0, 0.0 \le m \le 1.0)$

① $x = 0.0$ のとき, $F(0.0,m) = E(0.0,m) = 0.0$

② $x = 1.0$ のとき, 完全楕円積分より

$$\begin{aligned} F(1.0,m) &= K(m) \\ E(1.0,m) &= E(m) \end{aligned}$$

とする.

③ ①, ②以外のとき

(a) $m = 0.0$ のとき

$$\begin{aligned} F(x,0.0) &= \sin^{-1}(x) \\ E(x,0.0) &= \sin^{-1}(x) \end{aligned}$$

とする.

(b) $0.0 < m < 1.0$ のとき

$a_0 = 1.0, b_0 = \sqrt{1-m}, c_0 = \sqrt{m}$ を初期値として

$$\begin{aligned} a_{k+1} &= \frac{a_k + b_k}{2} \\ b_{k+1} &= \sqrt{a_k b_k} \\ c_{k+1} &= \frac{a_k - b_k}{2} \ (k = 0, 1, \cdots) \end{aligned}$$

を求める. 収束判定は,

$c_N < a_k \cdot$ (誤差判定のための単位).

とする. $\phi_0 = \sin^{-1}(x)$ とし,

$$\tan(\phi_{k+1} - \phi_k) - \frac{b_k}{a_k}\tan(\phi_k) = 0 \ (k = 0, \cdots, N-1)$$

を $\phi_{k+1}(> \phi_k)$ についてニュートン法で解きながら, $\phi_1, \cdots, \phi_N$ を求める.

$$\begin{aligned} F(x,m) &= \frac{\phi_N}{2^N a_N} \\ E(x,m) &= \left(1 - \frac{1}{2}\sum_{k=0}^{N} 2^k c_k^2\right) F(x,m) + \sum_{k=1}^{N} c_k \sin(\phi_k) \end{aligned}$$

を計算する. 他の方法を用いて, 第 1 種不完全楕円積分 $F(x,m)$ を求めることもできる. ランデン変換 (母数 $m = k^2$ を, 以下の k_1^2 に換える変換) により,

$$k_1 = \frac{1 - \sqrt{1-m}}{1 + \sqrt{1-m}}, \ m_1 = k_1^2, \ m = k^2, \ x_1 = \frac{2}{1 + \sqrt{1 - mx^2}} \frac{x}{1 + k_1}$$

とするとき, $F(x,m) = (1 + k_1)F(x_1, m_1)$ であるから, $X_0 = x, \ K_0 = k, \ (m = k^2, \ k > 0)$ として, $n = 0, \ 1, \cdots$ について逐次,

$$K_{n+1} = \frac{K_n^2}{\left(1 + \sqrt{1 - K_n^2}\right)^2}, \ X_{n+1} = \frac{2}{1 + \sqrt{1 - K_n^2 X_n^2}} \frac{X_n}{1 + K_{n+1}}$$

とすると,

$$F(x,m) = (1 + K_1)(1 + K_2) \cdots (1 + K_{n+1}) F(X_{n+1}, \ K_{n+1}^2)$$

であるが, $K_{n+1} \le K_n^2$ などにより, K_{n+1} は非常に速く 0 に収束し, $F(X_{n+1}, \ K_{n+1}^2)$ は $\sin^{-1} X_{n+1}$ に等しいとみなすことができる.

ガウスの算術幾何平均の方法は, ランデン変換の方法と本質的には同じである.

(c) $m = 1.0$ のとき

$$\begin{aligned} F(x,m) &= \text{artanh(x)} \\ E(x,m) &= x \end{aligned}$$

とする.

(3) 不完全変形楕円積分, ワイエルシュトラス型の不完全楕円積分

母数 $m(0 \leq m < 1)$ の第 1 種不完全楕円積分 $F(r,m)$ および第 2 種不完全楕円積分 $E(r,m)$ を, 各々

$$F(r,m) = \int_0^r \frac{dx}{\sqrt{(1-x^2)(1-mx^2)}},$$

$$E(r,m) = \int_0^r \sqrt{\frac{1-mx^2}{1-x^2}}dx$$

で定義する. ただし, $0 \leq r < 1$.

① 不完全変形楕円積分

$m \geq 0, a, b$ と $p \geq 0$ に対して, 不完全変形楕円積分

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}}$$

は以下の手順で求める.

ただし, $m = 0,\ 1$ である場合, すなわち不完全変形楕円積分が退化し初等関数で表現できる場合は対数関数または逆正接関数によって求める.

(a) m が 1 に近くない場合で, $m > 1$ ならば, 置換積分によって $m < 1$ の場合 (b) に帰着する.

(b) m が 1 に近くない場合で, $m < 1$ の場合, $p = r/\sqrt{1-r^2}$ とおいて

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}} = \frac{b-ma}{1-m}F(r,1-m) + \frac{a-b}{1-m}E(r,1-m)$$

(c) m が 1 に近いときは, $p = \tan\alpha, 0 \leq \alpha < \pi/2$ と置いて,

$$\int_0^p \frac{a+bt^2}{1+t^2}\frac{dt}{\sqrt{(1+t^2)(1+mt^2)}} = \int_0^\alpha \frac{a+(b-a)\sin^2 u}{\sqrt{1-(1-m)\sin^2 u}}du$$

を用いる. $I_n(\alpha) = \int_0^\alpha \sin^{2n} u du$ は漸化式

$$(2n+2)I_{n+1}(\alpha) = (2n+1)I_n(\alpha) - \sin^{2n+1}\alpha\cos\alpha,\ n \geq 0$$

によって求め, 被積分関数の展開にべき級数展開

$$\frac{1}{\sqrt{1-v}} = 1 + \sum_{n=1}^{\infty}\frac{I_n(\pi/2)}{\pi/2}v^n,\ |v| < 1$$

を利用する.

② ワイエルシュトラス型の不完全楕円積分

不完全楕円積分が退化し初等関数で表現できる場合は対数関数または逆正接関数によって表示する. それ以外の場合は, $z > y > x > 0$ の場合を取り扱えばよく

$$v_1 = \sqrt{\frac{(z-x)p}{1+zp}}, v_2 = \sqrt{\frac{z-x}{z}}, m = \frac{z-y}{z-x}$$

とおいて

$$\frac{1}{2}\int_0^{1/p} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}} = \frac{1}{\sqrt{z-x}}(F(v_2,m) - F(v_1,m))$$

である. $x = 0$ のときは不完全変形楕円積分を用いれば良い.

(4) ヤコビの楕円関数 $\mathrm{sn}(u,m), \mathrm{cn}(u,m), \mathrm{dn}(u,m)$ $(0.0 \leq m \leq 1.0)$

① $u = 0.0$ のとき

$$\mathrm{sn}(0.0,m) = 0.0, \mathrm{cn}(0.0,m) = \mathrm{dn}(0.0,m) = 1.0$$

② $m = 1.0$ のとき

$$\begin{aligned} \mathrm{sn}(u, 1.0) &= \tanh(u) \\ \mathrm{cn}(u, 1.0) &= \mathrm{dn}(u, 1.0) = \mathrm{sech}(u) \end{aligned}$$

③ $m = 0.0$ のとき

$$\begin{aligned} \mathrm{sn}(u, 0.0) &= \sin(u) \\ \mathrm{cn}(u, 0.0) &= \cos(u) \\ \mathrm{dn}(u, 0.0) &= 1.0 \end{aligned}$$

④ ①, ②, ③以外のとき

$a_0 = 1.0, b_0 = \sqrt{1-m}$ を初期値として

$$\begin{aligned} a_{k+1} &= \frac{a_k + b_k}{2} \\ b_{k+1} &= \sqrt{a_k b_k} \\ c_{k+1} &= \frac{a_k - b_k}{2} \ (k = 0, 1, \cdots) \end{aligned}$$

を求める. 収束判定は,

$c_N < a_{N-1} \cdot$ (誤差判定のための単位).

とする.

$$\begin{aligned} K(m) &= \frac{\pi}{2a_N} \\ y_N &= \frac{a_N}{\sin(a_N u)} \\ y_{N-1} &= y_N + \frac{a_N c_N}{y_N} \\ &\vdots \\ y_0 &= y_1 + \frac{a_1 c_1}{y_1} \end{aligned}$$

を求め

$$\begin{aligned} \mathrm{sn}(u, m) &= \frac{1}{y_0} \\ \mathrm{cn}(u, m) &= \begin{cases} \sqrt{1 - (\frac{1}{y_0})^2} & (\mathrm{mod}(\lfloor \frac{|u|}{K(m)} \rfloor, 4) = 0 \text{ or } 3 \text{ のとき}) \\ -\sqrt{1 - (\frac{1}{y_0})^2} & (\mathrm{mod}(\lfloor \frac{|u|}{K(m)} \rfloor, 4) = 1 \text{ or } 2 \text{ のとき}) \end{cases} \\ \mathrm{dn}(u, m) &= \sqrt{1 - m(\frac{1}{y_0})^2} \end{aligned}$$

なお, $\lfloor \frac{u}{K(m)} \rfloor$ と $\frac{u}{K(m)}$ の差が 0.03125 以下のときは, $\mathrm{sn}(u, m) \simeq 1.0$ で $\sqrt{1 - \mathrm{sn}^2(u, m)}$ の計算で大きなケタ落ちが生じるため, 次式により求める.

$$\begin{aligned} v &= \frac{u}{2K(m)} \\ \mathrm{sn}(u, m) &= \frac{\vartheta_1(v, q)}{\sqrt[4]{m}\vartheta_4(v, q)} \\ \mathrm{cn}(u, m) &= \frac{\vartheta_2(v, q)\sqrt[4]{\frac{1-m}{m}}}{\vartheta_4(v, q)} \\ \mathrm{dn}(u, m) &= \frac{\vartheta_3(v, q)\sqrt[4]{(1-m)}}{\vartheta_4(v, q)} \end{aligned}$$

(q は, 次項ノーム q 参照)

(5) ノーム q $(0.0 \leq m \leq 1.0)$

$m > 0.5$ ならば,

$m = 1 - m$ とし, 最終的に q と $q', K(m)$ と $K(m'), E(m)$ と $E(m')$ の値を, それぞれ入れ替える.

① $m = 0.0$ のとき,

$$\begin{aligned} q &= 0.0 \\ q' &= 1.0 \\ K(m) &= \frac{\pi}{2} \\ K(m') &= \text{(最大値)} \\ E(m) &= \frac{\pi}{2} \\ E(m') &= 1.0 \end{aligned}$$

② ① 以外のとき,

$$\varepsilon = \frac{0.5m}{(2(1 + \sqrt[4]{1-m}(1 + \sqrt{1-m}) + \sqrt{1-m}) - m)}$$

とおくと,

$$q = c_4\varepsilon^{13} + c_3\varepsilon^9 + c_2\varepsilon^5 + c_1\varepsilon$$

(ただし, 倍精度では $c_1 = 1, c_2 = 2, c_3 = 15, c_4 = 150$, 単精度では $c_1 = 1, c_2 = 2, c_3 = 0, c_4 = 0$)

$$q' = \exp(\frac{\pi^2}{\log(q)})$$

δ を

$$\delta = (d_4q^9 + d_3q^4 + d_2q + d_1)^2$$

(ただし, 倍精度では $d_1 = 1, d_2 = 2, d_3 = 1, d_4 = 1$ 単精度では $d_1 = 1, d_2 = 2, d_3 = 1, d_4 = 0$) とおくと,

$$\begin{aligned} K(m) &= \frac{\pi}{2}\delta \\ K(m') &= (-0.5)\log(q)\delta \\ E(m) &= (\frac{\pi}{6\delta})(1 + (2-m)\delta^2 - (e_7q^{14} + e_6q^{12} + e_5q^{10} + e_4q^8 + e_3q^6 + e_2q^4 + e_1q^2) \\ E(m') &= K(m')(1 - \frac{E(m)}{K(m)}) + \frac{1}{\delta} \end{aligned}$$

(ただし, 倍精度では $e_1 = 24, e_2 = 72, e_3 = 96, e_4 = 168, e_5 = 144, e_6 = 288, e_7 = 192$ 単精度では $e_1 = 24, e_2 = 72, e_3 = 96, e_4 = 0, e_5 = 0, e_6 = 0, e_7 = 0$)

(6) 楕円テータ関数 $\vartheta_i(v, q)$ $(0 \leq i \leq 4, 0.0 \leq q \leq 1.0)$

$i = 0$ のときは $i = 4$ とし, $i = 2, 3$ のときはそれぞれ $v = v + 0.5, v = v - 0.5$ としておく.

① $i = 1, 2$ のとき

$s = SIGN(v), v = \mid v \mid$ とおき,

$v = v - INT(v), INT(v)$ が奇数のとき $s = -s$ とする.

i) $q = 0.0$ または $v = 0.0$ ならば,

$\vartheta(v, q) = 0.0$

とする.

ii) $q \leq 0.125$ ならば,

$sw = \sin(\pi \cdot v)$

$s3w = (-4 \cdot sw^2 + 3) \cdot sw$

$s5w = ((16 \cdot sw^2 - 20) \cdot sw^2 + 5) \cdot sw$

とおくと,

・単精度

$\vartheta(v, q) = 2 \cdot s \cdot \sqrt[4]{q} \cdot ((s5w \cdot q^4 - s3w) \cdot q^2 + sw)$

・倍精度

$s7w = (((-64 \cdot sw^2 + 112) \cdot sw^2 - 56) \cdot sw^2 + 7) \cdot sw$

$s9w = ((((256 \cdot (1 - sw^2) - 448) \cdot (1 - sw^2) + 240) \cdot (1 - sw^2) - 40) \cdot (1 - sw^2) + 1) \cdot sw$

とおくと,

$\vartheta_i(v, q) = 2 \cdot s \cdot \sqrt[4]{q} \cdot ((((s9w \cdot q^8 - s7w) \cdot q^6 + s5w) \cdot q^4 - s3w) \cdot q^2 + sw)$

iii) $q > 0.125$ のとき

$v > 0.5$ ならば $v = 1.0 - v$ とする.

$PLQ = \pi^2 \cdot (1/\log(q))$

$wQ = \exp(PLQ)$

$w = -\exp(2 \cdot v \cdot PLQ)$

とおくと,

・単精度

$$\vartheta_i(v, q) = s \cdot \sqrt{\pi \cdot (-1/\log(q))} \cdot \exp((v^2 - v + 0.25) \cdot PLQ) \cdot (((wQ^6 \cdot w + wQ^2) \cdot w + 1) \cdot w + 1 + (wQ^4/w + 1) \cdot (wQ^2/w))$$

・倍精度

$$\vartheta_i(v, q) = s \cdot \sqrt{\pi \cdot (-1/\log(q))} \cdot \exp((v^2 - v + 0.25) \cdot PLQ) \cdot ((((wQ^{12} \cdot w + wQ^6) \cdot w + wQ^2) \cdot w + 1) \cdot w + 1 + ((wQ^8/w + wQ^2) \cdot (wQ^2/w) + 1) \cdot (wQ^2/w))$$

② $i = 3, 4$ のとき,

$v = |\ v\ | - INT(|\ v\ |)$ とする.

i) $q = 0.0$ ならば,

$\vartheta_i(v, q) = 1.0$

とする.

ii) $q \leq 0.125$ ならば,

$cw = \cos(2 \cdot \pi \cdot v)$

$c2w = 2 \cdot cw^2 - 1$

$c3w = (4 \cdot cw^2 - 3) \cdot cw$

とおくと,

・単精度

$\vartheta_i(v, q) = 2 \cdot ((-c3w \cdot q^5 + c2w) \cdot q^3 - cw) \cdot q + 1$

・倍精度

$c4w = (8 \cdot cw^2 - 8) \cdot cw^2 + 1$

とおくと,

$\vartheta_i(v, q) = 2 \cdot (((c4w \cdot q^7 - c3w) \cdot q^5 + c2w) \cdot q^3 - cw) \cdot q + 1$

iii) $q > 0.125$ のとき, $v > 0.5$ ならば $v = 1.0 - v$ とする.

$PLQ = \pi^2 \cdot (1/\log(q))$

$wQ = \exp(PLQ)$

$w = \exp(2 \cdot v \cdot PLQ)$

とおくと,

・単精度

$$\vartheta_i(v,q) = \sqrt{\pi \cdot (-1/\log(q))} \cdot \exp((v^2 - v + 0.25) \cdot PLQ) \cdot (((wQ^6 \cdot w + wQ^2) \cdot w + 1) \cdot w + 1 + (wQ^4/w + 1) \cdot (wQ^2/w))$$

・倍精度

$$\vartheta_i(v,q) = \sqrt{\pi \cdot (-1/\log(q))} \cdot \exp((v^2 - v + 0.25) \cdot PLQ) \cdot ((((wQ^{12} \cdot w + wQ^6) \cdot w + wQ^2) \cdot w + 1) \cdot w + 1 + ((wQ^6/w + wQ^2) \cdot (wQ^2/w) + 1) \cdot (wQ^2/w))$$

(7) ヤコビのゼータ関数 $Z(u)$ $(0.0 \leq m \leq 1.0)$

m より $K(m), K(m')$, ノーム q, 補ノーム q' を求める.

$v = u/(2 \cdot K(m))$

$s = SIGN(v), v = \mid v \mid -INT(\mid v \mid), w = 2\pi \cdot v$ として

① $q \leq 0.125$ のとき

・単精度

$$Z(u) = \frac{\pi \cdot 2q}{K(m)} \cdot \frac{\sin(w) - 2q^3 \cdot \sin(2w) + 3q^8 \cdot \sin(3w)}{1 - 2q(\cos(w) - q^3 \cdot \cos(2w) + q^8 \cdot \cos(3w))} \cdot s$$

・倍精度

$$Z(u) = \frac{\pi \cdot 2q}{K(m)} \cdot \frac{\sin(w) - 2q^3 \cdot \sin(2w) + 3q^8 \cdot \sin(3w) - 4q^{15} \cdot \sin(4w)}{1 - 2q(\cos(w) - q^3 \cdot \cos(2w) + q^8 \cdot \cos(3w) - q^{15} \cdot \cos(4w))} \cdot s$$

② $q > 0.125$ のとき

$z = \exp(-\pi u/K(m'))$ として

・単精度

$$Z(u) = \frac{\pi}{2k(m')} \cdot \left(\frac{-5q'^6 z^5 - 3q'^2 z^4 - z^3 + z^2 + 3q'^2 z + 5q'^6}{q'^6 z^5 + q'^2 z^4 + z^3 + z^2 + q'^2 z + q'^6} - 2v \right) \cdot s$$

・倍精度

$$Z(u) = \frac{\pi}{2k(m')} \cdot \left(\frac{(z^3 - z^4) + 3q'^2(z^2 - z^5) + 5q'^6(z - z^6) + 7q'^{12}(1 - z^7)}{(z^3 + z^4) + q'^2(z^2 + z^5) + q'^6(z + z^6) + q'^{12}(1 + z^7)} - 2v \right) \cdot s$$

③ $q >$ 倍精度 0.8 / 単精度 0.6 でしかも, $v \leq 0.5$ のとき

$Z(u) = \tanh(u) - 2v$

(8) ヤコビのエプシロン関数 $E(u)$ $(0.0 \leq m \leq 1.0)$

$E(u) = Z(u) + \frac{E(m)}{K(m)} \cdot u$ ($E(m), K(m)$ は, ノーム q により計算する.)

(9) ヤコビのテータ関数 $\Theta(u)$ $(0.0 \leq m \leq 1.0)$

$\Theta(u) = \vartheta_4(u/2 \cdot K(m), q)$ ($q, K(m)$ はノーム q, ϑ_4 はテータ関数により計算する.)

ただし, $m = 1.0$ のときは $\Theta(u) = 0.0$ とする.

(10) パイ関数 $\Pi(u, \alpha)$ $(0.0 \leq m < 1.0)$

$\Pi(u, \alpha) = \frac{1}{2} \log \frac{\Theta(u-\alpha)}{\Theta(u+\alpha)} + u \cdot Z(\alpha)$

2.1.2.8 初等関数の不定積分

(1) 指数積分 $\overline{\mathrm{Ei}}(x), \mathrm{Ei}(-x)$ $(x > 0.0)$

$x < 0.0$ のときは $\mathrm{Ei}(x), x > 0.0$ のときは $\overline{\mathrm{Ei}}(x)$ を計算する. 以下にその方法を述べる.

① $x < -4.0$ のとき

$$\mathrm{Ei}(x) = \frac{e^x}{x}\left\{1 + (-\frac{1}{x})\frac{\sum_{k=0}^{m} a_k^{(1)}(-\frac{1}{x})^k}{\sum_{k=0}^{m} b_k^{(1)}(-\frac{1}{x})^k}\right\}$$

② $-4.0 \leq x < -1.0$ のとき

$$\mathrm{Ei}(x) = -e^x\frac{\sum_{k=0}^{m} a_k^{(2)}(-\frac{1}{x})^k}{\sum_{k=0}^{m} b_k^{(2)}(-\frac{1}{x})^k}$$

③ $-1.0 \leq x < 0.0$ のとき

$$\mathrm{Ei}(x) = \log(-x) + \gamma + \sum_{n=1}^{\infty}\frac{(-x)^n}{n!n}$$

を最良近似式化して求める (γ はオイラーの定数).
最良近似式化については 2.1.2.22 に記述されている.

④ $0.0 < x \leq 1.0$ のとき

$$\overline{\mathrm{Ei}}(x) = \log(x) + \gamma + \sum_{n=1}^{\infty}\frac{x^n}{n!n}$$

を最良近似式化して求める.
最良近似式化については 2.1.2.22 に記述されている.

⑤ $1.0 < x \leq 6.0$ のとき

$$\overline{\mathrm{Ei}}(x) = \log(\frac{x}{x_0}) + (x - x_0)\frac{\sum_{k=0}^{m} a_k^{(3)}x^k}{\sum_{k=0}^{m} b_k^{(3)}x^k}$$

($x_0 = 0.37250741078136663446$)

⑥ $6.0 < x \leq 12.0$ のとき

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left(a_0^{(4)} + \mathop{\Phi}_{k=1}^{m}\frac{b_{k-1}^{(4)}}{a_k^{(4)} + x}\right)$$

を計算する.

⑦ $12.0 < x \leq 24.0$ のとき

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left(a_0^{(5)} + \mathop{\Phi}_{k=1}^{m}\frac{b_{k-1}^{(5)}}{a_{k-1}^{(5)} + x}\right)$$

⑧ $x > 24.0$ のとき

$$\overline{\mathrm{Ei}}(x) = \frac{e^x}{x}\left\{1 + \frac{1}{x}\left(a_0^{(6)} + \mathop{\Phi}_{k=1}^{m}\frac{b_{k-1}^{(6)}}{a_k^{(6)} + x}\right)\right\}$$

係数 $a_k^{(1)} \sim a_k^{(6)}, b_k^{(1)} \sim b_k^{(6)}$ は文献 (14), (15) に記述されている. また, $a_k^{(3)}, b_k^{(3)}$ は, ずらしチェビシェフ多項式の係数をテレスコーピングして求める.

(2) 対数積分 $\mathrm{Li}(x)$ $(x \geq 1.0)$

指数積分を利用して

$$\mathrm{Li}(x) = \overline{\mathrm{Ei}}(\log(x))$$

を計算する.

(対数積分 $\mathrm{Li}(x)$ は $\mathrm{li}(x)$ とも書かれる.)

(3) 正弦積分 $\mathrm{Si}(x)$

① $x < 0.0$ のとき

$\mathrm{Si}(x) = -\mathrm{Si}(-x)$ として $\mathrm{Si}(-x)$ について以下に述べる方法を適用する.

② $x \geq 0.0$ のとき

i) $x \geq \left\{ \begin{array}{l} \text{倍精度} : 2^{50} \cdot \pi \\ \text{単精度} : 2^{18} \cdot \pi \end{array} \right\}$ の場合

$\mathrm{Si}(x) = \frac{\pi}{2}$ とする.

ii) i) 以外のとき

イ) $x \leq 5.0$ のとき

$$\mathrm{Si}(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1) \cdot (2k+1!)}$$

を最良近似式化して求める.

最良近似式化については 2.1.2.22 に記述されている.

ロ) $x \geq 42.0$ のとき

$\mathrm{Si}(x) = \frac{\pi}{2} - f(x) \cdot \cos(x) - g(x) \cdot \sin(x)$ で計算する.

ここで $f(x), g(x)$ は次の計算式を最良近似式化して求める.

$$f(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (2k)!}{x^{2k+1}}$$

$$g(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (2k+1)!}{x^{2k+2}}$$

最良近似式については 2.1.2.22 に記述されている.

ハ) $5.0 < x < 42.0$ のとき

$Q = \mathrm{Si}(x_n)$ $\left(\begin{array}{l} x_n \text{ は } x \text{ に最も近い整数値で } \mathrm{Si}(x_n) \text{ は } x_n \text{ における正弦積分} \\ \text{の真値} \end{array} \right)$

$z = x - x_n$ とする.

a) $|z| <$ (誤差判定のための単位) $\cdot\, x$ ならば

$\mathrm{Si}(x) = \mathrm{Si}(x_n)$ とする.

b) その他のとき

$f^{(J)}(x_n) = \{(\sin(x_n))^{(J-1)} - (J-1) \cdot f^{(J-1)}(x_n)\}/x_n$

$QQ = f^{(J)}(x_n) \cdot Z^J/J!$

$Q = Q + QQ \qquad (J = 1, 2, 3 \cdots\cdots\cdots)$

を, $|\, QQ \,| < \| Q \cdot$ (誤差判定のための単位) $\|$ になるまで繰返し,

$\mathrm{Si}(x) = Q$

とする.

(4) 余弦積分 $\mathrm{Ci}(x)$ $(x \geq 0.0)$

① $x \leq 2.0$ のとき

$$\mathrm{Ci}(x) = \gamma + \log(x) + \sum_{k=1}^{\infty} \frac{(-1)^k \cdot x^{2k}}{2k \cdot (2k)!}$$

$$(\gamma = 0.57721566490153286061)$$

を最良近似式化して求める.

最良近似式化については 2.1.2.22 に記述されている.

② $x \geq 42.0$ のとき

$\mathrm{Ci}(x) = f(x) \cdot \sin(x) - g(x) \cdot \cos(x)$

ここで, $f(x)$, $g(x)$ は, (35) 正弦積分と同じ.

③ $2.0 < x < 42.0$ のとき

$Q = \mathrm{Ci}(x_n)$ (x_n は x に最も近い整数値で $\mathrm{Ci}(x_n)$ は x_n における余弦積分の真値)

$z = x - x_n$ とする.

イ) $|z| <$ (誤差判定のための単位) $\cdot\, x$ ならば

$\mathrm{Ci}(x) = \mathrm{Ci}(x_n)$ とする.

ロ) その他のとき

$f^{(1)}(x_n) = (\cos(x_n) - 1)/x_n$

$Q = Q + f^{(1)}(x_n) \cdot z$

とおき,

$f^{(J)}(x_n) = \{(\cos(x_n))^{(J-1)} - (J-1) \cdot f^{(J-1)}(x_n)\}/x_n$

$QQ = f^{(J)}(x_n) \cdot z^J/J!$

$Q = Q + QQ$ $(J = 2, 3, \cdots\cdots)$

を, $|\, QQ \,| <| \, Q \cdot$ (誤差判定のための単位) $|$ になるまで繰返し,

$\mathrm{Ci}(x) = Q + \log(\frac{x}{x_n})$

とする.

(5) フレネル積分 $S(x), C(x)$

次の式で定義されるフレネル積分を計算する.

$S(x) = \int_0^x \sin(\frac{\pi}{2} \cdot t^2)dt$

$C(x) = \int_0^x \cos(\frac{\pi}{2} \cdot t^2)dt$

① $x < 0.0$ のとき

$S(x) = -S(-x), C(x) = -C(-x)$ として $S(-x), C(-x)$ について以下に述べる方法を適用する.

② $x \geq 0.0$ のとき

i) $x \leq 1.6$ のとき

$$S(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (\frac{\pi}{2})^{2k+1}}{(2k+1)! \cdot (4k+3)} \cdot x^{4k+3}$$

$$C(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (\frac{\pi}{2})^{2k}}{(2k)! \cdot (4k+1)} \cdot x^{4k+1}$$

を最良近似式化して求める.

最良近似式化については 2.1.2.22 に記述されている.

ii) $x \geq 5.0$ のとき

$$S(x) = \frac{1}{2} - \left\{ f(x) \cdot \cos\left(\frac{\pi}{2} \cdot x^2\right) + g(x) \cdot \sin\left(\frac{\pi}{2} \cdot x^2\right) \right\}$$

$$C(x) = \frac{1}{2} + \left\{ f(x) \cdot \sin\left(\frac{\pi}{2} \cdot x^2\right) - g(x) \cdot \cos\left(\frac{\pi}{2} \cdot x^2\right) \right\}$$

ここで, $f(x), g(x)$ は, 次の計算式を最良近似式化して求める.

$$f(x) = x \cdot \sum_{k=0}^{m} \frac{(-1)^k \cdot (4k-1)!!}{(\pi \cdot x^2)^{2k+1}}$$

$$g(x) = x \cdot \sum_{k=0}^{m} \frac{(-1)^k \cdot (4k+1)!!}{(\pi \cdot x^2)^{2k+2}}$$

最良近似式化については 2.1.2.22 に記述されている.

iii) $1.6 < x < 5.0$ のとき

$Y = \frac{\pi}{2} \cdot x^2$ で変換し,

$$S(x) = \frac{1}{2} - \{f(Y) \cdot \cos(Y) + g(Y) \cdot \sin(Y)\}$$
$$C(x) = \frac{1}{2} + \{f(Y) \cdot \sin(Y) - g(Y) \cdot \cos(Y)\}$$

ここで, $f(Y), g(Y)$ は,

単精度 :

$$f(Y) = \frac{1}{x} \sum_{k=0}^{7} a_k^{(1)} (\frac{4}{Y})^k$$
$$g(Y) = \frac{1}{x} \sum_{k=0}^{8} b_k^{(1)} (\frac{4}{Y})^k$$

倍精度 :

$$f(Y) = \frac{1}{x} \sum_{k=0}^{34} a_k^{(2)} (\frac{4}{Y})^k$$
$$g(Y) = \frac{1}{x} \sum_{k=0}^{34} b_k^{(2)} (\frac{4}{Y})^k$$

係数 $a_k^{(1)}, b_k^{(1)}, a_k^{(2)}, b_k^{(2)}$ はテレスコーピングにより求めた.

(6) ドーソン積分 $F(x)$

ドーソン積分の値は次の式で定義される.

$F(x) = e^{-x^2} \int_0^x e^{t^2} dt$

① $x \geq 0.0$ のとき

$F(x) = -F(-x)$ として $F(-x)$ について以下に述べる方法を適用する.

② $x \geq 0.0$ のとき

i) $x \leq 2.5$ ならば

$$F(x) = x \frac{\sum_{k=0}^{m} a_k^{(1)} x^{2k}}{\sum_{k=0}^{m} b_k^{(1)} x^{2k}}$$

ii) $2.5 < x \leq 3.5$ ならば

$$F(x) = \frac{1}{x} \cdot (a_0^{(2)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(2)}}{a_k^{(2)} + x^2})$$

iii) $3.5 < x \leq 5.0$ ならば

計算式は ii) と同じで係数 $a_k^{(3)}, b_k^{(3)}$ とする

iv) $5.0 < x \leq 1/\varepsilon$ ならば (ε : 誤差判定のための単位)

$$F(x) = \frac{1}{2x} \cdot \{1 + x^{-2} \cdot (a_0^{(4)} + \mathop{\Phi}_{k=1}^{m} \frac{b_{k-1}^{(4)}}{a_k^{(4)} + x^2})\}$$

v) $x > 1/\varepsilon$ ならば

$F(x) = \frac{1}{2x}$

係数 $a_k^{(1\sim4)}, b_k^{(1\sim4)}$ は文献 (15) に記述されている.

(7) 正規分布関数, 余正規分布関数 $\Phi(x), \Psi(x)$

次の式で計算する.

$$\Phi(x) = \frac{1}{2} \cdot \mathrm{Erf}(x/\sqrt{2})$$

$$\Psi(x) = \frac{1}{2} \cdot \mathrm{Erfc}(x/\sqrt{2})$$

ここに Erf, Erfc は, 誤差関数 および 補誤差関数 (余誤差関数と呼ぶこともある) で, これらの定義および計算方法については 2.1.2.21 を参照のこと.

2.1.2.9 ルジャンドル陪関数

(1) 第 1 種ルジャンドル陪関数 $P_n^m(x)$

① 正規化されたものについては, 球面調和関数の項を参照のこと.

② $n < 0$ のとき
$P_n^m(x) = P_{-n-1}^m(x)$ として $P_{-n-1}^m(x)$ について以下に述べる方法を適用する.

③ $m < 0$ かつ $n \geq 0$ のとき
③ または ⑤ に述べる方法によって $P_n^{|m|}(x)$ を求め次の関係式より $P_n^m(x)$ を計算する.

i) $|x| \leq 1.0$ ならば
$$P_n^m(x) = (-1)^m \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} P_n^{|m|}(x)$$

ii) $|x| > 1.0$ ならば
$$P_n^m(x) = \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} P_n^{|m|}(x)$$

④ $m > n \geq 0$ のとき
$P_n^m(x) = 0.0$

⑤ $m = 0$ のとき, $P_n^m(x)$ はルジャンドル多項式 $P_n(x)$ に等しいので $P_0(x) = 1, P_1(x) = x$ を初期値として次の漸化式を計算し $P_n^m(x)$ を求める.
$$P_k(x) = \frac{2k-1}{k} \cdot x \cdot P_{k-1}(x) - \frac{k-1}{k} \cdot P_{k-2}(x) (k = 2, \cdots, n)$$

⑥ $n \geq m > 0$ のとき

i) $n = 1$ で $x \geq \sqrt{1/\varepsilon}$ ならば
$$P_n^m(x) = x$$

ii) $n = m$ ならば
$$P_n^m(x) = (2n-1)!! \cdot (\sqrt{\mid 1 - x^2 \mid})^m$$

iii) $n = m + 1$ ならば
$$P_n^m(x) = (2n-1)!! \cdot x \cdot (\sqrt{\mid 1 - x^2 \mid})^m$$

iv) $n \geq m + 2$ ならば
$$F_m = (2m-1)!!, F_{m+1} = (2m+1)!! \cdot x = F_m \cdot (2m+1) \cdot x$$
を初期値に, 次の漸化式を計算し $P_n^m(x)$ を求める.
$$F_k = \frac{2k-1}{k-m} \cdot x \cdot F_{k-1} - \frac{k+m-1}{k-m} \cdot F_{k-2} \quad (k = m+2, \cdots n)$$
$$P_n^m(x) = (\sqrt{\mid 1 - x^2 \mid})^m \cdot F_n$$

(2) 第 2 種ルジャンドル陪関数 $Q_n^m(x)$ $(n \geq 0, |x| = 1.0)$

① $m < 0$ のとき
② または ③ に述べる方法によって $Q_n^{|m|}$ を求め, 次の関係式より $Q_n^m(x)$ を計算する.

i) $|x| < 1.0$ ならば
$$Q_n^m(x) = (-1)^m \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} Q_n^{|m|}(x)$$

ii) $|x| > 1.0$ ならば
$$Q_n^m(x) = \frac{(n- \mid m \mid)!}{(n+ \mid m \mid)!} Q_n^{|m|}(x)$$

② $|x| < 1.0$ かつ $m \geq 0$ のとき

$Q_0^0(x) = \mathrm{artanh(x)}, \mathrm{Q_1^0(x) = x \cdot Q_0^0(x) - 1}$ を初期値に次の漸化式を実行し,

$Q_n^0(x), Q_{n-1}^0(x)$ を求める ($m = 0$ のときは, これで打ち切る).

$$Q_{k+1}^0(x) = \{(2k+1) \cdot x \cdot Q_k^0(x) - k \cdot Q_{k-1}^0(x)\}/(k+1) \quad (k = 1, 2, \cdots, n-1)$$

次に, $Q_n^1(x) = \{-n \cdot x \cdot Q_n^0(x) + n \cdot Q_{n-1}^0(x)\}/\sqrt{1-x^2}$ により $Q_n^1(x)$ を求める.

なお, $n = 0$ のときは, $Q_n^1(x) = -1/\sqrt{1-x^2}$ とする ($m = 1$ のときは, これで打ち切る).

さらに次の漸化式を計算し, $Q_n^m(x)$ を求める.

$$Q_n^{k+2}(x) = 2 \cdot (k+1) \cdot (x/\sqrt{1-x^2}) \cdot Q_n^{k+1}(x) - (n-k) \cdot (n+k+1) \cdot Q_n^k(x) \quad (k = 0, 1, \cdots, m-2)$$

③ $|x| > 1.0$ かつ $m \geq 0$ のとき

i) $x > \sqrt{n+2}$ および $n \geq m$ のとき

イ) $n = m = 0$ のとき

$$Q_n^m(x) = \mathrm{artanh(1/x)}$$

ロ) $|x| > \sqrt{(n+0.5)/\varepsilon}$ のとき

$$Q_n^m(x) = (-1)^m \frac{(n+m)!}{(2n+1)!!} x^{-n-1}$$

ハ) その他のとき次の級数展開式を計算する.

$$Q_n^m(x) = (-1)^m (x^2-1)^{\frac{m}{2}} \frac{(n+m)!}{(2n+1)!!} x^{-n-m-1} \cdot \left(1 + \sum_{k=1}^{\infty} \frac{(n+m+1) \cdot (n+m+2) \cdots (n+m+2 \cdot k)}{2k!! \cdot (2n+3) \cdot (2n+5) \cdots (2n+2k+1)} \cdot \frac{1}{x^{2k}}\right)$$

ここで Σ 計算は, 末項が初項に対し十分小さくなるまで計算する.

ii) i) 以外のとき

まず, $Q_0^0(x)$ を次式より求める.

$$Q_0^0(x) = \mathrm{artanh(1/x)}$$

($n = m = 0$ のときは, これで打ち切る.)

次に

$$f = \sum_{k=n}^{\infty} \frac{1}{(k+1) \cdot P_k(x) \cdot P_{k+1}(x)}$$

として級数展開式を計算し, $Q_n^0(x),\ Q_{n-1}^0(x)$ を次式より求める.

$$\begin{aligned} Q_n^0(x) &= P_n(x) \cdot f \\ Q_{n-1}^0(x) &= P_{n-1}(x) \cdot f + \frac{1}{n \cdot P_n \cdot (x)} \end{aligned}$$

ここで, $P_k(x)$ は 2.1.2.10 (1) に示すアルゴリズムを用いて解く.

($m = 0$ のときは, これで打ち切る.)

次に,

$$Q_n^1(x) = \frac{n \cdot x \cdot Q_n^0(x) - n \cdot Q_{n-1}^0(x)}{\sqrt{x^2-1}}$$

より $Q_n^1(x)$ を求める.

なお, $n = 0$ のときは $Q_n^1(x) = -1/\sqrt{x^2-1}$ とする

($m = 1$ のときは, これで打ち切る).

さらに次の漸化式を計算し, $Q_n^m(x)$ を求める.

$$Q_n^{k+2}(x) = -2(k+1)(x/\sqrt{x^2-1})Q_n^{k+1}(x) + (n-k)(n+k+1)Q_n^k(x) \quad (k = 0, 1, \cdots, m-2)$$

2.1.2.10 直交多項式

(1) ルジャンドル多項式 $P_n(x)$ $(n \geq 0)$

$P_0(x) = 1.0, P_1(x) = x$ を初期値に次の漸化式を計算し $P_n(x)$ を求める.

$$P_k(x) = \frac{2k-1}{k} \cdot x \cdot P_{k-1}(x) - \frac{k-1}{k} \cdot P_{k-2}(x) \ (k = 2, 3, \cdots, n)$$

(2) ラゲール多項式 $L_n(x)$ $(n \geq 0)$

$L_0(x) = 1.0, L_1(x) = 1.0 - x$ を初期値に次の漸化式を計算し $L_n(x)$ を求める.

$$L_k(x) = \frac{(2k-x-1)}{k} \cdot L_{k-1}(x) - \frac{(k-1)}{k} \cdot L_{k-2}(x) \ (k = 2, 3, \cdots, n)$$

(3) エルミート多項式 $H_n(x)$ $(n \geq 0)$

$H_0(x) = 1.0, H_1(x) = 2 \cdot x$ を初期値に次の漸化式を計算し $H_n(x)$ を求める.

$$H_k(x) = 2 \cdot x \cdot H_{k-1}(x) - 2 \cdot (k-1) \cdot H_{k-2}(x) \ (k = 2, 3, \cdots, n)$$

(4) チェビシェフ多項式 $T_n(x)$ $(n \geq 0)$

$T_0(x) = 1.0, T_1(x) = x$ を初期値に次の漸化式を計算し $T_n(x)$ を求める.

$$T_k(x) = 2 \cdot x \cdot T_{k-1}(x) - T_{k-2}(x) \ (k = 2, 3, \cdots, n)$$

(5) 第 2 種チェビシェフ関数 $U_n(x)$ $(n \geq 0, |x| \leq 1.0)$

$U_0(x) = 0.0,$

$U_1(x) = \sqrt{1-x^2}$ として,

$U_{n+1}(x) = 2 \cdot x \cdot U_n(x) - U_{n-1}(x)$ を計算する.

(6) 一般ラゲール多項式 $L_n^{(\alpha)}(x)$ $(n \geq 0)$

$L_0^{(\alpha)}(x) = 1.0,$

$L_1^{(\alpha)}(x) = (1+\alpha) - x$ として,

$L_n^{(\alpha)}(x) = ((2 \cdot n + \alpha - x - 1) \cdot L_{n-1}^{(\alpha)}(x) + (1 - \alpha - n) \cdot L_{n-2}^{(\alpha)}(x))/n$

を計算する.

2.1.2.11 整数次マシュー関数

以下のような常微分方程式を考える:

$$\frac{d^2y}{dx^2} + (a - 2q\cos 2x)y = 0.$$

実数のパラメータ q に対して, 周期 2π の偶関数解 $y = y(x)$ をもつような実数 $a = c_0, c_1, \cdots$ の全体を

$$c_0 < c_1 < c_2 < \cdots$$

とする. また, 周期 2π の奇関数解をもつような実数 $a = s_1, s_2, \cdots$ の全体を

$$s_1 < s_2 < s_3 < \cdots$$

とする. $a = c_n (n \geq 0)$ に対して, 周期偶関数解を以下の条件で正規化する:

$$\int_0^{2\pi} ce_n(x,q)^2 dx = \pi.$$

この周期偶関数解 $ce_n(x,q)$ は, 整数次マシュー関数 $ce_n(x,q)$ と呼ばれている.

同様に, $a = s_n (n \geq 1)$ のときの周期奇関数解を以下の条件で正規化する:

$$\int_0^{2\pi} se_n(x,q)^2 dx = \pi.$$

この周期奇関数解 $se_n(x,q)$ も, 整数次マシュー関数 $se_n(x,q)$ と呼ばれている.

$ce_n(x,q), se_n(x,q)$ は, 以下のフーリエ展開で与えられることが知られている:

(1) n が偶数の場合の $ce_n(x,q)$ (s=0 とする)

$$ce_n(x,q) = \sum_{r=0}^{\infty} X_r \cos(2rx).$$

(2) n が奇数の場合の $se_n(x,q)$ (s=1 とする)

$$se_n(x,q) = \sum_{r=0}^{\infty} X_r \sin((2r+1)x).$$

(3) n が奇数の場合の $ce_n(x,q)$ (s=2 とする)

$$ce_n(x,q) = \sum_{r=0}^{\infty} X_r \cos((2r+1)x).$$

(4) n が正の偶数の場合の $se_n(x,q)$ (s=3 とする)

$$se_n(x,q) = \sum_{r=0}^{\infty} X_r \sin((2r+2)x).$$

これらは, n 次のマシュー関数と呼ばれている. 展開係数 X_r を求めるために, これらの式を常微分方程式に代入すると, 以下のような実対称 3 重対角行列 $A_s(q)$ の固有値問題に帰着する.

$$A_s(q) = \begin{bmatrix} K_s^2 + d_s q & R_s q & & & \\ R_s q & (2+K_s)^2 & q & 0 & \\ & q & (4+K_s)^2 & q & \\ & 0 & q & (6+K_s)^2 & \ddots \\ & & & \ddots & \ddots \end{bmatrix}$$

ここで,

(1) $R_0 = \sqrt{2}, R_1 = R_2 = R_3 = 1$

(2) $K_0 = 0, K_1 = K_2 = 1, K_3 = 2$

(3) $d_0 = 0, d_1 = -1, d_2 = 1, d_3 = 0$

で, s=0, 1, 2, 3 である. この s は, 計算すべきマシュー関数の種類 (つまり, $ce_n(x,q)$ もしくは $se_n(x,q)$) および次数 n の奇偶で定まる. $A_s(q)$ の固有ベクトル $[R_s X_0, X_1, X_2, X_3, \cdots]^t$ は, 大きさを 1 とする. また, 整定数 l を

(1) $ce_n(x,q)$ を求めるとき
$l = 1 + [n/2]$

(2) $se_n(x,q)$ を求めるとき
$l = n/2$ (n が偶数のとき)
$l = (n+1)/2$ (n が奇数のとき)

としたとき, 小さい方から l 番目の固有値に対応する固有ベクトルの向きは, $X_l > 0$ となるようにとる.
この手法によって得られたマシュー関数 $ce_n(x,q)$, $se_n(x,q)$ は, $q \to 0$ のとき

(1) $ce_0(x,q) \to \frac{1}{\sqrt{2}}$

(2) $ce_n(x,q) \to \cos nx (n = 1, 2, \cdots)$

(3) $se_n(x,q) \to \sin nx (n = 1, 2, \cdots)$

と収束する.

2.1.2.12　ランジュバン関数

(1) $x < 0.0$ のとき

$L(x) = -L(-x)$ として $x > 0.0$ の範囲で計算する.

(2) $x \geq 0.0$ のとき

i) $x \leq 1.5$ のとき

$L(x) = x \cdot \sum_{k=1}^{\infty}(-1)^{k-1}\{\frac{2^{2k} \cdot B_k \cdot x^{2k+2}}{(2k)!}\}$ を最良近似式化して求める.

最良近似式化については 2.1.2.22 に記述されている.

ii) $1.5 < x < \left\{\begin{matrix}\text{倍精度} : 45.0 \\ \text{単精度} : 20.0\end{matrix}\right\}$ のとき

$$L(x) = \frac{2 \cdot e^{-2 \cdot x}}{1 - e^{-2 \cdot x}} - \frac{1}{x} + 1$$

iii) $x \geq \left\{\begin{matrix}\text{倍精度} : 45.0 \\ \text{単精度} : 20.0\end{matrix}\right\}$ のとき

$$L(x) = 1 - \frac{1}{x}$$

2.1.2.13　ガウス・ルジャンドル積分公式

(1) ガウス・ルジャンドル積分公式

次数が $2N-1$ 以下の任意の多項式 $F(x)$ に対し, 定数 $b_0, b_1, \cdots, b_{N-1}$ を適当にとれば, N 次ルジャンドル多項式 $P_N(x)$ で $F(x)$ を割った余りを $P_j(x)(j = 0, 1, \cdots, N-1)$ の線型結合で表示して

$$F(x) = P_N(x)Q(x) + b_0 + \sum_{j=1}^{N-1}\sqrt{2j+1}b_jP_j(x)$$

($Q(x)$ は次数が $N-1$ 以下の多項式) とすることができる. ここで, 次数が $N-1$ 以下の任意の多項式 $Q(x)$ は $P_j(x)(j = 0, 1, \cdots, N-1)$ の線型結合で表示され, 直交性

$$\int_{-1}^{+1} P_N(x)P_j(x)dx = 0 \ (j = 0, 1, \cdots, N-1)$$

により

$$\int_{-1}^{+1} P_N(x)Q(x)dx = 0.$$

さらに,

$$\int_{-1}^{+1} P_j(x)dx = 0 \ (j = 1, 2, \cdots, N-1)$$

に注意して, $F(x)$ の表示式から

$$\int_{-1}^{+1} F(x)dx = 2b_0.$$

$P_N(x)$ の N 個の零点を $\alpha_k(k = 1, 2, \cdots, N)$ とする. $F(x)$ の表示式に $x = \alpha_k$ を代入して

$$F(\alpha_k) = b_0 + \sum_{j=1}^{N-1}\sqrt{2j+1}b_jP_j(\alpha_k) \ (k = 1, 2, \cdots, N)$$

である. いま, $k = 1, 2, \cdots, N$ について

$$d_k^2 = \sum_{n=0}^{N-1} (2n+1) P_n^2(\alpha_k)$$

となる d_k^2 をとると, $F(\alpha_k)$ の上記表示と

$$\sum_{k=1}^{N} d_k^{-2} P_j(\alpha_k) = 0 \ (j = 1, 2, \cdots, N-1),$$

$$\sum_{k=1}^{N} d_k^{-2} = 1$$

により

$$b_0 = \sum_{k=1}^{N} d_k^{-2} F(\alpha_k)$$

したがって, $\alpha_k (k = 1, 2, \cdots, N)$ において, そこでの値 $F(\alpha_k)$ が積分すべき関数 $f(x)$ の値と一致するように多項式 $F(x)$ を選べば, $2N-1$ 次の積分公式

$$\int_{-1}^{+1} f(x)dx \simeq \int_{-1}^{+1} F(x)dx = 2b_0 = 2\sum_{k=1}^{N} d_k^{-2} F(\alpha_k)$$

$$= 2\sum_{k=1}^{N} d_k^{-2} f(\alpha_k)$$

がえられる. ここで, 次数が $2N-1$ 以下の多項式 $F(x)$ は N 個の零点における値に対する条件を満たしていればよい. すなわち, $F(x)$ の選び方には最大 N 個の自由度がある. この積分公式は, 被積分関数 $f(x)$ を多項式 $F(x)$ で近似出来ない場合 (例えば振動する関数) には次数を十分に大きくとる等して, 適用しなければならない.

(2) ルジャンドル多項式の零点

$X_n = \sqrt{2n+1} P_n(\alpha)$ とおく. α が $P_N(\alpha) = 0$ をみたせば, $a_n = n/\sqrt{4n^2-1}$ として

$$\begin{aligned}
a_1 X_1 &= \alpha X_0 \\
a_2 X_2 + a_1 X_0 &= \alpha X_1 \\
a_3 X_3 + a_2 X_1 &= \alpha X_2 \\
&\vdots \\
a_{N-1} X_{N-1} + a_{N-2} X_{N-3} &= \alpha X_{N-2} \\
a_{N-1} X_{N-2} &= \alpha X_{N-1}
\end{aligned}$$

が成り立ち, ルジャンドル多項式の零点が単根であることから, これらは対角成分が 0, 副対角成分が $a_1, a_2, \cdots, a_{N-1}$ である実対称 3 重対角行列の固有値として得られる. 高次数のルジャンドル多項式では, すべての零点が開区間 (-1,1) に密集して存在するため, ニュートン法で零点を求めるかわりに, この実対称 3 重対角行列の固有値問題を無平方根 QR 法で解くことにより, N 個の零点すべてを確実に求めることができる.

2.1.2.14 ベッセル関数の零点

(1) ベッセル関数を含む超越方程式 $aJ_0(\alpha)+\alpha J_1(\alpha)=0$ の正の解
超越方程式

$$aJ_0(\alpha)+\alpha J_1(\alpha)=0$$

の正の解 α は以下の手順で求める.
この方法で超越方程式の正の解を M=50 個程度まで求めることができる.
(単精度では $M\leq 24$ すなわち $N\leq 192$ である.)
パラメータ a は, a=0, または, $10^{-10}\leq|a|\leq 10^4$ が, 適用範囲である.
1. 近似値を有限要素法によって求める.
微分方程式

$$u''(r)+u'(r)/r+\alpha^2 u(r)=0$$

の境界条件

$$u'(1)=au(1)$$

を満たす非自明な解 $u(r)(0\leq r\leq 1)$ で $u(+0)$ が有限なものは, α が $aJ_0(\alpha)+\alpha J_1(\alpha)=0$ を満たすときに限り存在することができて, $u(r)=CJ_0(\alpha r)$ である.
$r=0,1/N,2/N,\cdots,(N-1)/N,1$ に節点をとり, 各節点において $w_j(j/N)=1, w_j(k/N)=0(k\neq j)$ をみたす基底関数 $w_j(r)$ を考える. この w_j を使って, $u(r)$ を

$$u(r)=\sum_{j=0}^{N}u(j/N)w_j(r)$$

と基底関数補間する. 重み付き残差法を適用して

$$\int_0^1 w_j(r)(u''(r)+u'(r)/r+\alpha^2 u(r))rdr=0$$

が得られる. さらに 2 回微分を含む項には部分積分を用いて変形すれば, これは積分関係式

$$\int_0^1 w_j'(r)u'(r)rdr-aw_j(1)u(1)=\alpha^2\int_0^1 w_j(r)u(r)rdr$$

となる. これに基底関数補間を代入すれば

$$AX=\alpha^2 BX,\ X=(u(0),u(1/N),\cdots,u(1))^t,$$

となる. ここで, A,B は,

$$A_{i,j}=\int_0^1 w_j'(r)w_i'(r)rdr-\delta_{i,N}\delta_{j,N}a,$$

$$B_{i,j}=\int_0^1 w_j(r)w_i(r)rdr$$

である. よって, $aJ_0(\alpha)+\alpha J_1(\alpha)=0$ の解 $\alpha>0$ の近似値は, 一般化固有値問題 $AX=\beta BX$ の正の固有値 β を求めて, $\sqrt{\beta}$ とすることで得られる.

2. バイセクション法およびニュートン法による解の改良
$N = LM$ とする. ここで, L は近似倍率であり, 8 程度の値が望ましい.
一般化固有値の正の値の平方根は, 求める超越方程式の正の解の近似値である.
E を

$$E = 0(a < 0)$$

$$E = \alpha(0)^2(a \geq 0)$$

で定義する, ここで $\alpha(0)$ は $J_0(x)$ の最小正零点である.
以下, 一般化固有値のうちで E よりも大きいものを昇順に $\beta_1^*, \beta_2^*, \cdots, \beta_M^*$ とする.
ここで, E 以下の固有値が 2 個以上存在しないことを検査する.
また, β_j^* は重根でないことを検査する.

① バイセクション法
もし, 近似解 $\alpha^* = \sqrt{\beta_j^*}$ $(j \geq 2)$ が,

$$|aJ_0(\alpha^*) + \alpha^* J_1(\alpha^*)| < 0.1 * (|J_0(\alpha^*)| + |J_1(\alpha^*)|)$$

を満たすならば, ニュートン法による近似解の改良処理に進む.
この条件は, j が 10 以下程度なら成り立つ. しかし, j が大きいときは重み付き残差法の近似誤差によってこの条件がなりたたない. したがって, 後者の場合は, バイセクション法を用いて近似解 α^* を改良する.
バイセクション法では, 近似解 α^* が, 解に近づくにつれ $|aJ_0(\alpha^*) + \alpha^* J_1(\alpha^*)|$ は小さくなるという性質を利用する.
j 番目の解のみを含む区間 $(\sqrt{\beta_j^* - \delta}, \sqrt{\beta_j^* + \delta})$ を 2 分割し, 分割された区間の端点の 1 つを α^* に置き換える. ここで, δ は β_j^* の近似誤差程度の正数とする. α^* が上記の条件を満たすまで区間の分割を繰り返した後, ニュートン法による近似解の改良処理に進む.

② ニュートン法
ニュートン法により, $x = \alpha^*$ を $|aJ_0(x) + xJ_1(x)| < e$ を満たすまで反復改良する. ここで,
e=10^{-11} 倍精度
e=10^{-4} 単精度

(2) 整数次ベッセル関数 $J_{m+1}(x)$ の正零点
m=$1, 2, \cdots$ のとき

$$J_m(x) = x^m F_m(x)$$

なる正則関数 $F_m(x)$ を定めることができる. これは

$$F_m''(x) + (2m+1)F_m'(x)/x + F_m(x) = 0$$

を満たす. また, $x^m F_m'(x) = -J_{m+1}(x)$. よって, アルゴリズム (1) の方法を

$$A_{i,j} = \int_0^1 w_j'(r) w_i'(r) r^{2m+1} dr$$

$$B_{i,j} = \int_0^1 w_j(r) w_i(r) r^{2m+1} dr$$

に対して適用する. ただし, 正の値であることを判定する場合の基準 E を
$E = 10^{-3}$ (倍精度)
$E = 10^{-1}$ (単精度)
とし, バイセクション法の誤差判定は,

$$|J_{m+1}(\alpha^*)| \leq 0.001$$

を用いている.

(3) 第 0 次, 第 1 次のベッセル関数 J_0, J_1 の正零点
$J_1(x)$ の零点は, アルゴリズム (1) で a=0 としたときの超越方程式の正の零点として得られる. a を十分大きな値にしたとき, 超越方程式 $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ の解は $J_0(x)$ の零点の近似値となる. したがって, $J_0(x)$ の零点は, 十分大きな a についてアルゴリズム (1) を適用することで得られる.

2.1.2.15 第 2 種ベッセル関数の正零点

① 窓区間 $[x_1,\ x_1 + 2\pi)$ $(x_1 > 0)$ における第 2 種ベッセル関数 $Y_n(x)$ の正零点を以下の手順で求める.

(1) 各区間における正零点
初期値 $x = x_0$ を窓区間の中点にとる.
$F(x) = -Y_n(x)\cos x + J_n(x)\sin x$, $G(x) = Y_n(x)\sin x + J_n(x)\cos x$ とおき, ベクトル $(F(x), G(x))$ の偏角を α とする.

$$\cos\alpha = F(x)/S$$

$$\sin\alpha = G(x)/S$$

$$S = \sqrt{(F(x)^2 + G(x)^2)}$$

ここで, α は, この窓区間に入るように定める. 零点の近似値の改良値 x_{i+1} は, ベクトル $(F(x_i), G(x_i))$ の偏角で与えられる. この改良を反復する.

(2) 正零点の存在判定
もし, この反復改良が収束しないときは, この窓区間には正零点が存在しないとしてよい.

② 窓区間を 2π ずつずらした各区間について, 以上の手順を得られた正零点の個数が指定値になるまで反復する.

2.1.2.16 正定値 2 次形式 $x^2 + ay^2$ のゼータ関数

正定値 2 次形式 $x^2 + ay^2$ のゼータ関数

$$Z(s,a) = \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (m^2 + an^2)^{-s} (s > 1)$$

から, この関数の極を消すための関数を引くことにより, 解析接続は

$$\begin{aligned} & Z(s,a) - \frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)} \\ = & \frac{\pi^s a^{-s/2}}{\Gamma(s)} \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (A_{m,n}^{-s} \int_{A_{m,n}}^{\infty} e^{-t} t^{s-1} dt + A_{m,n}^{s-1} \int_{A_{m,n}}^{\infty} e^{-t} t^{-s} dt) - \frac{\pi^s a^{-s/2}}{\Gamma(s+1)} \\ = & \frac{\pi^s a^{-s/2}}{\Gamma(s)} \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (A_{m,n}^{-s} \Gamma(s, A_{m,n}) + A_{m,n}^{s-1} \Gamma(1-s, A_{m,n})) - \frac{\pi^s a^{-s/2}}{\Gamma(s+1)} \end{aligned}$$

である. ここに, $A_{m,n} = \pi(\sqrt{a}m^2 + n^2/\sqrt{a})$.
和をとるべき範囲は, $A_{m,n} < 40$ をみたす整数対 $(m,n) \neq (0,0)$ の全部である.

2.1.2.17 ディログ関数

$$Li_2(x) = -\int_0^x \log|t-1|\frac{dt}{t}$$

の値は以下のとおりに求める.

① $x = 1$ のとき, 関数値を $\pi^2/6$ とする.

② $x > 1$ のとき, 関数値を $\pi^2/3 - \frac{1}{2}(\log x)^2 - Li_2(\frac{1}{x})$ とする. $Li_2(\frac{1}{x})$ の値は以下によって求める.

③ $0 \leq x < 1$ のとき,

(1) もし, $x > \frac{1}{2}$ ならば, $Li_2(x) = \pi^2/6 - \log(1-x)\log x - Li_2(1-x)$ によって関数値を計算する. $Li_2(1-x)$ は, (2) によって得られる.

(2) もし, $x \leq \frac{1}{2}$ ならば, $Li_2(x)$ は

$$Li_2(x) = \alpha - \frac{1}{4}\alpha^2 - \sum_{k=1}^{8}\frac{c_{2k}}{2k+1}\alpha^{2k+1}$$

となる. ここで, $\alpha = -\log(1-x)$, c_{2k} はベルヌーイ数である.
この関係式から関数値 $Li_2(x)$ を求める.

2.1.2.18 デバイ関数

デバイ関数 $F_D(x)$ は, 以下で定義される関数である.

$$F_D(x) = \frac{3}{x^3}\int_0^x \frac{e^t t^4}{(e^t-1)^2}dt$$

これは,

$$F_D(x) = \frac{12}{x^3}\int_0^x \frac{t^3}{e^t-1}dt - \frac{3x}{e^x-1}$$

となる. この関数値は, x の範囲に応じ, 以下の 2 種類の方法を用いて計算される.

① $0 \leq x \leq \log 2$ のとき

$$F_D(x) = -\frac{3x}{e^x-1} + 12(\frac{1}{3} - \frac{x}{8} - \sum_{k=1}^{\infty}\frac{c_{2k}}{2k+3}x^{2k}) = 1 + \sum_{k=1}^{\infty}\frac{3(2k-1)}{2k+3}c_{2k}x^{2k}$$

により求める. ここで, c_{2k} はベルヌーイ数である. k についての級数和は, 加算項が誤差判定定数 (付録 B 参照) 以下になるまで計算する.

② $x > \log 2$ のとき

$$F_D(x) = \frac{12}{x^3}F(x) - \frac{3x}{e^x-1},$$

$$F(x) = \frac{\pi^4}{15} + S_1(y) + S_2(y)\log y + S_3(y)(\log y)^2 + y(\log y)^3,$$

$$S_j(y) = \sum_{k=1}^{\infty} b_{k,j} y^k (j = 1, 2, 3)$$

により求める. ここで, $y = -\log(1 - e^{-x})$ であり, k についての級数和は加算項が誤差判定定数 (付録 B 参照) 以下になるまで計算する. 係数 $b_{k,j}$ は, 以下の手順で求める.
$F(x) = \int_0^x t^3 dt/(e^t - 1)$ について, パラメータ $y(0 < y < \log 2)$ を用いると,

$$F(x) = \int_y^{\infty} (-\log(1 - e^{-u}))^3 du = \int_0^{\infty} - \int_0^y = I_1 - I_2$$

ここで,

$$I_1 = \int_0^{\infty} = F(\infty) = \frac{\pi^4}{15},$$

$$I_2 = \int_0^y = \int_0^y (\log(\frac{u}{1 - e^{-u}}) - \log u)^3 du.$$

さらに, 被積分関数の含む $\log(\frac{u}{1-e^{-u}})$ を級数展開

$$\log(\frac{u}{1 - e^{-u}}) = \frac{u}{2} + \sum_{k=1}^{\infty} \frac{c_{2k}}{2k} u^{2k}$$

で置き換え, べき級数と $\log u$ の低次べきの積和に変型し, 項別積分の係数 $b_{k,j}$ が得られる. ここで, c_{2k} はベルヌーイ数である.

2.1.2.19 正規化された球面調和関数

実数 $x(-1 \leq x \leq 1)$ に対して正規化された球面調和関数 (正規化されたルジャンドル陪関数)$P_n^{*m}(x)$

$$P_n^{*m}(x) = \frac{1}{4\pi i^m} A_{n,m} \int_{-\pi}^{\pi} (x + i\sqrt{1 - x^2} \cos\psi)^n \cos(m\psi) d\psi$$

を求める. ただし, $i = \sqrt{-1}$,

$$A_{n,0} = \sqrt{\frac{2n + 1}{\pi}}$$

$$A_{n,m} = \sqrt{\frac{2(2n + 1)(n - m)!(n + m)!}{\pi(n!)^2}} \ (1 \leq m \leq n)$$

である.
積分項を計算するために, 以下のフーリエ級数展開を用いる ($x = \cos\theta, 0 \leq \theta \leq \pi$ とする):

$$(\cos\theta + i \sin\theta \cos\psi)^n = \sum_{m=0}^{n} i^m C_{n,m}(\theta) \cos(m\psi)$$

ここで,

$$I_{n,m} = \frac{1}{i^m \sin^m \theta} \int_{-\pi}^{\pi} (\cos\theta + i \sin\theta \cos\psi)^n \cos(m\psi) d\psi$$

とおくと, フーリエ展開係数 $C_{n,m}(\theta)$ は, 以下の漸化式によって得られる:

$$I_{n,n} = \frac{\pi}{2^{n-1}}, \ I_{n,n+1} = 0$$

$$I_{n,m-2} = \frac{2}{m - n - 2} \{-(m - 1) \cos\theta I_{n,m-1} + (m + n) \frac{\sin^2\theta}{2} I_{n,m}\}$$

$$C_{n,m}(\theta) = \frac{(2-\delta_{m,0})\sin^m\theta}{2\pi} I_{n,m}$$

上述の手順によって得られたフーリエ展開係数 $C_{n,m}(\theta)$ を用いると, 次数 n の正規化された球面調和関数 (正規化されたルジャンドル陪関数) $P_n^{*m}(\cos\theta)$ は, 以下のようになる.

$$P_n^{*m}(\cos\theta) = \frac{C_{n,m}(\theta)}{h_{n,m}}$$

ここで, 係数 $h_{n,m}$ は

$$g_{n,0} = 1, g_{n,m} = g_{n,m-1}(1+\frac{m}{n})^{-1}(1-\frac{m-1}{n}) \quad (m = 1, 2, \cdots)$$

$$h_{n,m} = 2\sqrt{\frac{(2-\delta_{m,0})\pi g_{n,m}}{2n+1}}$$

である.

2.1.2.20 実変数フルビッツゼータ関数

(1) 定義

フルビッツゼータ関数 $\zeta(s,a)$ は, 実領域 $s > -1$ かつ $s \neq 1$ で, パラメータ $a > 0$ に対して, 以下のように定義される関数である:

$$\zeta(s,a) = \frac{a^{-s+1}}{s-1} + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx.$$

ここで, $\psi(x)$ は x の小数部分から $1/2$ を引いて定義される x の周期関数 (周期 1) である.

(a) $s > 1$ のときの定義

フルビッツゼータ関数は以下のようになる:

$$\begin{aligned}
\zeta(s,a) &= \int_0^\infty (x+a)^{-s}dx + \frac{a^{-s}}{2} + \int_0^\infty -s\psi(x)(x+a)^{-s-1}dx \\
&= \sum_{n=0}^\infty \{\int_n^{n+1} (x+a)^{-s}dx + \int_n^{n+1} -s\psi(x)(x+a)^{-s-1}dx\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^\infty \{\int_n^{n+1} \frac{d}{dx}(\psi(x)(x+a)^{-s})dx\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^\infty \{\frac{1}{2}(n+a)^{-s} + \frac{1}{2}(n+1+a)^{-s}\} + \frac{a^{-s}}{2} \\
&= \sum_{n=0}^\infty (n+a)^{-s}.
\end{aligned}$$

したがって, $s > 1$ について, フルビッツゼータ関数は有限値に収束する無限級数和 $\sum_{n=0}^\infty (n+a)^{-s}$ に一致する.

(b) $s > -1$ のときの定義

無限積分

$$\int_0^\infty -s\psi(x)(x+a)^{-s-1}dx$$

は, $s > -1$ に対しても収束する. このことは, 交代級数の部分和

$$\sum_{m=0}^M \int_{\frac{m}{2}}^{\frac{m+1}{2}} -s\psi(x)(x+a)^{-s-1}dx$$

が $M \to \infty$ のとき収束することから明らかである. したがって, この範囲においてもフルビッツゼータ関数が定義される.

以上より, フルビッツゼータ関数

$$\zeta(s,a)=\frac{a^{-s+1}}{s-1}+\frac{a^{-s}}{2}+\int_0^\infty -s\psi(x)(x+a)^{-s-1}dx$$

は, $s>1$ について定義された無限級数和 $\sum_{n=0}^{\infty}(n+a)^{-s}$ の定義域を拡張した関数である. また,

$$\zeta(s,a)-\frac{1}{s-1}=\int_0^1\{\zeta(s,a)-\zeta(s,x+1)\}dx$$

は, $s=1$ の場合も含め, $s>-1$ において成り立つ積分表示

$$\int_a^1 x^{-s}dx+\frac{a^{-s}}{2}+\int_0^\infty -s\psi(x)(x+a)^{-s-1}dx$$

をもつ.

(2) 計算のアルゴリズム

(a) 計算に用いる展開式

自然数 $N=10$ として, これを固定する. フルビッツゼータ関数は, 以下のように展開できる.

$$\zeta(s,a)=\sum_{n=0}^{N-1}(n+a)^{-s}+Z(s,a)+\frac{(N+a)^{-s}}{2}+R(s,a)$$

ここに,

$$Z(s,a)=\frac{(N+a)^{-s+1}}{s-1},$$

$$R(s,a)=\int_N^\infty -s(x+a)^{-s-1}\psi(x)dx$$

である. この展開式は, フルビッツゼータ関数 $\zeta(s,a)=\sum_{n=0}^{N-1}(n+a)^{-s}+\sum_{n=N}^{\infty}(n+a)^{-s}$ の右辺第 2 和に, 積分表示を用いれば得られる.

(b) $R(s,a)$ の計算

$R(s,a)$ の値は, 次の近似式で計算する. (近似誤差:単精度 10^{-9}, 倍精度 10^{-18})

$$R(s,a)=\sum_{j=0}^{m-2}(-s)\cdots(-s-j)(N+a)^{-s-1-j}\psi_{2+j}(0)$$

ここで, $\psi_k(x)$ は, 周期 1 の周期関数であり,

$$-\psi(x)=\psi_2'(x),\ -\psi_2(x)=\psi_3'(x),\ -\psi_3(x)=\psi_4'(x),\ \cdots$$

$$\int_0^1\psi_k(x)dx=0$$

を満たすように定める.

$R(s,a)$ の手順は単精度と倍精度とでは異なる. また, s の値によっても異なる.

i. 単精度実数版のとき

$s>10$ ならば $R(s,a)=0$ とする.

$0\le s\le 10$ ならば $m=10$ として $R(s,a)$ の近似式を用いる.

ii. 倍精度実数版のとき

$s>20$ ならば $R(s,a)=0$ とする.

$10<s\le 20$ ならば $m=10$ として $R(s,a)$ の近似式を用いる.

$5<s\le 10$ ならば $m=15$ として $R(s,a)$ の近似式を用いる.

$0\le s\le 5$ ならば $m=20$ として $R(s,a)$ の近似式を用いる.

$\psi_k(0)$ はベルヌーイ数 B_l により

$$\psi_{2j+1}(0) = 0$$

$$\psi_{2j}(0) = (-1)^j \frac{B_j}{(2j)!}$$

と表され, 任意の実数 x と t $(|t| < 2\pi)$ に対して

$$\frac{te^{x_1 t}}{e^t - 1} = 1 - \sum_{k=1}^{\infty} \psi_k(x)(-t)^k$$

が成り立つ. ここに x_1 は x の小数部分である. ここで, ベルヌーイ多項式 $\psi_k(x)$ は

$$(-1)^{\frac{k+1}{2}} 2 \sum_{n=1}^{\infty} \frac{\sin(2\pi n x)}{(2\pi n)^k} \ (k : \text{奇数})$$

$$(-1)^{\frac{k}{2}} 2 \sum_{n=1}^{\infty} \frac{\cos(2\pi n x)}{(2\pi n)^k} \ (k : \text{偶数})$$

とフーリエ展開される. これより, 絶対値が $(2\pi)^{-k}$ 程度であることがわかる.

(c) 特異点処理

フルビッツゼータ関数は $s = 1$ に極を持つので, s が 1 に等しいときは関数値を求めることができない. しかし, $\zeta(s, a) - 1/(s-1)$ は整関数として定義できるので, フルビッツゼータ関数 $\zeta(s, a)$ の展開式の中の $Z(s, a)$ を $|s-1| < 10^{-4}$ (単精度), $|s-1| < 10^{-8}$(倍精度) のときに限り,

$$\frac{(N+a)^{-s+1} - 1}{s-1}$$

の近似式 (理論精度は少なくとも:単精度 10^{-9}, 倍精度 10^{-18})

$$-\log(N+a)(1 - \frac{1}{2}\log(N+a)(s-1)(1 - \frac{1}{3}\log(N+a)(s-1)))$$

に置き換える. ここで,

$$\lim_{s \to 1} (\zeta(s, a) - \frac{1}{s-1}) = -\frac{\Gamma'(a)}{\Gamma(a)}$$

である.

2.1.2.21 誤差関数に関連した関数

(1) 複素変数の誤差関数 $e^{-z^2}\,\mathrm{Erfc}(-iz)$

複素変数の誤差関数 $e^{-z^2}\,\mathrm{Erfc}(-iz)$ は, 定積分

$$e^{-z^2} \int_0^z e^{w^2} dw$$

の値から求めることができる.

(a) $\Im(z)$ の絶対値が小さく 10^{-4} 以下のとき

$\Re(z)$ の特定の値 (0 もこれらの 1 つ) に対して分母が小さくなる可能性を避けるため,

$$f(z) = f_0 + f_1(z - z_0) + f_2(z - z_0)^2/2 + f_3(z - z_0)^3/6$$

で求める. ここに, $f(z) = e^{-z^2} + \frac{2i}{\sqrt{\pi}} D_w(z)$ ($D_w(z)$ はドーソン積分) で,

$$f_0 = f(z_0), f_1 = -2z_0 f_0 + \frac{2i}{\sqrt{\pi}}, f_2 = -2z_0 f_1 - 2f_0, f_3 = -2z_0 f_2 - 4f_1, z_0 = \Re(z)$$

である.

(b) $\Im(z)$ が負のとき

$F(z) = e^{-z^2}\,\mathrm{Erfc}(-iz)$ に対して

$$F(z) + F(-z) = 2e^{-z^2}$$

であることを用いて, 虚数部が正の場合に帰着させる.

(c) $\Im(z)$ が正のとき

$z = \beta = R + iI(R, I$ は実数で, $I > 0)$ とする.

$$f(x) = \int_{-\infty}^{\infty} e^{-xt^2}/(t+\beta)dt = \int_0^{\infty} 2\beta e^{-xt^2}/(\beta^2 - t^2)dt (x \geq 0)$$

とおく. $x > 0$ のとき,

$$(e^{\beta^2 x} f(x))' = (\sqrt{\pi}\beta/\sqrt{x})e^{\beta^2 x}$$

$$e^{\beta^2} f(1) - f(0) = \sqrt{\pi}\beta \int_0^1 \frac{e^{\beta^2 x}}{\sqrt{x}} dx$$

これより,

$$\int_0^{\beta} e^{w^2} dw = \frac{1}{2\sqrt{\pi}}(e^{\beta^2} f(1) - f(0))$$

さらに, $f(0) = \int_0^{\infty} 2\beta/(\beta^2 - t^2)dt$ において, 積分路を $[0, \infty)$ から $[0, i\beta\infty)$ に換えて, 留数を考慮すれば,

$$e^{-\beta^2} \int_0^{\beta} e^{w^2} dw = \frac{1}{2\sqrt{\pi}}(\int_{-\infty}^{\infty} e^{-t^2}/(t+\beta)dt + \pi i e^{-\beta^2})$$

である.

次にポアソンの和公式を適用する.

$$g(x) = h \sum_{n=-\infty}^{\infty} \frac{e^{-h^2(n+x)^2}}{h(n+x)+\beta}$$

とおくと, $g(x)$ は周期 1 の周期関数で, フーリエ級数に展開される:

$$\int_0^1 g(x)e^{-2\pi i n x} dx = \int_{-\infty}^{\infty} h\frac{e^{-h^2x^2}}{hx+\beta} e^{-2\pi i n x} dx$$

$$= \int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta} e^{-2\pi i n x/h} dx$$

$$= \int_{-\infty}^{\infty} \frac{e^{-(x+\pi i n/h)^2}}{x+\beta} dx e^{-\pi^2 n^2/h^2}$$

ここで, 留数を考慮すると,

$$= e^{-\pi^2 n^2/h^2} \int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta-\pi i n/h} dx (\pi n/h < I)$$

$$= e^{-\pi^2 n^2/h^2} (\int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta-\pi i n/h} dx - 2\pi i e^{-(-\beta+\pi i n/h)^2}) (\pi n/h > I)$$

となる. そこで, $\pi/h > \sqrt{\log(10^{16})}$ とれば, 10^{-16} のオーダーより小さい項を無視して,

$$h \sum_{n=-\infty}^{\infty} \frac{e^{-h^2 n^2}}{hn+\beta} = \int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta} dx - 2\pi i \sum_{\pi n/h > I} \exp(2\pi i n\beta/h - \beta^2)$$

が得られる. この両辺の無限和も少ない項で収束する. $e^{-\beta^2}\,\mathrm{Erfc}(-i\beta)$ は

$$\frac{i}{\pi} \int_{-\infty}^{\infty} \frac{e^{-x^2}}{x+\beta} dx$$

である.

(2) 余誤差関数の逆関数

余誤差関数 Erfc(x) は, 以下のように定義される.

$$y = \mathrm{Erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$

ここで, $y \geq 0$ である.

この逆関数 $y = \mathrm{Erfc}^{-1}(x)$ を求めるために, ニュートン法を用いて

$$f(y) = (\frac{2}{\sqrt{\pi}} \int_y^\infty e^{-t^2} dt - x) e^{y^2} = 0$$

を満たすような y を計算する. なお, この $f(y)$ の微分は, $f'(y) = 2yf(y) - 2/\sqrt{\pi}$ である. y の初期値は, x の範囲によって以下のようにとる.

(a) $x \geq \mathrm{Erfc}(2)$ のとき

初期値 y_{init} は, $y_{init} = 0$ とする.

(b) $x < \mathrm{Erfc}(2)$ のとき

初期値 y_{init} は, 与えられた x に対して以下の式を用いて計算する.

$$y_1 = 1/(\sqrt{\pi}x), y_2 = \sqrt{\log y_1}, y_3 = 1/(\sqrt{\pi}xy_2), y_{init} = \sqrt{\log y_3}$$

この手順によって得られた初期値 y_{init} は,

$$\frac{\sqrt{\pi}}{2} x = e^{-y^2} \frac{1}{2y} (1 + O(\frac{1}{y^2}))$$

を満たす y の近似値である.

(3) 誤差関数 Erf, 余誤差関数 Erfc

(a) $|x| \leq 0.476563$ のとき

i. 余誤差関数 $\mathrm{Erfc}(x) = 1 - \mathrm{Erf}(x)$ とする.

ii. 誤差関数 Erf(x) は, 以下による:

$$\mathrm{Erf}(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \cfrac{2x}{1 - \cfrac{2x^2}{3 + \cfrac{4x^2}{5 - \cfrac{6x^2}{7 + \cfrac{8x^2}{9 - \cdots}}}}}$$

(b) $|x| > 0.476563$ のとき

i. 余誤差関数 Erfc(x) は, 以下の手順による.

$0 \leq x \leq 8.0$ のとき, 最良近似式による.

$8.0 < x \leq 26.628736$ のとき

$$\mathrm{Erfc}(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \cfrac{1.0}{x + \cfrac{0.5}{x + \cfrac{1.0}{x + \cfrac{1.5}{x + \cdots}}}}$$

$x > 26.628736$ のとき, Erfc(x)=0 とする.

x が負のときは, $\mathrm{Erfc}(x) = 2 - \mathrm{Erfc}(-x)$ とする.

ii. 誤差関数 $\mathrm{Erf}(x) = 1 - \mathrm{Erfc}(x)$ とする.

2.1.2.22 係数算出法

本ライブラリの関数は, 精度を落さずに計算量を減らす技法を採用している (文献 (18) 参照).

2.1.2.23 関連した特殊関数の計算方法

(1) **第 3 種不完全楕円積分**

$$\begin{aligned}\Pi(\varphi;c,m) &= \int_0^{\varphi} \frac{d\theta}{(1+c\sin^2\theta)\sqrt{1-m\sin^2\theta}} \\ &= \int_0^{x} \frac{dt}{(1+c\,t^2)\sqrt{(1-t^2)(1-mt^2)}} \quad (x=\sin\varphi) \\ &= \int_0^{u} \frac{dw}{1+c\cdot\mathrm{sn}^2\,w} \qquad\qquad (x=\mathrm{sn}\ \ u)\end{aligned}$$

これを Π 関数で表すと, 次式になる.

$$\Pi(\varphi;c,m) = u + \frac{\mathrm{sn}\,\alpha}{\mathrm{cn}\,\alpha\cdot\mathrm{dn}\,\alpha}\cdot\Pi(u,\alpha)$$

ここで α は $\mathrm{sn}^2\,\alpha = -\frac{c}{m}$ となる α とする. また $u = F(x,m) = F(\sin\varphi, m))$ である. (参考文献 (3), (17) 参照)

また, 別の方法として

・ $-m < c < 0$ のとき

$$\epsilon = \sqrt{-\frac{c}{m}}$$
$$\beta = \frac{\pi}{2}\cdot F(\epsilon,m)/K(m)$$
$$v = \frac{\pi}{2}\cdot F(x,m)/K(m) \qquad (x=\sin\varphi)$$
$$\delta_1 = \sqrt{\frac{-c}{(1+c)(m+c)}}$$
$$\Pi(\varphi;c,m) = \delta_1[-\frac{1}{2}\log\frac{\vartheta_4(v+\beta)}{\vartheta_4(v-\beta)} + v\cdot\frac{\vartheta_1'(\beta)}{\vartheta_1(\beta)}]$$

ここで, q を母数 m に対するノーム q として

$$\frac{1}{2}\log\frac{\vartheta_4(v+\beta)}{\vartheta_4(v-\beta)} = 2\sum_{s=1}^{\infty}\frac{q^s}{s\cdot(1-q^{2s})}\cdot\sin(2\cdot s\cdot v)\cdot\sin(2\cdot s\cdot\beta)$$
$$\frac{\vartheta_1'(\beta)}{\vartheta_1(\beta)} = \cot\beta + 4\sum_{s=1}^{\infty}\frac{q^{2s}}{1-2\cdot q^{2s}\cdot\cos(2\cdot\beta)+q^{4s}}\cdot\sin(2\cdot\beta)$$

Σ 計算は, 末項が初項に対し十分小さくなるまで計算する.

・ $c < -1$ のとき

$N = \frac{m}{c}, p_1 = \sqrt{(-c-1)(1+\frac{m}{c})}$ として

$$\Pi(\varphi;c,m) = -\pi(\varphi;N,m) + F(x,m) + \frac{1}{2p_1}\log[\frac{\Delta(\varphi)+p_1\tan\varphi}{\Delta(\varphi)-p_1\tan\varphi}]$$

$\Delta(\varphi) = \sqrt{1-m\sin^2\varphi}, \Pi(\varphi;N,m)$ は $-m < c < 0$ のときの計算を利用

・ $-1 < c < -m$ のとき

$$\epsilon = \sqrt{\frac{1+c}{1-m}}$$
$$\beta = \frac{\pi}{2}\cdot F(\epsilon,1-m)/K(m)$$
$$v = \frac{\pi}{2}\cdot F(x,m)/K(m)$$
$$\delta_2 = \sqrt{\frac{c}{(1+c)(m+c)}}$$
$$\lambda = a\tan(\tanh\beta\cdot\tan v) + 2\sum_{s=1}^{\infty}\frac{(-1)^{s-1}\cdot q^{2s}}{s\cdot(1-q^{2s})}\cdot\sin(2\cdot s\cdot v)\cdot\sinh(2\cdot s\cdot\beta)$$
$$\mu = \frac{\sum_{s=1}^{\infty} s q^{s^2}\sinh(2\cdot s\cdot\beta)}{1+2\sum_{s=1}^{\infty} q^{s^2}\cosh(2\cdot s\cdot\beta)}$$

$\Pi(\varphi;c,m)=\delta_2\cdot(\lambda-4\cdot\mu\cdot v)$

・ $c>0.0$ のとき

$$N=-\frac{m+c}{1+c}$$

$$p_2=\sqrt{\frac{c\cdot(m+c)}{1+c}}$$

$$\Pi(\varphi;c,m)=\{\sqrt{(1+N)\cdot(1+\frac{m}{N})}\cdot\Pi(\varphi;N,m)+\frac{m\cdot F(x,m)}{p_2}$$
$$+a\tan(\frac{1}{2}p_2\cdot\sin(\frac{2\cdot\varphi}{\Delta(\varphi)}))\}/\sqrt{(1+c)(1+\frac{m}{c})}$$

$\Delta(\varphi)=\sqrt{1-m\sin^2\varphi}$, $\Pi(\varphi;N,m)$ は $-m<c<0.0$ のときの計算を利用 (参考文献 (1) 参照)

(2) **ホイマンのラムダ関数**

$$\Lambda_0(\varphi\backslash\alpha)=\frac{2}{\pi}\{K(\alpha)E(\varphi\backslash 90^\circ-\alpha)-(K(\alpha)-E(\alpha))\cdot F(\varphi\backslash 90^\circ-\alpha)\}$$

このとき $m=\sin^2\alpha$ とし

$$K(\alpha)=K(m)$$
$$E(\varphi\backslash 90^\circ-\alpha)=E(\sin\varphi,1-m)$$
$$E(\alpha)=E(m)$$
$$F(\varphi\backslash 90^\circ-\alpha)=F(\sin\varphi,1-m)$$

として求める. ここで K,E,F は楕円積分である. (参考文献 (1) 参照)

(3) **ルジャンドル関数**

$x>1.0$ のとき

$$P_{-1/2}(x)=\frac{2}{\pi}\sqrt{\frac{2}{x+1}}K(\frac{x-1}{x+1})$$

$$P_{1/2}(x)=\frac{2}{\pi}\sqrt{x-\sqrt{x^2-1}}E(\frac{2\sqrt{x^2-1}}{x+\sqrt{x^2-1}})$$

$$Q_{-1/2}(x)=\sqrt{\frac{2}{x+1}}K(\frac{2}{x+1})$$

$$Q_{1/2}(x)=x\sqrt{\frac{2}{x+1}}K(\frac{2}{x+1})-\sqrt{2(x+1)}E(\frac{2}{x+1})$$

$x\leq 1.0$ のとき

$$P_{-1/2}(x)=\frac{2}{\pi}K(\frac{1-x}{2})$$

$$P_{1/2}(x)=\frac{2}{\pi}\{2E(\frac{1-x}{2})-K(\frac{1-x}{2})\}$$

$$Q_{-1/2}(x)=K(\frac{1+x}{2})$$

$$Q_{1/2}(x)=K(\frac{1+x}{2})-2E(\frac{1+x}{2})$$

ここで E,K は完全楕円積分である.

2.1.3 参考文献

(1) Abramowitz, M. and Stegur, I. A. 編, “Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables”, Dover Publications, Inc. (1965).

(2) Hart, J. F. 編, “Computer approximation”, John Wiley and Sons (1968).

(3) 山内二朗, 宇野利雄, 一松信編, “電子計算機のための数値計算法 II, III”, 培風館 (1967, 1972).

(4) 森口繁一, 宇田川 久, 一松信, “数学公式 III–特殊函数–”, 岩波全書, (1960).

(5) 吉田年雄, 二宮市三, “x が小さい場合のベッセル関数 $Y_\nu(x)$ の数値計算”, 情報処理学会論文誌, Vol. 23, No. 3, pp. 269–303 (1982).

(6) 吉田年雄他, “漸化式を用いる複素変数のベッセル関数 $I_n(z)$ の数値計算”, 情報処理, Vol. 14, No. 1, pp. 23–29 (1973).

(7) 吉田年雄, “複素変数のベッセル関数副プログラムについて – $I_n(z)$ 及び $J_n(z)$ –”, 名古屋大学大型計算機センターニュース, Vol. 5, No. 3, pp. 179–185 (1974).

(8) 吉田年雄, 二宮市三, “x が小さい場合の変形ベッセル関数 $K_\nu(x)$ の数値計算”, 情報処理学会論文誌, Vol. 21, No. 3, pp. 238–245 (1980).

(9) 吉田年雄, 二宮市三, “τ –method による複素変数のベッセル関数 $K_n(z)$ の数値計算”, 情報処理, Vol. 14, No. 8, pp. 569–575 (1973).

(10) 渡部力, 名取亮, 小国力編, “第 3 版数値解析と FORTRAN”, 丸善, (1983).

(11) 井阪秀高, “連分数と漸化式を用いるベッセル関数 $J_\nu(x), I_\nu(x)$ の近似計算”, 情報処理学会全国大会講演論文集, Vol. 35, pp. 11–12 (1987).

(12) Cody W. J. , Strecok A. J. and Thacher H. C. , “Chebyshev approximations for the PSI function”, Math. Comp. , Vol. 27, No. 121, pp. 123–127 (1973).

(13) Cody W. J. and Thacher H. C. , “Rational Chebyshev approximations for the exponential integral $E_1(x)$”, Math. Comp. , Vol. 22, pp. 641–649 (1968).

(14) Cody W. J. and Thacher H. C. , “Rational Chebyshev approximations for the exponential integral $E_1(x)$”, Math. Comp. , Vol. 23, pp. 289–303 (1969).

(15) Cody W. J. , Paciorek K. A. and Thacher H. C. Jr. , “Chebyshev approximations for Dawson’s integral”, Math. Comp. , Vol. 24, pp. 171–178 (1970).

(16) 多谷虎男, “ベッセル関数と弾性波動理論”, 山海堂, (1986).

(17) 安藤四郎, “楕円積分, 楕円関数入門”, 日新出版, (1970).

(18) 浜田穂積, “最良近似システム”, 情報処理学会研究報告, 数値解析研究会資料, 6, (1983).

(19) 井阪秀高, “x が中間領域の場合のベッセル関数の近似計算”, 情報処理学会全国大会講演論文集, Vol. 37, pp. 47–48 (1988).

(20) 吉田年雄, 二宮市三, “ベッセル関数 $K_\nu(x)$ の計算法”, 情報処理学会全国大会講演論文集, Vol. 20, pp. 427–428 (1979).

(21) Luke Y. L. , “The special functions and their approximations, II”, Academic Press, (1969).

(22) 吉田年雄, 二宮市三, “x が小さい場合の不完全ガンマ関数 $\Gamma(v,x)$ の数値計算”, 情報処理学会論文誌, Vol. 23, No. 5, pp. 522–528 (1982).

2.2 ベッセル関数

2.2.1 ASL_wibj0x, ASL_vibj0x
第 1 種 0 次ベッセル関数

(1) **機 能**

$x = X_i$ に対して 0 次の第 1 種ベッセル関数

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t))dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibj0x (nv, xi, xo);

単精度関数:

ierr = ASL_vibj0x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$J_0(X_i)$ の値
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) nv ≥ 1

(b) |xi[i − 1]| $\leq M$
ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 第 1 種 ν 次ベッセル関数 $J_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

の基本解であり,

$$J_\nu(z)=\left(\frac{z}{2}\right)^\nu\sum_{m=0}^{\infty}\frac{(-1)^m}{m!\Gamma(m+\nu+1)}\left(\frac{z}{2}\right)^{2m}$$

で定義される.

(b) 第 1 種ベッセル関数は第 1 種円柱関数とも呼ばれる.

(7) **使用例**

(a) 問 題

$J_0(x)$ の値を $x=0.0,0.1,0.2,\cdots,0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibj0x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibj0x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibj0x(nv,xt, xo);

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

    printf( "\n\tValue of J0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wibj0x ***

     ** Input **

      xt =        0
      xt =      0.1
      xt =      0.2
      xt =      0.3
      xt =      0.4
      xt =      0.5
      xt =      0.6
```

```
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of J0(x)

  xo =        1
  xo =    0.998
  xo =     0.99
  xo =    0.978
  xo =     0.96
  xo =    0.938
  xo =    0.912
  xo =    0.881
  xo =    0.846
  xo =    0.808
```

2.2.2 ASL_wiby0x, ASL_viby0x
第 2 種 0 次ベッセル関数

(1) **機 能**

$x = X_i$ に対して 0 次の第 2 種ベッセル関数

$$Y_0(x) = -\frac{2}{\pi}\int_0^{\infty} \cos(x\cosh(t))dt \quad (x > 0.0)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiby0x (nv, xi, xo);

単精度関数:

ierr = ASL_viby0x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$Y_0(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] ≥ 0.0

(c) xi[i − 1] $\leq M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000	xi[i − 1] = 0.0 (overflow)	xo[i − 1] =(最小値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) 第 2 種 ν 次ベッセル関数 $Y_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

の基本解であり,

$$Y_\nu(z)=\frac{J_\nu(z)\cos\nu\pi-J_{-\nu}(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$Y_n(z)=\lim_{\nu\to n}Y_\nu(z)$$

とする.

(b) 第 2 種ベッセル関数は第 2 種円柱関数とも呼ばれる.

(c) ノイマン関数 $N_\nu(z)$ は, 第 2 種ベッセル関数 $Y_\nu(z)$ と同じものである.

(7) **使用例**

(a) 問 題

$Y_0(x)$ の値を $x=0.1,0.2,\cdots,1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiby0x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiby0x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiby0x(nv,xt, xo);

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

    printf( "\n\tValue of Y0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiby0x ***

 ** Input **
```

```
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of Y0(x)

  xo =    -1.53
  xo =    -1.08
  xo =   -0.807
  xo =   -0.606
  xo =   -0.445
  xo =   -0.309
  xo =   -0.191
  xo =  -0.0868
  xo =  0.00563
  xo =   0.0883
```

2.2.3 ASL_wibj1x, ASL_vibj1x
第 1 種 1 次ベッセル関数

(1) **機 能**

$x = X_i$ に対して 1 次の第 1 種ベッセル関数

$$J_1(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t) - t)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibj1x (nv, xi, xo);

単精度関数:

ierr = ASL_vibj1x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$J_1(X_i)$ の値
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) nv ≥ 1

(b) |xi[i − 1]| $\leq M$
ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処 理 内 容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) 注意事項

(a) 第 1 種 ν 次ベッセル関数 $J_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

の基本解であり,

$$J_\nu(z)=\left(\frac{z}{2}\right)^\nu\sum_{m=0}^{\infty}\frac{(-1)^m}{m!\Gamma(m+\nu+1)}\left(\frac{z}{2}\right)^{2m}$$

で定義される.

(b) 第 1 種ベッセル関数は第 1 種円柱関数とも呼ばれる.

(7) 使用例

(a) 問 題

$J_1(x)$ の値を $x=0.0,0.1,0.2,\cdots,0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibj1x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibj1x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibj1x(nv,xt, xo);

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

    printf( "\n\tValue of J1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wibj1x ***

     ** Input **

      xt =        0
      xt =      0.1
      xt =      0.2
      xt =      0.3
      xt =      0.4
      xt =      0.5
      xt =      0.6
```

```
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of J1(x)

  xo =        0
  xo =   0.0499
  xo =   0.0995
  xo =    0.148
  xo =    0.196
  xo =    0.242
  xo =    0.287
  xo =    0.329
  xo =    0.369
  xo =    0.406
```

2.2.4 ASL_wiby1x, ASL_viby1x
第 2 種 1 次ベッセル関数

(1) **機 能**

$x = X_i$ に対して 1 次の第 2 種ベッセル関数

$$Y_1(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin(t) - t)dt - \frac{1}{\pi}\int_0^{\infty} e^{-x\sinh(t)}[e^t - e^{-t}]dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiby1x (nv, xi, xo);

単精度関数:

ierr = ASL_viby1x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$Y_1(X_i)$ の値
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) nv ≥ 1

(b) xi[i $-$ 1] ≥ 0.0

(c) xi[i $-$ 1] $\leq M$
ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000+i	xi[i $-$ 1] $\leq$ 1.0/(最大値) (overflow)	xo[i $-$ 1] =(最小値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i $-$ 1] が制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) 第 2 種 ν 次ベッセル関数 $Y_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}+(z^2-\nu^2)w=0$$

の基本解であり,

$$Y_\nu(z)=\frac{J_\nu(z)\cos\nu\pi-J_{-\nu}(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$Y_n(z)=\lim_{\nu\to n}Y_\nu(z)$$

とする.

(b) 第 2 種ベッセル関数は第 2 種円柱関数とも呼ばれる.

(c) ノイマン関数 $N_\nu(z)$ は, 第 2 種ベッセル関数 $Y_\nu(z)$ と同じものである.

(7) **使用例**

(a) 問 題

$Y_1(x)$ の値を $x=0.1,0.2,\cdots,1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiby1x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiby1x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiby1x(nv,xt, xo);

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

    printf( "\n\tValue of Y1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiby1x ***

 ** Input **
```

```
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of Y1(x)

  xo =    -6.46
  xo =    -3.32
  xo =    -2.29
  xo =    -1.78
  xo =    -1.47
  xo =    -1.26
  xo =     -1.1
  xo =   -0.978
  xo =   -0.873
  xo =   -0.781
```

2.2.5 ASL_dibjnx, ASL_ribjnx
第 1 種整数次ベッセル関数

(1) **機 能**

整数次の第 1 種ベッセル関数

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin(t) - nt)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibjnx (n, xi, & xo);

単精度関数:

ierr = ASL_ribjnx (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$J_n(x)$ の値
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) $|\mathrm{xi}| \le M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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	$\lvert n\rvert(\log_e \lvert\frac{n}{x}\rvert - M_1) > M_2$ であった. (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$)(underflow)	xo $= 0.0$ を返す.
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) $J_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に |n| < 1000, |xi| < 1000.0 とすることが望ましい.

(b) $J_n(x), J_{n+1}(x), J_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方がこの関数を幾度も呼ぶよりも速い.
ただし, n を増大させていくと不安定になるので, n が減少する方向へ適用する.
漸化式:

$$J_{n-1}(x) = \frac{2n}{x}J_n(x) - J_{n+1}(x)$$

(c) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 1 種 ν 次ベッセル関数 $J_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}$$

で定義される.

(e) 第 1 種ベッセル関数は第 1 種円柱関数とも呼ばれる.

(7) **使用例**

(a) 問 題
$J_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibjnx */

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

int main()
{
    int nt;
    double xt;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    fscanf( fp, "%lf", &xt );
    printf( "\tn = %6d\t\txi = %8.3g\n", nt, xt );

    fclose( fp );

    ierr = ASL_dibjnx(nt, xt, &xo);

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

    printf( "\n\tValue of Jn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibjnx ***
 ** Input **
n =      5      xi =      1.5
 ** Output **
ierr =      0
Value of Jn(x)
  xo =   0.0018
```

2.2.6 ASL_dibynx, ASL_ribynx
第 2 種整数次ベッセル関数

(1) **機 能**

整数次の第 2 種ベッセル関数

$$Y_n(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin(t) - nt)dt - \frac{1}{\pi}\int_0^{\infty} e^{-x\sinh(t)}[e^{nt} + (-1)^n e^{-nt}]dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibynx (n, xi, & xo);

単精度関数:

ierr = ASL_ribynx (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$Y_n(x)$ の値
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) xi ≥ 0.0

(b) xi $\leq M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000	xi $\leq 2.0/$ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$) (overflow)	n ≥ 0 のとき xo =(最小値) n < 0 のとき xo =(最小値)$\times(-1)^{\mathrm{n}}$ を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) 注意事項

(a) $Y_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に |n| < 1000, xi < 1000.0 とすることが望ましい.

(b) $Y_n(x), Y_{n+1}(x), Y_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
漸化式:

$$Y_{n+1}(x) = \frac{2n}{x}Y_n(x) - Y_{n-1}(x)$$

(c) この関数で ierr=2000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 2 種 ν 次ベッセル関数 $Y_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$Y_n(z) = \lim_{\nu\to n} Y_\nu(z)$$

とする.

(e) 第 2 種ベッセル関数は第 2 種円柱関数とも呼ばれる.

(f) ノイマン関数 $N_\nu(z)$ は, 第 2 種ベッセル関数 $Y_\nu(z)$ と同じものである.

(7) 使用例

(a) 問 題
$Y_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibynx */

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

int main()
{
    int nt;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", nt, xi );

    fclose( fp );

    ierr = ASL_dibynx(nt, xi, &xo);

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

```
        printf( "\n\tValue of Yn(x)\n\n" );
        printf( "\t  xo = %8.3g\n", xo );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_dibynx ***

 ** Input **

n =      5      xi =      1.5

 ** Output **

ierr =      0

Value of Yn(x)

  xo =    -37.2
```

2.2.7 ASL_dibjmx, ASL_ribjmx
第 1 種実数次ベッセル関数

(1) **機 能**

実数次の第 1 種ベッセル関数

$$J_\nu(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin(t) - \nu t)dt - \frac{\sin(\pi\nu)}{\pi}\int_0^\infty e^{-x\sinh(t)}e^{-\nu t}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibjmx (r, xi, & xo);

単精度関数:

ierr = ASL_ribjmx (r, xi, & xo);

(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}	1	入 力	次数 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$J_\nu(x)$ の値
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) r が整数値と一致するとき

$|\mathrm{r}| \le M_1$

$|\mathrm{xi}| \le M_2$

(b) r が整数値と一致しないとき

$0 < \mathrm{r} \le M_1$

$0 < \mathrm{xi} \le M_2$

ここで, M_1 ={ 倍精度: 2^{31}, 単精度: 2^{31}},

M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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	$\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ であった. (注意事項 (e) 参照) (xi $\neq$ 0.0, r $\neq$ 0.0) (underflow) **注** ν が整数値と一致するときは $\lvert\nu\rvert$, $\lvert x\rvert$ に対して判定	xo = 0.0 を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $J_\nu(x)$ は, x や ν が大きいほど計算時間がかかる. 一般に r $<$ 1000.0, xi $<$ 1000.0 とすることが望ましい.

(b) 半奇数次の場合は, 球ベッセル関数を利用する方がよい.

$$J_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} j_n(x)$$

(c) $\nu < 0$ で ν が整数値と一致しないときは, この関数では計算ができない. したがって漸化式を利用して求める.

(d) $J_\nu(x), J_{\nu+1}(x), J_{\nu+2}(x) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い. ただし, ν を増大させていくと不安定となるので, ν が減少する方向へ適用する.
漸化式:

$$J_{\nu-1}(x) = \frac{2\nu}{x} J_\nu(x) - J_{\nu+1}(x)$$

(e) この関数で ierr=1000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(f) 第 1 種 ν 次ベッセル関数 $J_\nu(z)$ はベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)} \left(\frac{z}{2}\right)^{2m}$$

で定義される.

(g) 第 1 種ベッセル関数は第 1 種円柱関数とも呼ばれる.

(7) 使用例

(a) 問 題

$J_\nu(x)$ の値を $\nu = 3.3$, $x = 1.5$ について求める.

(b) 入力データ

r = 3.3, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibjmx */

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

int main()
{
    double r;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &r );
    fscanf( fp, "%lf", &xi );
    printf( "\tr = %8.3g\t\txi = %8.3g\n", r, xi );

    fclose( fp );

    ierr = ASL_dibjmx(r, xi, &xo);

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

    printf( "\n\tValue of Jm(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibjmx ***

 ** Input **

r =      3.3        xi =      1.5

 ** Output **

ierr =      0

Value of Jm(x)

  xo =   0.0383
```

2.2.8 ASL_dibymx, ASL_ribymx
第 2 種実数次ベッセル関数

(1) **機 能**

実数次の第 2 種ベッセル関数

$$Y_\nu(x) = \frac{1}{\pi}\int_0^\pi \sin(x\sin(t) - \nu t)dt - \frac{1}{\pi}\int_0^\infty e^{-x\sinh(t)}[e^{\nu t} + \cos(\pi\nu)e^{-\nu t}]dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibymx (r, xi, & xo);

単精度関数:

ierr = ASL_ribymx (r, xi, & xo);

(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}	1	入 力	次数 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$Y_\nu(x)$ の値
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) r が整数値と一致するとき

$|\mathrm{r}| \le M_1$

(b) r が整数値と一致しないとき

$0 < \mathrm{r} \le M_1$

ここで, M_1={ 倍精度: 2^{31}, 単精度: 2^{31}}

(c) $\mathrm{xi} \ge 0.0$

(d) $\mathrm{xi} \le M_2$

ここで, M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000	xi $\leq$ 2.0/ (最大値) または $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ であった. (注意事項 (e) 参照) (xi $\neq$ 0.0, r $\neq$ 0) (overflow) **注** ν が整数値と一致するときは $\|\nu\|$ に対して判定	xo =(最小値) を返す. **注** ν が整数で負のときは xo = $(-1)^{\nu+1}\times$ (最大値) を返す.
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) $Y_\nu(x)$ は, x や ν が大きいほど計算時間がかかる. 一般に r $<$ 1000.0, xi $<$ 1000.0 とすることが望ましい.

(b) 半奇数次の場合は, 球ベッセル関数を利用するのが良い.

$$Y_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} y_n(x)$$

(c) $\nu < 0$ で ν が整数値と一致しないときは, この関数では計算ができない. したがって, 漸化式を利用し求める.

$$Y_{\nu-1}(x) = \frac{2\nu}{x} Y_\nu(x) - Y_{\nu+1}(x)$$

(d) $Y_\nu(x), Y_{\nu+1}(x), Y_{\nu+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.

(e) この関数で ierr=2000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(f) 第 2 種 ν 次ベッセル関数 $Y_\nu(z)$ はベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$Y_n(z) = \lim_{\nu\to n} Y_\nu(z)$$

とする.

(g) 第 2 種ベッセル関数は第 2 種円柱関数とも呼ばれる.

(h) ノイマン関数 $N_\nu(z)$ は, 第 2 種ベッセル関数 $Y_\nu(z)$ と同じものである.

(7) 使用例

(a) 問 題

$Y_\nu(x)$ の値を $\nu = 3.3$, $x = 1.5$ について求める.

(b) 入力データ

r = 3.3, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibymx */

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

int main()
{
    double r;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &r );
    fscanf( fp, "%lf", &xi );
    printf( "\tr = %8.3g\t\txi = %8.3g\n", r, xi );

    fclose( fp );

    ierr = ASL_dibymx(r, xi, &xo);

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

    printf( "\n\tValue of Ym(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibymx ***

 ** Input **

r =      3.3        xi =      1.5

 ** Output **

ierr =      0

Value of Ym(x)

  xo =     -2.9
```

2.2.9 ASL_zibjnz, ASL_cibjnz
複素変数第 1 種整数次ベッセル関数

(1) **機 能**

複素変数で整数次の第 1 種ベッセル関数

$$J_n(z) = \frac{1}{\pi}\int_0^{\pi} \cos(z\sin(t) - nt)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibjnz (n, & zi, & zo);

単精度関数:

ierr = ASL_cibjnz (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$J_n(z)$ の値
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) $|\Im(\text{zi})| \le M_1$
ここで, M_1 ={ 倍精度: 709.7827, 単精度: 88.72284}

(b) $|\text{zi}| \le M_2$
ここで, M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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	$\lvert n\rvert(\log_e \frac{\lvert n\rvert}{z} - M_3) > M_4$ (注意事項 (c) 参照) ($\lvert\text{zi}\rvert \ne 0.0$, n $\ne 0$) (underflow)	zo = (0.0, 0.0) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $J_n(z)$ は, $|z|$ や n が大きいほど時間がかかり, 一般に |n| < 1000, |zi| < 1000.0 とすることが望ましい.

(b) $J_n(z), J_{n+1}(z), J_{n+2}(z)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い. ただし, n を増大させていくと不安定となるので, n が減少する方向へ適用する.
漸化式:

$$J_{n-1}(z) = \frac{2n}{z}J_n(z) - J_{n+1}(z)$$

(c) この関数で ierr=1000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, M_4 =\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 1 種 ν 次ベッセル関数 $J_\nu(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\nu+1)}\left(\frac{z}{2}\right)^{2m}$$

で定義される.

(e) 第 1 種ベッセル関数は第 1 種円柱関数とも呼ばれる.

(7) **使用例**

(a) 問 題
$J_n(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
n = 3, zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_zibjnz */

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

int main()
{
    int n;
    double _Complex zi;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_zibjnz ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &n );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zi = tmp_re + tmp_im * _Complex_I;

    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", n, creal(zi), cimag(zi) );

    fclose( fp );

    ierr = ASL_zibjnz(n, &zi, &zo);

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

    printf( "\n\tValue of Jn(z)\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zibjnz ***
 ** Input **
n =      3      zi = (       1,       2)
 ** Output **
ierr =      0
Value of Jn(z)
  zo = (  -0.281,  0.0172)
```

2.2.10 ASL_zibynz, ASL_cibynz
複素変数第 2 種整数次ベッセル関数

(1) **機 能**

複素変数で整数次の第 2 種ベッセル関数

$$Y_n(z) = \frac{1}{\pi}\int_0^{\pi} \sin(z\sin(t) - nt)dt - \frac{1}{\pi}\int_0^{\infty} e^{-z\sinh(t)}[e^{nt} + (-1)^n e^{-nt}]dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibynz (n, & zi, & zo);

単精度関数:

ierr = ASL_cibynz (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$Y_n(z)$ の値
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) $|\mathrm{zi}| > 0.0$

(b) $|\Im(\mathrm{zi})| \le M_1$
ここで, M_1 ={ 倍精度: 709.7827, 単精度: 88.72284}

(c) $|\mathrm{zi}| \le M_2$
ここで, M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	$\|zi\| \leq 2.0/$ (最大値) または $\|n\|(\log_e \frac{\|n\|}{\|z\|} - M_3) > M_4$ (注意事項 (d) 参照) ($\|zi\| \neq 0.0$, $n \neq 0$)	

(6) **注意事項**

(a) 第 2 種ベッセル関数 $N_n(z)$ は, $Y_n(z)$ と同じものである.

(b) $Y_n(z)$ は, $|z|$ や n が大きいほど計算時間がかかる. 一般に $|n| < 1000, |zi| < 1000.0$ とすることが望ましい.

(c) $Y_n(z), Y_{n+1}(z), Y_{n+2}(z) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
漸化式:

$$Y_{n+1}(z) = \frac{2n}{z} Y_n(z) - Y_{n-1}(z)$$

(d) この関数で ierr=4000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(e) 第 2 種 ν 次ベッセル関数 $Y_\nu(z)$ はベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の基本解であり,

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$Y_n(z) = \lim_{\nu \to n} Y_\nu(z)$$

とする.

(f) 第 2 種ベッセル関数は第 2 種円柱関数とも呼ばれる.

(g) ノイマン関数 $N_\nu(z)$ は, 第 2 種ベッセル関数 $Y_\nu(z)$ と同じものである.

(7) **使用例**

(a) 問 題
$Y_n(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
n = 3, zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_zibynz */

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

int main()
{
    int nt;
```

```
    double _Complex zt;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_zibynz ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &nt );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zt = tmp_re + tmp_im * _Complex_I;

    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", nt, creal(zt), cimag(zt) );

    fclose( fp );

    ierr = ASL_zibynz(nt, &zt, &zo);

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

    printf( "\n\tValue of Yn(z)\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zibynz ***

 ** Input **

n =      3      zi = (       1,       2)

 ** Output **

ierr =      0

Value of Yn(z)

  zo = (    0.29,  -0.212)
```

2.3 ベッセル関数の零点

2.3.1 ASL_dizbs0, ASL_rizbs0
第 1 種 0 次ベッセル関数の正零点

(1) **機 能**

第 1 種 0 次ベッセル関数 $J_0(x)$ の正の零点を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dizbs0  (n, z);
```

単精度関数:

```
ierr = ASL_rizbs0  (n, z);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	正の零点の個数
2	z	{ D∗ / R∗ }	n	出 力	第 0 次の第 1 種ベッセル関数の正の零点 (小さい方から順に入る)
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq n \leq 50$

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

なし

(7) 使用例

(a) 問 題

N = 20 として,$J_0(x)$ の 20 番目までの正の零点を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    double *z,y;
    int     i,n,ierr;
    n=20;
    z=(double * )malloc((size_t)( sizeof(double)* n ));
    if( z  == NULL )
    {
        printf( "no enough memory for array z\n" );
        return -1;
    }
    printf( "\n\t *** ASL_dizbs0 \n\n" );
        printf( "\n\t input \n\n" );
        printf( "\n\t order of bessel function  =  0\n\n");
        ierr = ASL_dizbs0(n, z);
        printf( "\n\t *** OUTPUT ***\n\n" );
        printf( "\n\tierr = %6d\n", ierr );
        printf( "\n\t zero points of the bessel function of the order 0\n\n");
        printf( "\n\t i    z[i]       abs error \n\n" );
        for ( i=0 ; i<n ; i++ )
        {
            ierr=ASL_dibj0x(z[i],&y);
            printf( "\n%6d, %13.8g,%13.8g\n " ,i,z[i],y);
        }
    free(z);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dizbs0

 input

 order of bessel function  =  0

 *** OUTPUT ***

ierr =      0
 zero points of the bessel function of the order 0

 i    z[i]       abs error

 0,     2.4048256,            0
 1,     5.5200781,-5.5511151e-17
 2,     8.6537279,-8.550601e-17
 3,     11.791534,5.6074844e-16
 4,     14.930918,3.1425706e-17
 5,     18.071064,2.0607712e-17
 6,     21.211637,2.3465576e-16
 7,     24.352472,-2.8684606e-16
 8,     27.493479,-1.6343398e-16
 9,     30.634606,-4.6072464e-17
10,      33.77582,-1.7036219e-16
11,     36.917098,-4.5752467e-16
12,     40.058426,-3.7350766e-16
13,     43.199792,2.2315316e-16
14,     46.341188,4.1065487e-16
15,      49.48261,3.5326335e-17
16,     52.624052,2.9204024e-16
```

```
17,     55.765511,-2.3113754e-16
18,     58.906984,-8.4474973e-17
19,     62.048469,-8.6245239e-17
```

2.3.2 ASL_dizbs1, ASL_rizbs1
第 1 種 1 次ベッセル関数の正零点

(1) **機 能**

第 1 種 1 次ベッセル関数 $J_1(x)$ の正の零点を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dizbs1 (n, z);

単精度関数:

ierr = ASL_rizbs1 (n, z);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	正の零点の個数
2	z	{ D∗ / R∗ }	n	出 力	第 1 次の第 1 種ベッセル関数の正の零点 (小さい方から順に入る)
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $1 \leq n \leq 50$

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

N = 20 として,$J_1(x)$ の 20 番目までの正の零点を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    double *z,y;
    int     i,n,ierr;
    n=20;
    z=(double * )malloc((size_t)( sizeof(double)* n ));
    if( z  == NULL )
    {
        printf( "no enough memory for array z\n" );
        return -1;
    }
    printf( "\n\t *** ASL_dizbs1 \n\n" );
```

```
        printf( "\n\t input \n\n" );
        printf( "\n\t order of bessel function  =  1\n\n");
        ierr = ASL_dizbs1(n, z);
        printf( "\n\t *** OUTPUT ***\n\n" );
        printf( "\n\tierr = %6d\n", ierr );
        printf( "\n\t zero points of the bessel function of the order 1\n\n");
        printf( "\n\t i    z[i]       abs error \n\n" );
        for ( i=0 ; i<n ; i++ )
        {
            ierr=ASL_dibj1x(z[i],&y);
            printf( "\n%6d, %13.8g,%13.8g\n " ,i,z[i],y);
        }
    free(z);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dizbs1

 input

 order of bessel function  =  1

 *** OUTPUT ***

ierr =      0
 zero points of the bessel function of the order 1

 i    z[i]       abs error

 0,      3.831706,            0
 1,     7.0155867,5.5511151e-17
 2,     10.173468,8.9194811e-17
 3,     13.323692,1.5873198e-16
 4,      16.47063,1.8636384e-16
 5,     19.615859,-2.8122313e-16
 6,     22.760084,-2.4689546e-16
 7,     25.903672,7.7711347e-17
 8,     29.046829,-1.8668477e-16
 9,      32.18968,3.5559172e-16
10,     35.332308,3.8406094e-16
11,     38.474766,3.3699954e-17
12,     41.617094,-5.1091421e-17
13,     44.759319,-1.6050153e-16
14,     47.901461,3.0616063e-16
15,     51.043535,-1.9242396e-16
16,     54.185554,1.3208823e-16
17,     57.327525,-4.9030061e-17
18,     60.469458,-1.2525988e-16
19,     63.611357,-7.8518163e-17
```

2.3.3 ASL_dizbsn, ASL_rizbsn
第 1 種整数次ベッセル関数の正零点

(1) **機 能**

第 1 種整数次ベッセル関数 $J_m(x)$ の正の零点を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dizbsn  (n,m,lf, z, work);
```

単精度関数:

```
ierr = ASL_rizbsn  (n,m,lf, z, work);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	正の零点の個数
2	m	I	1	入 力	次数 m
3	lf	I	1	入 力	近似倍率
4	z	{ D∗ / R∗ }	n	出 力	正の零点 (小さい方から順に入る)
5	work	{ D∗ / R∗ }	内容参照	ワーク	作業領域 大きさ:2 × (lf×n+1) × (lf×n+2)
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) $1 \leq$ n ≤ 50 (m $= -1, 0, +1$) または $1 \leq$ n (それ以外)

(b) lf ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3500	反復改良による解の改良ができなかった.	
3600	固有値問題の解が求められなかった.	

(6) 注意事項

(a) n は 50 程度を上限とすべきである.

(b) 反復改良は lf×n 回で打ち切られる. さらに, この値は反復改良の初期値を求める際に解かれる固有値問題の次数としても使われる. この値の大きさが十分ではない場合, 反復改良に使われる初期値の近似精度が悪くなり, ierr=3500,3600 が発生する原因となる. 逆に, この値が大きすぎると, 反復改良に使われる初期値を求める計算にかかる処理時間が増大する.
目安として, m が 10 程度の場合は n×lf を 24 以上, m が 18 程度の場合は n×lf を 30 以上にとるとよい.

(c) m=-1,0,+1 のときは, 数値テーブルを参照して計算しているので, 処理時間は比較的に短くなる.

(7) 使用例

(a) 問 題
N=20,M=10 として,$J_{10}(x)$ の 20 番目までの正の零点を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
        double *z;
    double *work;
    double *b;
        int     i,ia,n,nn,ierr,i8;
        n=20;i8=8;
    nn=2*(1+8*n)*(2+8*n);
        z=(double * )malloc((size_t)( sizeof(double)* n ));
        if( z  == NULL )
        {
                printf( "no enough memory for array z\n" );
                return -1;
        }
    work=(double * )malloc((size_t)( sizeof(double)* nn ));
    if( work == NULL )
    {
        printf( "no enough memory for array work \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( "\n\t *** ASL_dizbsn \n\n" );
    for ( ia = 10 ; ia < 11 ; ia++ )
    {
    printf( "\n\t            \n\n" );
    printf( "\n\t *** INPUT order = %6d\n" , ia );
    ierr=ASL_dizbsn(n,ia,i8,z,work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    for (i=0 ; i<n ; i++ )
    {
        ierr=ASL_dibjnx(ia,z[i],&b[i]);
    }
    printf( "\n\t i    z[i]       abs error \n\n" );
    for ( i=0 ; i<n ; i++ )
    {
        printf( "\n%6d,%13.8g,%13.8g\n " ,i, z[i], b[i]);
    }
    }
    free(z);
    free(work);
    free(b);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dizbsn



 *** INPUT order =     10

 *** OUTPUT ***
```

```
ierr =      0
 i    z[i]       abs error

 0,    14.475501,-1.6653345e-16
 1,    18.433464,1.9428903e-16
 2,    22.046985,1.9428903e-16
 3,    25.509451,-2.6367797e-16
 4,    28.887375,1.3877788e-17
 5,    32.211856,-3.1918912e-16
 6,    35.499909,4.4408921e-16
 7,    38.761807,1.9428903e-16
 8,     42.00419,5.5511151e-17
 9,    45.231574,2.4980018e-16
10,    48.447151,-5.5511151e-17
11,    51.653252,1.6653345e-16
12,    54.851619,-1.5959456e-16
13,    58.043588,-6.9388939e-17
14,    61.230198,-3.400058e-16
15,    64.412272,3.5388359e-16
16,    67.590472,-3.5388359e-16
17,    70.765334,-6.0715322e-16
18,    73.937299,4.6143644e-16
19,    77.106734,-1.6306401e-16
```

2.3.4 ASL_dizbyn, ASL_rizbyn
第 2 種整数次ベッセル関数の正零点

(1) **機 能**

第 2 種整数次ベッセル関数 $Y_m(x)$ の正の零点を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dizbyn  (n, m, z, & nconv);
```

単精度関数:

```
ierr = ASL_rizbyn  (n, m, z, & nconv);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	正の零点の個数
2	m	I	1	入 力	次数 m
3	z	{ D∗ / R∗ }	n	出 力	正の零点 (小さい方から順に入る.)
4	nconv	I*	1	出 力	最大反復回数
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $n \geq 1$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	最大反復回数以内で収束しなかった.	

(6) **注意事項**

(a) 次数の絶対値が 5 以上の第 2 種ベッセル関数の零点を求めるときは, 倍精度版を使用する.

(7) 使用例

(a) 問 題

$Y_{10}(x)$ の 20 番目までの正零点を求める.

(b) 入力データ

n=20, m=10

(c) 主プログラム

```
/*       C interface example for ASL_dizbyn */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int n,m,ierr,nconv,i;
    double *z,derr;
    printf( "    *** ASL_dizbyn ***\n" );
    n=20;m=10;
    printf( "\n     ** Input **\n\n" );
    z = ( double * )malloc(sizeof(double)*n);
    if( z == NULL )
    {
        printf( "no enough memory for array z\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tm   = %6d\n" , m );
    ierr = ASL_dizbyn(n,m,z,&nconv);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\tnconv = %6d\n", nconv);
    printf( "\n\t          Zero Point         Valid" );
    for(i=0; i<n; i++)
    {
        printf( "\n\t" );
        printf( "%6d",i);
        printf( "%8.3g",z[i]);
        printf( "\t              ");
        ierr = ASL_dibynx(m, z[i], &derr);
        if( ierr > 0 )
        {
            printf( "error in ASL_dibynx\n");
            return -1;
        }
        printf( "%8.3g",derr);
    }
    printf( "\n" );
    free(z);
    return 0;
}
```

(d) 出力結果

```
*** ASL_dizbyn ***

 ** Input **

n   =     20
m   =     10

 ** Output **

ierr =      0
nconv =     29

           Zero Point         Valid
     0    12.1               -3.35e-12
     1    16.5               -4.68e-13
     2    20.3               6.95e-14
     3    23.8               1.45e-13
     4    27.2               -1.44e-14
     5    30.6               2.47e-14
     6    33.9               -3.22e-15
     7    37.1               1.78e-14
     8    40.4               -3.57e-15
     9    43.6               5.41e-16
    10    46.8               7.98e-16
    11    50.1               3.52e-15
    12    53.3               1.25e-16
    13    56.4               -1.01e-15
    14    59.6               1.94e-15
    15    62.8               -1.96e-15
    16      66                3.3e-15
    17    69.2               -2.51e-14
    18    72.4               3.51e-15
    19    75.5               -4.1e-15
```

2.3.5 ASL_dizbsl, ASL_rizbsl
$aJ_0(\alpha) + xJ_1(\alpha)$ の正零点

(1) **機 能**

任意の実パラメータ a を与えて, ベッセル関数を含む方程式 $aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ の正の解 α を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dizbsl  (n, a, lf, z, work);
```

単精度関数:

```
ierr = ASL_rizbsl  (n, a, lf, z, work);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	正の解の個数
2	a	{D / R}	1	入 力	パラメータ a
3	lf	I	1	入 力	近似倍率
4	z	{D∗ / R∗}	n	出 力	正の解 α(小さい方から順に入る)
5	work	{D∗ / R∗}	内容参照	ワーク	作業領域 大きさ:2 × (lf×n+1) × (lf×n+2)
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 ≥ 1

(b) lf ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3500	反復改良による解の改良ができなかった.	
3600	固有値問題の解が求められなかった.	

(6) **注意事項**

(a) パラメータ a(入力値 a) の有効な適用範囲は $10^{-10} \leq |a| \leq 10^4$ および $a = 0$ である.

(b) n は 50 程度を上限とすべきである.

(c) 反復改良は lf×n 回で打ち切られる.

(d) lf は 8 程度が望ましい.

(7) **使用例**

(a) 問 題

$a = -\beta \dfrac{J_1(\beta)}{J_0(\beta)}(\beta = 2.304780)$,n=20,lf=8 に対して,$aJ_0(\alpha) + \alpha J_1(\alpha) = 0$ の正の解 α を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
        double *z;
    double *work;
    double a,f,d,p;
    double *b0,*b1;
        int     i,n,nn,ierr,i8;
        n=20;i8=8;
    nn=2*(1+8*n)*(2+8*n);
        z=(double * )malloc((size_t)( sizeof(double)* n ));
        if( z  == NULL )
        {
                printf( "no enough memory for array z\n" );
                return -1;
        }
    work=(double * )malloc((size_t)( sizeof(double)* nn ));
    if( work == NULL )
    {
        printf( "no enough memory for array work \n" );
        return -1;
    }
    b0=(double * )malloc((size_t)( sizeof(double)* n ));
        if( b0  == NULL )
        {
                printf( "no enough memory for array b0\n" );
                return -1;
        }
        b1=(double * )malloc((size_t)( sizeof(double)* n ));
        if( b1  == NULL )
        {
                printf( "no enough memory for array b1\n" );
                return -1;
        }
        printf( "\n\t  *** ASL_dizbsl \n\n" );
    a=2.304780;
    ierr=ASL_dibj0x(a,&f);
    ierr=ASL_dibj1x(a,&d);
    a=-d*a/f;
    printf( "\n\t input : a \n\n" );
    printf( "\n%13.8g\n " ,a);
    ierr=ASL_dizbsl(n,a,i8,z,work);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    for (i=0 ; i<n ; i++ )
    {
        ierr=ASL_dibj0x(z[i],&b0[i]);
        ierr=ASL_dibj1x(z[i],&b1[i]);
    }
    printf( "\n\t i    z[i]       abs error \n\n" );
    for ( i=0 ; i<n ; i++ )
    {
        p=a*b0[i]+b1[i]*z[i];
        printf( "\n%6d,%13.8g,%13.8g\n " ,i, z[i], p);
    }
    free(z);
    free(work);
    free(b0);
    free(b1);
    return 0;
}
```

(c) 出力結果

```
        *** ASL_dizbsl

       input : a
```

```
-23.456847
  *** OUTPUT ***

 ierr =      0
  i    z[i]       abs error

  0,      2.30478,            0
  1,    5.2934884,1.3322676e-15
  2,    8.3065969,-4.8849813e-15
  3,    11.333025,-1.2434498e-14
  4,    14.371644,6.2172489e-15
  5,    17.421959,-5.7731597e-15
  6,    20.483189,4.4408921e-15
  7,    23.554295,1.3322676e-15
  8,    26.634127,            0
  9,    29.721552,5.7731597e-15
 10,    32.815519,-3.5527137e-15
 11,    35.915098,1.4210855e-14
 12,    39.019482,-1.8651747e-14
 13,    42.127984,1.5099033e-14
 14,     45.24002,-4.4408921e-16
 15,    48.355101,-6.6613381e-15
 16,    51.472814,1.5543122e-14
 17,     54.59281,1.7319479e-14
 18,    57.714796,2.3092639e-14
 19,    60.838523,1.1990409e-14
```

2.4 変形ベッセル関数

2.4.1 ASL_wibi0x, ASL_vibi0x
第 1 種 0 次変形ベッセル関数

(1) **機 能**

$x = X_i$ に対して 0 次の第 1 種変形ベッセル関数

$$I_0(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibi0x (nv, xi, xo);

単精度関数:

ierr = ASL_vibi0x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$I_0(X_i)$ の値
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) nv ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000	制限条件 (a) を満足しなかった.	処理を打ち切る.
2000+i	$\|$xi[i − 1]$\| > M$ であった. (注意事項 (a) 参照) (overflow)	xo[i − 1] = (最大値) を返す.

(6) **注意事項**

(a) この関数で ierr=2000 となるときの M の値は以下の通りである.
M ={ 倍精度: 713.067, 単精度: 90.978}

(b) 第 1 種 ν 次変形ベッセル関数 $I_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0$$

の特殊解であり,

$$I_{\pm\nu}(z)=e^{\mp\sqrt{-1}\pi/2}J_{\pm\nu}(\sqrt{-1}z)$$

で定義される.

(7) **使用例**

(a) 問 題
$I_0(x)$ の値を $x=0.0,0.1,\cdots,0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibi0x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibi0x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibi0x(nv,xt, xo);

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

    printf( "\n\tValue of I0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibi0x ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
```

```
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of I0(x)

  xo =        1
  xo =        1
  xo =     1.01
  xo =     1.02
  xo =     1.04
  xo =     1.06
  xo =     1.09
  xo =     1.13
  xo =     1.17
  xo =     1.21
```

2.4.2 ASL_wibk0x, ASL_vibk0x
第 2 種 0 次変形ベッセル関数

(1) **機 能**

$x = X_i$ に対して 0 次の第 2 種変形ベッセル関数

$$K_0(x) = \int_0^\infty e^{-x\cosh(t)}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibk0x (nv, xi, xo);

単精度関数:

ierr = ASL_vibk0x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$K_0(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] ≥ 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	xi[i − 1] $> M$ であった. (注意事項 (a) 参照) (underflow)	xo[i − 1] = 0.0 を返す.
2000	xi[i − 1] = 0.0 (overflow)	xo[i − 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) この関数で ierr=1000 となるときの M の値は以下の通りである.
M ={ 倍精度: 705.117, 単精度: 85.114}

(b) 第 2 種 ν 次変形ベッセル関数 $K_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0$$

の特殊解であり,

$$K_\nu(z)=\frac{\pi}{2}\frac{I_{-\nu}(z)-I_\nu(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$K_n(z)=\lim_{\nu\to n}K_\nu(z)$$

とする.

(7) **使用例**

(a) 問 題
$K_0(x)$ の値を $x=0.1,0.2,\cdots,1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibk0x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibk0x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibk0x(nv,xt, xo);

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

    printf( "\n\tValue of K0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wibk0x ***

     ** Input **
```

```
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of K0(x)

  xo =     2.43
  xo =     1.75
  xo =     1.37
  xo =     1.11
  xo =    0.924
  xo =    0.778
  xo =    0.661
  xo =    0.565
  xo =    0.487
  xo =    0.421
```

2.4.3 ASL_wibi1x, ASL_vibi1x
第 1 種 1 次変形ベッセル関数

(1) **機 能**

$x = X_i$ に対して 1 次の第 1 種変形ベッセル関数

$$I_1(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}\cos(t)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibi1x (nv, xi, xo);

単精度関数:

ierr = ASL_vibi1x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$I_1(X_i)$ の値
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) nv ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000	制限条件 (a) を満足しなかった.	処理を打ち切る.
2000+i	$\|$xi[i − 1]$\|$ $> M$ であった. (注意事項 (a) 参照) (overflow)	xi[i − 1] ≥ 0.0 のとき xo[i − 1] =(最大値) xi[i − 1] < 0.0 のとき xo[i − 1] = −(最大値)

(6) **注意事項**

(a) この関数で ierr=2000 となるときの M の値は以下の通りである.
M ={ 倍精度: 713.067, 単精度: 90.978}

(b) 第 1 種 ν 次変形ベッセル関数 $I_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0$$

の特殊解であり,

$$I_{\pm\nu}(z)=e^{\mp\sqrt{-1}\pi/2}J_{\pm\nu}(\sqrt{-1}z)$$

で定義される.

(7) **使用例**

(a) 問 題
$I_1(x)$ の値を $x=0.0,0.1,0.2,\cdots,0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibi1x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibi1x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibi1x(nv,xt, xo);

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

    printf( "\n\tValue of I1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibi1x ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
```

```
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of I1(x)

  xo =        0
  xo =   0.0501
  xo =    0.101
  xo =    0.152
  xo =    0.204
  xo =    0.258
  xo =    0.314
  xo =    0.372
  xo =    0.433
  xo =    0.497
```

2.4.4 ASL_wibk1x, ASL_vibk1x
第 2 種 1 次変形ベッセル関数

(1) **機 能**

$x = X_i$ に対して 1 次の第 2 種変形ベッセル関数

$$K_1(x) = \int_0^\infty e^{-x\cosh(t)}\cosh(t)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibk1x (nv, xi, xo);

単精度関数:

ierr = ASL_vibk1x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$K_1(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] ≥ 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	xi[i − 1] $> M$ であった. (注意事項 (a) 参照) (underflow)	xo[i − 1] = 0.0 を返す.
2000	xi[i − 1] $\leq$ 1.0/(最大値) (overflow)	xo[i − 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) この関数で ierr=1000 となるときの M の値は以下の通りである.
M ={ 倍精度: 705.117, 単精度: 85.114}

(b) 第 2 種 ν 次変形ベッセル関数 $K_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2}+z\frac{dw}{dz}-(z^2+\nu^2)w=0$$

の特殊解であり,

$$K_\nu(z)=\frac{\pi}{2}\frac{I_{-\nu}(z)-I_\nu(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$K_n(z)=\lim_{\nu\to n}K_\nu(z)$$

とする.

(7) **使用例**

(a) 問 題
$K_1(x)$ の値を $x=0.1,0.2,\cdots,1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibk1x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibk1x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibk1x(nv,xt, xo);

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

    printf( "\n\tValue of K1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wibk1x ***

     ** Input **
```

```
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of K1(x)

  xo =     9.85
  xo =     4.78
  xo =     3.06
  xo =     2.18
  xo =     1.66
  xo =      1.3
  xo =     1.05
  xo =    0.862
  xo =    0.717
  xo =    0.602
```

2.4.5 ASL_dibinx, ASL_ribinx
第 1 種整数次変形ベッセル関数

(1) **機 能**

整数次の第 1 種変形ベッセル関数

$$I_n(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(t)}\cos(nt)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dibinx  (n, xi, & xo);
```

単精度関数:

```
ierr = ASL_ribinx  (n, xi, & xo);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$I_n(x)$ の値
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) $|\mathrm{xi}| \le M$

ここで, M =\{ 倍精度: 713.067, 単精度: 90.978\}

(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	$\lvert n\rvert(\log_e\lvert\frac{n}{x}\rvert - M_1) > M_2$ であった. (注意事項 (c) 参照) (xi $\neq$ 0.0, n $\neq$ 0) (underflow)	xo = 0.0 を返す.
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) $I_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に $|\mathrm{n}| < 1000, |\mathrm{xi}| < 1000.0$ とすることが望ましい.

(b) $I_n(x), I_{n+1}(x), I_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い. ただし, n を増大させていくと不安定となるので, n が減少する方向へ適用する.
漸化式

$$I_{n-1}(x) = \frac{2n}{x} I_n(x) + I_{n+1}(x)$$

(c) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 1 種 ν 次変形ベッセル関数 $I_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0$$

の特殊解であり,

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z)$$

で定義される.

(7) 使用例

(a) 問 題
$I_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibinx */

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

int main()
{
    int nt;
    double xt;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    fscanf( fp, "%lf", &xt );
    printf( "\tn = %6d\t\txi = %8.3g\n", nt, xt );

    fclose( fp );

    ierr = ASL_dibinx(nt, xt, &xo);

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

    printf( "\n\tValue of In(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibinx ***
 ** Input **
n =       5       xi =       1.5
 ** Output **
ierr =       0
Value of In(x)
  xo =  0.00217
```

2.4.6 ASL_dibknx, ASL_ribknx
第 2 種整数次変形ベッセル関数

(1) **機 能**

整数次の第 2 種変形ベッセル関数

$$K_n(x) = \int_0^\infty e^{-x\cosh(t)}\cosh(nt)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibknx (n, xi, & xo);

単精度関数:

ierr = ASL_ribknx (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$K_n(x)$ の値
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) xi ≥ 0.0

(b) xi $\leq M$

ここで, M =\{ 倍精度: 705.117, 単精度: 85.114\}

(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.	
2000	xi $\leq 2.0/$ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$) (overflow)	xo =(最大値) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $K_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に |n| < 1000, xi < 1000.0 とすることが望ましい.

(b) $K_n(x), K_{n+1}(x), K_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
漸化式

$$K_{n+1}(x) = \frac{2n}{x}K_n(x) + K_{n-1}(x)$$

(c) この関数で ierr=2000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 2 種 ν 次変形ベッセル関数 $K_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2+\nu^2)w = 0$$

の特殊解であり,

$$K_\nu(z) = \frac{\pi}{2}\frac{I_{-\nu}(z) - I_\nu(z)}{\sin\nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$K_n(z) = \lim_{\nu\to n} K_\nu(z)$$

とする.

(7) **使用例**

(a) 問 題
$K_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibknx */

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

int main()
{
    int nt;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", nt, xi );

    fclose( fp );

    ierr = ASL_dibknx(nt, xi, &xo);

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

    printf( "\n\tValue of Kn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibknx ***
 ** Input **
n =      5      xi =      1.5
 ** Output **
ierr =      0
Value of Kn(x)
  xo =     44.1
```

2.4.7 ASL_dibimx, ASL_ribimx
第 1 種実数次変形ベッセル関数

(1) **機 能**

整数次の第 1 種変形ベッセル関数

$$I_\nu(x) = \frac{1}{\pi}\int_0^\pi e^{x\cos(t)}\cos(\nu t)dt - \frac{\sin(\pi\nu)}{\pi}\int_0^\infty e^{-x\cosh(t)-\nu t}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibimx (r, xi, & xo);

単精度関数:

ierr = ASL_ribimx (r, xi, & xo);

(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}	1	入 力	次数 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$I_\nu(x)$ の値
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) r が整数値と一致するとき

$|\mathrm{r}| \le M_1$

$|\mathrm{xi}| \le M_2$

(b) r が整数値と一致しないとき

$0 < \mathrm{r} \le M_1$

$0 < \mathrm{xi} \le M_2$

ここで, M_1 ={ 倍精度: 2^{31}, 単精度: 2^{31}},

M_2 ={ 倍精度: 713.067, 単精度: 90.978}

(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	$\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ であった. (注意事項 (e) 参照) (xi $\neq 0.0$, r $\neq 0.0$) (underflow) **注** ν が整数値と一致するときは $\lvert\nu\rvert$, $\lvert x\rvert$ に対して判定	xo $= 0.0$ を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $I_\nu(x)$ は, x や ν が大きいほど計算時間がかかる. 一般に r < 1000.0, xi < 1000.0 とすることが望ましい.

(b) 半奇数次の場合は, 球ベッセル関数を利用する方が良い.

$$I_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} i_n(x)$$

(c) $\nu < 0$ で ν が整数値と一致しないときは, この関数では計算できない. したがって, 漸化式を利用して求める.

(d) $I_\nu(x), I_{\nu+1}(x), I_{\nu+2}(x) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方がこの関数を幾度も使用するよりも速い.
ただし, ν を増大させていくと不安定となるので, ν が減少する方向へ適用する.
漸化式

$$I_{\nu-1}(x) = \frac{2\nu}{x} I_\nu(x) + I_{\nu+1}(x)$$

(e) この関数で ierr=1000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(f) 第 1 種 ν 次変形ベッセル関数 $I_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0$$

の特殊解であり,

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z)$$

で定義される.

(7) 使用例

(a) 問 題

$I_\nu(x)$ の値を $\nu = 3.3$, $x = 1.5$ について求める.

(b) 入力データ

r = 3.3, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibimx */

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

int main()
{
    double r;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &r );
    fscanf( fp, "%lf", &xi );
    printf( "\tr = %8.3g\t\txi = %8.3g\n", r, xi );

    fclose( fp );

    ierr = ASL_dibimx(r, xi, &xo);

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

    printf( "\n\tValue of Im(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibimx ***

 ** Input **

r =      3.3        xi =      1.5

 ** Output **

ierr =      0

Value of Im(x)

  xo =   0.0497
```

2.4.8 ASL_dibkmx, ASL_ribkmx
第 2 種実数次変形ベッセル関数

(1) **機 能**

実数次の第 2 種変形ベッセル関数

$$K_\nu(x) = \int_0^\infty e^{-x\cosh(t)} \cosh(\nu t) dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibkmx (r, xi, & xo);

単精度関数:

ierr = ASL_ribkmx (r, xi, & xo);

(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}	1	入 力	次数 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$K_\nu(x)$ の値
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) $|\mathrm{r}| \leq M_1$
ここで, M_1={ 倍精度: 2^{31}, 単精度: 2^{31}}

(b) xi ≥ 0.0

(c) xi $\leq M_2$
ここで, M_2 ={ 倍精度: 705.117, 単精度: 85.114}

(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.	
2000	xi $\leq$ 2.0/ (最大値) または $\nu(\log_e \frac{\nu}{x} - M_3) > M_4$ であった. (注意事項 (d) 参照) (xi $\neq$ 0.0, r $\neq$ 0.0) (overflow)	xo =(最大値) を返す.
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) $K_\nu(x)$ は, x または ν が大きいほど計算時間がかかる. 一般に r $< 1000.0,$ xi < 1000.0 とすることが望ましい.

(b) 半奇数次の場合は, 球ベッセル関数を利用する.

$$K_{n+\frac{1}{2}}(x) = \sqrt{\frac{2x}{\pi}} k_n(x)$$

(c) $K_\nu(x), K_{\nu+1}(x), K_{\nu+2}(x) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を反復使用するよりも速い.
漸化式

$$K_{\nu+1}(x) = \frac{2\nu}{x} K_\nu(x) + K_{\nu-1}(x)$$

(d) この関数で ierr=2000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(e) 第 2 種 ν 次変形ベッセル関数 $K_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2 \frac{d^2 w}{dz^2} + z \frac{dw}{dz} - (z^2 + \nu^2) w = 0$$

の特殊解であり,

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$K_n(z) = \lim_{\nu \to n} K_\nu(z)$$

とする.

(7) 使用例

(a) 問 題

$K_\nu(x)$ の値を $\nu = 3.3$, $x = 1.5$ について求める.

(b) 入力データ

r = 3.3, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibkmx */

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

int main()
{
    double r;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &r );
    fscanf( fp, "%lf", &xi );
    printf( "\tr = %8.3g\t\txi = %8.3g\n", r, xi );

    fclose( fp );

    ierr = ASL_dibkmx(r, xi, &xo);

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

    printf( "\n\tValue of Km(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibkmx ***

 ** Input **

r =      3.3        xi =      1.5

 ** Output **

ierr =      0

Value of Km(x)

  xo =     2.76
```

2.4.9 ASL_zibinz, ASL_cibinz
複素変数第 1 種整数次変形ベッセル関数

(1) **機 能**

複素変数で整数次の第 1 種変形ベッセル関数

$$I_n(z) = \frac{1}{\pi}\int_0^{\pi} e^{z\cos(t)}\cos(nt)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibinz (n, & zi, & zo);

単精度関数:

ierr = ASL_cibinz (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$I_n(z)$ の値
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) $|\mathrm{zi}| \le M_1$
ここで, M_1={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(b) $|\Re(\mathrm{zi})| \le M_2$
ここで, M_2 ={ 倍精度: 709.7827, 単精度: 88.72284}

(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	$\lvert n\rvert(\log_e \frac{\lvert n\rvert}{\lvert z\rvert} - M_3) > M_4$ (注意事項 (c) 参照) ($\lvert\mathrm{zi}\rvert \ne 0.0$, $\mathrm{n} \ne 0$) (underflow)	zo = (0.0, 0.0) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) 注意事項

(a) $I_n(z)$ は, $|z|$ や n が大きいほど計算時間がかかる. 一般に $|\mathrm{n}| < 1000, |\mathrm{zi}| < 1000.0$ とすることが望ましい.

(b) $I_n(z), I_{n+1}(z), I_{n+2}(z)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
ただし, n を増大させていくと不安定になるので, n が減少する方向へ適用する.
漸化式

$$I_{n-1}(z) = \frac{2n}{z} I_n(z) + I_{n+1}(z)$$

(c) この関数で ierr=1000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 1 種 ν 次変形ベッセル関数 $I_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0$$

の特殊解であり,

$$I_{\pm\nu}(z) = e^{\mp\sqrt{-1}\pi/2} J_{\pm\nu}(\sqrt{-1}z)$$

で定義される.

(7) 使用例

(a) 問 題
$I_n(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
$\mathrm{n} = 3$, $\mathrm{zi} = (1.0, 2.0)$

(c) 主プログラム

```
/*      C interface example for ASL_zibinz */

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

int main()
{
    int nt;
    double _Complex zt;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zt = tmp_re + tmp_im * _Complex_I;
    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", nt, creal(zt), cimag(zt) );

    fclose( fp );

    ierr = ASL_zibinz(nt, &zt, &zo);

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

    printf( "\n\tValue of In(z)\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zibinz ***
 ** Input **
n =      3      zi = (       1,       2)
 ** Output **
ierr =      0
Value of In(z)
  zo = (  -0.175, -0.0824)
```

2.4.10 ASL_zibknz, ASL_cibknz
複素変数第 2 種整数次変形ベッセル関数

(1) **機 能**

複素変数で整数次の第 2 種変形ベッセル関数

$$K_n(z) = \int_0^\infty e^{-z\cosh(t)} \cosh(nt) dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibknz (n, & zi, & zo);

単精度関数:

ierr = ASL_cibknz (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$K_n(z)$ の値
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) $|\text{zi}| > 0.0$

(b) $|\text{zi}| \leq M_1$
ここで, M_1={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(c) $|\Re(\text{zi})| \leq M_2$
ここで, M_2 ={ 倍精度: 709.7827, 単精度: 88.72284}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	$\|\mathrm{zi}\| \leq 2.0/$(最大値) または $\|n\|(\log_e \frac{\|n\|}{\|z\|} - M_3) > M_4$ (注意事項 (c) 参照) ($\|\mathrm{zi}\| \neq 0.0$, $\mathrm{n} \neq 0$)	

(6) **注意事項**

(a) $K_n(z)$ は, $|z|$ や n が大きいほど計算時間がかかる. 一般に $|\mathrm{n}| < 1000$, $|\mathrm{zi}| < 1000.0$ とすることが望ましい.

(b) $K_n(z), K_{n+1}(z), K_{n+2}(z) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方がこの関数を幾度も使用するよりも速い.
漸化式

$$K_{n+1}(z) = \frac{2n}{z} K_n(z) + K_{n-1}(z)$$

(c) この関数で ierr=4000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 2 種 ν 次変形ベッセル関数 $K_\nu(z)$ は変形されたベッセルの微分方程式

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2 + \nu^2)w = 0$$

の特殊解であり,

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}$$

で定義される. ただし, ν が整数 n に等しい場合は,

$$K_n(z) = \lim_{\nu \to n} K_\nu(z)$$

とする.

(7) 使用例

(a) 問 題

$K_n(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ

n = 3, zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_zibknz */

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

int main()
{
    int nt;
    double _Complex zt;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nt );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zt = tmp_re + tmp_im * _Complex_I;
    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", nt, creal(zt), cimag(zt) );

    fclose( fp );

    ierr = ASL_zibknz(nt, &zt, &zo);

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

    printf( "\n\tValue of Kn(z)\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zibknz ***

 ** Input **

n =      3      zi = (       1,       2)

 ** Output **

ierr =      0

Value of Kn(z)

  zo = (  -0.681,   0.625)
```

2.5 球ベッセル関数

2.5.1 ASL_dibsjn, ASL_ribsjn
第 1 種整数次球ベッセル関数

(1) **機 能**

整数次の第 1 種球ベッセル関数

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+\frac{1}{2}}(x)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibsjn (n, xi, & xo);

単精度関数:

ierr = ASL_ribsjn (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	1	出 力	$j_n(x)$ の値
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) xi ≥ 0.0

(b) xi $\leq M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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	$\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ であった. (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$) (underflow or overflow)	n ≥ 0 のとき xo $= 0.0$ n < 0 のとき xo $= (-1)^{n+1}\times$(最大値).
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) 注意事項

(a) $j_n(x)$ は x や n が大きいほど計算時間がかかる. 一般に |n| < 1000, xi < 1000.0 とすることが望ましい.

(b) $j_n(x), j_{n+1}(x), j_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
ただし, n を増大させていくと不安定となるので, n が減少する方向へ適用する.
漸化式

$$j_{n-1}(x) = \frac{2n+1}{x} j_n(x) - j_{n+1}(x)$$

(c) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 1 種 n 次球ベッセル関数 $j_n(z)$ は微分方程式

$$z^2 \frac{d^2w}{dz^2} + 2z\frac{dw}{dz} + \{z^2 - n(n+1)\}w = 0 \quad (n = 0, \pm1, \pm2, \cdots)$$

の特殊解であり,

$$j_n(z) = \sqrt{\frac{\pi}{2z}} J_{n+\frac{1}{2}}(z)$$

で定義される.

(7) 使用例

(a) 問 題
$j_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibsjn */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibsjn(n, xi, &xo);

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

    printf( "\n\tValue of Spherical Jn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibsjn ***
 ** Input **
n =      5      xi =      1.5
 ** Output **
ierr =      0
Value of Spherical Jn(x)
  xo =  0.00067
```

2.5.2 ASL_dibsyn, ASL_ribsyn
第 2 種整数次球ベッセル関数

(1) **機 能**

整数次の第 2 種球ベッセル関数

$$y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+\frac{1}{2}}(x)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibsyn (n, xi, & xo);

単精度関数:

ierr = ASL_ribsyn (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	1	出 力	$y_n(x)$ の値
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) xi ≥ 0.0

(b) xi $\leq M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000	xi $\leq 2.0/$ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$) (overflow or underflow)	n ≥ 0 のとき xo =(最小値) n < 0 のとき xo $= 0.0$
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $y_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に $|\mathrm{n}| < 1000, \mathrm{xi} < 1000.0$ とすることが望ましい.

(b) $y_n(x), y_{n+1}(x), y_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
漸化式

$$y_{n+1}(x) = \frac{2n+1}{x} y_n(x) - y_{n-1}(x)$$

(c) この関数で ierr=2000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 2 種 n 次球ベッセル関数 $y_n(z)$ は微分方程式

$$z^2 \frac{d^2w}{dz^2} + 2z\frac{dw}{dz} + \{z^2 - n(n+1)\}w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

の特殊解であり,

$$y_n(z) = \sqrt{\frac{\pi}{2z}} Y_{n+\frac{1}{2}}(z)$$

で定義される.

(e) 球ノイマン関数 $n_n(z)$ は, 第 2 種球ベッセル関数 $y_n(z)$ と同じものである.

(7) **使用例**

(a) 問 題
$y_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
$\mathrm{n} = 5$, $\mathrm{xi} = 1.5$

(c) 主プログラム

```
/*      C interface example for ASL_dibsyn */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibsyn(n, xi, &xo);

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

    printf( "\n\tValue of Spherical Yn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibsyn ***
 ** Input **
n =      5      xi =      1.5
 ** Output **
ierr =      0
Value of Spherical Yn(x)
  xo =    -94.2
```

2.5.3 ASL_dibsin, ASL_ribsin
第 1 種整数次変形球ベッセル関数

(1) **機 能**

整数次の第 1 種変形球ベッセル関数

$$i_n(x) = \sqrt{\frac{\pi}{2x}} I_{n+\frac{1}{2}}(x)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibsin (n, xi, & xo);

単精度関数:

ierr = ASL_ribsin (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	1	出 力	$i_n(x)$ の値
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) n ≥ 0

(b) xi ≥ 0.0

(c) xi $\leq M$

ここで, M =\{ 倍精度: 709.7827, 単精度: 88.72284\}

(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	$n(\log_e \frac{n}{x} - M_1) > M_2$ であった. (注意事項 (d) 参照) (xi $\neq 0.0$, n $\neq 0$) (underflow)	xo = 0.0 を返す.
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) $i_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に $\mathrm{n} < 1000, \mathrm{xi} < 1000.0$ とすることが望ましい.

(b) $n < 0$ のときは, この関数では計算できない. 従って, 漸化式を利用して求める.

(c) $i_n(x), i_{n+1}(x), i_{n+2}(x) \cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
ただし, n を増大させていくと不安定になるので, n が減少する方向へ適用する.
漸化式

$$i_{n-1}(x) = \frac{2n+1}{x} i_n(x) + i_{n+1}(x)$$

(d) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(e) 第 1 種 n 次変形球ベッセル関数 $i_n(z)$ は微分方程式

$$z^2 \frac{d^2w}{dz^2} + 2z\frac{dw}{dz} - \{z^2 + n(n+1)\}w = 0 \quad (n = 0, \pm 1, \pm 2, \cdots)$$

の特殊解であり,

$$i_n(z) = \sqrt{\frac{\pi}{2z}} I_{n+\frac{1}{2}}(z)$$

で定義される.

(7) 使用例

(a) 問 題
$i_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
$\mathrm{n} = 5$, $\mathrm{xi} = 1.5$

(c) 主プログラム

```
/*      C interface example for ASL_dibsin */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibsin(n, xi, &xo);

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

    printf( "\n\tValue of Spherical In(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibsin ***
 ** Input **
n =      5      xi =      1.5
 ** Output **
ierr =      0
Value of Spherical In(x)
  xo = 0.000796
```

2.5.4 ASL_dibskn, ASL_ribskn
第 2 種整数次変形球ベッセル関数

(1) **機 能**

整数次の第 2 種変形球ベッセル関数

$$k_n(x) = \sqrt{\frac{\pi}{2x}} K_{n+\frac{1}{2}}(x)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibskn (n, xi, & xo);

単精度関数:

ierr = ASL_ribskn (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	1	出 力	$k_n(x)$ の値
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) xi ≥ 0.0

(b) xi $\leq M$

ここで, M =\{ 倍精度: 702.293, 単精度: 83.364\}

(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.	
2000	xi $\leq 2.0/$ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ (注意事項 (c) 参照) (xi $\neq 0.0$, n $\neq 0$) (overflow)	xo =(最大値) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) $k_n(x)$ は, x や n が大きいほど計算時間がかかる. 一般に |n| < 1000, xi < 1000.0 とすることが望ましい.

(b) $k_n(x), k_{n+1}(x), k_{n+2}(x)\cdots$ を同時に求める場合は, 漸化式を用いて順次求めた方が, この関数を幾度も使用するよりも速い.
漸化式

$$k_{n+1}(x) = \frac{2n+1}{x}k_n(x) + k_{n-1}(x)$$

(c) この関数で ierr=2000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(d) 第 2 種 n 次変形球ベッセル関数 $k_n(z)$ は微分方程式

$$z^2\frac{d^2w}{dz^2} + 2z\frac{dw}{dz} - \{z^2 + n(n+1)\}w = 0 \quad (n = 0, \pm1, \pm2, \cdots)$$

の特殊解であり,

$$k_n(z) = \sqrt{\frac{\pi}{2z}}K_{n+\frac{1}{2}}(z)$$

で定義される.

(7) **使用例**

(a) 問 題
$k_n(x)$ の値を $n = 5$, $x = 1.5$ について求める.

(b) 入力データ
n = 5, xi = 1.5

(c) 主プログラム

```
/*      C interface example for ASL_dibskn */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n,xi );

    fclose( fp );

    ierr = ASL_dibskn(n, xi, &xo);

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

    printf( "\n\tValue of Spherical Kn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibskn ***
 ** Input **
n =       5       xi =       1.5
 ** Output **
ierr =       0
Value of Spherical Kn(x)
  xo =       115
```

2.6 ベッセル関数に関連した関数

2.6.1 ASL_zibh1n, ASL_cibh1n
第 1 種ハンケル関数

(1) **機 能**

第 1 種整数次ハンケル関数

$$H_n^{(1)}(z) = -\frac{2\sqrt{-1}}{\pi} e^{-\sqrt{-1}n\pi/2} \int_0^\infty e^{\sqrt{-1}z\cosh(t)} \cosh(nt) dt \quad (0 < \arg z < \pi)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibh1n (n, & zi, & zo);

単精度関数:

ierr = ASL_cibh1n (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$H_n^{(1)}(z)$ の値
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) $|\text{zi}| > 0.0$

(b) $|\Im(\text{zi})| \leq M_1$
ここで, M_1 ={ 倍精度: 709.7827, 単精度: 88.72284}

(c) $|\text{zi}| \leq M_2$
ここで, M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	$\|zi\| \leq 2.0/$ (最大値) または $\|n\|(\log_e \frac{\|n\|}{\|z\|} - M_3) > M_4$ (注意事項 (a) 参照)	

(6) **注意事項**

(a) この関数で ierr=4000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) 第 1 種 ν 次ハンケル関数 $H_\nu^{(1)}(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の特殊解であり,

$$H_\nu^{(1)}(z) = -\frac{1}{\pi}\int_{L_1} e^{-\sqrt{-1}z\sin\tau + \sqrt{-1}\nu\tau}d\tau$$

で定義される. ここで, 積分路 L_1 は $(0,-\infty) \to (0,0) \to (-\pi,0) \to (-\pi,\infty)$ ととる.

(c) 第 1 種ならびに第 2 種ハンケル関数は第 3 種ベッセル関数 (第 3 種円柱関数) とも呼ばれる.

(7) **使用例**

(a) 問 題
$H_n^{(1)}(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
n = 3, zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_zibh1n */

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

int main()
{
    int n;
    double _Complex zi;
    double _Complex zo;
    int ierr;
    FILE *fp;

    fp = fopen( "zibh1n.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zibh1n ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zi = tmp_re + tmp_im * _Complex_I;
    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", n, creal(zi), cimag(zi) );

    fclose( fp );
```

```
        ierr = ASL_zibh1n(n, &zi, &zo);

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

        printf( "\n\tValue of H1n(z)\n\n" );
        printf( "\t  zo =(%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_zibh1n ***

     ** Input **

    n =      3      zi = (       1,       2)

     ** Output **

    ierr =      0

    Value of H1n(z)

      zo =( -0.0689,   0.307)
```

2.6.2 ASL_zibh2n, ASL_cibh2n
第 2 種ハンケル関数

(1) **機 能**

第 2 種整数次ハンケル関数

$$H_n^{(2)}(z) = \frac{2\sqrt{-1}}{\pi} e^{\sqrt{-1}n\pi/2} \int_0^\infty e^{-\sqrt{-1}z\cosh(t)} \cosh(nt) dt \quad (0 < \arg z < \pi)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_zibh2n (n, & zi, & zo);

単精度関数:

ierr = ASL_cibh2n (n, & zi, & zo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	zi	{ Z∗ / C∗ }	1	入 力	変数値 z
3	zo	{ Z∗ / C∗ }	1	出 力	$H_n^{(2)}(z)$ の値
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) $|\text{zi}| > 0.0$

(b) $|\Im(\text{zi})| \leq M_1$
ここで, M_1 ={ 倍精度: 709.7827, 単精度: 88.72284}

(c) $|\text{zi}| \leq M_2$
ここで, M_2={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	$\|\mathrm{zi}\| \leq 2.0/$ (最大値) または $\|n\|(\log_e \frac{\|n\|}{\|z\|} - M_3) > M_4$ (注意事項 (a) 参照)	

(6) **注意事項**

(a) この関数で ierr=4000 となるときの M_3, M_4 の値は以下の通りである.
$M_3 = 0.3068$, $M_4 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) 第 2 種 ν 次ハンケル関数 $H_\nu^{(2)}(z)$ はベッセルの微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0$$

の特殊解であり,

$$H_\nu^{(2)}(z) = -\frac{1}{\pi}\int_{L_2} e^{-\sqrt{-1}z\sin\tau + \sqrt{-1}\nu\tau}d\tau$$

で定義される. ここで, 積分路 L_2 は $(\pi, \infty) \to (\pi, 0) \to (0, 0) \to (0, -\infty)$ ととる.

(c) 第 1 種ならびに第 2 種ハンケル関数は第 3 種ベッセル関数 (第 3 種円柱関数) とも呼ばれる.

(7) **使用例**

(a) 問 題
$H_n^{(2)}(z)$ の値を $n = 3$, $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
n = 3, zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_zibh2n */

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

int main()
{
    int n;
    double _Complex zi;
    double _Complex zo;
    int ierr;
    FILE *fp;

    fp = fopen( "zibh2n.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_zibh2n ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zi = tmp_re + tmp_im * _Complex_I;
    printf( "\tn = %6d\t\tzi = (%8.3g,%8.3g)\n", n, creal(zi), cimag(zi) );

    fclose( fp );
```

```
    ierr = ASL_zibh2n(n, &zi, &zo);

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

    printf( "\n\tValue of H2n(z)\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zibh2n ***

 ** Input **

n =      3      zi = (       1,       2)

 ** Output **

ierr =      0

Value of H2n(z)

  zo = (  -0.493,  -0.273)
```

2.6.3 ASL_dibber, ASL_ribber
ケルビン関数 $\mathrm{ber}_n(x)$

(1) **機 能**

ケルビン関数

$$\mathrm{ber}_n(x) = \Re\{J_n(xe^{3\sqrt{-1}\pi/4})\}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibber (n, xi, & xo);

単精度関数:

ierr = ASL_ribber (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\mathrm{ber}_n(x)$ の値
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) $|\mathrm{xi}| \leq M$

ここで, M =\{ 倍精度: 1003.784, 単精度: 125.473\}

(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	$\lvert n\rvert(\log_e \frac{\lvert n\rvert}{\lvert x\rvert} - M_1) > M_2$ であった. (注意事項 (a) 参照) (underflow)	xo = 0.0 を返す.
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) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ は微分方程式

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0$$

の解である.

(7) 使用例

(a) 問 題
$\mathrm{ber}_n(x)$ の値を $n = 3$, $x = 1.0$ について求める.

(b) 入力データ
n = 3, xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_dibber */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibber(n, xi, &xo);

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

    printf( "\n\tValue of Bernx\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibber ***

 ** Input **

n =      3      xi =        1

 ** Output **

ierr =      0

Value of Bernx

  xo =   0.0138
```

2.6.4 ASL_dibbei, ASL_ribbei
ケルビン関数 $\mathrm{bei}_n(x)$

(1) **機 能**

ケルビン関数

$$\mathrm{bei}_n(x) = \Im\{J_n(xe^{3\sqrt{-1}\pi/4})\}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibbei (n, xi, & xo);

単精度関数:

ierr = ASL_ribbei (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\mathrm{bei}_n(x)$ の値
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) $|\mathrm{xi}| \leq M$

ここで, M =\{ 倍精度: 1003.784, 単精度: 125.473\}

(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	$\lvert n\rvert(\log_e \frac{\lvert n\rvert}{\lvert x\rvert} - M_1) > M_2$ であった. (注意事項 (a) 参照) (underflow)	xo = 0.0 を返す.
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) この関数で ierr=1000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ は微分方程式

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0$$

の解である.

(7) 使用例

(a) 問 題
$\mathrm{bei}_n(x)$ の値を $n = 3$, $x = 1.0$ について求める.

(b) 入力データ
n = 3, xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_dibbei */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibbei(n, xi, &xo);

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

    printf( "\n\tValue of Beinx\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibbei ***

 ** Input **

n =      3      xi =        1

 ** Output **

ierr =      0

Value of Beinx

  xo =   0.0156
```

2.6.5 ASL_dibker, ASL_ribker
ケルビン関数 $\ker_n(x)$

(1) **機 能**

ケルビン関数

$$\ker_n(x) = \Re\{e^{-\sqrt{-1}n\pi/2} K_n(xe^{\sqrt{-1}\pi/4})\}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibker (n, xi, & xo);

単精度関数:

ierr = ASL_ribker (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\ker_n(x)$ の値
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) $0.0 < \text{xi} \leq M$

ここで, M =\{ 倍精度: 1003.784, 単精度: 125.473\}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	xi $\leq$ 2.0/ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ であった. (注意事項 (a) 参照)	

(6) 注意事項

(a) この関数で ierr=4000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ は微分方程式

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0$$

の解である.

(7) 使用例

(a) 問 題
$\mathrm{ker}_n(x)$ の値を $n = 3$, $x = 1.0$ について求める.

(b) 入力データ
n = 3, xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_dibker */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibker(n, xi, &xo);

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

    printf( "\n\tValue of Kernx\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibker ***

 ** Input **

n =      3      xi =        1

 ** Output **

ierr =      0

Value of Kernx

  xo =     4.89
```

2.6.6 ASL_dibkei, ASL_ribkei
ケルビン関数 $\mathrm{kei}_n(x)$

(1) **機 能**

ケルビン関数

$$\mathrm{kei}_n(x) = \Im\{e^{-\sqrt{-1}n\pi/2} K_n(xe^{\sqrt{-1}\pi/4})\}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibkei (n, xi, & xo);

単精度関数:

ierr = ASL_ribkei (n, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\mathrm{kei}_n(x)$ の値
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) $0.0 < \mathrm{xi} \leq M$

ここで, M =\{ 倍精度: 1003.784, 単精度: 125.473\}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	xi $\leq$ 2.0/ (最大値) または $\lvert n\rvert(\log_e \frac{\lvert n\rvert}{x} - M_1) > M_2$ であった. (注意事項 (a) 参照)	

(6) **注意事項**

(a) この関数で ierr=4000 となるときの M_1, M_2 の値は以下の通りである.
$M_1 = 0.3068$, $M_2 =$\{ 倍精度: 709.7827, 単精度: 88.72284\}

(b) $w = \mathrm{ber}_\nu(x) + \sqrt{-1}\,\mathrm{bei}_\nu(x)$, $\mathrm{ber}_{-\nu}(x) + \sqrt{-1}\,\mathrm{bei}_{-\nu}(x)$, $\mathrm{ker}_\nu(x) + \sqrt{-1}\,\mathrm{kei}_\nu(x)$, $\mathrm{ker}_{-\nu}(x) + \sqrt{-1}\,\mathrm{kei}_{-\nu}(x)$ は微分方程式

$$x^2\frac{d^2w}{dx^2} + x\frac{dw}{dx} - (\sqrt{-1}x^2 + \nu^2)w = 0$$

の解である.

(7) **使用例**

(a) 問 題
$\mathrm{kei}_n(x)$ の値を $n = 3$, $x = 1.0$ について求める.

(b) 入力データ
n = 3, xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_dibkei */

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

int main()
{
    int n;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_dibkei(n, xi, &xo);

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

    printf( "\n\tValue of Keinx\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibkei ***

 ** Input **

n =      3      xi =        1

 ** Output **

ierr =      0

Value of Keinx

  xo =    -6.27
```

2.6.7 ASL_wibh0x, ASL_vibh0x 0 次ストルーブ関数

(1) **機 能**

$x = X_i$ に対して 0 次のストルーブ関数

$$\mathbf{H}_0(x) = \frac{2}{\pi}\int_0^{\frac{\pi}{2}} \sin(x\cos(t))dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibh0x (nv, xi, xo);

単精度関数:

ierr = ASL_vibh0x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D, R}	nv	入 力	変数値 X_i
3	xo	{D, R}	nv	出 力	$\mathbf{H}_0(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] $\leq M$ ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) 注意事項

(a) ν 次ストルーブ関数を $\mathbf{H}_\nu(z)$ とすると次の漸化式が成立する.

$$\mathbf{H}_{\nu-1}(z) + \mathbf{H}_{\nu+1}(z) = \frac{2\nu}{z}\,\mathbf{H}_\nu(z) + \frac{\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})}$$

(b) 微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2-\nu^2)w = \frac{4\left(\frac{z}{2}\right)^{\nu+1}}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}$$

の一般解は

$$w = aJ_\nu(z) + bY_\nu(z) + \mathbf{H}_\nu(z)$$

である. ここで, $J_\nu(z)$, $Y_\nu(z)$ は第 1 種ならびに第 2 種ベッセル関数, a, b は定数である.

(7) 使用例

(a) 問 題

$\mathbf{H}_0(x)$ の値を $x = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibh0x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibh0x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibh0x(nv,xt, xo);

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

    printf( "\n\tValue of H0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );
    free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibh0x ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
```

```
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of H0(x)

  xo =        0
  xo =   0.0636
  xo =    0.127
  xo =    0.189
  xo =     0.25
  xo =     0.31
  xo =    0.367
  xo =    0.422
  xo =    0.474
  xo =    0.523
```

2.6.8 ASL_wibh1x, ASL_vibh1x
1 次ストルーブ関数

(1) **機 能**

$x = X_i$ に対して 1 次のストルーブ関数

$$\mathbf{H}_1(x) = \frac{2x}{\pi} \int_0^{\frac{\pi}{2}} \sin(x\cos(t)) \sin^2(t) dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibh1x (nv, xi, xo);

単精度関数:

ierr = ASL_vibh1x (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$\mathbf{H}_1(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] $\leq M$

ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(5) **エラーインディケータ (戻り値)**

戻り値	意 味	処 理 内 容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) ν 次ストルーブ関数を $\mathbf{H}_\nu(z)$ とすると次の漸化式が成立する.

$$\mathbf{H}_{\nu-1}(z) + \mathbf{H}_{\nu+1}(z) = \frac{2\nu}{z}\,\mathbf{H}_\nu(z) + \frac{\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})}$$

(b) 微分方程式

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2-\nu^2)w = \frac{4\left(\frac{z}{2}\right)^{\nu+1}}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}$$

の一般解は

$$w = aJ_\nu(z) + bY_\nu(z) + \mathbf{H}_\nu(z)$$

である. ここで, $J_\nu(z)$, $Y_\nu(z)$ は第 1 種ならびに第 2 種ベッセル関数, a, b は定数である.

(7) **使用例**

(a) 問 題

$\mathbf{H}_1(x)$ の値を $x = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibh1x */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibh1x ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibh1x(nv,xt, xo);

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

    printf( "\n\tValue of H1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibh1x ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
```

```
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of H1(x)

  xo =        0
  xo =  0.00212
  xo =  0.00847
  xo =    0.019
  xo =   0.0336
  xo =   0.0522
  xo =   0.0746
  xo =    0.101
  xo =     0.13
  xo =    0.163
```

2.6.9 ASL_wibhy0 , ASL_vibhy0
0 次ストルーブ関数とベッセル関数の差

(1) **機 能**

$x = X_i$ に対して 0 次ストルーブ関数と第 2 種 0 次ベッセル関数の差

$$\mathbf{H}_0(x) - Y_0(x) = \frac{2}{\pi}\int_0^{\infty} e^{-xt}(1+t^2)^{-\frac{1}{2}}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibhy0 (nv, xi, xo);

単精度関数:

ierr = ASL_vibhy0 (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\mathbf{H}_0(X_i) - Y_0(X_i)$ の値
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) nv ≥ 1

(b) xi[i $-$ 1] ≥ 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.	
2000	xi[i $-$ 1] $= 0.0$ (overflow)	xo[i $-$ 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i $-$ 1] が制限条件 (b) を満足しなかった.	

(6) 注意事項

(a) 以下の関係が成立する.

$$\mathbf{H}_\nu(z) - Y_\nu(z) = \frac{2\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}\int_0^\infty e^{-zt}(1+t^2)^{\nu-\frac{1}{2}}dt \quad (|\arg z| < \frac{\pi}{2})$$

(7) 使用例

(a) 問 題

$\mathbf{H}_0(x) - Y_0(x)$ の値を $x = 0.1, 0.2, \cdots, 1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibhy0 */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibhy0 ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibhy0(nv,xt, xo);

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

    printf( "\n\tValue of HY0(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibhy0 ***

 ** Input **

  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of HY0(x)

  xo =      1.6
  xo =     1.21
  xo =    0.996
```

```
xo =    0.856
xo =    0.754
xo =    0.675
xo =    0.613
xo =    0.561
xo =    0.517
xo =     0.48
```

2.6.10 ASL_wibhy1, ASL_vibhy1
1 次ストルーブ関数とベッセル関数の差

(1) **機 能**

$x = X_i$ に対して 1 次ストルーブ関数と第 2 種 1 次ベッセル関数の差

$$\mathbf{H}_1(x) - Y_1(x) = \frac{2x}{\pi} \int_0^\infty e^{-xt}(1+t^2)^{\frac{1}{2}} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wibhy1 (nv, xi, xo);

単精度関数:

ierr = ASL_vibhy1 (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\mathbf{H}_1(X_i) - Y_1(X_i)$ の値
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) nv ≥ 1

(b) xi[i $-$ 1] ≥ 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.	
2000	xi[i $-$ 1] $\leq 1.0/$(最大値) (overflow)	xo[i $-$ 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i $-$ 1] が制限条件 (b) を満足しなかった.	

(6) 注意事項

(a) 以下の関係が成立する.

$$\mathbf{H}_\nu(z) - Y_\nu(z) = \frac{2\left(\frac{z}{2}\right)^\nu}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})}\int_0^\infty e^{-zt}(1+t^2)^{\nu-\frac{1}{2}}dt \quad (|\arg z| < \frac{\pi}{2})$$

(7) 使用例

(a) 問 題

$\mathbf{H}_1(x) - Y_1(x)$ の値を $x = 0.1, 0.2, \cdots, 1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wibhy1 */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wibhy1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wibhy1(nv,xt, xo);

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

    printf( "\n\tValue of HY1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wibhy1 ***

 ** Input **

  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of HY1(x)

  xo =     6.46
  xo =     3.33
  xo =     2.31
```

```
xo =     1.81
xo =     1.52
xo =     1.33
xo =      1.2
xo =     1.11
xo =     1.04
xo =     0.98
```

2.6.11 ASL_dibaix, ASL_ribaix
エアリ関数 Ai(x)

(1) **機 能**

エアリ関数

$$\mathrm{Ai}(x) = \begin{cases} \pi^{-1}\sqrt{\frac{x}{3}}K_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) & (x \geq 0.0) \\ \frac{1}{3}\sqrt{|x|}[J_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + J_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibaix (xi, & xo);

単精度関数:

ierr = ASL_ribaix (xi, & xo);

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	Ai(x) の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xi $\geq -M$

ここで, M={ 倍精度: $(3 \times 2^{49} \times \pi)^{2/3}$, 単精度: $(3 \times 2^{17} \times \pi)^{2/3}$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xi $> M_1$ であった. (注意事項 (a) 参照)	xo = 0.0 を返す.
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) この関数で ierr=1000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 103.8, 単精度: 25.3}

(b) 微分方程式

$$\frac{d^2w}{dz^2} - zw = 0$$

の 1 次独立な解の組は $\{\mathrm{Ai}(z), \mathrm{Bi}(z)\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{2\sqrt{-1}\pi}{3}})\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{-2\sqrt{-1}\pi}{3}})\}$ である.

(7) 使用例

(a) 問 題
$x = -5.3$ におけるエアリ関数 Ai(x) の値を求める.

(b) 入力データ
xi $= -5.3$

(c) 主プログラム

```
/*      C interface example for ASL_dibaix */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\tValue of Variable x = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_dibaix(xi, &xo);

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

    printf( "\n\tValue of Airy Function Ai(x) = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibaix ***

 ** Input **

Value of Variable x =     -5.3

 ** Output **

ierr =      0

Value of Airy Function Ai(x) =    0.183
```

2.6.12 ASL_dibbix, ASL_ribbix
エアリ関数 Bi(x)

(1) **機 能**

エアリ関数

$$\mathrm{Bi}(x) = \begin{cases} \sqrt{\frac{x}{3}}[I_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + I_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x \geq 0.0) \\ \sqrt{\frac{|x|}{3}}[J_{-\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) - J_{\frac{1}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibbix (xi, & xo);

単精度関数:

ierr = ASL_ribbix (xi, & xo);

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	Bi(x) の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xi $\geq -M$

ここで, M={ 倍精度: $(3 \times 2^{49} \times \pi)^{2/3}$, 単精度: $(3 \times 2^{17} \times \pi)^{2/3}$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000	xi $>$ M_1 であった. (注意事項 (a) 参照) (overflow)	xo =(最大値) を返す.
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) この関数で ierr=2000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 104.266, 単精度: 20.066}

(b) 微分方程式

$$\frac{d^2w}{dz^2} - zw = 0$$

の 1 次独立な解の組は $\{\mathrm{Ai}(z), \mathrm{Bi}(z)\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{2\sqrt{-1}\pi}{3}})\}$, $\{\mathrm{Ai}(z), \mathrm{Ai}(ze^{\frac{-2\sqrt{-1}\pi}{3}})\}$ である.

(7) 使用例

(a) 問 題
$x = -5.3$ におけるエアリ関数 Bi(x) の値を求める.

(b) 入力データ
xi $= -5.3$

(c) 主プログラム

```
/*      C interface example for ASL_dibbix */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\tValue of Variable x = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_dibbix(xi, &xo);

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

    printf( "\n\tValue of Airy Function Bi(x) = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibbix ***

 ** Input **

Value of Variable x =     -5.3

 ** Output **

ierr =      0

Value of Airy Function Bi(x) =   -0.324
```

2.6.13 ASL_dibaid, ASL_ribaid
エアリ関数の導関数 $\mathrm{Ai}'(x)$

(1) **機 能**

エアリ関数の導関数

$$\mathrm{Ai}'(x) = \begin{cases} -\frac{x}{\sqrt{3}\pi} K_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) & (x \geq 0.0) \\ \frac{x}{3}[J_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) - J_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases}$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dibaid  (xi, & xo);
```

単精度関数:

```
ierr = ASL_ribaid  (xi, & xo);
```

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	$\mathrm{Ai}'(x)$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{xi} \geq -M$

ここで, M={ 倍精度: $(3 \times 2^{49} \times \pi)^{2/3}$, 単精度: $(3 \times 2^{17} \times \pi)^{2/3}$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xi $>$ M_1 であった. (注意事項 (a) 参照) (underflow)	xo = 0.0 を返す.
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) この関数で ierr=1000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 104.1, 単精度: 25.7}

(7) 使用例

(a) 問 題
$x = -5.3$ におけるエアリ関数 $\mathrm{Ai}(x)$ の導関数 $\mathrm{Ai}'(x)$ の値を求める.

(b) 入力データ
xi $= -5.3$

(c) 主プログラム

```
/*      C interface example for ASL_dibaid */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\tValue of Variable x = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_dibaid(xi, &xo);

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

    printf( "\n\tValue of Derived Airy Function Ai'(x) = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibaid ***

 ** Input **

Value of Variable x =     -5.3

 ** Output **

ierr =      0

Value of Derived Airy Function Ai'(x) =    0.755
```

2.6.14 ASL_dibbid, ASL_ribbid
エアリ関数の導関数 $\mathrm{Bi}'(x)$

(1) **機 能**

エアリ関数の導関数

$$\mathrm{Bi}'(x) = \begin{cases} \frac{x}{\sqrt{3}}[I_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + I_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x \geq 0.0) \\ -\frac{x}{\sqrt{3}}[J_{-\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}}) + J_{\frac{2}{3}}(\frac{2}{3}|x|^{\frac{3}{2}})] & (x < 0.0) \end{cases}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dibbid (xi, & xo);

単精度関数:

ierr = ASL_ribbid (xi, & xo);

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	$\mathrm{Bi}'(x)$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xi $\geq -M$

ここで, M={ 倍精度: $(3 \times 2^{49} \times \pi)^{2/3}$, 単精度: $(3 \times 2^{17} \times \pi)^{2/3}$}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000	xi $>$ M_1 であった. (注意事項 (a) 参照) (overflow)	xo =(最大値) を返す.
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) この関数で ierr=2000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 104.266, 単精度: 20.066}

(7) **使用例**

(a) 問 題
$x = -5.3$ におけるエアリ関数 $\mathrm{Bi}(x)$ の導関数 $\mathrm{Bi}'(x)$ の値を求める.

(b) 入力データ
xi $= -5.3$

(c) 主プログラム

```
/*      C interface example for ASL_dibbid */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\tValue of Variable x = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_dibbid(xi, &xo);

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

    printf( "\n\tValue of Derived Airy Function Bi'(x) = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dibbid ***

 ** Input **

Value of Variable x =     -5.3

 ** Output **

ierr =      0

Value of Derived Airy Function Bi'(x) =    0.406
```

2.7 ガンマ関数

2.7.1 ASL_wigamx, ASL_vigamx 実変数ガンマ関数

(1) **機 能**

$x = X_i$ に対して実変数ガンマ関数

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wigamx (nv, xi, xo);

単精度関数:

ierr = ASL_vigamx (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\Gamma(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] は負の整数値または 0.0 と一致しないこと

(c) xi[i − 1] $\geq -M$
ここで, M ={ 倍精度: 170.4, 単精度: 34.0}

(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.	
2000	$\|$xi[i − 1]$\| \leq 1.0/$(最大値) または xi[i − 1] $> M_1$ であった. (注意事項 (c) 参照) (overflow)	(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) ierr = 3000 + i で制限条件 (b) を満足しないときは, xi[i − 1] が負の整数値に近い値でない限り, $\Gamma(X_i)$ の値は 0.0 に非常に近い値である.

(b) $x = -n$ $(n = 0, 1, 2, \cdots)$ はガンマ関数 $\Gamma(x)$ の, 有理型関数としての単純極で,

$$\lim_{x \to -n} \frac{1}{\Gamma(x)} = 0 \ (n = 0, 1, 2, \cdots)$$

であるので, ゼロや負の整数値に近い点でのガンマ関数の計算値について高い精度を期待することはできない.

(c) この関数で ierr=2000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 171.4, 単精度: 35.0}

(d) ガンマ関数 $\Gamma(z)$ は第 2 種 Euler 積分とも呼ばれる. また, $\Gamma(z+1) = z\Gamma(z) = z!$ であるので階乗関数と呼ばれることもある.

(7) **使用例**

(a) 問 題
$\Gamma(x)$ の値を $x = 0.1, 0.2, \cdots, 0.9, 1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wigamx */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wigamx ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
```

```
        printf( "\t  xt = %8.3g\n", xt[i] );
        }

        ierr = ASL_wigamx(nv,xt, xo);

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

        printf( "\n\tValue of gamx(x)\n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", xo[i] );

            free(xt);
            free(xo);
        return 0;
    }
```

(c) 出力結果

```
    *** ASL_wigamx ***

     ** Input **

      xt =      0.1
      xt =      0.2
      xt =      0.3
      xt =      0.4
      xt =      0.5
      xt =      0.6
      xt =      0.7
      xt =      0.8
      xt =      0.9
      xt =        1

     ** Output **

    ierr =      0

    Value of gamx(x)

      xo =     9.51
      xo =     4.59
      xo =     2.99
      xo =     2.22
      xo =     1.77
      xo =     1.49
      xo =      1.3
      xo =     1.16
      xo =     1.07
      xo =        1
```

2.7.2 ASL_wiglgx, ASL_viglgx
実変数対数ガンマ関数

(1) 機 能

$x = X_i$ に対して実変数対数ガンマ関数 $\log_e(\Gamma(x))$ の値を求める.

(2) 使用法

倍精度関数:

ierr = ASL_wiglgx (nv, xi, xo);

単精度関数:

ierr = ASL_viglgx (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$\log_e(\Gamma(X_i))$ の値
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) nv ≥ 1

(b) xi[i − 1] は負の整数値または 0.0 と一致しないこと

(c) xi[i − 1] $> -M$
ここで, M ={ 倍精度: 2^{50}, 単精度: 2^{18}}

(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	Γ(xi[i − 1]) が負であった.	|Γ(xi[i − 1])| に対して自然対数をとり値を返す.
2000	xi[i − 1] $> M_1$ であった. (注意事項 (b) 参照) (overflow)	(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) または (c) を満足しなかった.	

(6) 注意事項

(a) ガンマ関数値は一般に exp(xo[i−1]) として得られるが, ierr = 1000 のときは, ある i に対して − exp(xo[i−1]) として得られる.

(b) この関数で ierr=2000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 2.545×10^{305}, 単精度: 4.08×10^{36}}

(7) **使用例**

(a) 問 題
$\log_e(\Gamma(x))$ の値を $x = 0.1, 0.2, \cdots, 0.9, 1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiglgx */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiglgx ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiglgx(nv,xt, xo);

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

    printf( "\n\tValue of glgx(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiglgx ***

 ** Input **

  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9
  xt =        1

 ** Output **

ierr =      0

Value of glgx(x)

  xo =     2.25
  xo =     1.52
  xo =      1.1
  xo =    0.797
  xo =    0.572
  xo =    0.398
  xo =    0.261
  xo =    0.152
  xo =   0.0664
  xo =        0
```

2.7.3 ASL_digig1, ASL_rigig1
第 1 種不完全ガンマ関数

(1) **機 能**

第 1 種不完全ガンマ関数

$$\gamma(\nu, x) = \int_0^x e^{-t} t^{\nu-1} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_digig1  (v, xi, & xo);
```

単精度関数:

```
ierr = ASL_rigig1  (v, xi, & xo);
```

(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	v	{D / R}	1	入 力	変数値 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\gamma(\nu, x)$ の値
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) xi ≥ 0.0

(b) v ≥ 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	$\nu \log_e(x) < -M_1$ であった. (注意事項 (b) 参照) (underflow)	xo = 0.0 を返す.
2000	v $\leq$ 1.0/(最大値) (overflow)	xo =(最大値) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	xi > 1.0 で, 次のいずれかが成立した. • $\nu > (M_2 + x)/\log_e(x)$ • $x > x_m$ かつ $\nu > \nu_m$ (注意事項 (c) 参照)	処理を打ち切る. (注意事項 (a) 参照)

(6) **注意事項**

(a) ierr = 4000 のときは, $\gamma(\nu, x)$ の値はほぼ (最大値) である.

(b) この関数で ierr=1000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 708.396, 単精度: 87.336}

(c) この関数で ierr=4000 となるときの x_m, ν_m, M_2 の値は以下の通りである.
x_m ={ 倍精度: 171.0, 単精度: 35.0},
ν_m ={ 倍精度: 171.4, 単精度: 35.0},
M_2 ={ 倍精度: 709.782, 単精度: 88.722}

(7) **使用例**

(a) 問 題
$\gamma(\nu, x)$ の値を $\nu = 4.0$, $x = 3.0$ について求める.

(b) 入力データ
v = 4.0, xi = 3.0

(c) 主プログラム

```
/*      C interface example for ASL_digig1 */

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

int main()
{
    double v;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "digig1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_digig1 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%lf", &v );
    fscanf( fp, "%lf", &xi );
    printf( "\tv = %8.3g\t\txi = %8.3g\n", v, xi );

    fclose( fp );

    ierr = ASL_digig1(v, xi, &xo);
```

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

    printf( "\n\tValue of Incomplete Gamma Function of The First Kind\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_digig1 ***

 ** Input **

v =        4        xi =        3

 ** Output **

ierr =      0

Value of Incomplete Gamma Function of The First Kind

  xo =     2.12
```

2.7.4 ASL_digig2, ASL_rigig2
第 2 種不完全ガンマ関数

(1) **機 能**

第 2 種不完全ガンマ関数

$$\Gamma(\nu, x) = \int_x^\infty e^{-t} t^{\nu-1} dt = e^{-x} \int_0^\infty e^{-t} (x+t)^{\nu-1} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_digig2 (v, xi, & xo);

単精度関数:

ierr = ASL_rigig2 (v, xi, & xo);

(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	v	{D / R}	1	入 力	変数値 ν
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	1	出 力	$\Gamma(\nu, x)$ の値
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) xi ≥ 0.0

(b) $0.0 \leq$ v $\leq M$
ここで, M ={ 倍精度: 171.4, 単精度: 35.0}

(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	xi > v で $(\nu-1)\log_e(x)-x<-M_1$ であった. (注意事項 (b) 参照) (underflow)	xo=0.0 を返す.
2000	v = 0.0 かつ xi = 0.0 (overflow)	xo =(最大値) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る. (注意事項 (a) 参照)
4000	xi ≤ v − 0.5 のとき $\nu\log_e(x)>(M_2+x)$ または $x>x_m$ であった. (注意事項 (c) 参照)	処理を打ち切る.

(6) **注意事項**

(a) ierr = 3000 で ν が大きな値のときは, $\Gamma(\nu,x)$ の値はほぼ (最大値) である.

(b) この関数で ierr=1000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 708.396, 単精度: 87.336}

(c) この関数で ierr=4000 となるときの x_m, M_2 の値は以下の通りである.
x_m ={ 倍精度: 171.0, 単精度: 35.0},
M_2 ={ 倍精度: 709.7827, 単精度: 88.72284}

(7) **使用例**

(a) 問 題
$\Gamma(\nu,x)$ の値を $\nu=4.0$, $x=3.0$ について求める.

(b) 入力データ
v = 4.0, xi = 3.0

(c) 主プログラム

```
/*      C interface example for ASL_digig2 */

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

int main()
{
    double v;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "digig2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_digig2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%lf", &v );
    fscanf( fp, "%lf", &xi );
    printf( "\tv = %8.3g\t\txi = %8.3g\n", v,xi );

    fclose( fp );

    ierr = ASL_digig2(v, xi, &xo);

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

    printf( "\n\tValue of Incomplete Gamma Function of The Second Kind\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_digig2 ***
 ** Input **
v =        4        xi =        3
 ** Output **
ierr =      0
Value of Incomplete Gamma Function of The Second Kind
  xo =     3.88
```

2.7.5 ASL_zigamz, ASL_cigamz
複素変数ガンマ関数

(1) **機 能**

複素変数ガンマ関数

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt \quad (\Re\{z\} > 0)$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zigamz ( & zi, & zo);
```

単精度関数:

```
ierr = ASL_cigamz ( & zi, & zo);
```

(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	zi	{Z∗ / C∗}	1	入 力	変数値 z
2	zo	{Z∗ / C∗}	1	出 力	$\Gamma(z)$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) zi は負の整数値または 0.0 と一致しないでかつ |zi| > 1.0/(最大値) であること.

(b) $-M_1 \le \Re(\text{zi}) \le M_2$
ここで, M_1 ={ 倍精度: 170.4, 単精度: 34.0},
M_2 ={ 倍精度: 171.4, 単精度: 35.0}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る. (注意事項 (a) 参照)

(6) **注意事項**

(a) $\Re(\mathrm{zi}) < -M_1$ のときは, ierr = 3000 が出力されるが, $\Gamma(z)$ の値は zi が負の整数値に近い値でない限り 0.0 に非常に近い値である.
ここで, M_1 ={ 倍精度: 170.4, 単精度: 34.0}

(b) $z = -n \ \ (n = 0, 1, 2, \cdots)$ はガンマ関数 $\Gamma(z)$ の特異点すなわち

$$\lim_{z \to -n} \frac{1}{\Gamma(z)} = 0 \ (n = 0, 1, 2, \cdots)$$

であるので, ゼロや負の整数値に近い点でのガンマ関数の計算値について高い精度を期待することはできない.

(c) ガンマ関数 $\Gamma(z)$ は第 2 種 Euler 積分とも呼ばれる. また, $\Gamma(z+1) = z\Gamma(z) = z!$ であるので階乗関数と呼ばれることもある.

(7) **使用例**

(a) 問 題
$\Gamma(z)$ の値を $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ
$\mathrm{zi} = (1.0, 2.0)$

(c) 主プログラム

```
/*      C interface example for ASL_zigamz */

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

int main()
{
    double _Complex zi;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

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

    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zi = tmp_re + tmp_im * _Complex_I;
    printf( "\tzi = (%8.3g,%8.3g)\n", creal(zi), cimag(zi));

    fclose( fp );

    ierr = ASL_zigamz(&zi, &zo);

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

    printf( "\n\tValue of Gamma Function of Complex Variable\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zigamz ***

 ** Input **

zi = (       1,       2)

 ** Output **

ierr =      0

Value of Gamma Function of Complex Variable

  zo = (   0.152,  0.0198)
```

2.7.6 ASL_ziglgz, ASL_ciglgz
複素変数対数ガンマ関数

(1) **機 能**

複素変数対数ガンマ関数 $\log_e(\Gamma(z))$ の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_ziglgz  ( & zi, & zo);
```

単精度関数:

```
ierr = ASL_ciglgz  ( & zi, & zo);
```

(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	zi	{Z∗ / C∗}	1	入 力	変数値 z
2	zo	{Z∗ / C∗}	1	出 力	$\log_e(\Gamma(z))$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) zi は負の整数値または 0.0 と一致しないこと.

(b) $\Re(\mathrm{zi}) > -M_1$
ここで, M_1 ={ 倍精度: 2^{50}, 単精度: 2^{18}}

(c) $|\Im(\mathrm{zi})|, \Re(\mathrm{zi}) \leq M_2$
ここで, M_2 ={ 倍精度: 2.545×10^{305}, 単精度: 4.08×10^{36}}

(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	$\Gamma(z)$ が負の実数であった.	$\log_e(\lvert\Gamma(z)\rvert) + \pi\sqrt{-1}$ の値を返す.
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) ガンマ関数値は, exp(zo) として得られるが, ierr = 1000 のときは, exp($\Re$(zo)) としてガンマ関数値を求めたほうがよい.

(b) 複素変数対数ガンマ関数 $\log_e(\Gamma(z))$ は無限多価関数であり, それらは $2\pi\sqrt{-1}$ の整数倍の差を持つ. この関数では $-\pi < \Im\{\log_e(\Gamma(z))\} \leq \pi$ となるように定義した主値を計算する.

(7) **使用例**

(a) 問 題

$\log_e(\Gamma(z))$ の値を $z = 1 + 2\sqrt{-1}$ について求める.

(b) 入力データ

zi = (1.0, 2.0)

(c) 主プログラム

```
/*      C interface example for ASL_ziglgz */

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

int main()
{
    double _Complex zi;
    double _Complex zo;
    int ierr;
    FILE *fp;

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

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

    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    zi = tmp_re + tmp_im * _Complex_I;
    printf( "\tzi = (%8.3g,%8.3g)\n", creal(zi), cimag(zi) );

    fclose( fp );

    ierr = ASL_ziglgz(&zi, &zo);

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

    printf( "\n\tValue of Logarithmic Gamma Function of Complex Variable\n\n" );
    printf( "\t  zo = (%8.3g,%8.3g)\n", creal(zo), cimag(zo) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_ziglgz ***

 ** Input **

zi = (       1,       2)

 ** Output **

ierr =      0

Value of Logarithmic Gamma Function of Complex Variable

  zo = (   -1.88,    0.13)
```

2.8 ガンマ関数に関連した関数

2.8.1 ASL_wigdig, ASL_vigdig ディガンマ関数

(1) **機 能**

$x = X_i$ に対してディガンマ関数 (プシー関数)

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)} = \frac{d}{dx}\log_e(\Gamma(x))$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wigdig (nv, xi, xo);

単精度関数:

ierr = ASL_vigdig (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\psi(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] は負の整数値または 0.0 と一致しないこと.

(c) xi[i − 1] $> -M$
ここで, M ={ 倍精度: 2^{50}, 単精度: 2^{18}}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) または (c) を満足しなかった.	

(6) 注意事項

(a) ディガンマ関数 $\psi(x)$ は

$$\psi(x+1)-\psi(x)=\frac{1}{x},\quad \psi(1)=-\gamma,\quad \lim_{n\to\infty}\{\psi(x+n)-\psi(1+n)\}=0$$

の解である. ここで, γ は Euler の定数:

$$\gamma=\lim_{n\to\infty}\left(1+\frac{1}{2}+\cdots+\frac{1}{n}-\log_e n\right)$$

(b) ディガンマ関数 $\psi(x)$ の導関数 $\psi'(x)$, $\psi''(x)$, $\psi'''(x)$ はそれぞれ, トリガンマ関数, テトラガンマ関数, ペンタガンマ関数と呼ばる. また, これらは総称してポリガンマ関数と呼ばれる.

(7) 使用例

(a) 問 題

$\psi(x)$ の値を $x=0.1,0.2,\cdots,0.9,1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wigdig */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wigdig ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wigdig(nv,xt, xo);

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

    printf( "\n\tValue of gdig(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wigdig ***

     ** Input **

      xt =      0.1
      xt =      0.2
      xt =      0.3
      xt =      0.4
      xt =      0.5
      xt =      0.6
      xt =      0.7
      xt =      0.8
      xt =      0.9
```

```
  xt =        1

 ** Output **

ierr =      0

Value of gdig(x)

  xo =    -10.4
  xo =    -5.29
  xo =     -3.5
  xo =    -2.56
  xo =    -1.96
  xo =    -1.54
  xo =    -1.22
  xo =   -0.965
  xo =   -0.755
  xo =   -0.577
```

2.8.2 ASL_wigbet, ASL_vigbet
ベータ関数

(1) **機 能**

$p = X_i, q = Y_i$ に対してベータ関数

$$B(p,q) = \int_0^1 t^{p-1}(1-t)^{q-1}dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wigbet (nv, p, q, xo);

単精度関数:

ierr = ASL_vigbet (nv, p, q, xo);

(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	nv	I	1	入 力	入力データ数
2	p	{D∗ / R∗}	nv	入 力	変数値 X_i
3	q	{D∗ / R∗}	nv	入 力	変数値 Y_i
4	xo	{D∗ / R∗}	nv	出 力	$B(X_i, Y_i)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nv ≥ 1

(b) 2.0/(最大値)< p[i − 1] $\leq M$

(c) 2.0/(最大値)< q[i − 1] $\leq M$

ここで, M ={ 倍精度: 1.2×10^{305}, 単精度: 2.0×10^{36}}

(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.	
2000	p[i − 1] > 1000.0, または q[i − 1] > 1000.0	処理を終了するが, 得られた値の精度が低い.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	p[i − 1] および q[i − 1] が制限条件 (b) または (c) を満足しなかった.	

(6) **注意事項**

(a) ベータ関数 $B(p,q)$ は第 1 種 Euler 積分とも呼ばれる.

(b) ベータ関数 $B(p,q)$ はガンマ関数 $\Gamma(z)$ を用いて次のように表すことができる.

$$B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$$

(7) **使用例**

(a) 問 題

$B(p,q)$ の値を $p = 0.1, 0.2, \cdots, 0.9, 1.0$, $q = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wigbet */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *yt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    yt=(double *)malloc((size_t)(sizeof(double) * nv));
        if( yt == NULL )
        {
            printf("no enough memory for array yt\n");
                return -1;
        }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wigbet ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/nv;
    yt[i]=0.5;
    printf( "\t  xt = %8.3g\n", xt[i] );
    printf( "\t  yt = %8.3g\n", yt[i] );
    }

    ierr = ASL_wigbet(nv,xt,yt, xo);

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

    printf( "\n\tValue of gbet(x) \n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(yt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
    *** ASL_wigbet ***

     ** Input **

      xt =      0.1
      yt =      0.5
      xt =      0.2
      yt =      0.5
      xt =      0.3
      yt =      0.5
      xt =      0.4
```

```
  yt =      0.5
  xt =      0.5
  yt =      0.5
  xt =      0.6
  yt =      0.5
  xt =      0.7
  yt =      0.5
  xt =      0.8
  yt =      0.5
  xt =      0.9
  yt =      0.5
  xt =        1
  yt =      0.5

 ** Output **

ierr =      0

Value of gbet(x)

  xo =     11.3
  xo =     6.27
  xo =     4.55
  xo =     3.68
  xo =     3.14
  xo =     2.77
  xo =     2.51
  xo =      2.3
  xo =     2.13
  xo =        2
```

2.9 楕円関数と楕円積分

2.9.1 ASL_wieci1, ASL_vieci1
第 1 種完全楕円積分

(1) **機 能**

$m = X_i$ に対して第 1 種完全楕円積分

$$K(m) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}} = \int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1-m\sin^2\theta}} d\theta$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wieci1 (nv, rm, xo);

単精度関数:

ierr = ASL_vieci1 (nv, rm, xo);

(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	nv	I	1	入 力	入力データ数
2	rm	{ D∗ / R∗ }	nv	入 力	母数 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$K(X_i)$ の値
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) nv ≥ 1

(b) $0.0 \leq$ rm[i $-$ 1] ≤ 1.0

(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.	
2000	rm[i $-$ 1] = 1.0 (overflow)	xo[i $-$ 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	rm[i $-$ 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 第 1 種完全楕円積分が $K(k) = \displaystyle\int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1-k^2\sin^2\theta}} d\theta$ と表されているときは, $\mathrm{rm}[i-1]$ に k^2 の値を入力すること.

(b) $K(m)$, $E(m)$ を共に求めたい場合は 2.9.8 $\left\{\begin{matrix}\text{ASL_wienmq}\\ \text{ASL_vienmq}\end{matrix}\right\}$を利用した方が効率が良い.

(7) **使用例**

(a) 問 題

$K(m)$ の値を $m = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wieci1 */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wieci1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wieci1(nv,xt, xo);

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

    printf( "\n\tValue of eci1(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wieci1 ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of eci1(x)
```

```
xo =     1.57
xo =     1.61
xo =     1.66
xo =     1.71
xo =     1.78
xo =     1.85
xo =     1.95
xo =     2.08
xo =     2.26
xo =     2.58
```

2.9.2 ASL_wieci2, ASL_vieci2
第 2 種完全楕円積分

(1) **機 能**

$m = X_i$ に対して第 2 種完全楕円積分

$$E(m) = \int_0^1 \sqrt{(1-t^2)(1-mt^2)}dt = \int_0^{\frac{\pi}{2}} \sqrt{1-m\sin^2\theta}d\theta$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wieci2 (nv, rm, xo);

単精度関数:

ierr = ASL_vieci2 (nv, rm, xo);

(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	nv	I	1	入 力	入力データ数
2	rm	{D∗ / R∗}	nv	入 力	母数 X_i
3	xo	{D∗ / R∗}	nv	出 力	$E(X_i)$ の値
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) nv ≥ 1

(b) $0.0 \leq$ rm[i $-$ 1] ≤ 1.0

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	rm[i $-$ 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 第 2 種完全楕円積分が $E(k) = \int_0^{\frac{\pi}{2}} \sqrt{1 - k^2 \sin^2\theta} d\theta$ と表されているときは, rm[i − 1] に k^2 の値を入力すること.

(b) $E(m)$, $K(m)$ を共に求めたい場合は 2.9.8 $\left\{\begin{matrix}\text{ASL_wienmq} \\ \text{ASL_vienmq}\end{matrix}\right\}$を利用した方が効率が良い.

(7) **使用例**

(a) 問 題

$E(m)$ の値を $m = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wieci2 */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wieci2 ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wieci2(nv,xt, xo);

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

    printf( "\n\tValue of eci2(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wieci2 ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of eci2(x)
```

```
xo =     1.57
xo =     1.53
xo =     1.49
xo =     1.45
xo =      1.4
xo =     1.35
xo =      1.3
xo =     1.24
xo =     1.18
xo =      1.1
```

2.9.3 ASL_dieii1, ASL_rieii1
第 1 種不完全楕円積分

(1) **機 能**

第 1 種不完全楕円積分

$$F(x,m)=\int_0^x \frac{1}{\sqrt{(1-t^2)(1-mt^2)}}dt=\int_0^\psi \frac{1}{\sqrt{1-m\sin^2\theta}}d\theta \quad (\text{ただし, } x=\sin\psi)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dieii1 (xi, rm, & xo);

単精度関数:

ierr = ASL_rieii1 (xi, rm, & xo);

(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	xi	{D / R}	1	入 力	変数値 x
2	rm	{D / R}	1	入 力	母数 m
3	xo	{D∗ / R∗}	1	出 力	$F(x,m)$ の値
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) $0.0 \le \text{xi} \le 1.0$

(b) $0.0 \le \text{rm} \le 1.0$ (注意事項 (c) 参照)

(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.	
2000	xi = 1.0 かつ rm = 1.0 (overflow)	xo =(最大値) を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	ガウスの算術幾何平均法またはニュートン法が収束しなかった.	

(6) **注意事項**

(a) 第 1 種不完全楕円積分が $F(x,k)=\displaystyle\int_0^{\psi}\frac{1}{\sqrt{1-k^2\sin^2\theta}}d\theta$ と表されているときは, rm に k^2 の値を入力すること.

(b) 第 1 種不完全楕円積分は

$$F(\varphi\backslash\alpha)=F(\varphi|m)=\int_0^{\varphi}\frac{1}{\sqrt{1-\sin^2\alpha\sin^2\theta}}d\theta\quad(\text{ただし, } m=\sin^2\alpha)$$

と表される場合がある.

(c) 母数 $m<0.0$ のときは 2.9.5 $\left\{\begin{matrix}\text{ASL_dieii3}\\ \text{ASL_rieii3}\end{matrix}\right\}$を用いる.

(7) **使用例**

(a) 問 題

$F(x,m)$ の値を $x=0.3$, $m=0.5$ について求める.

(b) 入力データ

xi $=0.3$, rm $=0.5$

(c) 主プログラム

```
/*      C interface example for ASL_dieii1 */

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

int main()
{
    double xi;
    double rm;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "dieii1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dieii1 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%lf", &xi );
    fscanf( fp, "%lf", &rm );
    printf( "\txi = %8.3g\t\trm = %8.3g\n", xi, rm );

    fclose( fp );

    ierr = ASL_dieii1(xi, rm, &xo);

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

    printf( "\n\tValue of F(x,m)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dieii1 ***

 ** Input **

xi =      0.3       rm =      0.5

 ** Output **

ierr =      0

Value of F(x,m)

  xo =    0.307
```

2.9.4 ASL_dieii2, ASL_rieii2
第 2 種不完全楕円積分

(1) **機 能**

第 2 種不完全楕円積分

$$E(x,m)=\int_0^x\sqrt{\frac{1-mt^2}{1-t^2}}dt=\int_0^\psi\sqrt{1-m\sin^2\theta}d\theta\quad(\text{ただし},\ x=\sin\psi)$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dieii2  (xi, rm, & xo);
```

単精度関数:

```
ierr = ASL_rieii2  (xi, rm, & xo);
```

(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	xi	{D / R}	1	入 力	変数値 x
2	rm	{D / R}	1	入 力	母数 m
3	xo	{D∗ / R∗}	1	出 力	$E(x,m)$ の値
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) $0.0 \le$ xi ≤ 1.0

(b) $0.0 \le$ rm ≤ 1.0 (注意事項 (c) 参照)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	ガウスの算術幾何平均法またはニュートン法が収束しなかった.	

(6) **注意事項**

(a) 第 2 種不完全楕円積分が $E(x,k)=\int_0^{\psi}\sqrt{1-k^2\sin^2\theta}d\theta$ と表されているときは, rm に k^2 の値を入力すること.

(b) 第 2 種不完全楕円積分は

$$E(\varphi\backslash\alpha)=E(u|m)=\int_0^{\varphi}\sqrt{1-\sin^2\alpha\sin^2\theta}d\theta \quad (\text{ただし, } m=\sin^2\alpha,\quad \sin\varphi=\mathrm{sn}\,u)$$

と表される場合がある. $E(u|m)$ の値を u をパラメータとして計算したい場合には 2.9.11 $\left\{\begin{matrix}\text{ASL\_wiejep}\\ \text{ASL\_viejep}\end{matrix}\right\}$ を用いる.

(c) 母数 $m<0.0$ のときは 2.9.5 $\left\{\begin{matrix}\text{ASL\_dieii3}\\ \text{ASL\_rieii3}\end{matrix}\right\}$ を用いる.

(7) **使用例**

(a) 問 題

$E(x,m)$ の値を $x=0.3$, $m=0.5$ について求める.

(b) 入力データ

xi = 0.3, rm = 0.5

(c) 主プログラム

```
/*      C interface example for ASL_dieii2 */

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

int main()
{
    double xi;
    double rm;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "dieii2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dieii2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%lf", &xi );
    fscanf( fp, "%lf", &rm );
    printf( "\txi = %8.3g\t\trm = %8.3g\n", xi, rm );

    fclose( fp );

    ierr = ASL_dieii2(xi, rm, &xo);

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

    printf( "\n\tValue of E(x,m)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dieii2 ***

 ** Input **

xi =      0.3       rm =      0.5

 ** Output **

ierr =      0

Value of E(x,m)

  xo =    0.302
```

2.9.5 ASL_dieii3, ASL_rieii3
不完全変形楕円積分

(1) **機 能**

実数 $m \geq 0, a, b$ と $x \geq 0$ に対して, 不完全変形楕円積分

$$f(x, m, a, b) \equiv \int_0^x \frac{a + bt^2}{\sqrt{(1+t^2)^3\ (1+mt^2)}} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dieii3 (x, dm, a, b, & y);

単精度関数:

ierr = ASL_rieii3 (x, dm, a, b, & y);

(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	x	{D / R}	1	入 力	変数値 x
2	dm	{D / R}	1	入 力	母数値 m
3	a	{D / R}	1	入 力	$a + bt^2$ の係数 a
4	b	{D / R}	1	入 力	$a + bt^2$ の係数 b
5	y	{D∗ / R∗}	1	出 力	不完全変形楕円積分 $f(x, m, a, b)$ の値
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) x ≥ 0

(b) dm ≥ 0

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) 不完全楕円積分

$$F(r,m)=\int_0^r \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}}, E(r,m)=\int_0^r \sqrt{\frac{1-mt^2}{1-t^2}}dt$$

と, 因数 $1+mt^2$ が違い, 制限 $m<1$ がない.

(b) 第 1 種不完全楕円積分 $\int_0^\psi \sqrt{(1-a\sin^2\theta)^{-1}}d\theta$ は $f(\tan\psi, 1-a, 1, 1)$ である. ここで $0\le a<1$ であるが, 第 1 種不完全楕円積分を $a<1$ に拡張して定義できる.

(c) 第 2 種不完全楕円積分 $\int_0^\psi \sqrt{1-a\sin^2\theta}d\theta$ は $f(\tan\psi, 1-a, 1, 1-a)$ である. ここで $0\le a<1$ であるが, 第 2 種不完全楕円積分を $a<1$ に拡張して定義できる.

(7) **使用例**

(a) 問 題

$x=1.0,\ m=3.0,\ a=4.0,\ b=2.0$ に対して不完全変形楕円積分の値を求める.

(b) 入力データ

x=1.0, dm=3.0, a=4.0, b=2.0

(c) 主プログラム

```
/*       C interface example for ASL_dieii3 */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int ierr;
    double x,dm,a,b,y;
    x=1.0,dm=3.0,a=4.0,b=2.0;
    printf( "    *** ASL_dieii3 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\n\tx = %8.3g\n",x );
    printf( "\n\tdm= %8.3g\n",dm);
    printf( "\n\ta = %8.3g\n",a );
    printf( "\n\tb = %8.3g\n",b );
    ierr = ASL_dieii3(x, dm, a, b,&y);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\ty= %8.3g\n",y );
    return 0;
}
```

(d) 出力結果

```
*** ASL_dieii3 ***
 ** Input **

x =        1
dm=        3
a =        4
b =        2
 ** Output **
ierr =      0
y=     2.51
```

2.9.6 ASL_dieii4, ASL_rieii4
ワイエルシュトラス型の不完全楕円積分

(1) **機 能**

正数 x, y, z と実数 $p \geq 0$ に対して, ワイエルシュトラス型の不完全楕円積分

$$f(x,y,z,p) \equiv \frac{1}{2}\int_0^{1/p} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dieii4  (x, y, z, p, & di);
```

単精度関数:

```
ierr = ASL_rieii4  (x, y, z, p, & di);
```

(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	x	{D / R}	1	入 力	変数値 x
2	y	{D / R}	1	入 力	変数値 y
3	z	{D / R}	1	入 力	変数値 z
4	p	{D / R}	1	入 力	上端値の逆数 p (注意事項 (a) 参照)
5	di	{D∗ / R∗}	1	出 力	不完全楕円積分 $f(x,y,z,p)$ の値 (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) x, y, z > 0.0

(b) p ≥ 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.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) 変数 p を 0.0 にするとワイエルシュトラス型の完全楕円積分

$$\frac{1}{2}\int_0^{\infty} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$$

が di に代入される.

(b) $f(x,y,z,p)$ を定義する定積分の上端が $1/p$ であることに注意されたい (前項参照).

(c) $f(x,y,z,p)$ は, $x,\ y,\ z$ について対称である.

(d) $h \geq 0$ のとき,

$$f(x+h,y+h,z+h,p) = \frac{1}{2}\int_h^{1/p+h} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$$

である.

(7) **使用例**

(a) 問 題

$x = 1.0,\ y = 3.0,\ z = 4.0,\ p = 2.0$ に対してワイエルシュトラス型の不完全楕円積分の値を求める.

(b) 入力データ

x=1.0, dm=3.0, a=4.0, b=2.0

(c) 主プログラム

```
/*       C interface example for ASL_dieii4 */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int ierr;
    double x, y, z, p, di;
    x=1.0;y=3.0;z=4.0;p=2.0;
    printf( "    *** ASL_dieii4 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\n\tx = %8.3g\n",x );
    printf( "\n\ty = %8.3g\n",y );
    printf( "\n\tz = %8.3g\n",z );
    printf( "\n\tp = %8.3g\n",p );
    ierr = ASL_dieii4(x, y, z, p,&di);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tdi = %8.3g\n",di );
    return 0;
}
```

(d) 出力結果

```
*** ASL_dieii4 ***
 ** Input **

x =        1

y =        3

z =        4

p =        2
 ** Output **

ierr =      0

di =   0.0607
```

2.9.7 ASL_wiejac, ASL_viejac
ヤコビの楕円関数

(1) 機 能

$u = X_i$ に対してヤコビの楕円関数 $\mathrm{sn}(u,m)$, $\mathrm{cn}(u,m)$, $\mathrm{dn}(u,m)$ の値を求める.
ここで, u, m は $u = F(x,m)$: 第 1 種不完全楕円積分とした場合,
$\mathrm{sn}(u,m) = \sin\psi = x$, $\mathrm{cn}(u,m) = \cos\psi$, $\mathrm{dn}(u,m) = \sqrt{1 - m\sin^2\psi}$ となるように定める.

(2) 使用法

倍精度関数:

```
ierr = ASL_wiejac  (nv, ui, rm, sn, cn, dn);
```

単精度関数:

```
ierr = ASL_viejac  (nv, ui, rm, sn, cn, dn);
```

(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	nv	I	1	入 力	入力データ数
2	ui	{D∗ / R∗}	nv	入 力	変数値 X_i
3	rm	{D / R}	1	入 力	母数 m
4	sn	{D∗ / R∗}	nv	出 力	$\mathrm{sn}(X_i,m)$ の関数値
5	cn	{D∗ / R∗}	nv	出 力	$\mathrm{cn}(X_i,m)$ の関数値
6	dn	{D∗ / R∗}	nv	出 力	$\mathrm{dn}(X_i,m)$ の関数値
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) $\mathrm{nv} \geq 1$

(b) $0.0 \leq \mathrm{rm} \leq 1.0$

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3001	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) この関数は第 1 種不完全楕円積分 u を $u = \int_0^{\psi} \frac{d\theta}{\sqrt{1 - m\sin^2\theta}}$ と表したときに $\mathrm{sn}(u, m) = \sin\psi$ となるように $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$, $\mathrm{dn}(u, m)$ を求めるが,
$u = \int_0^{\psi} \frac{d\theta}{\sqrt{1 - k^2\sin^2\theta}}$ と表したときに $\mathrm{sn}(u, k) = \sin\psi$ となるように $\mathrm{sn}(u, k)$, $\mathrm{cn}(u, k)$, $\mathrm{dn}(u, k)$ を求めるときは, rm に k^2 の値を入力すること. 一般には $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$, $\mathrm{dn}(u, m)$ はそれぞれ $\mathrm{sn}(u|m)$, $\mathrm{cn}(u|m)$, $\mathrm{dn}(u|m)$ と表される.

(7) **使用例**

(a) 問 題

m=0.5 として $\mathrm{sn}(u, m)$, $\mathrm{cn}(u, m)$, $\mathrm{dn}(u, m)$ の値を $u = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiejac */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *sn;
    double *cn;
    double *dn;
    double rm;
    int nv;
    int  i;
    int ierr;
    nv=10;
    rm=0.5;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    sn=(double *)malloc((size_t)(sizeof(double) * nv));
    if( sn == NULL )
    {
        printf("no enough memory for array \n");
        return -1;
    }
        cn=(double *)malloc((size_t)(sizeof(double) * nv));
        if( cn == NULL )
        {
            printf("no enough memory for array \n");
                return -1;
        }
        dn=(double *)malloc((size_t)(sizeof(double) * nv));
        if( dn == NULL )
        {
            printf("no enough memory for array \n");
                return -1;
        }

    printf( "    *** ASL_wiejac ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiejac(nv,xt, rm, sn, cn, dn);

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

    printf( "\n\tValue of sn \n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  sn = %8.3g\n", sn[i] );
        printf( "\n\tValue of cn \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  cn = %8.3g\n", cn[i] );
        printf( "\n\tValue of dn \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  dn = %8.3g\n", dn[i] );

        free(xt);
        free(sn);
```

```
        free(cn);
        free(dn);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiejac ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of sn

  sn = -3.79e-16
  sn =   0.0998
  sn =    0.198
  sn =    0.293
  sn =    0.385
  sn =    0.471
  sn =    0.551
  sn =    0.624
  sn =    0.691
  sn =     0.75

Value of cn

  cn =        1
  cn =    0.995
  cn =     0.98
  cn =    0.956
  cn =    0.923
  cn =    0.882
  cn =    0.835
  cn =    0.781
  cn =    0.723
  cn =    0.661

Value of dn

  dn =        1
  dn =    0.998
  dn =     0.99
  dn =    0.978
  dn =    0.962
  dn =    0.943
  dn =    0.921
  dn =    0.897
  dn =    0.873
  dn =    0.848
```

2.9.8 ASL_wienmq, ASL_vienmq
ノーム q および完全楕円積分

(1) **機 能**

$m = X_i$ に対して,

ノーム $q = e^{-\pi K(m')/K(m)}$,

補ノーム $q' = e^{-\pi K(m)/K(m')}$ (ただし, $m' = 1 - m$),

第 1 種完全楕円積分 $K(m)$, $K(m')$ および

第 2 種完全楕円積分 $E(m)$, $E(m')$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wienmq (nv, rm, q, qd, k, kd, e, ed);

単精度関数:

ierr = ASL_vienmq (nv, rm, q, qd, k, kd, e, ed);

(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	nv	I	1	入 力	入力データ数
2	rm	{ D∗ / R∗ }	nv	入 力	母数 $m = X_i$
3	q	{ D∗ / R∗ }	nv	出 力	ノーム q の値
4	qd	{ D∗ / R∗ }	nv	出 力	補ノーム q' の値
5	k	{ D∗ / R∗ }	nv	出 力	第 1 種完全楕円積分 $K(m)$ の値
6	kd	{ D∗ / R∗ }	nv	出 力	第 1 種完全楕円積分 $K(m')$ の値
7	e	{ D∗ / R∗ }	nv	出 力	第 2 種完全楕円積分 $E(m)$ の値
8	ed	{ D∗ / R∗ }	nv	出 力	第 2 種完全楕円積分 $E(m')$ の値
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) nv ≥ 1

(b) $0.0 \leq$ rm[i $- 1] \leq 1.0$

(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.	
2000	rm[i − 1] = 0.0 または 1.0 (overflow)	rm[i − 1] = 0.0 のとき kd[i − 1] =(最大値) を返す. rm[i − 1] = 1.0 のとき k[i − 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	rm[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) 第 1 種完全楕円積分 $K(m)$ または第 2 種完全楕円積分 $E(m)$ の値のみを求めればよい場合は
2.9.1 $\left\{\begin{matrix}\text{ASL\_wieci1}\\ \text{ASL\_vieci1}\end{matrix}\right\}$, 2.9.2 $\left\{\begin{matrix}\text{ASL_wieci2}\\ \text{ASL_vieci2}\end{matrix}\right\}$を利用した方が効率がよい.

(7) **使用例**

(a) 問 題
ノーム q, 補ノーム q', 完全楕円積分 $K(m)$, $E(m)$,
$m' = 1 - m$ に対する完全楕円積分 $K(m')$, $E(m')$ を母数 $m = 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wienmq */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *q ;
    double *qd;
    double *k ;
    double *kd;
    double *e ;
    double *ed;
    int nv;
    int  i;
    int ierr;
    nv=9;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    q =(double *)malloc((size_t)(sizeof(double) * nv));
    if( q  == NULL )
    {
        printf("no enough memory for array q  \n");
        return -1;
    }
        qd =(double *)malloc((size_t)(sizeof(double) * nv));
        if( qd  == NULL )
        {
            printf("no enough memory for array qd  \n");
                return -1;
        }
        k  =(double *)malloc((size_t)(sizeof(double) * nv));
        if( k   == NULL )
        {
            printf("no enough memory for array k   \n");
                return -1;
        }
        kd  =(double *)malloc((size_t)(sizeof(double) * nv));
        if( kd   == NULL )
        {
            printf("no enough memory for array kd   \n");
                return -1;
        }
        e  =(double *)malloc((size_t)(sizeof(double) * nv));
```

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

    printf( "    *** ASL_wienmq ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
    xt[i]=xt[i]/(nv+1);
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wienmq(nv,xt, q,qd,k,kd,e,ed);

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

    printf( "\n\tValue of q  \n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", q[i] );
        printf( "\n\tValue of qd  \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", qd[i] );

        printf( "\n\tValue of k  \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", k[i] );
        printf( "\n\tValue of kd  \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", kd[i] );

        printf( "\n\tValue of e  \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", e[i] );
        printf( "\n\tValue of ed  \n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", ed[i] );

        free(xt);
        free(q);
        free(qd);
        free(k);
        free(kd);
        free(e);
        free(ed);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wienmq ***

 ** Input **

  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of q

  xo =  0.00658
  xo =   0.0139
  xo =   0.0223
  xo =   0.0319
  xo =   0.0432
  xo =    0.057
  xo =   0.0747
  xo =   0.0993
  xo =     0.14

Value of qd

  xo =     0.14
  xo =   0.0993
  xo =   0.0747
  xo =    0.057
  xo =   0.0432
  xo =   0.0319
```

```
  xo =   0.0223
  xo =   0.0139
  xo =  0.00658

Value of k

  xo =     1.61
  xo =     1.66
  xo =     1.71
  xo =     1.78
  xo =     1.85
  xo =     1.95
  xo =     2.08
  xo =     2.26
  xo =     2.58

Value of kd

  xo =     2.58
  xo =     2.26
  xo =     2.08
  xo =     1.95
  xo =     1.85
  xo =     1.78
  xo =     1.71
  xo =     1.66
  xo =     1.61

Value of e

  xo =     1.53
  xo =     1.49
  xo =     1.45
  xo =      1.4
  xo =     1.35
  xo =      1.3
  xo =     1.24
  xo =     1.18
  xo =      1.1

Value of ed

  xo =      1.1
  xo =     1.18
  xo =     1.24
  xo =      1.3
  xo =     1.35
  xo =      1.4
  xo =     1.45
  xo =     1.49
  xo =     1.53
```

2.9.9 ASL_wiethe, ASL_viethe
楕円テータ関数

(1) **機 能**

$v = X_j$ に対してヤコビの楕円テータ関数 $\vartheta_i(v,q)$ の値を求める.

$$\vartheta_0(v,q) = \vartheta_4(v,q) = 1 + 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos(2n\pi v)$$

$$\vartheta_1(v,q) = 2q^{1/4}\sum_{n=0}^{\infty}(-1)^n q^{n(n+1)}\sin((2n+1)\pi v)$$

$$\vartheta_2(v,q) = 2q^{1/4}\sum_{n=0}^{\infty} q^{n(n+1)}\cos((2n+1)\pi v)$$

$$\vartheta_3(v,q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}\cos(2n\pi v)$$

(2) **使用法**

倍精度関数:

ierr = ASL_wiethe (nv, i, v, q, xo);

単精度関数:

ierr = ASL_viethe (nv, i, v, q, xo);

(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	nv	I	1	入 力	入力データ数
2	i	I	1	入 力	次数 i
3	v	{D∗ / R∗}	nv	入 力	変数値 X_j
4	q	{D / R}	1	入 力	ノーム q
5	xo	{D∗ / R∗}	nv	出 力	$\vartheta_i(X_j,q)$ の値
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) nv ≥ 1

(b) $0 \leq$ i ≤ 4

(c) $0 \leq$ q < 1.0

(d) |v[j − 1]| $< M$
ここで, M ={ 倍精度: 2^{31}, 単精度: 2^{18}}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a),(b) または (c) を満足しなかった.	処理を打ち切る.
4000+j	v[j − 1] が制限条件 (d) を満足しなかった.	

(6) **注意事項**

(a) $\vartheta_0(v,q) = \vartheta_4(v,q)$ である.

(b) 第 1 種不完全楕円積分 $F(x,m)$ の引数 (x,m) に対応する値を求める場合は, 2.9.3 $\left\{\begin{matrix}\text{ASL_dieii1}\\ \text{ASL_rieii1}\end{matrix}\right\}$から $u = F(x,m)$ を, 2.9.8 $\left\{\begin{matrix}\text{ASL_wienmq}\\ \text{ASL_vienmq}\end{matrix}\right\}$からノーム q と第 1 種完全楕円積分 $K(m)$ とをそれぞれ求め, $v = \frac{u}{2K(m)}$ としてこの関数を適用する.

(c) $\vartheta_4'(v,q)$ は $\vartheta_4'(v,q) = 2Z(u)K(m)\vartheta_4(v,q)$ により求めることができる.

(7) **使用例**

(a) 問 題

$\vartheta_i(v,q)$ の値を $i = 3$, $v = 0.0, 0.1, 0.2, \cdots, 0.9$, $q = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiethe */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    double   q;
    int nv;
    int  i;
    int ii;
    int ierr;
    nv=10;
    ii=3;
    q=0.5;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiethe ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiethe(nv,ii,xt, q, xo);

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

    printf( "\n\tValue of ethe(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
```

```
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiethe ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of ethe(x)

  xo =     2.13
  xo =     1.85
  xo =      1.2
  xo =    0.593
  xo =    0.231
  xo =    0.121
  xo =    0.231
  xo =    0.593
  xo =      1.2
  xo =     1.85
```

2.9.10 ASL_wiejzt, ASL_viejzt
ヤコビのゼータ関数

(1) **機 能**

$u = X_i$ に対してヤコビのゼータ関数

$$Z(u) = \frac{\Theta'(u)}{\Theta(u)} \quad (\text{ただし, } \Theta(u) = \vartheta_4(v, q) = \vartheta_4(u/2K(m), q))$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiejzt (nv, ui, rm, xo);

単精度関数:

ierr = ASL_viejzt (nv, ui, rm, xo);

(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	nv	I	1	入 力	入力データ数
2	ui	{D∗ / R∗}	nv	入 力	変数値 X_i (第 1 種不完全楕円積分値 $F(x, m)$)
3	rm	{D / R}	1	入 力	母数 m
4	xo	{D∗ / R∗}	nv	出 力	$Z(X_i)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nv} \geq 1$

(b) $0.0 \leq \mathrm{rm} \leq 1.0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3001	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) $u = 2K(m)v$ により u は v と $K(m)$ から求めることができる.

(7) 使用例

(a) 問 題

$Z(u)$ の値を $u = 0.0, 0.1, 0.2, \cdots, 0.9$, 母数 $m = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiejzt */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    double  rm;
    int nv;
    int  i;
    int ierr;
    nv=10;
    rm=0.5;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiejzt ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiejzt(nv,xt, rm, xo);

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

    printf( "\n\tValue of ejzt(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiejzt ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of ejzt(x)

  xo =        0
  xo =    0.027
  xo =    0.053
  xo =   0.0771
  xo =   0.0984
  xo =    0.116
  xo =     0.13
  xo =     0.14
  xo =    0.146
  xo =    0.147
```

2.9.11 ASL_wiejep, ASL_viejep
ヤコビのエプシロン関数

(1) **機 能**

$u = X_i$ に対してヤコビのエプシロン関数

$$E(u|m) = \int_0^x \sqrt{\frac{1-mt^2}{1-t^2}}dt = \int_0^u \mathrm{dn}^2(t)dt \quad \left(\text{ただし, } u = \int_0^x \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}}\right)$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_wiejep  (nv, ui, rm, xo);
```

単精度関数:

```
ierr = ASL_viejep  (nv, ui, rm, xo);
```

(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	nv	I	1	入 力	入力データ数
2	ui	{D∗ / R∗}	nv	入 力	変数値 X_i (第 1 種不完全楕円積分値 $F(x,m)$)
3	rm	{D / R}	1	入 力	母数 m
4	xo	{D∗ / R∗}	nv	出 力	$E(X_i\|m)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{nv} \geq 1$

(b) $0.0 \leq \mathrm{rm} \leq 1.0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3001	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) ヤコビのエプシロン関数 $E(u|m)$ は第 2 種不完全楕円積分 $E(\varphi\backslash\alpha)$ と同じもので以下の関係がある.

$$E(\varphi\backslash\alpha) = E(u|m) = \int_0^{\varphi} \sqrt{1-\sin^2\alpha\sin^2\theta}d\theta \quad (\text{ただし}, m = \sin^2\alpha, \quad \sin\varphi = \mathrm{sn}\,u)$$

(2.9.2 $\left\{\begin{matrix}\text{ASL_wieci2}\\ \text{ASL_vieci2}\end{matrix}\right\}$参照)

(7) **使用例**

(a) 問 題

$E(u|m)$ の値を $u = 0.0, 0.1, 0.2, \cdots, 0.9$, 母数 $m = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiejep */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    double  rm;
    int nv;
    int  i;
    int ierr;
    nv=10;
    rm=0.5;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiejep ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiejep(nv,xt, rm, xo);

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

    printf( "\n\tValue of ejep(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiejep ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **
```

```
ierr =      0

Value of ejep(x)

  xo =         0
  xo =    0.0998
  xo =     0.199
  xo =     0.296
  xo =      0.39
  xo =     0.481
  xo =     0.568
  xo =      0.65
  xo =     0.729
  xo =     0.802
```

2.9.12 ASL_wiejte, ASL_viejte
ヤコビのテータ関数

(1) **機 能**

$u = X_i$ に対してヤコビのテータ関数 $\Theta(u) = \vartheta_4(v, q) = \vartheta_4(u/2K(m), q)$ の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiejte (nv, ui, rm, xo);

単精度関数:

ierr = ASL_viejte (nv, ui, rm, xo);

(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	nv	I	1	入 力	入力データ数
2	ui	{D∗ / R∗}	nv	入 力	変数値 X_i (第 1 種不完全楕円積分値 $F(x, m)$)
3	rm	{D / R}	1	入 力	母数 m
4	xo	{D∗ / R∗}	nv	出 力	$\Theta(X_i)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nv ≥ 1

(b) $0.0 \leq$ rm ≤ 1.0

(c) $\frac{|\text{ui[i}-1]|}{2K(m)} < M$
ここで, $M =$ { 倍精度: 2^{31}, 単精度: 2^{18}}, $K(m)$ は $m =$ rm での第 1 種完全楕円積分値.

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3001	制限条件 (b) を満足しなかった.	
4000+i	ui[i − 1] が制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) $\Theta'(u)$ の値は $Z(u)$ の値を求める 2.9.10 $\left\{\begin{matrix}\text{ASL_wiejzt}\\ \text{ASL_viejzt}\end{matrix}\right\}$とこの関数を用いて以下の式から求めることができる.

$$\Theta'(u) = Z(u)\Theta(u)$$

(7) **使用例**

(a) 問 題

$\Theta(u)$ の値を $u = 0.0, 0.1, 0.2, \cdots, 0.9$, 母数 $m = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiejte */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    double  rm;
    int nv;
    int  i;
    int ierr;
    nv=10;
    rm=0.5;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiejte ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiejte(nv,xt, rm, xo);

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

    printf( "\n\tValue of ejte(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiejte ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0
```

```
Value of ejte(x)

  xo =    0.914
  xo =    0.915
  xo =    0.918
  xo =    0.925
  xo =    0.933
  xo =    0.943
  xo =    0.955
  xo =    0.968
  xo =    0.982
  xo =    0.996
```

2.9.13 ASL_wiepai, ASL_viepai
パイ関数

(1) **機 能**

$u = X_i$ に対してパイ関数

$$\Pi(u, \alpha) = m \,\mathrm{sn}\,\alpha \,\mathrm{cn}\,\alpha \,\mathrm{dn}\,\alpha \int_0^u \frac{\mathrm{sn}^2\, t dt}{1 - m\,\mathrm{sn}^2\,\alpha\,\mathrm{sn}^2\, t}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiepai (nv, ui, alf, rm, xo);

単精度関数:

ierr = ASL_viepai (nv, ui, alf, rm, xo);

(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	nv	I	1	入 力	入力データ数
2	ui	{D∗ / R∗}	nv	入 力	変数値 X_i (第 1 種不完全楕円積分値 $F(x, m)$)
3	alf	{D / R}	1	入 力	変数値 α
4	rm	{D / R}	1	入 力	母数 m
5	xo	{D∗ / R∗}	nv	出 力	$\Pi(X_i, \alpha)$ の値
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) nv ≥ 1

(b) $0.0 \leq$ rm < 1.0

(c) $|\frac{|\mathrm{ui[i-1]}| - \mathrm{alf}}{2K(m)}| < M$
ここで, $M =$ { 倍精度: 2^{31}, 単精度: 2^{18}}, $K(m)$ は $m =$ rm での第 1 種完全楕円積分値.

(d) $|\frac{\mathrm{alf}}{2K(m)}| < M$
ここで, M および $K(m)$ は制限条件 (c) に同じ.

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3001	制限条件 (b) を満足しなかった.	
4000	制限条件 (d) を満足しなかった.	
4000+i	ui[i − 1] が制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) パイ関数 $\Pi(u,\alpha)$ はヤコビのテータ関数 $\Theta(u)$ を用いて次のように表せる.

$$\Pi(u,\alpha) = u\frac{\Theta'(\alpha)}{\Theta(\alpha)} + \frac{1}{2}\log_e\frac{\Theta(u-\alpha)}{\Theta(u+\alpha)}$$

(b) 第 3 種不完全楕円積分 $F(z)$ は

$$F(z) = \int_0^z \frac{dz}{(1-a^2z^2)\sqrt{(1-z^2)(1-k^2z^2)}} = \frac{\mathrm{sn}\,\alpha}{\mathrm{cn}\,\alpha\,\mathrm{dn}\,\alpha}\Pi(u,\alpha) + u$$

と表せる. ただし, $z = \mathrm{sn}\,u$, $a^2 = k^2\,\mathrm{sn}^2\,\alpha$ である.

(7) **使用例**

(a) 問 題

$\Pi(u,\alpha)$ の値を $u = 0.0, 0.1, 0.2, \cdots, 0.9$, $\alpha = 1.7$, 母数 $m = 0.5$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiepai */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    double  rm;
    double alf;
    int nv;
    int  i;
    int ierr;
    nv=10;
    rm=0.5;
    alf=1.7;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiepai ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiepai(nv,xt,  alf, rm, xo);

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

    printf( "\n\tValue of epai(x)\n\n" );
    for(i=0;i<nv;i++)
```

```
        printf( "\t  xo = %8.3g\n", xo[i] );

            free(xt);
            free(xo);
        return 0;
    }
```

(c) 出力結果

```
        *** ASL_wiepai ***

         ** Input **

          xt =        0
          xt =      0.1
          xt =      0.2
          xt =      0.3
          xt =      0.4
          xt =      0.5
          xt =      0.6
          xt =      0.7
          xt =      0.8
          xt =      0.9

         ** Output **

        ierr =      0

        Value of epai(x)

          xo =        0
          xo = 1.28e-05
          xo = 0.000103
          xo = 0.000346
          xo = 0.000821
          xo =   0.0016
          xo =  0.00276
          xo =  0.00437
          xo =   0.0065
          xo =   0.0092
```

2.10 初等関数の不定積分

2.10.1 ASL_wiiexp, ASL_viiexp 指数積分

(1) **機 能**

$x = X_i$ に対して指数積分

$$\overline{\mathrm{Ei}}(x) = P\int_{-\infty}^{x} \frac{e^t}{t}dt \quad (x > 0.0)$$

$$\mathrm{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}dt = \int_{-\infty}^{x} \frac{e^t}{t}dt \quad (x < 0.0)$$

の値を求める. ただし, P は Cauchy の主値を表す.

(2) **使用法**

倍精度関数:

ierr = ASL_wiiexp (nv, xi, xo);

単精度関数:

ierr = ASL_viiexp (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$X_i > 0.0 : \overline{\mathrm{Ei}}(X_i)$ の値 $X_i < 0.0 : \mathrm{Ei}(X_i)$ の値
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) nv ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	xi[i − 1] < $-M_1$ であった. (注意事項 (b) 参照) (underflow)	xo[i − 1] = 0.0 を返す.
2000	xi[i − 1] = 0.0 (overflow)	xo[i − 1] =(最小値) を返す.
2100	xi[i − 1] > M_2 であった. (注意事項 (c) 参照) (overflow)	xo[i − 1] =(最大値) を返す.
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) $x = \mathrm{xi}[i-1] > 0.0$ のときは $\overline{\mathrm{Ei}}(x) = P\displaystyle\int_{-\infty}^{x} \frac{e^t}{t}dt$, $x = \mathrm{xi}[i-1] < 0.0$ のときは $\mathrm{Ei}(x) = -\displaystyle\int_{-x}^{\infty} \frac{e^{-t}}{t}dt = \int_{-\infty}^{x} \frac{e^t}{t}dt$ を計算する.

(b) この関数で ierr = 1000 となるときの M_1 の値は以下の通りである.
M_1 ={ 倍精度: 702.0, 単精度: 83.0}

(c) この関数で ierr = 2100 となるときの M_2 の値は以下の通りである.
M_2 ={ 倍精度: 709.782, 単精度: 88.722}

(d) 指数積分を

$$E_1(z) = \int_z^{\infty} \frac{e^t}{t}dt \quad (|\arg z| < \pi)$$

と定義する場合もある. この場合 $E_1(x) = -\mathrm{Ei}(-x)$ $(x > 0)$ となる.

(7) **使用例**

(a) 問 題
$\overline{\mathrm{Ei}}(x)$ の値を $x = 0.1, 0.2, \cdots, 1.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiiexp */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiiexp ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1;
```

```
        xt[i]=xt[i]/nv;
        printf( "\t  xt = %8.3g\n", xt[i] );
        }

        ierr = ASL_wiiexp(nv,xt, xo);

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

        printf( "\n\tValue of iexp(x)\n\n" );
        for(i=0;i<nv;i++)
        printf( "\t  xo = %8.3g\n", xo[i] );

            free(xt);
            free(xo);
        return 0;
    }
```

(c) 出力結果

```
        *** ASL_wiiexp ***

         ** Input **

          xt =      0.1
          xt =      0.2
          xt =      0.3
          xt =      0.4
          xt =      0.5
          xt =      0.6
          xt =      0.7
          xt =      0.8
          xt =      0.9
          xt =        1

         ** Output **

        ierr =      0

        Value of iexp(x)

          xo =    -1.62
          xo =   -0.822
          xo =   -0.303
          xo =    0.105
          xo =    0.454
          xo =     0.77
          xo =     1.06
          xo =     1.35
          xo =     1.62
          xo =      1.9
```

2.10.2 ASL_wiilog, ASL_viilog
対数積分

(1) **機 能**

$x = X_i$ に対して対数積分

$$\mathrm{Li}(x) = \int_0^x \frac{1}{\log_e(t)} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiilog (nv, xi, xo);

単精度関数:

ierr = ASL_viilog (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\mathrm{Li}(X_i)$ の値
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) nv ≥ 1

(b) xi[i − 1] ≥ 1.0

(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.	
2000	xi[i − 1] = 1.0 (overflow)	xo[i − 1] =(最小値) を返す.
2100	log(xi[i − 1]) > M_2 であった. (注意事項 (a) 参照)	xo[i − 1] =(最大値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) この関数で ierr = 2100 となるときの M_2 の値は以下の通りである.
M_2 ={ 倍精度: 709.782, 単精度: 88.722}

(7) 使用例

(a) 問 題

Li(x) の値を $x = 1.1, 1.2, \cdots, 2.0$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiilog */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiilog ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i+1+nv;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiilog(nv,xt, xo);

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

    printf( "\n\tValue of ilog(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiilog ***

 ** Input **

  xt =      1.1
  xt =      1.2
  xt =      1.3
  xt =      1.4
  xt =      1.5
  xt =      1.6
  xt =      1.7
  xt =      1.8
  xt =      1.9
  xt =        2

 ** Output **

ierr =      0

Value of ilog(x)

  xo =    -1.68
  xo =   -0.934
  xo =    -0.48
  xo =   -0.145
  xo =    0.125
  xo =    0.354
  xo =    0.554
  xo =    0.733
  xo =    0.895
  xo =     1.05
```

2.10.3 ASL_diisin, ASL_riisin
正弦積分

(1) **機 能**

正弦積分

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_diisin (xi, & xo);

単精度関数:

ierr = ASL_riisin (xi, & xo);

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	$\mathrm{Si}(x)$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

なし

(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.	

(6) **注意事項**

なし

(7) 使用例

(a) 問 題

Si(x) の値を $x = 1.0$ について求める.

(b) 入力データ

xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_diisin */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\txi = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_diisin(xi, &xo);

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

    printf( "\n\tValue of Si(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diisin ***

 ** Input **

xi =        1

 ** Output **

ierr =      0

Value of Si(x)

  xo =    0.946
```

2.10.4 ASL_diicos, ASL_riicos
余弦積分

(1) **機 能**

余弦積分

$$\mathrm{Ci}(x) = -\int_x^\infty \frac{\cos t}{t} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_diicos  (xi, & xo);
```

単精度関数:

```
ierr = ASL_riicos  (xi, & xo);
```

(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	xi	{D / R}	1	入 力	変数値 x
2	xo	{D∗ / R∗}	1	出 力	$\mathrm{Ci}(x)$ の値
3	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $0.0 \leq \mathrm{xi} \leq M$
ここで, M={ 倍精度: $2^{50}\pi$, 単精度: $2^{18}\pi$}

(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.	
2000	xi = 0.0 (overflow)	xo =(最小値) を返す.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る. (注意事項 (a) 参照)

(6) **注意事項**

(a) ierr = 3000 で xi が十分大きいときは, Ci(x) の値は 0.0 に非常に近い値である.

(7) **使用例**

(a) 問 題

Ci(x) の値を $x = 1.0$ について求める.

(b) 入力データ

xi = 1.0

(c) 主プログラム

```
/*      C interface example for ASL_diicos */

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

int main()
{
    double xi;
    double xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%lf", &xi );
    printf( "\txi = %8.3g\n", xi );

    fclose( fp );

    ierr = ASL_diicos(xi, &xo);

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

    printf( "\n\tValue of Ci(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diicos ***

 ** Input **

xi =        1

 ** Output **

ierr =      0

Value of Ci(x)

  xo =    0.337
```

2.10.5 ASL_wiifsi, ASL_viifsi
フレネル正弦積分

(1) **機 能**

$x = X_i$ に対してフレネル正弦積分

$$S(x) = \int_0^x \sin(\frac{\pi}{2}t^2)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiifsi (nv, xi, xo);

単精度関数:

ierr = ASL_viifsi (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$S(X_i)$ の値
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) nv ≥ 1

(b) |xi[i − 1]| $\leq M$
ここで, M={ 倍精度: 47453132.0, 単精度: 724.07734}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	処理を打ち切る. (注意事項 (b) 参照)

(6) **注意事項**

(a) フレネル正弦積分が $S(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\sin t}{\sqrt{t}} dt$ と表されているときは, xi[i − 1] に $\sqrt{\frac{2x}{\pi}}$ の値を入力すること.

(b) ierr=3000+i で $|x|$ が十分大きいときは (x = xi[i − 1]), $S(x)$ の値は 0.5(xi[i − 1] > 0.0 のとき) または −0.5(xi[i − 1] < 0.0 のとき) に非常に近い値である.

(7) 使用例

(a) 問 題

$S(x)$ の値を $x = 0.0, 0.1, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiifsi */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiifsi ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiifsi(nv,xt, xo);

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

    printf( "\n\tValue of ifsi(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiifsi ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of ifsi(x)

  xo =        0
  xo = 0.000524
  xo =  0.00419
  xo =   0.0141
  xo =   0.0334
  xo =   0.0647
  xo =    0.111
  xo =    0.172
  xo =    0.249
  xo =     0.34
```

2.10.6 ASL_wiifco, ASL_viifco
フレネル余弦積分

(1) **機 能**

$x = X_i$ に対してフレネル余弦積分

$$C(x) = \int_0^x \cos(\frac{\pi}{2}t^2)dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiifco (nv, xi, xo);

単精度関数:

ierr = ASL_viifco (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$C(X_i)$ の値
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) nv ≥ 1

(b) |xi[i − 1]| $\leq M$
ここで, M={ 倍精度: 47453132.0, 単精度: 724.07734}

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3000+i	xi[i − 1] が制限条件 (b) を満足しなかった.	処理を打ち切る. (注意事項 (b) 参照)

(6) **注意事項**

(a) フレネル余弦積分が $C(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\cos t}{\sqrt{t}} dt$ と表されているときは, xi[i − 1] に $\sqrt{\frac{2x}{\pi}}$ の値を入力すること.

(b) ierr = 3000 + i で $|x|$ が十分大きいときは (x = xi[i − 1]), $C(x)$ の値は 0.5(xi[i − 1] > 0.0 のとき) または −0.5(xi[i − 1] < 0.0 のとき) に非常に近い値である.

(7) 使用例

(a) 問 題

$C(x)$ の値を $x = 0.0, 0.1, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiifco */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiifco ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiifco(nv,xt, xo);

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

    printf( "\n\tValue of ifco(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiifco ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of ifco(x)

  xo =        0
  xo =      0.1
  xo =      0.2
  xo =    0.299
  xo =    0.397
  xo =    0.492
  xo =    0.581
  xo =     0.66
  xo =    0.723
  xo =    0.765
```

2.10.7 ASL_wiidaw, ASL_viidaw
ドーソン積分

(1) 機 能

$x = X_i$ に対してドーソン積分

$$e^{-x^2} \int_0^x e^{t^2} dt$$

の値を求める.

(2) 使用法

倍精度関数:

ierr = ASL_wiidaw (nv, xi, xo);

単精度関数:

ierr = ASL_viidaw (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$e^{-X_i^2} \int_0^{X_i} e^{t^2} dt$ の値
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) nv ≥ 1

(5) エラーインディケータ (戻り値)

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

(6) 注意事項

なし

(7) 使用例

(a) 問 題

$e^{-x^2}\int_0^x e^{t^2}dt$ の値を $x = 0.0, 0.1, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiidaw */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiidaw ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiidaw(nv,xt, xo);

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

    printf( "\n\tValue of idaw(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiidaw ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of idaw(x)

  xo =        0
  xo =   0.0993
  xo =    0.195
  xo =    0.283
  xo =     0.36
  xo =    0.424
  xo =    0.475
  xo =    0.511
  xo =    0.532
  xo =    0.541
```

2.10.8 ASL_wiicnd, ASL_viicnd
正規分布関数

(1) 機 能

$x = X_i$ に対して正規分布関数

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$$

の値を求める.

(2) 使用法

倍精度関数:

```
ierr = ASL_wiicnd  (nv, xi, xo);
```

単精度関数:

```
ierr = ASL_viicnd  (nv, xi, xo);
```

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$\Phi(X_i)$ の値
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) nv ≥ 1

(5) エラーインディケータ (戻り値)

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

(6) 注意事項

なし

(7) 使用例

(a) 問 題

$\Phi(x)$ の値を $x = 0.0, 0.1, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiicnd */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiicnd ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiicnd(nv,xt, xo);

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

    printf( "\n\tValue of icnd(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiicnd ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of icnd(x)

  xo =        0
  xo =   0.0398
  xo =   0.0793
  xo =    0.118
  xo =    0.155
  xo =    0.191
  xo =    0.226
  xo =    0.258
  xo =    0.288
  xo =    0.316
```

2.10.9 ASL_wiicnc, ASL_viicnc 余正規分布関数

(1) **機 能**

$x = X_i$ に対して余正規分布関数

$$\Psi(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{t^2}{2}} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wiicnc (nv, xi, xo);

単精度関数:

ierr = ASL_viicnc (nv, xi, xo);

(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	nv	I	1	入 力	入力データ数
2	xi	{D∗ / R∗}	nv	入 力	変数値 X_i
3	xo	{D∗ / R∗}	nv	出 力	$\Psi(X_i)$ の値
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) nv ≥ 1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処 理 内 容
0	正常終了.	
1000	xi[i − 1] > M であった. (注意事項 (a) 参照) (underflow)	xo[i − 1] = 0.0 を返す.
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) この関数で ierr=1000 となるときの M の値は以下の通りである.
M ={ 倍精度: 38.485, 単精度: 13.0}

(7) 使用例

(a) 問 題

$\Phi(x)$ の値を $x = 0.0, 0.1, 0.2, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wiicnc */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wiicnc ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wiicnc(nv,xt, xo);

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

    printf( "\n\tValue of icnc(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wiicnc ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of icnc(x)

  xo =      0.5
  xo =     0.46
  xo =    0.421
  xo =    0.382
  xo =    0.345
  xo =    0.309
  xo =    0.274
  xo =    0.242
  xo =    0.212
  xo =    0.184
```

2.11 誤差関数に関連した関数

2.11.1 ASL_wierrf, ASL_vierrf
誤差関数

(1) **機 能**

$x = X_i$ に対して, 誤差関数 Erf(x) の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wierrf (nv, xv, yv);

単精度関数:

ierr = ASL_vierrf (nv, xv, yv);

(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	nv	I	1	入 力	入力値の個数
2	xv	{D∗ / R∗}	nv	入 力	x_i
3	yv	{D∗ / R∗}	nv	出 力	Erf(x_i)
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) nv>0

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

なし

(7) 使用例

(a) 問 題

x_i を, 関数値が $1-0.1^i$ となるように与えて,$i=1,2,\cdots,10$ に対して $1-\mathrm{Erf}(x_i)$ の値を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    int nv,i;
    double *xv,*yv;
    int ierr;
    double o1,x;
    o1=0.1;nv=10;x=1.0;
    xv=(double * )malloc((size_t)( sizeof(double)* nv ));
    if( xv  == NULL )
    {
        printf( "no enough memory for array xv\n" );
        return -1;
    }
    yv=(double * )malloc((size_t)( sizeof(double)* nv ));
    if( yv  == NULL )
    {
        printf( "no enough memory for array yv\n" );
        return -1;
    }
    for ( i=0 ; i<nv ; i++ )
    {
        x=x*o1;
        ierr=ASL_diierf(x,&xv[i], 0);
        if( ierr > 0 )
        {
            printf( "error in ASL_diierf\n" );
        }
    }
    printf( "\n\t *** ASL_wierrf \n\n" );
    printf( "\n\t *** input  \n\n" );
    for ( i=0 ; i<nv ; i++ )
    {
        printf( "\n%13.8g\n " ,xv[i] );
    }
    ierr=ASL_wierrf(nv, xv, yv);
    printf( "\n\t *** output   \n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    for ( i=0 ; i<nv ; i++ )
    {
        printf( "\n%6d,%13.8g\n " ,i,1.0-yv[i] );
    }
    free(xv);
    free(yv);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_wierrf

 *** input

1.1630872

1.8213864

2.3267538

2.7510639

3.1234133

3.4589107

3.7665626

4.0522372

4.3200054

 4.572825

 *** output

ierr =      0

 0,          0.1

 1,         0.01

 2,        0.001
```

```
3,       0.0001
4,        1e-05
5,        1e-06
6,        1e-07
7,        1e-08
8,9.9999997e-10
9,1.0000001e-10
```

2.11.2 ASL_wierfc, ASL_vierfc
余誤差関数

(1) **機 能**

$x = X_i$ に対して, 余誤差関数 Erfc(x) の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_wierfc  (nv, xv, yv);
```

単精度関数:

```
ierr = ASL_vierfc  (nv, xv, yv);
```

(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	nv	I	1	入 力	入力値の個数
2	xv	{ D∗ / R∗ }	nv	入 力	x_i
3	yv	{ D∗ / R∗ }	nv	出 力	Erfc(x_i)
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) nv>0

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

なし

(7) 使用例

(a) 問 題

x_i を, 関数値が 0.1^i となるように与えて,$i = 1, 2, \cdots, 10$ に対して $\mathrm{Erfc}(x_i)$ の値を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    int nv,i;
    double *xv,*yv;
    int ierr;
    double o1,x;
    o1=0.1;nv=10;x=1.0;
    xv=(double * )malloc((size_t)( sizeof(double)* nv ));
    if( xv  == NULL )
    {
        printf( "no enough memory for array xv\n" );
        return -1;
    }
    yv=(double * )malloc((size_t)( sizeof(double)* nv ));
    if( yv  == NULL )
    {
        printf( "no enough memory for array yv\n" );
        return -1;
    }
    for ( i=0 ; i<nv ; i++ )
    {
        x=x*o1;
        ierr=ASL_diierf(x,&xv[i], 0);
        if( ierr > 0 )
        {
            printf( "error in ASL_diierf\n" );
        }
    }
    printf( "\n\t *** ASL_wierfc \n\n" );
    printf( "\n\t *** input  \n\n" );
    for ( i=0 ; i<nv ; i++ )
    {
        printf( "\n%13.8g\n " ,xv[i] );
    }
    ierr=ASL_wierfc(nv, xv, yv);
    printf( "\n\t *** output   \n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    for ( i=0 ; i<nv ; i++ )
    {
        printf( "\n%6d,%13.8g\n " ,i,yv[i] );
    }
    free(xv);
    free(yv);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_wierfc

 *** input

1.1630872

1.8213864

2.3267538

2.7510639

3.1234133

3.4589107

3.7665626

4.0522372

4.3200054

 4.572825

 *** output

ierr =      0

 0,          0.1

 1,         0.01

 2,        0.001
```

```
3,       0.0001
4,        1e-05
5,        1e-06
6,        1e-07
7,        1e-08
8,        1e-09
9,        1e-10
```

2.11.3 ASL_diierf, ASL_riierf
余誤差関数の逆関数

(1) **機 能**

余誤差関数 Erfc の逆関数を求める. すなわち, $0 < x \leq 1$ に対して $\mathrm{Erfc}(y) = x$ となる y を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_diierf  (x, & y, iter);
```

単精度関数:

```
ierr = ASL_riierf  (x, & y, iter);
```

(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	x	{D / R}	1	入 力	x
2	y	{D∗ / R∗}	1	出 力	y
3	iter	I	1	入 力	最大反復回数 (既定値:10)(注意事項 (d) 参照)
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 <x ≤ 1

(b) iter ≥ 1 (既定値にするために 0 または負の値を入力する場合は除く)

(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	x=1 であった.	y=0 とする.
3000	x> 1 であった.	処理を打ち切る.
3500	x≤ 0 であった.	
4000	反復回数が最大反復回数でも収束しなかった.	

(6) **注意事項**

(a) $\mathrm{Erfc}(y) = x$ を満たす y を $0 < x \leq 1$ に対して求めるので, $2 > x > 1$ に対しては,

$$\frac{2}{\sqrt{\pi}} \int_{-y}^{\infty} e^{-t^2} dt = 2 - x$$

を用いて, $y = -\mathrm{Erfc}^{-1}(2 - x)$ とすればよい.

(b) $\mathrm{Erfc}^{-1}(0) = \infty$, $\mathrm{Erfc}^{-1}(2) = -\infty$ である.

(c) $\mathrm{Erf}(x)$ の逆関数は $\mathrm{Erfc}^{-1}(1 - x)$ である.

(d) 最大反復回数は, iter が 0 または負のときは, 10 に設定される.

(7) **使用例**

(a) 問 題

$\mathrm{Erf}(y)$=0.9999 となる y を求め, 結果を確かめる.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    double x,y,z,x1;
    int m,ierr;
    x=0.9999;
    x1=1.0-x;
    m=0;
    printf( "\n\t *** ASL_diierf \n\n" );
    printf( "\n\t *** input ***  \n\n" );
    printf( "\n%13.8g\n",x1);
    ierr=ASL_diierf(x1,&y,m);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    ierr=ASL_wierrf(1,&y,&z);
    printf( "\n\ttestv          output          testv\n\n");
    printf( "\n%13.8g , %13.8g ,%13.8g\n", x,y,z);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_diierf

 *** input ***

   0.0001
 *** OUTPUT ***

ierr =      0
testv           output          testv

   0.9999 ,     2.7510639 ,       0.9999
```

2.11.4 ASL_jiierf, ASL_iiierf
複素変数の誤差関数

(1) **機 能**

z=Z_i に対して, 複素変数の誤差関数 $e^{-z^2}\,\mathrm{Erfc}(-iz)$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_jiierf (nv, z, w);

単精度関数:

ierr = ASL_iiierf (nv, z, w);

(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	nv	I	1	入 力	入力値の個数
2	z	{ Z∗ / C∗ }	nv	入 力	z
3	w	{ Z∗ / C∗ }	nv	出 力	$e^{-z^2}\,\mathrm{Erfc}(-iz)$
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) nv $>$ 0

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処 理 内 容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	オーバフローが発生した. (注意事項 (b) 参照)	

(6) **注意事項**

(a) 虚数部が 0 に近いときは, 若干精度が落ちる (倍精度では 10^{-10} 程度の誤差).

(b) $\Im(z)$ が負かつ $-\Re(z^2)$ が log(正の最大値) 程度のときは, オーバフローが起きる.

(7) 使用例

(a) 問 題

積分路 $|z-3i|=1$ について, コーシーの積分表示を確かめる.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <asl.h>
int main()
{
    double _Complex s,z,z3,w,w3,paii,zerr;
    double _Complex *zinp,*wout;
    int n;
    int i;
    int ierr;
    n=100;
    zinp = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (n+1) ));
    if(  zinp == NULL )
    {
        printf( "no enough memory for array zinp\n" );
        return -1;
    }
    wout = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (n+1) ));
     if(  wout == NULL )
    {
         printf( "no enough memory for array wout\n" );
         return -1;
    }
    s=0.0;
    z3=3.0*_Complex_I;
    paii=-1.0;
    paii=clog(paii);
    printf( "\n\t *** ASL_jiierf \n\n" );
    for ( i=0 ; i<n ; i++ )
    {
        z=(i+1)*cimag(paii)*2.0/(double)n * _Complex_I;
        z=cexp(z);
        zinp[i]=z3+z;
    }
    zinp[n]=z3;
    n=n+1;
    ierr=ASL_jiierf(n,zinp,wout);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\t output value \n\n" );
    w3=wout[n-1];
    for ( i=0 ; i<n-1 ; i++ )
    {
        w=wout[i]/(double)(n-1);
        s=s+w;
    }
    printf( "\n%8.3g , %8.3g\n", creal(w3), cimag(w3));
    printf( "\n\t value obtained by Cauchy's theorem  \n\n" );
    printf( "\n%8.3g , %8.3g\n", creal(s), cimag(s));
    zerr=w3-s;
    printf( "\n\t *** abs error \n\n" );
    printf( "\n%8.3g , %8.3g\n", creal(zerr) , cimag(zerr));
    free(zinp);
    free(wout);
        return 0;
}
```

(c) 出力結果

```
     *** ASL_jiierf


     *** OUTPUT ***

    ierr =      0

     output value

   0.179 , 6.25e-20

     value obtained by Cauchy's theorem


   0.179 , -1.3e-18

     *** abs error

 5.55e-17 , 1.36e-18
```

2.12 ルジャンドル陪関数

2.12.1 ASL_dileg1, ASL_rileg1
第 1 種ルジャンドル陪関数

(1) **機 能**

第 1 種ルジャンドル陪関数

$$P_n^m(x) = (|1-x^2|)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dileg1 (n, m, xi, & xo);

単精度関数:

ierr = ASL_rileg1 (n, m, xi, & xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	m	I	1	入 力	陪数 m
3	xi	{D / R}	1	入 力	変数値 x
4	xo	{D∗ / R∗}	1	出 力	$P_n^m(x)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) m < 0 のとき |m| $\leq$ n (n < 0 なら |m| $\leq -$n $- 1$)

(b) |m| $\leq$ n (n < 0 なら |m| $\leq -$n $- 1$) のとき
|m| $< M_1$
ここで, M_1 ={ 倍精度: 150, 単精度: 27}

(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.	
2000	$\|$xi$\|$ > (最大値)$^{1/\mathrm{n}}/2$ および n ≥ 2 (overflow)	xi ≥ 0.0 のとき xo=(最大値) xi < 0.0 のとき xo=(最大値)$\times(-1)^{\mathrm{n}}$ を返す.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) $|x| > 1.0$ のときは,

$$P_n^m(x) = (x^2 - 1.0)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

(Hobson のルジャンドル陪関数) を計算し, $|x| \leq 1.0$ のときは,

$$P_n^m(x) = (1.0 - x^2)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

(Ferrers のルジャンドル陪関数) を計算する.

(b) この関数は精度を保つため, 内部で倍長演算を用いている.

(c) $|x| \leq 1.0$ のとき,

$$P_n^m(x) = (-1)^m (1.0 - x^2)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m}$$

と定義する場合や, $(n-m)!$ をかけて陪関数と定義する場合もあるので注意を要する.

(d) 次数 n が大きく, 多くの xi についての値を求める場合は, 2.15.3 $\left\{\begin{matrix}\text{ASL\_winplg}\\ \text{ASL\_vinplg}\end{matrix}\right\}$ を使用すると良い. 第 1 種ルジャンドル陪関数 $P_n^m(x)$ と正規化された球面調和関数 $P_n^{*m}(x)$ の間の関係は,

$$P_n^{*0}(x) = \sqrt{\frac{2n+1}{4\pi}} P_n^0(x) \ (-1 \leq x \leq 1),$$

$$P_n^{*m}(x) = \sqrt{\frac{2n+1}{4\pi}} \sqrt{2\frac{(n-m)!}{(n+m)!}} P_n^m(x) \ (-1 \leq x \leq 1; m = 1, 2, \cdots, n)$$

である. m が大きいときは, 第 1 種ルジャンドル陪関数の絶対値が階乗的に大きくなることに注意する必要がある.

(7) 使用例

(a) 問 題

$P_n^m(x)$ の値を $n = 4$, $m = 2$, $x = 0.8$ について求める.

(b) 入力データ

n = 4, m = 2, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_dileg1 */

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

int main()
{
    int ni;
    int mi;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "dileg1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dileg1 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &ni );
    fscanf( fp, "%d", &mi );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d    m = %6d    xi = %8.3g\n", ni, mi, xi );

    fclose( fp );

    ierr = ASL_dileg1(ni, mi, xi, &xo);

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

    printf( "\n\tValue of Pnm(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dileg1 ***

 ** Input **

n =      4    m =      2    xi =      0.8

 ** Output **

ierr =      0

Value of Pnm(x)

  xo =      9.4
```

2.12.2 ASL_dileg2, ASL_rileg2 第 2 種ルジャンドル陪関数

(1) **機 能**

第 2 種ルジャンドル陪関数

$$Q_n^m(x) = (|1-x^2|)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dileg2  (n, m, xi, & xo);
```

単精度関数:

```
ierr = ASL_rileg2  (n, m, xi, & xo);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	m	I	1	入 力	倍数 m
3	xi	{D / R}	1	入 力	変数値 x
4	xo	{D∗ / R∗}	1	出 力	$Q_n^m(x)$ の値
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $\mathrm{m} < 0$ のとき $|\mathrm{m}| \leq \mathrm{n}$

(c) $|\mathrm{xi}| \neq 1.0$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a), (b) または (c) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	
4100	級数展開式の計算が収束しなかった.	

(6) **注意事項**

(a) $|x| > 1.0$ のときは,

$$Q_n^m(x) = (x^2 - 1.0)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}$$

(Hobson のルジャンドル陪関数) を計算し, $|x| \leq 1.0$ のときは,

$$Q_n^m(x) = (1.0 - x^2)^{\frac{m}{2}} \frac{d^m Q_n(x)}{dx^m}$$

(Ferrers のルジャンドル陪関数) を計算する.

(b) この関数は精度を保つため, 内部で倍長演算を用いている.

(c) $|x| > 1.0$ または $|x| < 1.0$ のときに本定義の $Q_n^m(x)$ に $(-1)^m$ をかけて定義する場合や, $(n-m)!$ をかけて陪関数と定義する場合もあるので注意を要する.

(7) **使用例**

(a) 問 題

$Q_n^m(x)$ の値を $n = 4$, $m = 2$, $x = 1.8$ について求める.

(b) 入力データ

n = 4, m = 2, xi = 1.8

(c) 主プログラム

```
/*      C interface example for ASL_dileg2 */

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

int main()
{
    int n;
    int mi;
    double xi;
    double xo;
    int ierr;
    FILE *fp;

    fp = fopen( "dileg2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dileg2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &mi );
    fscanf( fp, "%lf", &xi );
    printf( "\tn = %6d    m = %6d    xi = %8.3g\n", n, mi, xi );

    fclose( fp );

    ierr = ASL_dileg2(n, mi, xi, &xo);

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

    printf( "\n\tValue of Qnm(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dileg2 ***

 ** Input **

n =      4    m =      2    xi =      1.8

 ** Output **

ierr =      0

Value of Qnm(x)

  xo =   0.0703
```

2.13 直交多項式

2.13.1 ASL_diople, ASL_riople
ルジャンドル多項式

(1) **機 能**

ルジャンドル多項式

$$P_i(x) = \frac{1}{2^i i!}\frac{d^i}{dx^i}(x^2-1)^i \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_diople (n, xi, xo);

単精度関数:

ierr = ASL_riople (n, xi, xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	n + 1	出 力	$P_i(x)$ の値 $(i = 0, 1, \cdots, n)$
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) 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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) 注意事項

(a) この関数は精度を保つため, 内部で倍長演算を用いている.

(7) 使用例

(a) 問 題

$P_n(x)$ の値を $n = 3$, $x = 0.8$ について求める.

(b) 入力データ

n = 3, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diople */

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

int main()
{
    int n;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_diople(n, xi, xo);

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

    printf( "\n\tValue of Pn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diople ***

 ** Input **

n =      3      xi =      0.8

 ** Output **

ierr =      0

Value of Pn(x)

  xo =     0.08
```

2.13.2 ASL_dizglw, ASL_rizglw
ガウス・ルジャンドル積分公式

(1) **機 能**

(高次数の) ガウス・ルジャンドル積分公式の積分点および重みを計算する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dizglw  (n, z, w, work);
```

単精度関数:

```
ierr = ASL_rizglw  (n, z, w, work);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	次数 n
2	z	{D∗ / R∗}	n	出 力	ルジャンドル多項式 $P_n(x)$ の零点 (積分点) (小さい方から順に入る)
3	w	{D∗ / R∗}	n	出 力	ガウス・ルジャンドル積分公式の重み (零点の小さい方から順に入る)
4	work	{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) $n \geq 1$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	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	積分点を求める処理で収束しなかった.	
5000	ルジャンドル多項式の計算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) ルジャンドル多項式 $P_n(x)$ の零点はこの関数の内部で計算される.

(b) ガウス・ルジャンドル積分公式は

$$\int_{-1}^{1} f(x)dx = \sum_{j=1}^{n} w_j f(z_j)$$

である. ここに, z_j, w_j は $P_n(x)$ の零点とそれに対応する重みである. 積分区間が $[-1,1]$ である場合に直接に適用できるが, これ以外の積分区間については置換積分により $[-1,1]$ に写すこと.

(c) ガウス・ルジャンドル積分公式を, 振動する関数の積分に用いることは避けることが望ましい. これを例を挙げて示すために定積分

$$\int_{-1}^{1} 5e^{-25x^2}\cos(10Ax)dx$$

を考える. この被積分関数は両端で急激に減少する. 両端を無限区間に換えて計算すると, この積分の値は $\sqrt{\pi}e^{-A^2}$ となる. また, この区間変換による誤差は 10^{-12} 程度である. 分点の数を 24 としてガウス・ルジャンドル積分公式を用いると, A =0.3 では $0.3*10^{-7}$ 程度の誤差である. A =2.4 では結果値-0.0004898 が得られるが, 真の値は 0.005585 程度である. このように $\cos(24x)$ 程度の振動項があるだけでも積分誤差を生じる.

(d) この関数は, 高次数のガウス・ルジャンドル積分公式についても適用できる.

(7) **使用例**

(a) 問 題

N=24 として, 定積分 $\int_{-1}^{1} \frac{1}{1+x+x^2}dx = \frac{\pi}{\sqrt{3}}$ を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    double *z,*w,d,trued,three;
    int n;
    int i,ierr;
    n=24;
    three=3.0;
    z=( double * )malloc((size_t)( sizeof(double) * n ));
    if( z == NULL )
    {
        printf( "no enough memory for array z\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;
    }
    printf( "\n\t *** ASL_dizglw \n\n" );
    ierr = ASL_dizglw(n, z, w, &w[n]);
    printf( "\n\t *** OUTPUT ***\n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\t *** Gauss points and weights \n\n" );
    for ( i=0 ; i<n ; i++ )
    {
        printf( "\n%6d, %13.8g , %13.8g\n", i , z[i] , w[i]);
    }
    d=0.0;
    for ( i=0 ; i<n ; i++ )
    {
        d=d+w[i]/(1.0+z[i]+z[i]*z[i]);
    }
    trued=M_PI/sqrt(three);
    printf( "\n\t *** value(Gauss-Legendre) and true value  \n\n" );
    printf("\n%13.8g, %13.8g \n", d, trued );
    free(z);
    free(w);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_dizglw

 *** OUTPUT ***

ierr =      0
 *** Gauss points and weights

 0,   -0.99518722 ,    0.01234123
 1,   -0.97472856 ,   0.028531389
 2,   -0.93827455 ,   0.044277439
 3,   -0.88641553 ,   0.059298585
 4,   -0.82000199 ,   0.073346481
 5,   -0.74012419 ,   0.086190162
 6,   -0.64809365 ,   0.097618652
 7,   -0.54542147 ,    0.10744427
 8,   -0.43379351 ,    0.11550567
 9,   -0.31504268 ,    0.12167047
10,   -0.19111887 ,    0.12583746
11,  -0.064056893 ,     0.1279382
12,   0.064056893 ,     0.1279382
13,    0.19111887 ,    0.12583746
14,    0.31504268 ,    0.12167047
15,    0.43379351 ,    0.11550567
16,    0.54542147 ,    0.10744427
17,    0.64809365 ,   0.097618652
18,    0.74012419 ,   0.086190162
19,    0.82000199 ,   0.073346481
20,    0.88641553 ,   0.059298585
21,    0.93827455 ,   0.044277439
22,    0.97472856 ,   0.028531389
23,    0.99518722 ,    0.01234123
 *** value(Gauss-Legendre) and true value

1.8137994,     1.8137994
```

2.13.3 ASL_diopla, ASL_riopla
ラゲール多項式

(1) **機 能**

ラゲール多項式

$$L_i(x) = \frac{e^x}{i!}\frac{d^i}{dx^i}(e^{-x}x^i) \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_diopla (n, xi, xo);

単精度関数:

ierr = ASL_riopla (n, xi, xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	n + 1	出 力	$L_i(x)$ の値 $(i = 0, 1, \cdots, n)$
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) 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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) この関数は精度を保つため, 内部で倍長演算を用いている.

(7) **使用例**

(a) 問 題

$L_n(x)$ の値を $n = 3$, $x = 0.8$ について求める.

(b) 入力データ

n = 3, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diopla */

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

int main()
{
    int n;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_diopla(n, xi, xo);

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

    printf( "\n\tValue of Ln(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diopla ***

 ** Input **

n =      3      xi =      0.8

 ** Output **

ierr =      0

Value of Ln(x)

  xo =   -0.525
```

2.13.4 ASL_diophe, ASL_riophe
エルミート多項式

(1) **機 能**

エルミート多項式

$$H_i(x) = (-1)^i e^{x^2} \frac{d^i}{dx^i}(e^{-x^2}) \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_diophe (n, xi, xo);

単精度関数:

ierr = ASL_riophe (n, xi, xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	n + 1	出 力	$H_i(x)$ の値 $(i = 0, 1, \cdots, n)$
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) 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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) 注意事項

(a) この関数は精度を保つため, 内部で倍長演算を用いている.

(7) 使用例

(a) 問 題

$H_n(x)$ の値を $n = 3$, $x = 0.8$ について求める.

(b) 入力データ

n = 3, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diophe */

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

int main()
{
    int n;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_diophe(n, xi, xo);

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

    printf( "\n\tValue of Hn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diophe ***

 ** Input **

n =      3      xi =      0.8

 ** Output **

ierr =      0

Value of Hn(x)

  xo =     -5.5
```

2.13.5 ASL_diopch, ASL_riopch
チェビシェフ多項式

(1) **機 能**

チェビシェフ多項式

$$T_i(x) = \frac{(-1)^i}{(2i-1)!!}\sqrt{1-x^2}\frac{d^i}{dx^i}(1-x^2)^{i-1/2} \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_diopch (n, xi, xo);

単精度関数:

ierr = ASL_riopch (n, xi, xo);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	xi	{ D / R }	1	入 力	変数値 x
3	xo	{ D∗ / R∗ }	n + 1	出 力	$T_i(x)$ の値 $(i = 0, 1, \cdots, n)$
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) 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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) **注意事項**

(a) この関数は精度を保つため, 内部で倍長演算を用いている.

(7) **使用例**

(a) 問 題

$T_n(x)$ の値を $n = 3$, $x = 0.8$ について求める.

(b) 入力データ

n = 3, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diopch */

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

int main()
{
    int n;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_diopch(n, xi, xo);

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

    printf( "\n\tValue of Tn(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diopch ***

 ** Input **

n =      3      xi =      0.8

 ** Output **

ierr =      0

Value of Tn(x)

  xo =   -0.352
```

2.13.6 ASL_diopc2, ASL_riopc2
第 2 種チェビシェフ関数

(1) **機 能**

第 2 種チェビシェフ関数

$$U_i(x) = \frac{\sqrt{1-x^2}}{i}\frac{dT_i(x)}{dx} \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_diopc2  (n, xi, xo);
```

単精度関数:

```
ierr = ASL_riopc2  (n, xi, xo);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	xi	{D / R}	1	入 力	変数値 x
3	xo	{D∗ / R∗}	n + 1	出 力	$U_i(x)$ の値 $(i = 0, 1, \cdots, n)$
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) n $\geq$ 0, |xi| $\leq$ 1.0

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

なし

(7) 使用例

(a) 問 題

$U_3(0.8)$ の値を求める.

(b) 入力データ

n = 3, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diopc2 */

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

int main()
{
    int n;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

    fp = fopen( "diopc2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_diopc2 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d\t\txi = %8.3g\n", n, xi );

    fclose( fp );

    ierr = ASL_diopc2(n, xi, xo);

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

    printf( "\n\tValue of Un(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diopc2 ***

 ** Input **

n =      3      xi =      0.8

 ** Output **

ierr =      0

Value of Un(x)

  xo =    0.936
```

2.13.7 ASL_diopgl, ASL_riopgl 一般ラゲール多項式

(1) 機 能

一般ラゲール多項式

$$L_i^{(\alpha)}(x) = \frac{e^x x^{-\alpha}}{i!}\frac{d^i}{dx^i}(e^{-x}x^{i+\alpha}) \quad (i = 0, 1, \cdots, n)$$

の値を求める.

(2) 使用法

倍精度関数:

ierr = ASL_diopgl (n, alf, xi, xo);

単精度関数:

ierr = ASL_riopgl (n, alf, xi, xo);

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	最高次数 n
2	alf	{D / R}	1	入 力	変数値 α
3	xi	{D / R}	1	入 力	変数値 x
4	xo	{D∗ / R∗}	n + 1	出 力	$L_i^{(\alpha)}(x)$ の値 $(i = 0, 1, \cdots, n)$
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) $n \geq 0$

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	演算の途中でオーバフローが発生した.	

(6) **注意事項**

なし

(7) **使用例**

(a) 問 題

$L_n^{(\alpha)}(x)$ を $n = 3$, $\alpha = 0.5$, $x = 0.8$ について求める.

(b) 入力データ

n = 3, alf = 0.5, xi = 0.8

(c) 主プログラム

```
/*      C interface example for ASL_diopgl */

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

int main()
{
    int n;
    double alf;
    double xi;
    double *xo;
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%lf", &alf );
    fscanf( fp, "%lf", &xi );

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

    printf( "\tn = %6d    alf = %8.3g    xi = %8.3g\n", n, alf, xi );

    fclose( fp );

    ierr = ASL_diopgl(n, alf, xi, xo);

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

    printf( "\n\tValue of Lna(x)\n\n" );
    printf( "\t  xo = %8.3g\n", xo[n] );

    free( xo );

    return 0;
}
```

(d) 出力結果

```
*** ASL_diopgl ***

 ** Input **

n =      3    alf =      0.5    xi =      0.8

 ** Output **

ierr =      0

Value of Lna(x)

  xo =   -0.278
```

2.14 マシュー関数

2.14.1 ASL_dimtce, ASL_rimtce
整数次マシュー関数 $ce_n(x, q)$

(1) **機 能**

整数次マシュー関数 $ce_n(x, q)$ の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dimtce  (nord, n, q, x, & ce, & isw, work);
```

単精度関数:

```
ierr = ASL_rimtce  (nord, n, q, x, & ce, & isw, work);
```

(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	nord	I	1	入 力	フーリエ展開項の個数 −1 (注意事項 (a) 参照)
2	n	I	1	入 力	次数 n
3	q	{D / R}	1	入 力	パラメータ q
4	x	{D / R}	1	入 力	変数 x
5	ce	{D∗ / R∗}	1	出 力	関数値 $ce_n(x, q)$
6	isw	I*	1	入 力	処理スイッチ isw =0 : 初期設定後, 関数値計算を行う isw =1 : 関数値計算のみ行う
				出 力	初期設定が行われた場合 1 を返す
7	work	{D∗ / R∗}	内容参照	入出力	作業領域 (係数テーブル) (注意事項 (b) 参照) **大きさ:** $2 \times \text{nord}^2 + 6 \times \text{nord} + 108$
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) $2 \leq \text{nord} \leq 50$

(b) $0 \leq \text{n} \leq 2 \times \text{nord} + 1$

(c) isw=0 または isw=1 であること.

(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.	
2000	十分な精度で計算できなくなった.	処理を打ち切る.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) nord の値が小さいとき, ce は十分な精度が得られないので, nord は大きいことが望ましい. パラメータ q および次数 n に対する nord の値の目安は, 次の範囲である.

$$\min(50.0, 0.5 \times \mathrm{n} + 10 + |\mathrm{q}|) \leq \mathrm{nord} \leq 50$$

(b) 特定のパラメータ q について, 複数のマシュー関数 $ce_n(x,q)$ を求める場合は, isw=0 としてこの関数を一度呼び出した後, x と n の値のみを変え, 続けてこの関数を使用すればよい. ただし, このとき nord の値は

$$\min(50.0, 0.5 \times \mathrm{n_{max}} + 10 + |\mathrm{q}|) \leq \mathrm{nord} \leq 50$$

の範囲に定めなければならない. ここで, n_{max} は次数 n の最大値である.
このようにすれば, 初期設定処理が一度だけしか行われないため, 演算回数の無駄を省くことができる.

(c) ierr=2000 で終了した場合, 計算結果の精度は保証できない.

(d) $ce_n(x,q)$ は, (余弦関数を展開項とする)Fourier 級数で表される. この関数では, この級数を (nord+1) 番目の展開項までの級数和として近似している. したがって, nord の値は $ce_n(x,q)$ の計算精度と計算効率に影響する.

(e) n と |q| が大きい程, 計算時間が増加する傾向がある. $\mathrm{n} \leq 90$, $|\mathrm{q}| \leq 70.0$ とすることが望ましい.

(7) **使用例**

(a) 問 題
展開項数 21 の近似条件で $ce_7(x, 5.0)$ の値を x=1.0, 2.0,$\cdots$, 10.0 について求める.

(b) 主プログラム

```
/*      C interface example for ASL_dimtce */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>
int main()
{
    double *work;
    double  q;
    double *x;
    double *ce;
    int     n;
    int     i;
    int     isw;
    int     ierr;
    int     ierr1;
    int     nord;
    int     size;
    int     nv;
    nv=10;
    nord=20;
    n   =7;
    q   =5;
    size=2*nord*nord+6*nord+108;
    work=(double *)malloc((size_t)(sizeof(double) * size));
    if( work == NULL )
    {
        printf("no enough memory for work\n");
        return -1;
    }
    x=(double *)malloc((size_t)(sizeof(double) * nv));
    if( x == NULL )
    {
        printf("no enough memory for x\n");
        return -1;
```

```
    }
    ce=(double *)malloc((size_t)(sizeof(double) * nv));
    if( ce == NULL )
    {
        printf("no enough memory for ce\n");
        return -1;
    }
    printf( "    *** ASL_dimtce ***\n" );
    printf( "\n     ** Input **\n\n" );
    isw=0;
    ierr1=0;
    for(i=0;i<nv;i++)
    {
        x[i]=1+i;
        printf( "\t  xi = %8.3g\n", x[i] );
        ierr=ASL_dimtce(nord, n, q, x[i], &ce[i], &isw, work);
        if( ierr > 0 )
        {
            ierr1=ierr;
            printf( "\tierr = %6d\n", ierr );
        }
    }
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr1 );
    for(i=0;i<nv;i++)
    {
        printf( "\t  xo = %8.3g\n", ce[i] );
    }
    free(work);
    free(x);
    free(ce);
    return 0;
}
```

(c) 出力結果

```
*** ASL_dimtce ***

 ** Input **

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

 ** Output **

ierr =      0
  xo =    0.902
  xo =   -0.129
  xo =   -0.666
  xo =   -0.796
  xo =    -0.77
  xo =   -0.218
  xo =  -0.0578
  xo =    0.859
  xo =    0.931
  xo =    0.869
```

2.14.2 ASL_dimtse, ASL_rimtse
整数次マシュー関数 $se_n(x,q)$

(1) **機 能**

整数次マシュー関数 $se_n(x,q)$ の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dimtse  (nord, n, q, x, & se, & isw, work);
```

単精度関数:

```
ierr = ASL_rimtse  (nord, n, q, x, & se, & isw, work);
```

(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	nord	I	1	入 力	フーリエ展開項の個数 −1 (注意事項 (a) 参照)
2	n	I	1	入 力	次数 n
3	q	{D / R}	1	入 力	パラメータ q
4	x	{D / R}	1	入 力	変数 x
5	se	{D∗ / R∗}	1	出 力	関数値 $se_n(x,q)$
6	isw	I*	1	入 力	処理スイッチ isw =0 : 初期設定後, 関数値計算を行う isw =1 : 関数値計算のみ行う
				出 力	初期設定が行われた場合 1 を返す
7	work	{D∗ / R∗}	内容参照	入出力	作業領域 (係数テーブル) (注意事項 (b) 参照) **大きさ:** $2 \times \mathrm{nord}^2 + 6 \times \mathrm{nord} + 108$
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) $2 \leq \mathrm{nord} \leq 50$

(b) $1 \leq \mathrm{n} \leq 2 \times \mathrm{nord} + 2$

(c) isw=0 または isw=1 であること.

(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.	
2000	十分な精度で計算できなくなった.	処理を打ち切る.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) nord の値が小さいとき, se は十分な精度が得られないので, nord は大きいことが望ましい. パラメータ q および次数 n に対する nord の値の目安は, 次の範囲である.

$$\min(50.0, 0.5 \times \mathrm{n} + 10 + |\mathrm{q}|) \leq \mathrm{nord} \leq 50$$

(b) 特定のパラメータ q について, 複数のマシュー関数 $se_n(x,q)$ を求める場合は, isw=0 としてこの関数を一度呼び出した後, x と n の値のみを変え, 続けてこの関数を使用すればよい. ただし, このとき nord の値は

$$\min(50.0, 0.5 \times \mathrm{n}_{\max} + 10 + |\mathrm{q}|) \leq \mathrm{nord} \leq 50$$

の範囲に定めなければならない. ここで, n_{max} は次数 n の最大値である.

このようにすれば, 初期設定処理が一度だけしか行われないため, 演算回数の無駄を省くことができる.

(c) ierr=2000 で終了した場合, 計算結果の精度は保証できない.

(d) $ce_n(x,q)$ は, (正弦関数を展開項とする)Fourier 級数で表される. この関数では, この級数を (nord+1) 番目の展開項までの級数和として近似している. したがって, nord の値は $se_n(x,q)$ の計算精度と計算効率に影響する.

(e) n と |q| が大きい程, 計算時間が増加する傾向がある. $\mathrm{n} \leq 90$, $|\mathrm{q}| \leq 70.0$ とすることが望ましい.

(7) **使用例**

(a) 問 題

展開項数 21 の近似条件で $se_7(x, 5.0)$ の値を x=1.0, 2.0,$\cdots$, 10.0 について求める.

(b) 主プログラム

```
/*      C interface example for ASL_dimtse */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>
int main()
{
    double *work;
    double  q;
    double *x;
    double *se;
    int     n;
    int     i;
    int     isw;
    int     ierr;
    int     ierr1;
    int     nord;
    int     size;
    int     nv;
    nv=10;
    nord=20;
    n   =7;
    q   =5;
    size=2*nord*nord+6*nord+108;
    work=(double *)malloc((size_t)(sizeof(double) * size));
    if( work == NULL )
    {
        printf("no enough memory for work\n");
        return -1;
    }
    x=(double *)malloc((size_t)(sizeof(double) * nv));
    if( x == NULL )
    {
```

```
        printf("no enough memory for x\n");
        return -1;
    }
    se=(double *)malloc((size_t)(sizeof(double) * nv));
    if( se == NULL )
    {
        printf("no enough memory for se\n");
        return -1;
    }
    printf( "    *** ASL_dimtse ***\n" );
    printf( "\n     ** Input **\n\n" );
    isw=0;
    ierr1=0;
    for(i=0;i<nv;i++)
    {
        x[i]=1+i;
        printf( "\t  xi = %8.3g\n", x[i] );
        ierr=ASL_dimtse(nord, n, q, x[i], &se[i], &isw, work);
        if( ierr > 0 )
        {
            ierr1=ierr;
            printf( "\tierr = %6d\n", ierr );
        }
    }
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr1 );
    for(i=0;i<nv;i++)
    {
        printf( "\t  xo = %8.3g\n", se[i] );
    }
    free(work);
    free(x);
    free(se);
    return 0;
}
```

(c) 出力結果

```
*** ASL_dimtse ***

 ** Input **

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

 ** Output **

ierr =      0
  xo =     0.37
  xo =    0.956
  xo =    0.815
  xo =    0.585
  xo =   -0.569
  xo =    -1.02
  xo =       -1
  xo =   -0.411
  xo =    0.448
  xo =     0.53
```

2.15 その他の関数

2.15.1 ASL_wixsps, ASL_vixsps ディログ関数

(1) **機 能**

N 個の実数 X_i (≥ 0 , $i = 1, \cdots, N$) に対するディログ関数

$$Li_2(X_i) = -\int_0^{X_i} \log|t-1|\frac{dt}{t}$$

を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_wixsps (nv,xv,yv);

単精度関数:

ierr = ASL_vixsps (nv,xv,yv);

(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	nv	I	1	入 力	X_i の個数 N
2	xv	{ D∗ / R∗ }	nv	入 力	X_i ($i = 1, \cdots$,nv)
3	yv	{ D∗ / R∗ }	nv	出 力	$Li_2(X_i)$ ($i = 1, \cdots$,nv)
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) nv ≥ 1

(b) xv[i $-$ 1] ≥ 0 ($i = 1, \cdots,$ nv)

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) yv[i − 1](i = 1, · · · , nv) には, Li_2(xv[i − 1]) が代入される.

(b) $Li_2(x)$ は, $0 \leq x < 2$ で単調増加し, $x = 2$ で最大値 $\frac{\pi^2}{4}$ をとる. $x > 2$ では単調減少して, 漸近式

$$Li_2(x) = -\frac{1}{2}(\log x)^2 + \frac{\pi^2}{3} + O(x^{-1})$$

が成り立つ.

(c) $Li_2(x)$ の正零点の近似値は, 12.59517037 である.

(7) **使用例**

(a) 問 題

$x_i = 0.2i$ $(i = 1, 2, \cdots, 10)$ に対するディログ関数の値 $Li_2(x_i)$ を求める.

(b) 入力データ

nv=10, 配列 xv

(c) 主プログラム

```
/*       C interface example for ASL_wixsps */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nv,ierr,i;
    double *xv, *yv;
    printf( "    *** ASL_wixsps ***\n" );
    nv=10;
    printf( "\n     ** Input **\n\n" );
    xv = ( double * )malloc(sizeof(double)*nv);
    if( xv == NULL )
    {
        printf( "no enough memory for array xv\n");
        return -1;
    }
    yv = ( double * )malloc(sizeof(double)*nv);
    if( yv == NULL )
    {
        printf( "no enough memory for array yv\n");
        return -1;
    }
    printf( "\tnv  = %6d\n" , nv);
    for(i=0;i<nv;i++)
    {
        xv[i]=0.2*(i+1);
    }
    ierr = ASL_wixsps(nv, xv, yv);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t           xv                    yv\n\n" );
    for(i=0; i<nv; i++)
    {
        printf( "\n\t" );
        printf( "%6d",i);
        printf( "%8.3g",xv[i]);
        printf( "\t              ");
        printf( "%8.3g",yv[i]);
    }
    printf( "\n" );
    free(xv);
    free(yv);
    return 0;
}
```

(d) 出力結果

```
*** ASL_wixsps ***

 ** Input **

nv  =     10

 ** Output **

ierr =      0

           xv                   yv
```

```
0      0.2                0.211
1      0.4                0.449
2      0.6                0.728
3      0.8                 1.07
4        1                 1.64
5      1.2                 2.13
6      1.4                 2.32
7      1.6                 2.41
8      1.8                 2.46
9        2                 2.47
```

2.15.2 ASL_widbey, ASL_vidbey デバイ関数

(1) **機 能**

N 個の実数 X_i $(\geq 0, i = 1, \cdots, N)$ に対して, デバイ関数

$$F_D(X_i) = \frac{3}{{X_i}^3} \int_0^{X_i} \frac{e^t t^4}{(e^t - 1)^2} dt$$

の値を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_widbey (nv,xv,yv);

単精度関数:

ierr = ASL_vidbey (nv,xv,yv);

(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	nv	I	1	入 力	X_i の値の個数 N
2	xv	{D∗ / R∗}	nv	入 力	$X_i(i = 1, \cdots$,nv)
3	yv	{D∗ / R∗}	nv	出 力	$F_D(X_i)(i = 1, \cdots$,nv)
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) nv ≥ 1

(b) xv[i $-$ 1] ≥ 0 (i $= 1, \cdots,$ nv)

(5) **エラーインディケータ (戻り値)**

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

(6) 注意事項

(a) yv[i − 1](i = 1, · · · , nv) には, F_D(xv[i − 1]) が代入される.

(b) デバイ関数 $F_D(y)$ は単調減少する.

(c) $F_D(0)$ は, $\lim_{y\to+0} F_D(y)$ を意味する.

(7) 使用例

(a) 問 題

$x_i = 0.2i$ $(i = 1, 2, \cdots, 10)$ に対するデバイ関数の値 $F_D(x_i)$ を求める.

(b) 入力データ

nv=10, 配列 xv

(c) 主プログラム

```
/*       C interface example for ASL_widbey */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nv,ierr,i;
    double *xv, *yv;
    printf( "    *** ASL_widbey ***\n" );
    nv=10;
    printf( "\n     ** Input **\n\n" );
    xv = ( double * )malloc(sizeof(double)*nv);
    if( xv == NULL )
    {
        printf( "no enough memory for array xv\n");
        return -1;
    }
    yv = ( double * )malloc(sizeof(double)*nv);
    if( yv == NULL )
    {
        printf( "no enough memory for array yv\n");
        return -1;
    }
    printf( "\tnv  = %6d\n" , nv);
    for(i=0;i<nv;i++)
    {
        xv[i]=0.2*(i+1);
    }
    ierr = ASL_widbey(nv, xv, yv);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t           xv                    yv\n\n" );
    for(i=0; i<nv; i++)
    {
        printf( "\n\t" );
        printf( "%6d",i);
        printf( "%8.3g",xv[i]);
        printf( "\t           ");
        printf( "%8.3g",yv[i]);
    }
    printf( "\n" );
    free(xv);
    free(yv);
    return 0;
}
```

(d) 出力結果

```
*** ASL_widbey ***

 ** Input **

nv  =     10

 ** Output **

ierr =      0

           xv                    yv


     0     0.2                 0.998
     1     0.4                 0.992
     2     0.6                 0.982
     3     0.8                 0.969
     4       1                 0.952
```

```
5     1.2                0.932
6     1.4                0.908
7     1.6                0.883
8     1.8                0.855
9       2                0.825
```

2.15.3 ASL_winplg, ASL_vinplg
球面調和関数

(1) **機 能**

N 個の実数 $X_i(|X_i| \leq 1; i = 1, \cdots, N)$ に対して正規化された n 次の球面調和関数系 (位数 $m = 0, \cdots, n$)

$$P_n^{*m}(X_i) = \frac{1}{4\pi\sqrt{-1}^m} A_{n,m} \int_{-\pi}^{\pi} (X_i + \sqrt{-1}\sqrt{1 - X_i^2}\cos\phi)^n \cos(m\phi) d\phi$$

$(m = 0, \cdots, n; i = 1, \cdots, N)$ を求める. ただし, 正規化定数 $A_{n,m}$ は

$$A_{n,0} = \sqrt{\frac{2n+1}{\pi}} \; ; \; A_{n,m} = \sqrt{\frac{2(2n+1)(n-m)!(n+m)!}{\pi(n!)^2}} \; (m = 1, \cdots, n)$$

である.

(2) **使用法**

倍精度関数:

ierr = ASL_winplg (nv,xv,n,plg,nvl,work);

単精度関数:

ierr = ASL_vinplg (nv,xv,n,plg,nvl,work);

(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	nv	I	1	入 力	X_i の値の個数 N
2	xv	{ D∗ / R∗ }	nv	入 力	$X_i(i = 1, \cdots$,nv)
3	n	I	1	入 力	次数 n
4	plg	{ D∗ / R∗ }	内容参照	出 力	球面調和関数 $P_n^{*m}(X_i)$ $(i = 1, \cdots$,nv; $m = 0, \cdots$,n) (注意事項 (a) 参照) **大きさ:** nvl×(n + 1)
5	nvl	I	1	入 力	plg の整合寸法
6	work	{ D∗ / R∗ }	内容参照	ワーク	作業領域 **大きさ:** 3 × nv + n + 1
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) $1 \leq$ nv $\leq$ nvl

(b) n ≥ 1

(c) $|$xv[i $- 1]| \leq 1$ (i $= 1, \cdots,$ nv)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3100	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) plg[i + nvl × m − 1] (i = 1, · · · , nv;$m = 0, \cdots,$ n) には, $P_n^{*\mathrm{m}}$(xv[i − 1]) が代入される.

(b) n を固定して $m = 0, \cdots, n$ について計算する場合は 2.12.1 $\left\{\begin{matrix}\text{ASL\_dileg1}\\ \text{ASL\_rileg1}\end{matrix}\right\}$ より効率が良い.

(c) 非負整数 m について, $n_1, n_2 \geq m$ である整数 n_1, n_2 をとるとき, 以下の積分関係式が成り立つ.

$$\int_{-1}^{1} P_{n_1}^{*\ m}(x) P_{n_2}^{*\ m}(x) dx = \frac{\delta_{n1,n2}}{(1+\delta_{0,m})\pi}$$

球面調和関数 $P_n^{*m}(\cos\theta)\cos(m\phi)(m = 0, \cdots, n)$ と $P_n^{*m}(\cos\theta)\sin(m\phi)(m = 1, \cdots, n)$ を $n = 1, 2, \cdots$ についてとれば, これらは単位球面上の面積分 $\int_{-\pi\leq\phi\leq\pi;0\leq\theta\leq\pi} \sin\theta d\theta d\phi$ について, 正規直交底となる.

(d) 関係式

$$\frac{2n+1}{4\pi} P_n(xy + \sqrt{(1-x^2)(1-y^2)}\cos\phi) = \sum_{m=0}^{n} P_n^{*m}(x) P_n^{*m}(y)\cos(m\phi) \quad (-1 \leq x, y \leq 1)$$

が成り立つ.

(7) **使用例**

(a) 問 題

$x_1 = 0.57735026919, x_2 = -0.57735026919$ に対する, $n = 10$ 次の球面調和関数の値

$$P_n^{*m}(x_i)\ (m = 0, \cdots, n)$$

を求める.

(b) 入力データ

nv=2,nvl= 2, 配列 xv, n= 10

(c) 主プログラム

```
/*       C interface example for ASL_winplg */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int nv,n,nvl,ierr,i;
    double xv[2], *plg, *work;
    printf( "    *** ASL_winplg ***\n" );
    nv=2;n=10,nvl=2;
    printf( "\n     ** Input **\n\n" );
    plg = ( double * )malloc(sizeof(double)*nvl*(n+1));
    if( plg == NULL )
    {
        printf( "no enough memory for array plg\n");
        return -1;
    }
    work = ( double * )malloc(sizeof(double)*(3*nv+n+1));
    if( work == NULL )
    {
        printf( "no enough memory for array work\n");
        return -1;
    }
    printf( "\tn   = %6d\n" , n );
    printf( "\tnv  = %6d\n" , nv);
    printf( "\tnvl = %6d\n" , nvl);
```

```
    xv[0]=0.57735026919;xv[1]=-0.57735026919;
    printf( "\txv  = %8.3g",xv[0]);printf( "\t     %8.3g",xv[1]);
    ierr = ASL_winplg( nv, xv, n, plg, nvl, work );
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\t     value[0]                value[1]\n\n" );
    for(i=0; i<n+1; i++)
    {
        printf( "\n\t" );
        printf( "%6d",i);
        printf( "%8.3g",plg[nvl*i]);
        printf( "\t             ");
        printf( "%8.3g",plg[nvl*i+1]);
    }
    printf( "\n" );
    free(plg);
    free(work);
    return 0;
}
```

(d) 出力結果

```
*** ASL_winplg ***

 ** Input **

n   =     10
nv  =      2
nvl =      2
xv  =    0.577         -0.577
 ** Output **

ierr =      0

     value[0]                value[1]


     0  -0.346                 -0.346
     1  0.0767                -0.0767
     2   0.504                  0.504
     3  0.0616                -0.0616
     4  -0.493                 -0.493
     5  -0.358                  0.358
     6    0.24                   0.24
     7   0.635                 -0.635
     8   0.587                  0.587
     9    0.32                  -0.32
    10   0.101                  0.101
```

2.15.4 ASL_wixsla, ASL_vixsla
ランジュバン関数

(1) **機 能**

$x = X_i$ に対するランジュバン関数

$$L(x) = \coth(x) - \frac{1}{x}$$

の値を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_wixsla  (nv, xi, xo);
```

単精度関数:

```
ierr = ASL_vixsla  (nv, xi, xo);
```

(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	nv	I	1	入 力	入力データ数
2	xi	{ D∗ / R∗ }	nv	入 力	変数値 X_i
3	xo	{ D∗ / R∗ }	nv	出 力	$L(X_i)$ の値
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) nv ≥ 1

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) $L(x)$ を定義式どおりに計算すると $x \simeq 0$ で精度が悪くなるのでこの関数を使用されたい.

(7) 使用例

(a) 問 題

$L(x)$ の値を $x = 0.0, 0.1, \cdots, 0.9$ について求める.

(b) 主プログラム

```
/*      C interface example for ASL_wixsla */
/*      R9.0    UPDD 03/02/05 N.Y.      CINT-SRC-02-0002   */

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

int main()
{
    double *xt;
    double *xo;
    int nv;
    int  i;
    int ierr;
    nv=10;
    xt=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xt == NULL )
    {
        printf("no enough memory for array xt\n");
        return -1;
    }
    xo=(double *)malloc((size_t)(sizeof(double) * nv));
    if( xo == NULL )
    {
        printf("no enough memory for array xo\n");
        return -1;
    }

    printf( "    *** ASL_wixsla ***\n" );
    printf( "\n     ** Input **\n\n" );
    for(i=0;i<nv;i++)
    {
    xt[i]=i;
    xt[i]=xt[i]/nv;
    printf( "\t  xt = %8.3g\n", xt[i] );
    }

    ierr = ASL_wixsla(nv,xt, xo);

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

    printf( "\n\tValue of xsla(x)\n\n" );
    for(i=0;i<nv;i++)
    printf( "\t  xo = %8.3g\n", xo[i] );

        free(xt);
        free(xo);
    return 0;
}
```

(c) 出力結果

```
*** ASL_wixsla ***

 ** Input **

  xt =        0
  xt =      0.1
  xt =      0.2
  xt =      0.3
  xt =      0.4
  xt =      0.5
  xt =      0.6
  xt =      0.7
  xt =      0.8
  xt =      0.9

 ** Output **

ierr =      0

Value of xsla(x)

  xo =        0
  xo =   0.0333
  xo =   0.0665
  xo =   0.0994
  xo =    0.132
  xo =    0.164
  xo =    0.195
  xo =    0.226
  xo =    0.256
  xo =    0.285
```

2.15.5 ASL_wixzta, ASL_vixzta
フルビッツゼータ関数

(1) **機 能**

$a > 0, s = X_i \geq 0$ に対して, フルビッツゼータ関数から $(s-1)^{-1}$ を引いた

$$\zeta(s,a) - (s-1)^{-1} = \frac{1}{\Gamma(s)}(\int_1^\infty \frac{e^{-at}}{1-e^{-t}} t^{s-1} dt + \int_0^1 (\frac{e^{-at}}{1-e^{-t}} - \frac{1}{t}) t^{s-1} dt + (s-1)^{-1}) - (s-1)^{-1}$$

の値を求める. この右辺は

$$\sum_{n=0}^{\infty} (n+a)^{-s} - (s-1)^{-1}$$

$(\Re(s) > 1)$ を $\Re(s) > 0$ にまで解析接続した積分表示の例である.

(2) **使用法**

倍精度関数:

ierr = ASL_wixzta (nv, x, a, y);

単精度関数:

ierr = ASL_vixzta (nv, x, a, y);

(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	nv	I	1	入 力	入力値の個数
2	x	{D∗ / R∗}	nv	入 力	変数値 s
3	a	{D / R}	1	入 力	パラメータ a
4	y	{D∗ / R∗}	nv	出 力	フルビッツゼータ関数の値 $\zeta(s,a) - 1/(s-1)$
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) a > 0.0

(b) x[i $- 1] \geq 0.0$

(c) nv > 0

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処 理 内 容
0	正常終了.	
3000	制限条件 (a),(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) $\zeta(s,a)$ は s を複素変数とする有理型関数に解析接続され, $s=1$ のみが極である. この関数は, フルビッツゼータ関数の値そのものを出力するのではなく, $1/(s-1)$ を引いた値を出力している. したがって, $\mathrm{x[i-1]}=1$ のとき $-\mathrm{y[i-1]}$ はディガンマ関数 $\psi(a)$ に等しい.

(b) ポリガンマ関数は,$\mathrm{x[i-1]}=2,3,\cdots$ としたときの $\mathrm{y[i-1]}$ の定数倍で与えることもできる.

(7) 使用例

(a) 問 題

$x=0.5i$ $(i=1,\cdots,10)$, $a=1$ について, $\zeta(x,a)-(x-1)^{-1}$ を求める.

(b) 主プログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <asl.h>
int main()
{
    int nv,i,ierr;
    double *x,*y,a;
    nv=10;
    x=(double * )malloc((size_t)( sizeof(double)* nv ));
     if( x  == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    y=(double * )malloc((size_t)( sizeof(double)* nv ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }
    a=1.0;
    for ( i=0 ; i<nv ; i++ )
    {
        x[i]=i+1;
        x[i]=x[i]*0.5;
    }
    printf( "\n\t *** ASL_wixzta \n\n" );
    printf( "\n\t *** input  a \n\n" );
    printf( "\n%13.8g\n " ,a);
    ierr=ASL_wixzta(nv, x, a, y);
    printf( "\n\t *** output \n\n" );
    printf( "\n\tierr = %6d\n", ierr );
    printf( "\n\t  x    y    \n\n" );
    for ( i=0 ; i<nv ; i++ )
    {
        printf( "\n%13.8g,%13.8g\n ",x[i],y[i]);
    }
    free(x);
    free(y);
    return 0;
}
```

(c) 出力結果

```
 *** ASL_wixzta

 *** input  a

        1

 *** output

ierr =      0

  x    y

      0.5,   0.53964549

        1,   0.57721566

      1.5,   0.61237535

        2,   0.64493407

      2.5,   0.67482059

        3,    0.7020569
```

```
3.5,   0.72673387
  4,    0.7489899
4.5,   0.76899323
  5,   0.78692776
```

2.15.6 ASL_dixeps, ASL_rixeps
正定値 2 次形式 $x^2 + ay^2$ のゼータ関数

(1) **機 能**

正定値 2 次形式 $x^2 + ay^2$ のゼータ関数から極を消すための関数を減じた

$$f(s;a) \equiv \sum_{(m,n)\in Z^2,(m,n)\neq(0,0)} (m^2 + an^2)^{-s} - \frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)} (s > 1)$$

の解析接続を, $s > -1$ に対して求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dixeps (s, a, & y);

単精度関数:

ierr = ASL_rixeps (s, a, & y);

(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	s	{D / R}	1	入 力	変数値 s
2	a	{D / R}	1	入 力	正定値 2 次形式の係数 a
3	y	{D∗ / R∗}	1	出 力	正定値 2 次形式のゼータ関数 $f(s;a)$ の値 (注意事項 (a) 参照)
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) s > -1

(b) a > 0

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) 正定値 2 次形式 $x^2 + ay^2$ のゼータ関数は $s = 1$ において 1 次の唯一の極をもつ有理型関数である. この関数はゼータ関数の値そのものを出力するのではなく, 極を消すために $\frac{\pi^s a^{-s/2}}{\Gamma(s)(s-1)}$ を引いた値を出力している.

(b) 整数対 (m, n) は, Z^2(0 および負も含めたすべての整数対) の (0, 0) 以外の全部を動く.

(7) 使用例

(a) 問 題

$s = 3.0,\ a = 1.0$ に対する正定値 2 次形式 $x^2 + ay^2$ のゼータ関数の値を求めて, フルビッツのゼータ関数値を用いて別の方法で計算した値との比較をする.

(b) 入力データ

s=3.0, a=1.0

(c) 主プログラム

```
/*       C interface example for ASL_dixeps */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    int ierr;
    double s,a,f1,f2,y,z,z1,z2,z3,pai3,f1s;
    pai3=31.006276680299820175476315067101/4.0;
    f1s=0.25*0.25*0.25;
    printf( "    *** ASL_dixeps ***\n" );
    s=3.0;a=1.0;f1=0.25;f2=0.75;
    printf( "\n     ** Input **\n\n" );
    printf( "\n\ts= %8.3g",s );
    printf( "\n\ta= %8.3g\n",a );
    ierr = ASL_dixeps(s,a,&y);
    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\ty= %8.3g\n",y );
    printf( "\n     ** Test values **\n\n" );
    z=y+pai3;
    ierr = ASL_dixzta(s, a, &z1);
    if( ierr > 0 )
    {
        printf( "\n\t dixzta_ierr = %6d\n", ierr );
    }
    ierr = ASL_dixzta(s,f1, &z2);
    if( ierr > 0 )
    {
        printf( "\n\t dixzta_ierr = %6d\n", ierr );
    }
    ierr = ASL_dixzta(s,f2, &z3);
    if( ierr > 0 )
    {
        printf( "\n\t dixzta_ierr = %6d\n", ierr );
    }
    z2=(z2 - z3)*f1s;
    z1=z1*z2/f1;
    printf( "\n\ttest1= %8.3g\n",z );
    printf( "\n\ttest2= %8.3g\n",z1);
    printf( "\n\tdist = %8.3g\n",z1-z);
    return 0;
}
```

(d) 出力結果

```
*** ASL_dixeps ***

 ** Input **


s=        3
a=        1

 ** Output **

ierr =      0

y=    -3.09

 ** Test values **


test1=     4.66

test2=     4.66

dist = -2.66e-15
```

第 3 章　ソート・順位付け

3.1　概 要

本章では, データのソートと順位付けおよびマージを行う関数について説明する.
本ライブラリでは, 以下の機能を持つ関数が用意されている.

(1) データ列のソート

(2) ペアデータ列のソート

(3) データ列の順位付け

(4) 上位 N 件の抽出

(5) ソート済みデータ列のマージ

(6) ソート済みペアデータ列のマージ

3.1.1 使用しているアルゴリズム

3.1.1.1 ソート

昇順にソートする場合のアルゴリズムを以下に示す. 降順にソートする場合のアルゴリズムは大小が異なるだけで同様である.

(1) シェル・ソート (shell sort)
(1) 間隔 h を設定する.
(2) データ列より, 間隔 h の部分列をすべて取り出す.
(3) 各部分列の中が小さい順に並ぶように隣同士を比較する.
逆順なら位置を交換し, 交換したデータはさらにその前のデータとの順序を確かめる.
さらに逆順ならば位置の交換が前にさかのぼって行われる.
(4) 間隔 h を小さくして (2) から (3) を繰り返し, h =1 の処理を行えばソートは終了する.

(2) ヒープ・ソート (heap sort)
(1) 与えられたデータをヒープ・ツリー (整列二分木. 親は子よりも大きいか等しい値を持っている) に構成する.
(2) ルートとツリーの一番後ろのデータを交換する.
(3) 一番後ろのデータを除いた部分を A とする.
(4) A を新しいツリーと考え, これを再びヒープ・ツリーに構成する.
(5) (2) から (4) を繰り返し, データがルートだけになればソートは終了する.

(3) クイック・ソート (quick sort)
(1) ソート区間内のデータ数を数える.
(2) データ数により以下のことをする.

- データ数が 1 以下のとき :
  何もしない.
- データ数が 2 のとき :
  逆順なら位置を交換する.
- データ数が 3 以上のとき :
  ① 区間内から枢軸値を一つ選ぶ.
  ② 区間内のデータを枢軸値より小さいものと大きいものとの二つの区間に振り分ける.

(3) (1) から (2) を繰り返し, すべてのデータ区間のデータ数が 2 以下になればソートは終了する.

(4) マージ・ソート (merge sort)
(1) ソート区間内のデータ数を数える.
(2) データ数により以下のことをする.

- データ数が 1 のとき :
  何もしない.
- データ数が 2 のとき :
  逆順なら位置を交換する.
- データ数が 3 以上のとき :
  ① 区間のデータを半分ずつ前半と後半に分ける.
  ② 前半のデータを再帰的にマージ・ソートする.

後半のデータを再帰的にマージ・ソートする.
③ ソートされた前半と後半のデータをマージする.

3.1.1.2 データ列の順位付け

n 個のデータが与えられた場合の各データに対応する昇順順位番号と同一順位にあるデータの数を返す.

3.1.1.3 上位 N 件の抽出

n 個のデータ $a_i(i = 1, 2, \cdots, n)$ が与えられたときこれを大きい順または小さい順に並べ換えたデータ列のうち先頭から m 個のデータ $a_j(j = j_1, j_2, \cdots, j_m)$ $(m < n)$ を求める.

3.1.1.4 ソート済みデータ列のマージ

昇順に整列された 2 つのデータ列 a_i $(i = 1, 2, \cdots, n)$ と b_j $(j = 1, 2, \cdots, m)$ をマージして, データ列 c_k $(k = 1, 2, \cdots, \ell)$ を求める.
ただし, c_k は

$$c_1 \leq c_2 \leq \cdots \leq c_\ell$$

を満たす.

3.1.1.5 ソート済みペアデータ列のマージ

a_i について昇順に整列されたデータの組 (a_i, b_i) $(i = 1, 2, \cdots, n)$ と c_j について昇順に整列されたデータの組 (c_j, d_j) $(j = 1, 2, \cdots, m)$ をマージして, データの組 (e_k, f_k) $(k = 1, 2, \cdots, \ell)$ を求める.
ただし, e_k は

$$e_1 \leq e_2 \leq \cdots \leq e_\ell$$

を満たす.
なお, 2 次ソートを指定した場合には, $e_k = e_{k+1}$ を満たす任意の k について

$$f_k \leq f_{k+1}$$

を満たすように $k = 1, 2, \cdots, \ell$ を決める.

3.1.2 参考文献

(1) Niklaus Wirth, “ALGORITHMS + DATA STRUCTURES = PROGRAMS”, Prentice–Hall Inc. (1976).

(2) 浪平博人, “ソート・検索”, CQ 出版.

(3) 近藤嘉雪, “アルゴリズムとデータ構造”, SOFTBANK.

3.2 ソート

3.2.1 ASL_dssta1, ASL_rssta1
データ列のソート

(1) **機 能**

n 個のデータ $a_{i_k}(k = 1, 2, \cdots, n)$ が与えられたとき a_i を並べ換えたデータ列 $a_{j_k}(k = 1, 2, \cdots, n)$ を求める. ただし, a_j は

昇順の場合 : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

降順の場合 : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

を満たす.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dssta1  (a,n,isw,wk,iwk);
```

単精度関数:

```
ierr = ASL_rssta1  (a,n,isw,wk,iwk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	n	入 力	ソートされるデータ a_i
				出 力	ソートされたデータ a_j
2	n	I	1	入 力	配列 a の大きさ
3	isw	I	1	入 力	ソート法の選択スイッチ (注意事項 (a) 参照)
4	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ :** n　(isw= 4, −4 のとき) 1　(それ以外のとき)
5	iwk	I*	内容参照	ワーク	作業領域 **大きさ :** 2×n　(isw= 3, −3 のとき) 1　　(それ以外のとき)
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≥1

(b) isw=1, 2, 3, 4, −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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	

(6) **注意事項**

(a) isw で選択するソート法は以下の通りである.

isw	ソート法	isw	ソート法
1	シェル・ソート (昇順)	−1	シェル・ソート (降順)
2	ヒープ・ソート (昇順)	−2	ヒープ・ソート (降順)
3	クイック・ソート (昇順)	−3	クイック・ソート (降順)
4	マージ・ソート (昇順)	−4	マージ・ソート (降順)

利用者は入力データの性質により適切なソート法を選べばよい. 各ソート法の特徴を以下に示す.

· シェル・ソート

計算量は平均 $O(n^{1.5})$ 程度である. どのようなデータに対しても安定して速いソートを行う. とくに, データ系列の一部がソートされている場合は速くなる.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· ヒープ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· クイック・ソート

計算量は平均 $O(n \log n)$ だが, 最初から部分的にソートされているなど, ある種の規則性があるものに対しては, 大変非効率的なソートになる. ランダムなデータに対してはもっとも速いソート方法である.

データに複数個同じ値が存在する場合, データの順序がソートを行う前後で保たれる保証はない.

· マージ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

同じ値のデータ間で整列前の順序関係が保たれる.

(7) 使用例

(a) 問 題

a [0] = 5.0

a [1] = 4.0

a [2] = 9.0

a [3] = 6.0

a [4] = 2.0

a [5] = 5.0

をシェル・ソートで昇順にソートする.

(b) 入力データ

配列 a, n=6, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dssta1 */

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

int main()
{
    double *a,*wk;
    int n,isw,*iwk,ierr;
    int i;
    FILE *fp;


    fp = fopen( "dssta1.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw=1;
    n=6;

    printf( "    *** ASL_dssta1 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

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

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

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

    printf( "\tArray a" );

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

    fclose( fp );

    ierr = ASL_dssta1(a, n, isw, wk, iwk);

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

    printf( "\tArray a" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "%8.3g", a[i] );
    }
```

```
        printf( "\n" );

        free( a );
        free( wk );
        free( iwk );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_dssta1 ***

     ** Input **

    n=  6

    Array a       5       4       9       6       2       5

     ** Output **

    ierr =      0

    Array a       2       4       5       5       6       9
```

3.2.2 ASL_dssta2, ASL_rssta2
ペアデータ列のソート

(1) **機 能**

2 組の n 個のデータ $a_{i_k}(k = 1, 2, \cdots, n), b_{i_k}(k = 1, 2, \cdots, n)$ が与えられたとき a_i を並べ換えたデータ列 $a_{j_k}(k = 1, 2, \cdots, n)$ と a_j に対応するデータ列 $b_{j_k}(k = 1, 2, \cdots, n)$ を求める. ただし, a_j は

昇順の場合 : $a_{j_1} \leq a_{j_2} \leq \cdots \leq a_{j_n}$

降順の場合 : $a_{j_1} \geq a_{j_2} \geq \cdots \geq a_{j_n}$

を満たす.

なお, 2 次ソートを指定した場合には, $a_{j_k} = a_{j_{k+1}}$ を満たす任意の k について

昇順の場合 : $b_{j_k} \leq b_{j_{k+1}}$

降順の場合 : $b_{j_k} \geq b_{j_{k+1}}$

を満たすように $j = j_1, j_2, \cdots, j_n$ を決める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dssta2  (a,n,b,isw1,isw2,wk,iwk);
```

単精度関数:

```
ierr = ASL_rssta2  (a,n,b,isw1,isw2,wk,iwk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	n	入 力	ソートされるデータ a_i
				出 力	ソートされたデータ a_j
2	n	I	1	入 力	配列 a の大きさ
3	b	{D∗ / R∗}	n	入 力	a_i に対するデータ b_i
				出 力	ソートされた a_j に対するデータ b_j
4	isw1	I	1	入 力	ソート法の選択スイッチ (注意事項 (a) 参照)
5	isw2	I	1	入力	2 次ソートスイッチ isw2=0：2 次ソートは行わない isw2=1：2 次ソートを行う
6	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ：** 2×n　(isw1= 4, −4 のとき) 1　　(それ以外のとき)
7	iwk	I*	内容参照	ワーク	作業領域 **大きさ：** 2×n　(isw1= 3, −3 のとき) 1　　(それ以外のとき)
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≥1

(b) isw1=1, 2, 3, 4, −1, −2, −3, −4

(c) isw2=0 または 1

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	

(6) **注意事項**

(a) isw1 で選択するソート法は以下の通りである.

isw1	ソート法	isw1	ソート法
1	シェル・ソート (昇順)	−1	シェル・ソート (降順)
2	ヒープ・ソート (昇順)	−2	ヒープ・ソート (降順)
3	クイック・ソート (昇順)	−3	クイック・ソート (降順)
4	マージ・ソート (昇順)	−4	マージ・ソート (降順)

利用者は入力データの性質により適切なソート法を選べばよい. 各ソート法の特徴を以下に示す.

· シェル・ソート

計算量は平均 $O(n^{1.5})$ 程度である. どのようなデータに対しても安定して速いソートを行う. とくに, データ系列の一部がソートされている場合は速くなる.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· ヒープ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

ワーク領域が不要である.

· クイック・ソート

計算量は平均 $O(n \log n)$ だが, 最初から部分的にソートされているなど, ある種の規則性があるものに対しては, 大変非効率的なソートになる. ランダムなデータに対してはもっとも速いソート方法である.

第 1 の組のデータに複数個同じ値が存在する場合, それに対応する第 2 の組のデータの順序がソートを行う前後で保たれる保証はない.

· マージ・ソート

計算量は $O(n \log n)$ だが, 定数項部分は大きめである. 入力データの性質によりソート時間があまり変わらない.

同じ値のデータ間で整列前の順序関係が保たれる.

(7) **使用例**

(a) 問 題

a [0] = 5.0, b [0] = 3.0
a [1] = 4.0, b [1] = 4.0
a [2] = 9.0, b [2] = 2.0
a [3] = 6.0, b [3] = 3.0
a [4] = 2.0, b [4] = 8.0
a [5] = 5.0, b [5] = 1.0

の a をシェル・ソートで昇順にソートし, それに対応するように b を並べかえる. 2 次ソートも行う.

(b) 入力データ

配列 a, 配列 b, n=6, isw1=1, isw2=1

(c) 主プログラム

```
/*      C interface example for ASL_dssta2 */
#include <stdio.h>
```

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

int main()
{
    double *a,*b,*wk;
    int n,isw1,isw2,*iwk,ierr;
    int i;
    FILE *fp;


    fp = fopen( "dssta2.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    isw1=1;
    isw2=1;
    n=6;

    printf( "    *** ASL_dssta2 ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

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

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

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

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

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

    fclose( fp );

    ierr = ASL_dssta2(a, n, b, isw1, isw2, wk, iwk);

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

    printf( "\t   <Array a>    <Array b>\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g     %8.3g\n", a[i], b[i] );
    }

    free( a );
    free( b );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
    *** ASL_dssta2 ***

     ** Input **

    n=  6

       <Array a>    <Array b>
           5            3
           4            4
           9            2
           6            3
```

```
       2            8
       5            1

 ** Output **

ierr =      0

   <Array a>    <Array b>
       2            8
       4            4
       5            1
       5            3
       6            3
       9            2
```

3.3 順位付け

3.3.1 ASL_dsstra, ASL_rsstra
データ列の順位付け

(1) **機 能**

n 個のデータが与えられた場合の各データに対応する昇順順位番号と同一順位にあるデータの数を返す (注意事項 (a) 参照).

n 個のデータ $a_i(i=1,2,\cdots,n)$ が与えられ, これを昇順に並べ換えたデータ列が a_j $(j=j_1,j_2,\cdots,j_{m_1+\cdots+m_k})$:

$$\begin{aligned} a_{j_1} = \quad & a_{j_2} \cdots = a_{j_{m_1}} \leq \\ & a_{j_{m_1+1}} = a_{j_{m_1+2}} \cdots = a_{j_{m_1+m_2}} \leq \\ & \cdots \leq \\ & a_{j_{m_1+\cdots+m_{k-1}+1}} = a_{j_{m_1+\cdots+m_{k-1}+2}} \cdots = a_{j_{m_1+\cdots+m_k}} \end{aligned}$$

$(m_1+\cdots+m_k=n)$ で与えられるとき $r_{j_{m_1+\cdots+m_{\ell-1}+1}} = r_{j_{m_1+\cdots+m_{\ell-1}+2}} = \cdots = r_{j_{m_1+\cdots+m_\ell}} = \ell$ で定義される順位データ $r_j(j=1,2,\cdots,n)$ を求める. ここで, m_ℓ は ℓ 番目に小さいデータに対する同一順位の数である. なお, 同一順位の数を求める場合には $c_j(j=1,2,\cdots,n)$ に $c_{j_{m_1+\cdots+m_{\ell-1}+1}} = c_{j_{m_1+\cdots+m_{\ell-1}+2}} = \cdots = c_{j_{m_1+\cdots+m_\ell}} = m_\ell$ を満たす様にデータを格納する.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dsstra  (a, n, ir, ic, isw, iw);
```

単精度関数:

```
ierr = ASL_rsstra  (a, n, ir, ic, isw, iw);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{ D∗ / R∗ }	n	入 力	順位付けられるデータ a_i
2	n	I	1	入 力	配列 a の大きさ
3	ir	I*	n	出 力	a に対応づけられて, 与えられる順位 r_j
4	ic	I*	n	出 力	isw=0 のとき 同順位のデータの個数 c_j isw=1 のとき 使用しない (注意事項 (b) 参照)
5	isw	I	1	入 力	同順位データ数出力スイッチ isw=0: ic に同順位のデータの個数を出力する. isw=1: 同順位のデータの個数は出力しない.
6	iw	I*	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) n≥2

(b) isw=0 または isw=1

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) a[i] は全体のうちで ir[i] 番目に小さいデータであり, isw=0 のとき, ir[i] 番目に小さいデータは ic[i] 個ある.

(b) isw=1 のとき, ic は使用されないので, ダミー配列を引数に設定することができる.

(7) **使用例**

(a) 問 題

a [0] = 1.2
a [1] = 3.2
a [2] = 4.2
a [3] = 5.2
a [4] = 7.2
a [5] = 1.2
a [6] = 9.2
a [7] = 1.2
a [8] = 1.2
a [9] = 7.2
a [10] = 6.2
a [11] = 8.2
a [12] = 7.2
a [13] = 5.2
a [14] = 0.2
a [15] = 2.2

に順位をつける.

(b) 入力データ

配列 a, n=16, isw=0

(c) 主プログラム

```
/*      C interface example for ASL_dsstra */

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

int main()
{
    double *a;
    int n,*ir,*ic,isw,*iw,ierr;
    int i;
    FILE *fp;
```

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

    n=16;
    isw=0;

    printf( "    *** ASL_dsstra ***\n" );
    printf( "\n     ** Input **\n\n" );
    printf( "\tn=%3d\n\n", n );

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

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

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

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

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

    fclose( fp );

    ierr = ASL_dsstra(a, n, ir, ic, isw, iw);

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

    printf( "\tArray  a    ir    ic\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%8.3g  %4d  %4d\n", a[i],ir[i],ic[i] );
    }

    free( a );
    free( ir );
    free( ic );
    free( iw );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dsstra ***

 ** Input **

n= 16

Array  a
     1.2
     3.2
     4.2
     5.2
     7.2
     1.2
     9.2
     1.2
     1.2
     7.2
     6.2
     8.2
     7.2
     5.2
     0.2
     2.2

 ** Output **
```

```
ierr =      0

Array  a    ir    ic
     1.2     2     4
     3.2     7     1
     4.2     8     1
     5.2     9     2
     7.2    12     3
     1.2     2     4
     9.2    16     1
     1.2     2     4
     1.2     2     4
     7.2    12     3
     6.2    11     1
     8.2    15     1
     7.2    12     3
     5.2     9     2
     0.2     1     1
     2.2     6     1
```

3.3.2 ASL_dsstpt, ASL_rsstpt
上位 N 件の抽出

(1) **機 能**

n 個のデータ $a_i(i = 1, 2, \cdots, n)$ が与えられたときこれを大きい順または小さい順に並べ換えたデータ列のうち先頭から m 個のデータ $a_j(j = j_1, j_2, \cdots, j_m)$ $(m < n)$ を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dsstpt  (a, n, & m, & p, isw);
```

単精度関数:

```
ierr = ASL_rsstpt  (a, n, & m, & p, isw);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	n	入 力	入力データ a_i
				出 力	大きい順または小さい順に整列されたデータ a_j
2	n	I	1	入 力	配列 a の大きさ
3	m	I*	1	入 力	大きい順または小さい順に整列するデータの数 (注意事項 (a) 参照)
				出 力	実際に大きい順または小さい順に整列されたデータの数
4	p	{D∗ / R∗}	1	入 力	閾値の初期値を求めるためのパラメータ (注意事項 (b) 参照)
				出 力	閾値の更新回数 (注意事項 (b) 参照)
5	isw	I	1	入 力	isw=0:小さい順に整列する isw=1:大きい順に整列する (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) isw∈ {0, 1}

(b) m≤0, n≤0, n<m

(5) **エラーインディケータ (戻り値)**

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

(6) **注意事項**

(a) 配列 a のはじめの m 個の要素に, 配列 a の要素を大きい順 (isw=1 の場合), または小さい順 (isw=0 の場合) に整列した場合の結果が求まる. なお, m には, 整列したい要素数を入力し, 実際に整列された要素数が出力される. ここで, (整列したい要素数) ≤(実際に整列された要素数) である.

(b) この関数では, ある閾値をもとに入力データを逐次この閾値よりも大きい集合と小さい集合とに分割しながら処理を行う. このようにして得られた集合の大きさが整列したい要素数に近付いたときその集合について並べ替えを行う. 最初の閾値は, パラメータ p に与えたデータをもとに次のように計算する.

$$\text{閾値の初期値} = \max \times \mathrm{p} + \min \times (1.0 - \mathrm{p})$$

ここで max は配列 a に含まれるデータの最大値, min は配列 a に含まれるデータの最小値をそれぞれ表す. したがって閾値の初期値は max と min とを $(1.0 - \mathrm{p}) : \mathrm{p}$ に内分する点として定義される. 並べ替えたいデータの性質が分かっている場合には p すなわち閾値の初期値に適切なデータを与えることによって処理スピードをあげることができる. 例えば n 個の (0, 1) 一様乱数が与えられてそのうち小さい方から m 個のデータを取り出したい場合, 最適な閾値の推定値は $\frac{m}{n}$ であり, したがって

$$\mathrm{p} = \frac{m}{n}$$

と指定すればよい. なお, p の出力には実際に閾値の更新を行った回数を$\left\{\begin{matrix}\text{倍精度}\\ \text{単精度}\end{matrix}\right\}$実数として出力する. この値が小さい程閾値の初期値が適切であったことを示している.

(7) **使用例**

(a) 問 題

a [0] = 5.0　a [1] = 39.0　a [2] = 15.0　a [3] = 8.0　a [4] = 23.0
a [5] = 45.0　a [6] = 61.0　a [7] = 25.0　a [8] = 33.0　a [9] = 45.0
a [10] = 39.0　a [11] = 10.0　a [12] = 21.0　a [13] = 5.0　a [14] = 23.0
a [15] = 38.0　a [16] = 41.0　a [17] = 55.0　a [18] = 61.0　a [19] = 39.0

を昇順にソートした場合の小さなものから 5 番目までを求める.

(b) 入力データ

配列 a, n=20, m=5, p=0.3, isw=0

(c) 主プログラム

```
/*      C interface example for ASL_dsstpt */

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

int main()
{
    double *a,p;
    int n,m,isw,ierr;
    int i;
    FILE *fp;

    isw=0;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%lf", &p );
    fscanf( fp, "%d", &isw );
```

```
    printf( "\tn  =%3d\n", n );
    printf( "\tm  =%3d\n", m );
    printf( "\tisw=%3d\n", isw );
    printf( "\tp  =%5.3g\n\n", p );

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

    printf( "\tArray a" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &a[i] );
        if( i%5 == 0 )
        {
            printf( "\n\t" );
        }
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    fclose( fp );

    ierr = ASL_dsstpt(a, n, &m, &p, isw);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\tm=%3d\n\n", m );

    printf( "\tArray a" );
    for( i=0 ; i<m ; i++ )
    {
        printf( "%8.3g", a[i] );
    }
    printf( "\n" );

    free( a );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dsstpt ***

 ** Input **

n  = 20
m  =  5
isw=  0
p  =  0.3

Array a
       5      39      15       8      23
      45      61      25      33      45
      39      10      21       5      23
      38      41      55      61      39

 ** Output **

ierr =      0

m=  6

Array a       5       5       8      10      15      21
```

3.4 マージ

3.4.1 ASL_dsmgon, ASL_rsmgon
ソート済みデータ列のマージ

(1) **機 能**

昇順に整列された 2 つのデータ列 $a_i \quad (i = 1, 2, \cdots, n)$ と $b_j \quad (j = 1, 2, \cdots, m)$ をマージして, データ列 $c_k \quad (k = 1, 2, \cdots, \ell)$ を求める.

ただし, c_k は

$$c_1 \leq c_2 \leq \cdots \leq c_\ell$$

を満たす.

(2) **使用法**

倍精度関数:

ierr = ASL_dsmgon (a, nn, b, nm, c, nl);

単精度関数:

ierr = ASL_rsmgon (a, nn, b, nm, c, nl);

(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∗}	nn	入 力	マージされるデータ a_i
2	nn	I	1	入 力	配列 a の大きさ
3	b	{D∗ / R∗}	nm	入 力	マージされるデータ b_j
4	nm	I	1	入 力	配列 b の大きさ
5	c	{D∗ / R∗}	nl	出 力	マージされたデータ c_k
6	nl	I	1	入 力	配列 c の大きさ
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) nn $\geq$ 1

(b) nm $\geq$ 1

(c) 1 $\leq$ nl $\leq$ nn + nm

(d) a [0] $\leq$ a [1] $\leq \cdots \leq$ a [nn-1]

(e) b [0] $\leq$ b [1] $\leq \cdots \leq$ b [nm-1]

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (c) を満足しなかった.	nl = nn + nm として処理する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (d) を満足しなかった.	
3300	制限条件 (e) を満足しなかった.	

(6) 注意事項

(a) nl < nn + nm のとき, 小さいものから nl 個のみマージした結果が出力される.

(7) 使用例

(a) 問 題

配列 x に格納された n 個のデータからなる数列 a_i $(i = 1, 2, \cdots, n)$ を n_s 個ずつの部分列に分割して, 部分列間のマージを繰り返すことにより元の数列全体を昇順にソートする. なお, プログラム内の配列のかわりに外部記憶装置上のファイルを用いれば, この問題は n 個のデータ a_i を外部ソートする問題となる.

(b) 入力データ

配列 x, 数列の長さ n

(c) 主プログラム

```
/*      C interface example for ASL_dsmgon */

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

int main()
{
    double *a,*b,*c,*x;
    int n,ierr;
    int i,j,k;
    int ista,istb,istc,isizea,isizeb,isizec,isize,isize2;
    int iloop,icrest,ia,ib,ic,na;
    FILE *fp;

    na = 100;
    fp = fopen( "dsmgon.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

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

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

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

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

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

    c = ( double * )malloc((size_t)( sizeof(double) * na ));
```

```
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}

fclose( fp );

printf( "\tInput data\n\n" );
printf("\tn  = %6d\n\n", n );

for( i=0 ; i<n ; i++ )
{
    x[i] = (double)((int)(sin((double)i+1.0)*100.0));
    printf( "\t%8.3g\n", x[i] );

}

for( i=0 ; i<n ; i++ )
{
    c[i] = x[i];
}

iloop = 1;
while(1<<iloop<n){
    iloop++;
}

for(i=0;i<iloop;i++){
    isize = 1<<i;
    isize2 = 1<<(i+1);

    ia = 0;
    ib = 0;
    ic = 0;
    for(j=0;j<n/isize2;j++){
        for(k=0;k<isize;k++){
            ia++;
            a[ia-1] = c[ic+k];
        }
        ic += isize;
        for(k=0;k<isize;k++){
            ib++;
            b[ib-1] = c[ic+k];
        }
        ic += isize;
    }
    icrest = n - ic;
    if( (0<icrest) && (icrest<=isize)){
        for(k=0;k<icrest;k++){
            ia++;
            a[ia-1] = c[ic+k];
        }
    }
    if( (isize<icrest) && (icrest<isize*2) ){
        for(k=0;k<isize;k++){
            ia++;
            a[ia-1] = c[ic+k];
        }
        ic += isize;
        for(k=0;k<icrest-isize;k++){
            ib++;
            b[ib-1] = c[ic+k];
        }
    }

    for(j=0;j<n/isize2;j++){
        ista = j*isize;
        istb = j*isize;
        istc = j*isize2;
        isizea = isize;
        isizeb = isize;
        isizec = isizea + isizeb;
        ierr = ASL_dsmgon
        (a+ista,isizea,b+istb,isizeb,c+istc,isizec);
    }
    if( (0<icrest) && (icrest<=isize) ){
        ista = n/isize2*isize;
        istc = n/isize2*isize2;
        isizea = icrest;
        for(k=0;k<isizea;k++){
            c[istc+k] = a[ista+k];
        }
    }
    if( (isize<icrest) && (icrest<isize*2) ){
        ista = n/isize2*isize;
        istb = n/isize2*isize;
        istc = n/isize2*isize2;
        isizea = isize;
        isizeb = icrest-isize;
        isizec = isizea+isizeb;
        ierr =
        ASL_dsmgon(a+ista,isizea,b+istb,isizeb,c+istc,isizec);
    }
}

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

```
        printf( "\tOutput data\n\n" );
        for(i=0;i<n;i++){
            printf( "\t%8.3g\n", c[i] );
        }

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

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dsmgon ***

         ** Input **

        Input data

        n  =     17

              84
              90
              14
             -75
             -95
             -27
              65
              98
              41
             -54
             -99
             -53
              42
              99
              65
             -28
             -96

         ** Output **

        ierr =      0

        Output data

             -99
             -96
             -95
             -75
             -54
             -53
             -28
             -27
              14
              41
              42
              65
              65
              84
              90
              98
              99
```

3.4.2 ASL_dsmgpa, ASL_rsmgpa
ソート済みペアデータ列のマージ

(1) **機 能**

a_i について昇順に整列されたデータの組 (a_i, b_i) $(i = 1, 2, \cdots, n)$ と c_j について昇順に整列されたデータの組 (c_j, d_j) $(j = 1, 2, \cdots, m)$ をマージして, データの組 (e_k, f_k) $(k = 1, 2, \cdots, \ell)$ を求める.
ただし, e_k は

$$e_1 \leq e_2 \leq \cdots \leq e_\ell$$

を満たす. なお, 2 次ソートを指定した場合には, $e_k = e_{k+1}$ を満たす任意の k について

$$f_k \leq f_{k+1}$$

を満たすように $k = 1, 2, \cdots, \ell$ を決める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dsmgpa  (a, nn, b, c, nm, d, e, nl, f, isw);
```

単精度関数:

```
ierr = ASL_rsmgpa  (a, nn, b, c, nm, d, e, nl, f, 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∗}	nn	入 力	マージされるデータ a_i
2	nn	I	1	入 力	配列 a の大きさ
3	b	{D∗ / R∗}	nn	入 力	a_i に対応するデータ b_i
4	c	{D∗ / R∗}	nm	入 力	マージされるデータ c_j
5	nm	I	1	入 力	配列 c の大きさ
6	d	{D∗ / R∗}	nm	入 力	c_j に対応するデータ d_j
7	e	{D∗ / R∗}	nl	出 力	マージされたデータ e_k
8	nl	I	1	入 力	配列 e の大きさ
9	f	{D∗ / R∗}	nl	出 力	e_k に対応するデータ f_k
10	isw	I	1	入 力	2 次ソートスイッチ isw=0 : 2 次ソートは行わない isw=1 : 2 次ソートを行う
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) nn ≥ 1

(b) nm ≥ 1

(c) $1 \leq$ nl $\leq$ nn + nm

(d) a [0] $\leq$ a [1] $\leq \cdots \leq$ a [nn-1]

(e) c [0] $\leq$ c [1] $\leq \cdots \leq$ c [nm-1]

(f) isw=0 または isw=1

(g) isw=1 を指定した場合には a [i] =a [i+1] を満たす任意の i について b [i] $\leq$b [i+1] を満たすこと

(h) isw=1 を指定した場合には c [j] =c [j+1] を満たす任意の j について d [j] $\leq$d [j+1] を満たすこと

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (c) を満足しなかった.	nl = nn + nm として処理する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (d) を満足しなかった.	
3300	制限条件 (e) を満足しなかった.	
3400	制限条件 (f) を満足しなかった.	
3500	制限条件 (g) を満足しなかった.	
3600	制限条件 (h) を満足しなかった.	

(6) **注意事項**

(a) nl < nn + nm のとき, 小さいものから nl 個のみマージした結果が出力される.

(7) **使用例**

(a) 問 題

配列 x と y に格納された n 個のデータの組からなる列 (a_i, b_i) $(i = 1, 2, \cdots, n)$ を n_s 個ずつの部分列に分割して, 部分列間のマージを繰り返すことにより元の n 個のデータの組全体を昇順にソートする. 2 次ソートも行う. なお, プログラム内の配列のかわりに外部記憶装置上のファイルを用いれば, この問題は n 個のデータの組 (a_i, b_i) を外部ソートする問題となる.

(b) 入力データ

配列 x, 数列の長さ n, isw=1

(c) 主プログラム

```
/*      C interface example for ASL_dsmgpa */

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

int main()
{
    double *a,*a2,*b,*b2,*c,*c2,*x,*y;
    int n,isw,ierr;
    int i,j,k;
    int ista,istb,istc,isizea,isizeb,isizec,isize,isize2;
    int iloop,icrest,ia,ib,ic,na;
    FILE *fp;
```

```
na = 100;
isw = 1;
fp = fopen( "dsmgpa.dat", "r" );
if( fp == NULL )
{
    printf( "file open error\n" );
    return -1;
}

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

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

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

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

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

a2 = ( double * )malloc((size_t)( sizeof(double) * na ));
if( a2 == NULL )
{
    printf( "no enough memory for array a2\n" );
    return -1;
}

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

b2 = ( double * )malloc((size_t)( sizeof(double) * na ));
if( b2 == NULL )
{
    printf( "no enough memory for array b2\n" );
    return -1;
}

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

c2 = ( double * )malloc((size_t)( sizeof(double) * na ));
if( c2 == NULL )
{
    printf( "no enough memory for array c2\n" );
    return -1;
}

fclose( fp );

printf( "\tInput data\n\n" );

printf( "\t     n = %6d\n\n", n );
printf( "\t   isw = %6d\n\n", isw );
for( i=0 ; i<n ; i++ )
{
    x[i] = (double)((int)(sin((double)i+1.0)*100.0));
    y[i] = (double)((int)(sin((double)i+1.0+0.5*M_PI)*100.0));
    printf( "\t%8.3g %8.3g\n", x[i], y[i] );

}

for( i=0 ; i<n ; i++ )
{
    c[i] = x[i];
    c2[i] = y[i];
}

iloop = 1;
while(1<<iloop<n){
    iloop++;
}

for(i=0;i<iloop;i++){
    isize = 1<<i;
    isize2 = 1<<(i+1);
```

```
            ia = 0;
            ib = 0;
            ic = 0;
            for(j=0;j<n/isize2;j++){
                for(k=0;k<isize;k++){
                    ia++;
                    a[ia-1] = c[ic+k];
                    a2[ia-1] = c2[ic+k];
                }
                ic += isize;
                for(k=0;k<isize;k++){
                    ib++;
                    b[ib-1] = c[ic+k];
                    b2[ib-1] = c2[ic+k];
                }
                ic += isize;
            }
            icrest = n - ic;
            if( (0<icrest) && (icrest<=isize)){
                for(k=0;k<icrest;k++){
                    ia++;
                    a[ia-1] = c[ic+k];
                    a2[ia-1] = c2[ic+k];
                }
            }
            if( (isize<icrest) && (icrest<isize*2) ){
                for(k=0;k<isize;k++){
                    ia++;
                    a[ia-1] = c[ic+k];
                    a2[ia-1] = c2[ic+k];
                }
                ic += isize;
                for(k=0;k<icrest-isize;k++){
                    ib++;
                    b[ib-1] = c[ic+k];
                    b2[ib-1] = c2[ic+k];
                }
            }

            for(j=0;j<n/isize2;j++){
                ista = j*isize;
                istb = j*isize;
                istc = j*isize2;
                isizea = isize;
                isizeb = isize;
                isizec = isizea + isizeb;
                ierr = ASL_dsmgpa
                (a+ista,isizea,a2+ista,b+istb,isizeb,b2+istb,
                 c+istc,isizec,c2+istc,isw);
            }
            if( (0<icrest) && (icrest<=isize) ){
                ista = n/isize2*isize;
                istc = n/isize2*isize2;
                isizea = icrest;
                for(k=0;k<isizea;k++){
                    c[istc+k] = a[ista+k];
                    c2[istc+k] = a2[ista+k];
                }
            }
            if( (isize<icrest) && (icrest<isize*2) ){
                ista = n/isize2*isize;
                istb = n/isize2*isize;
                istc = n/isize2*isize2;
                isizea = isize;
                isizeb = icrest-isize;
                isizec = isizea+isizeb;
                ierr = ASL_dsmgpa
                (a+ista,isizea,a2+ista,b+istb,isizeb,b2+istb,
                 c+istc,isizec,c2+istc,isw);
            }
        }

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

        printf( "\tOutput data\n\n" );
        for(i=0;i<n;i++){
            printf( "\t%8.3g %8.3g\n", c[i], c2[i] );
        }

        free( x );
        free( y );
        free( a );
        free( a2 );
        free( b );
        free( b2 );
        free( c );
        free( c2 );

        return 0;
    }
```

(d) 出力結果

```
    *** ASL_dsmgpa ***

     ** Input **
```

```
Input data

     n =     17

   isw =      1

      84       54
      90      -41
      14      -98
     -75      -65
     -95       28
     -27       96
      65       75
      98      -14
      41      -91
     -54      -83
     -99        0
     -53       84
      42       90
      99       13
      65      -75
     -28      -95
     -96      -27

 ** Output **

ierr =      0

Output data

     -99        0
     -96      -27
     -95       28
     -75      -65
     -54      -83
     -53       84
     -28      -95
     -27       96
      14      -98
      41      -91
      42       90
      65      -75
      65       75
      84       54
      90      -41
      98      -14
      99       13
```

第 4 章　方程式の根

4.1　概 要

本章では, 代数方程式や非線形方程式, 連立非線形方程式の根を求める関数について説明する.
本ライブラリでは, 代数方程式として次の 2 種類を用意している.

(1) 実係数実数型入力で複素根実数型出力タイプ

(2) 複素係数復素型入力で複素根複素型出力タイプ

また非線形方程式としては次の 3 種類を用意している.

(1) 初期値を与えて 1 根を求めるもの

(2) 区間を与えて 1 根を求めるもの

(3) 区間内のすべての根を求めるもの

このうち初期値を与えて 1 根を求める実数型のものは, 初期値が根から遠い, あるいは根と初期値の間で関数が振動する場合でも根が求まるように特別の配慮がなされている.
連立非線形方程式では, ヤコビ行列計算関数が与えられる場合と与えられない場合がある. どちらも大域的収束性があり, 連立する各式がスケーリングされていなくても解けるような配慮がされている.

4.1.1 使用上の注意

(1) 代数方程式は, 実係数代数方程式については 0 に近い根から求まる傾向があるが, 複素係数代数方程式についてはこの傾向はない.

(2) 重根があると, 根の多重度を n としてそこでの解の精度は, 代数方程式では $\sqrt[n]{(\text{誤差判定のための単位})}$ 程度, 初期値を与えて解を得る非線形方程式では $n \times$ (要求精度) 程度になる.

(3) 非線形方程式の収束判定は, 区間内の 1 根を求めるものを除き, e_r を要求精度, e_r を誤差判定のための単位, Δx を x の更新量として

$$|\Delta x| < e_r \max(1, |x|) \text{ and } |f(x)| < e_r + 64\varepsilon|x|$$

または

$$f(x) = 0$$

が成立すれば収束したものと見なす. すなわち, 関数値が 0 または, 関数値も解の動きもともに 0 に近いときに収束したものとする. これにより条件の悪い場合でも正しく収束判定がなされ, しかも重根の場合には演算量が増えるが精度を保つことができる.

ただし, 解の動きのみで判定する方法や関数値のみで判定する方法, または解の動きと関数値のいずれかで判定する方法に比べて判定がきびしくなる傾向がある.

(4) 連立非線形方程式の収束判定は,
$(|\Delta x_i| < e_r \max(1, |x_i|) \text{ and } \|f(x)\|_\infty < e_r + 64\varepsilon|x_i|)$ or $f_i(x) = 0$
$(i = 1, \cdots,$ 元数) がすべての i に対し成立したとき収束したものとする.

(5) 関数 f, df の作り方は次のようにする
例 非線形方程式

```
/* C interface example for ASL_dlnrds */
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <asl.h>
double FORTRAN f(double *x) ··· 主プログラムにおいて, 第 1 引数 f と同じ名前とする.
{
      return (f(*x));
}
double FORTRAN df(double *x) ··· 主プログラムにおいて, 第 1 引数 df と同じ名前とする.
{
      return (f'(*x));
}
int main()
{
         ≀
      ierr=ASL_dinrds(f, df, & x, er, m);
         ≀
}
```

例 連立非線形方程式

```
/* C interface example for ASL_disrds */
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <asl.h>
void FORTRAN sub(double *x, int *n, double *f) ··· 主プログラムにおいて, 第 1 引数 sub
と同じ名前とする.
{
      f[0]=f_1(x_1, \cdots, x_{(*n)});
```

```
        ≀
      f[(*n)-1]=f_(*n)(x_1,···,x_(*n));
}
void FORTRAN subj(double *x, int *n, double *f) ··· 主プログラムにおいて，第2引数 subj
と同じ名前とする．
{
      a[0]=∂f_1/∂x_1;
        ≀
      a[(*n)*(*n)-1]=∂f_(*n)/∂x_(*n);
}
int main()
{
        ≀
      ierr=ASL_dlsrds
        (sub, subj, x, n, er, m, isw, ip, wk, dwk);
        ≀
}
```

(6) いくつかのエラーが重なって発生したとき，エラーインディケータ (戻り値) には最も重大なエラー値が出力されるので, 他のエラー情報が隠れてしまう場合がある．

4.1.2 使用しているアルゴリズム

4.1.2.1 実係数代数方程式の根

4.1.2.1.1 次数 $n = 2$ の場合

2 次方程式の根の公式を用いてつぎのように根を求める.
2 次方程式

$$x^2 + a_1 x + a_0 = 0$$

において

$$\begin{aligned} r &= -\frac{a_1}{2} \\ D &= r^2 - a_0 \end{aligned}$$

とすると

(1) $|D| \leq \varepsilon$ のとき

$$x = r, r$$

(2) $D < 0$ のとき

$$x = (r, \pm\sqrt{-D})$$

(3) $D > 0$ のとき

$$x = \alpha, \frac{a_0}{\alpha}$$

ただし

$$\alpha = \begin{cases} r + \sqrt{D} & (r \geq 0) \\ r - \sqrt{D} & (r < 0) \end{cases}$$

である.

4.1.2.1.2 次数 $n = 3$ の場合

カルダノ法を用いて次のように根を求める.
3 次方程式

$$x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

において

(1) $|a_0| \leq \varepsilon$ の場合 $(x(x^2 + a_2 x + a_1) = 0)$
(4.1.2.1.1) に述べた方法により 2 次方程式

$$x^2 + a_2 x + a_1 = 0$$

を解き根 α, β を求めることによって

$$x = 0, \alpha, \beta$$

と求まる.

(2) $|a_2| \leq \varepsilon$ の場合 $(x^3 + a_1 x + a_0 = 0)$

(a) $|a_1| \leq \varepsilon$ のとき $(x^3 + a_0 = 0)$

$$x = r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2} \right)$$

ただし

$$r = \begin{cases} -\sqrt[3]{a_0} & (a_0 \geq 0) \\ \sqrt[3]{-a_0} & (a_0 < 0) \end{cases}$$

である.

(b) それ以外のとき

$$b_1 = \frac{a_1}{3}$$
$$b_0 = -\frac{a_0}{2}$$

として,

$$x^3 + 3b_1 x - 2b_0 = 0$$

を次の (3) に示す方法を用いて解く.

(3) それ以外の場合

3 次方程式

$$x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

は

$$x = y - YMX$$
$$YMX = \frac{a_2}{3}$$

なる変数変換によって

$$y^3 + 3b_1 y - 2b_0 = 0$$

と変形できる.

ここで,

$$b_1 = \frac{3a_1 - a_2^2}{9}$$
$$b_0 = \frac{(9a_1 - 2a_2^2)a_2 - 27a_0}{54}$$

である.

(a) $|b_0| \leq \varepsilon$ の場合 $(y(y^2 + 3b_1) = 0)$

i. $b_1 < 0$ のとき

$$y = 0, \pm\sqrt{-3b_1}$$
$$x = -YMX, \pm\sqrt{-3b_1} - YMX$$

なお, 計算上桁落ちが生じる場合には根と係数の関係を用いることで桁落ちを防止する.

ii. $b_1 \geq 0$ のとき

$$\begin{aligned} y &= 0, (0, \pm\sqrt{-3b_1}) \\ x &= -YMX, (-YMX, \pm\sqrt{-3b_1}) \end{aligned}$$

(b) $|b_1| \leq \varepsilon$ の場合 $(y^3 - 2b_0 = 0)$

$$\begin{aligned} y &= r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2}\right) \\ x &= r - YMX, \left(-\frac{r}{2} - YMX, \pm\frac{\sqrt{3}r}{2}\right) \end{aligned}$$

ただし,

$$r = \begin{cases} \sqrt[3]{2b_0} & (b_0 \geq 0) \\ -\sqrt[3]{-2b_0} & (b_0 < 0) \end{cases}$$

なお, 計算上桁落ちが生じる場合には根と係数の関係を用いることで桁落ちを防止する.

(c) それ以外の場合 $(y^3 + 3b_1 y - 2b_0 = 0)$

いま,

$$y = s + t$$

とおくと,

$$s^3 + t^3 - 2b_0 + 3(st + b_1)(s + t) = 0$$

が成立する.

s, t を

$$\begin{cases} st &= -b_1 \\ s^3 + t^3 &= 2b_0 \end{cases}$$

が成り立つように決定すれば, これから y を求めることができる. s^3, t^3 は

$$z^2 - 2b_0 z - b_1^3 = 0$$

の 2 根となっている.

$$\begin{aligned} D &= b_0^2 + b_1^3 = D_s^2 D_d \\ D_s &= \begin{cases} b_0 & (|b_0| \geq |b_1|) \\ b_1 & (|b_0| < |b_1|) \end{cases} \\ D_d &= \begin{cases} 1 + b_1(\dfrac{b_1}{b_0})^2 & (|b_0| \geq |b_1|) \\ (\dfrac{b_0}{b_1})^2 + b_1 & (|b_0| < |b_1|) \end{cases} \end{aligned}$$

とおくと

i. $|D_d| \leq \varepsilon^2$ の場合 $((z - b_0)^2 = 0)$

$$\begin{aligned} s^3 &= t^3 = b_0 \\ s, t &= r, \left(-\frac{r}{2}, \pm\frac{\sqrt{3}r}{2}\right) \end{aligned}$$

ただし,

$$r = \begin{cases} \sqrt[3]{b_0} & (b_0 \geq 0) \\ -\sqrt[3]{-b_0} & (b_0 < 0) \end{cases}$$

s, t は $st = -b_1$ を満たすことから

$$\begin{aligned} y &= 2r, -r, -r \\ x &= 2r - YMX, -r - YMX, -r - YMX \end{aligned}$$

が得られる.

なお, 計算上桁落ちが生じる場合には根と係数の関係を用いることで桁落ちを防止する.

ii. $D < 0$ の場合

$$\begin{aligned} s^3 &= (b_0, \sqrt{-D}) \\ t^3 &= (b_0, -\sqrt{-D}) \end{aligned}$$

いま,

$$|s^3|^2 = |t^3|^2 = -b_1^3$$

であるから,

$$\begin{aligned} s &= \frac{r}{2}e^{\sqrt{-1}\frac{\theta}{3}}, \frac{r}{2}e^{\sqrt{-1}(\pi+\frac{\theta-\pi}{3})}, \frac{r}{2}e^{\sqrt{-1}(\frac{\theta+\pi}{3}-\pi)} \\ t &= \frac{r}{2}e^{-\sqrt{-1}\frac{\theta}{3}}, \frac{r}{2}e^{-\sqrt{-1}(\pi+\frac{\theta-\pi}{3})}, \frac{r}{2}e^{-\sqrt{-1}(\frac{\theta+\pi}{3}-\pi)} \end{aligned}$$

したがって,

$$\begin{aligned} y &= r\cos\frac{\theta}{3}, -r\cos\frac{\pi-\theta}{3}, -r\cos\frac{\pi+\theta}{3} \\ x &= r\cos\frac{\theta}{3} - YMX, -r\cos\frac{\pi-\theta}{3} - YMX, -r\cos\frac{\pi+\theta}{3} - YMX \end{aligned}$$

ただし,

$$\begin{aligned} r &= \begin{cases} -2\sqrt{b_1} & (b_1 \geq 0) \\ 2\sqrt{-b_1} & (b_1 < 0) \end{cases} \\ \theta &= \begin{cases} \tan^{-1} D_\theta & (b_0 \geq 0) \\ \pi - \tan^{-1} D_\theta & (b_0 < 0) \end{cases} \\ D_\theta &= \frac{|D|}{|b_0|} = \begin{cases} \sqrt{|D_d|} & (|b_0| \geq |b_1|) \\ \dfrac{|D_s|\sqrt{|D_d|}}{|b_0|} & (|b_0| < |b_1|) \end{cases} \end{aligned}$$

なお, 計算上桁落ちが生じる場合には根と係数の関係を用いることで桁落ちを防止する.

iii. $D > 0$ の場合

$$\begin{aligned} s^3 &= \alpha \\ t^3 &= \beta \\ \alpha &= \begin{cases} b_0 + \sqrt{D} = |D_s| r & (b_0 \geq 0) \\ b_0 - \sqrt{D} = -|D_s| r & (b_0 < 0) \end{cases} \\ \beta &= -\frac{b_1^3}{\alpha} \end{aligned}$$

ただし,

$$r = \frac{|b_0| + \sqrt{D}}{|D_s|} = \begin{cases} 1 + \sqrt{|D_d|} & (|b_0| \geq |b_1|) \\ \dfrac{|b_0|}{|D_s|} + \sqrt{|D_d|} & (|b_0| < |b_1|) \end{cases}$$

である. したがって,

$$\begin{aligned} s &= \alpha', \left(-\frac{\alpha'}{2}, \pm\frac{\sqrt{3}\alpha'}{2}\right) \\ t &= \beta', \left(-\frac{\beta'}{2}, \pm\frac{\sqrt{3}\beta'}{2}\right) \end{aligned}$$

ここで,

$$\alpha' = \begin{cases} \sqrt[3]{|\alpha|} & (b_0 \geq 0) \\ -\sqrt[3]{|\alpha|} & (b_0 < 0) \end{cases}$$

$$\beta' = \begin{cases} \sqrt[3]{|\beta|} & (b_0 b_1 < 0) \\ -\sqrt[3]{|\beta|} & (b_0 b_1 \geq 0) \end{cases}$$

s, t は $st = -b_1$ を満たすことから

$$y = u, \left(-\frac{u}{2}, \pm\frac{\sqrt{3}v}{2} \right)$$

$$x = u - YMX, \left(-\frac{u}{2} - YMX, \pm\frac{\sqrt{3}v}{2} \right)$$

ただし,

$$u = \alpha' + \beta', v = \alpha' - \beta'$$

である.

なお, 計算上桁落ちが生じる場合には根と係数の関係を用いることで桁落ちを防止する.

4.1.2.1.3 次数 $n = 4$ の場合

フェラーリ法を用いて次のように根を求める.

4 次方程式

$$x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

において

(1) $|a_0| \leq \varepsilon$ の場合 $(x(x^3 + a_3 x^2 + a_2 x + a_1) = 0)$

(4.1.2.1.2) に述べた方法によって 3 次方程式

$$x^3 + a_3 x^2 + a_2 x + a_1 = 0$$

を解き, その根 α, β, γ を求めることによって

$$x = 0, \alpha, \beta, \gamma$$

と求まる.

(2) $|a_3| < \varepsilon$ の場合 $(x^4 + a_2 x^2 + a_1 x + a_0 = 0)$

(a) $|a_1| \leq \varepsilon$ のとき $(x^4 + a_2 x^2 + a_0 = 0)$

$$r = -\frac{a_2}{2}$$

$$D = r^2 - a_0$$

とおくと

i. $|D| \leq \varepsilon$ のとき $((x^2 - r)^2 = 0)$

$$x = \begin{cases} \sqrt{r}, \sqrt{r}, -\sqrt{r}, -\sqrt{r} & (r \geq 0) \\ (0, \sqrt{-r}), (0, \sqrt{-r}), (0, -\sqrt{-r}), (0, -\sqrt{-r}) & (r < 0) \end{cases}$$

ii. $D < 0$ のとき

$$x^2 = (r, \pm\sqrt{-D})$$

したがって,

$$x = (\alpha, \pm\beta), (-\alpha, \pm\beta)$$

ただし,

$$\begin{cases} \alpha = \sqrt{\dfrac{r+NRD}{2}}, & \beta = \dfrac{\sqrt{-D}}{2\alpha} \quad (r>0) \\ \beta = \sqrt{\dfrac{-r+NRD}{2}}, & \alpha = \dfrac{\sqrt{-D}}{2\beta} \quad (r\le 0) \end{cases}$$

$$NRD = \sqrt{r^2-D} = \begin{cases} |r|\sqrt{1+\dfrac{-D}{r^2}} & (|r| \ge \sqrt{-D}) \\ \sqrt{-D}\sqrt{\dfrac{r^2}{-D}+1} & (|r| < \sqrt{-D}) \end{cases}$$

である.

iii. $D>0$ のとき

$$x^2 = \alpha, \beta$$

ただし,

$$\alpha = \begin{cases} r+\sqrt{D} & (r \ge 0) \\ r-\sqrt{D} & (r<0) \end{cases}$$

$$\beta = \frac{a_0}{\alpha}$$

したがって,

$$x = \begin{cases} \pm\sqrt{\alpha}, \pm\sqrt{\beta} & (r, \beta \ge 0) \\ \pm\sqrt{\alpha}, (0, \pm\sqrt{-\beta}) & (r \ge 0, \beta \le 0) \\ (0 \pm \sqrt{-\alpha}), \pm\sqrt{\beta} & (r<0, \beta \ge 0) \\ (0 \pm \sqrt{-\alpha}), (0 \pm \sqrt{-\beta}) & (r, \beta < 0) \end{cases}$$

である.

(b) それ以外のとき

$$x^4 + a_2x^2 + a_1x + a_0 = 0$$

を次の (3) に示す方法を用いて解く.

(3) それ以外の場合

4 次方程式

$$x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

が与えられた時

$$\begin{aligned} x &= y - YMX \\ YMX &= \frac{a_3}{4} \end{aligned}$$

なる変数変換を行うと

$$y^4 + b_2y^2 + b_1y + b_0 = 0$$

となる.

ただし,

$$\begin{aligned} b_2 &= \frac{8a_2 - 3a_3^2}{8} \\ b_1 &= \frac{8a_1 - a_3(4a_2 - a_3^2)}{8} \\ b_0 &= \frac{256a_0 - a_3(64a_1 - a_3(16a_2 - 3a_3^2))}{256} \end{aligned}$$

である.

(a) $b_1^2 \leq \varepsilon^2 \max(|4b_0b_2|, |b_2^3|)$ のとき $(y^4 + b_2y^2 + b_0 = 0)$

$$\begin{aligned} r &= -\frac{b_2}{2} \\ D &= r^2 - b_0 \end{aligned}$$

とおくと

i. $|D| \leq \varepsilon$ のとき

$$\begin{aligned} (y^2 - r)^2 &= 0 \\ y &= \begin{cases} \pm\sqrt{r}, \pm\sqrt{r} & (r \geq 0) \\ (0, \pm\sqrt{-r}), (0, \pm\sqrt{-r}) & (r < 0) \end{cases} \\ x &= \begin{cases} \pm\sqrt{r} - YMX, \pm\sqrt{r} - YMX & (r \geq 0) \\ (-YMX, \pm\sqrt{-r}), (-YMX, \pm\sqrt{-r}) & (r < 0) \end{cases} \end{aligned}$$

ii. $D < 0$ のとき

$$\begin{aligned} y^2 &= (r, \pm\sqrt{-D}) \\ y &= (\alpha, \pm\beta), (-\alpha, \pm\beta) \\ x &= (\alpha - YMX, \pm\beta), (-\alpha - YMX, \pm\beta) \end{aligned}$$

ただし,

$$\begin{cases} \alpha = \sqrt{\dfrac{r + NRD}{2}}, & \beta = \dfrac{\sqrt{-D}}{2\alpha} \quad (r > 0) \\ \beta = \sqrt{\dfrac{-r + NRD}{2}}, & \alpha = \dfrac{\sqrt{-D}}{2\beta} \quad (r \leq 0) \end{cases}$$

$$NRD = \sqrt{r^2 - D} = \begin{cases} \mid r \mid \sqrt{1 + \dfrac{-D}{r^2}} & (|r| > \sqrt{-D}) \\ \sqrt{-D}\sqrt{\dfrac{r^2}{-D} + 1} & (|r| < \sqrt{-D}) \end{cases}$$

である.

iii. $D > 0$ のとき

$$\begin{aligned} y^2 &= \alpha, \beta \\ \alpha &= \begin{cases} r + \sqrt{D} & (r \geq 0) \\ r - \sqrt{D} & (r < 0) \end{cases} \\ \beta &= \frac{b_0}{\alpha} \\ y &= \begin{cases} \pm\sqrt{\alpha}, \pm\sqrt{\beta} & (r, \beta \geq 0) \\ (0, \pm\sqrt{-\alpha}), \pm\sqrt{\beta} & (r < 0, \beta \geq 0) \\ \pm\sqrt{\alpha}, (0, \pm\sqrt{-\beta}) & (r \geq 0, \beta < 0) \\ (0, \pm\sqrt{-\alpha}), (0, \pm\sqrt{\beta}) & (r, \beta < 0) \end{cases} \\ x &= \begin{cases} \pm\sqrt{\alpha} - YMX, \pm\sqrt{\beta} - YMX & (r, \beta \geq 0) \\ (-YMX, \pm\sqrt{-\alpha}), \pm\sqrt{\beta} - YMX & (r < 0, \beta \geq 0) \\ \pm\sqrt{\alpha} - YMX, (-YMX, \pm\sqrt{-\beta}) & (r \geq 0, \beta < 0) \\ (-YMX, \pm\sqrt{-\alpha}), (-YMX, \pm\sqrt{-\beta}) & (r, \beta < 0) \end{cases} \end{aligned}$$

(b) $|b_0| \leq \varepsilon$ の場合 $(y(y^3 + b_2y + b_1) = 0)$

(4.1.2.1.2) に述べた方法によって 3 次方程式

$$y^3 + b_2y + b_1 = 0$$

を解き根 α, β, γ を求めることによって

$$x = -YMX, \alpha - YMX, \beta - YMX, \gamma - YMX$$

と求まる.

(c) $b_1^2 > 10^{-4}|b_2(b_2^2 - 4b_0)|$ の場合
4 次方程式

$$y^4 + b_2y^2 + b_1y + b_0 = 0$$

の両辺に $py^2 + \frac{p^2}{4}$ を加えて変形すると

$$\left(y^2 + \frac{p}{2}\right)^2 = (p - b_2)y^2 - b_1y + \frac{p^2}{4} - b_0$$

となる. 右辺が一次式の平方の形になるためには

$$b_1^2 - (p - b_2)(p^2 - 4b_0) = 0$$

すなわち

$$p^3 - b_2p^2 - 4b_0p + (4b_2b_0 - b_1^2) = 0$$

を満足していれば良い. (4.1.2.1.2) に述べた方法によってこの方程式を解き, 得られたこの方程式の実根を p ととれば,

$$\left(y^2 + \frac{p}{2}\right)^2 = (p - b_2)\left\{y - \frac{b_1}{2(p - b_2)}\right\}^2$$

が成立するので, 2 次方程式

$$y^2 \pm \sqrt{p - b_2}y \mp \frac{b_1}{2\sqrt{p - b_2}} + \frac{p}{2} = 0 \quad \text{(複合同順)}$$

を解くことによって解が次のように得られる.

$$\begin{aligned} y &= \alpha \pm \sqrt{\beta - \gamma}, -\alpha \pm \sqrt{\beta + \gamma} \\ \alpha &= \frac{\sqrt{p - b_2}}{2} \\ \beta &= -\frac{p + b_2}{4} \\ \gamma &= \frac{4b_1}{\alpha} \end{aligned}$$

ただし, $p \simeq b_2$ の場合, この方法では精度よく解を求められない. $p \simeq b_2$ となるのは

$$|p - b_2| = \left|\frac{b_1^2}{p^2 - 4b_0}\right| \simeq \left|\frac{b_1^2}{b_2^2 - 4b_0}\right| < \delta|b_2|$$

すなわち,

$$b_1^2 < \delta|b_2(b_2^2 - 4b_0)|$$

のときである. ここで, δ は十分小さな正数である.

$$b_1^2 \leq \varepsilon^2 \max(|b_2^3|, |4b_0b_2|)$$

の場合には前述のように b_1 を無視して

$$y^4 + b_2y^2 + b_0 = 0$$

を解いて解を求める. 一方,

$$10^{-4}|b_2(b_2^2 - 4b_0)| \geq b_1^2 > \varepsilon^2 \max(|b_2^3|, |4b_0b_2|)$$

の場合には次の (d) に述べる方法を用いて解を求める.

(d) それ以外の場合
4 次方程式

$$y^4 + b_2y^2 + b_1y + b_0 = 0$$

の根は 3 次方程式

$$z^3 + \frac{b_2}{2}z^2 + \frac{b_2^2 - 4b_0}{16}z - \frac{b_1^2}{64} = 0$$

の 3 根 z_1, z_2, z_3 (z_1, z_2, z_3 は一般には複素数) の 2 乗根を用いて

$$\begin{aligned} y \quad = \quad & \sqrt{z_1} + \sqrt{z_2} + \sqrt{z_3}, \\ & \sqrt{z_1} - (\sqrt{z_2} + \sqrt{z_3}), \\ & -\sqrt{z_1} + \sqrt{z_2} - \sqrt{z_3}, \\ & -\sqrt{z_1} - (\sqrt{z_2} - \sqrt{z_3}) \end{aligned}$$

と表せることが知られている. ただし, ここでは $\sqrt{z}$ は z の 2 つある 2 乗根 ($\pm\sqrt{z}$) の一方を表す. 2 乗根のどちらを選べば良いかは前もって分からないが, 4 根の組み合わせは前述の y かまたはすべての根の符号を反転した $-y$ となるので, 根と係数の関係から b_1 の符号によってどちらが根かの判定を行う. なお, 3 次方程式は (4.1.2.1.2) に述べた方法で解く.

4.1.2.1.4 次数 $n > 4$ の場合

平野法を用いて根を求める. 与えられた方程式を

$$P_n(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 = 0$$

とする. 平野法では, 近似根の反復改良による根の探索と方程式の減次を交互に行うことによって方程式のすべての根を求める. 近似根は原点を出発値とし, 多項式 $P_n(x)$ をこの近似根の回りで展開したときの展開係数を用いて順次真根に近づけていく. このようにして得られた根 $x = (x_R, x_I)$ を用いて方程式の減次を行い, 減次後の方程式について同様の操作を順次繰り返すことによって, 方程式の全根を求める.
根 $x = (x_R, x_I)$ は次のように決定する.

(1) 根 x の近似値 $z = (z_R, z_I)$ の初期値を $z = 0$ とする.

(2) $\mid P_n(z + \zeta_m) \mid \leq \delta$ が成立するまで z を $z + \zeta_m$ と置き換え, 次の計算を繰り返す.

(a) 多項式 $P_n(x)$ を z を中心として
$P_n(\zeta + z) = c_n\zeta^n + c_{n-1}\zeta^{n-1} + \cdots + c_1\zeta + c_0 \quad (x = \zeta + z)$
と展開し, その係数 c_k, $(k = n, n-1, \cdots, 0)$ を組立除法をもちいて計算する.

(b) $\mu = 1$ とおく.

(c) $\zeta_k(\mu) \quad (k = 1, \cdots, n)$ の内で絶対値最小のものを ζ_m とすると, $\mid P_n(z + \zeta_m) \mid$
$\leq (1 - \frac{\mu}{4}) \mid P_n(z) \mid$ が成立するまで順次 μ を $\frac{\mu}{2}$ と置き換えて計算を進める.
ただし,
$\zeta_k(\mu) = (-\mu\frac{c_0}{c_k})^{\frac{1}{k}} \quad (k = 1, \cdots, n)$ である.

(3) $z + \zeta_m$ を方程式 $P_n(x) = 0$ の根とする.

ここで, 収束判定値 δ は次のように決定する.
いま, 係数 $c_k, (k = n, n-1, \cdots, 0)$ の計算に $\varepsilon \mid c_k \mid$ の誤差があり, ζ の計算誤差が $\varepsilon \mid x \mid$ (x は真の根の値, $x = z + \zeta$) まで許されるとする. このとき $P_n(z + \zeta)$ の計算誤差 δ は, $P_n(z + \zeta)$ が
$b_n = c_n$

$b_k = b_{k+1}\zeta + c_k \quad (k = n-1, n-2, \cdots, 0)$
$P_n(z+\zeta) = b_0$
から計算できることから,
$d_n = \varepsilon \mid c_n \mid$
$d_k = d_{k+1}(\mid \zeta \mid + \varepsilon \mid x \mid) + \varepsilon(\mid x \mid\mid b_{k+1} \mid + \mid c_k \mid) \quad (k = n-1, n-2, \cdots, 0)$
$\delta = d_0$
から計算できる. なお, 計算では, $\zeta = \zeta_m$ とし, x は $z + \zeta_m$ で近似する.
平野法では, 方程式を減次しながら根を順次求めて行くが, 各減次過程で誤差解析を行い, 減次の成功の判定を次のように行う.

(1) 1 実根による減次
方程式
$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$
をその実根 z を用いて
$P_n(x) = (x-z)(b_n x^{n-1} + b_{n-1}x^{n-2} + \cdots + b_1) + b_0 = 0$
と減次した場合,
$b_n = a_n$
$b_k = a_k + b_{k+1}z \quad (k = n-1, n-2, \cdots, 0)$
として各係数は計算される. b_0 は本来 0.0 であるが, z に $\sqrt{\varepsilon} \mid z \mid, a_k$ に $\varepsilon \mid a_k \mid$ の誤差まで許されるとすると, b_k に許される最大誤差 δ_k は
$\delta_n = \varepsilon \mid a_n \mid$
$\delta_k = \varepsilon \mid a_k \mid + (\delta_{k+1} + \sqrt{\varepsilon} \mid b_{k+1} \mid) \mid z \mid \quad (k = n-1, n-2, \cdots, 0)$
と表せる. したがって, $\mid b_0 \mid < \delta_0$ のとき減次が成功したものとする.

(2) 複素共役根による減次
方程式
$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$
をその複素共役根 $z = (z_R, z_I), \bar{z} = (z_R, -z_I)$ を用いて
$P_n(x) = (x-z)(x-\bar{z})(b_n x^{n-2} + b_{n-1}x^{n-3} + \cdots + b_2) + b_1 x + b_0 = 0$
と減次した場合,
$b_n = a_n$
$b_{n-1} = a_{n-1} + 2z_R b_n$
$b_k = a_k + 2z_R b_{k+1} - ({z_R}^2 + {z_I}^2)b_{k+2} \quad (k = n-1, n-3, \cdots, 0)$
として各係数は計算される. b_0, b_1 は本来 0.0 であるが, z_R に $\sqrt{\varepsilon} \mid z_R \mid, z_I$ に $\sqrt{\varepsilon} \mid z_I \mid, a_k$ に $\varepsilon \mid a_k \mid$ の誤差まで許されるとすると, b_k に許される最大誤差 δ_k は
$\delta_n = \varepsilon \mid a_n \mid$
$\delta_{n-1} = \varepsilon \mid a_{n-1} \mid + \delta_n \mid 2z_R \mid + \sqrt{\varepsilon} \mid 2z_R b_n \mid$
$\delta_k = \varepsilon \mid a_k \mid + \delta_{k+1} \mid 2z_R \mid + \sqrt{\varepsilon} \mid 2z_R b_{k+1} \mid + \delta_{k+2}({z_R}^2 + {z_I}^2) + 2\sqrt{\varepsilon}$
$(\mid z_R \mid + \mid z_I \mid) \mid b_{k+2} \mid \quad (k = n-2, n-3, \cdots, 0)$
と表せる. したがって, $\mid b_0 \mid < \delta_0, \mid b_1 \mid < \delta_1$ のとき減次が成功したものとする.

4.1.2.2 複素係数代数方程式の根

デュラン–ケルナー (Durand-Kerner) の 3 次法で解く.
与えられた方程式を

$$P_n(Z) = a_0 Z^n + a_1 Z^{n-1} + \cdots + a_{n-1}Z + a_n = 0 \qquad (a_0 = 0)$$

とする.
全根を同時に求める反復公式は,

$$\begin{cases} z_i^{(\nu+1)} = z_i^{(\nu)} + \phi_i(z_1^{(\nu)}, \cdots, z_n^{(\nu)}) \\ \phi_i(z_1^{(\nu)}, \cdots, z_n^{(\nu)}) = \dfrac{P_n(z_i^{(\nu)})/P_n'(z_i^{(\nu)})}{1 - \dfrac{P_n(z_i^{(\nu)})}{P_n'(z_i^{(\nu)})} \displaystyle\sum_{\substack{j=1 \\ j\neq i}}^{n} \dfrac{1}{z_i^{(\nu)} - z_j^{(\nu)}}} \quad (i = 1, \cdots, n) \end{cases}$$

によって求める. ここで $p_n'(z_i^{(\nu)})$ は, 次式により求める.

$$\begin{cases} z_i{}^{(\nu+1)} = z_i{}^{(\nu)} \Phi_i \left(z_1{}^{(\nu)}, \cdots, z_n{}^{(\nu)} \right) \\ \Phi_i(z_1^{(\nu)}, \cdots, z_n^{(\nu)}) = \dfrac{P_n(z_i{}^{(\nu)})/P_n'(z_i{}^{(\nu)})}{1 - \dfrac{P_n(z_i{}^{(\nu)})}{P_n'(z_i{}^{(\nu)})} \displaystyle\sum_{\substack{j=1 \\ j\neq i}}^{n} \dfrac{1}{z_i{}^{(\nu)} - z_j{}^{(\nu)}}} \quad \text{for} \quad (i = 1, \cdots, n) \end{cases}$$

$$\begin{cases} z_i{}^{(\nu+1)} = z_i{}^{(\nu)} \Phi_i \left(z_1{}^{(\nu)}, \cdots, z_n{}^{(\nu)} \right) \\ \Phi_i(z_1^{(\nu)}, \cdots, z_n^{(\nu)}) = \dfrac{P_n(z_i{}^{(\nu)})/P_n'(z_i{}^{(\nu)})}{1 - \dfrac{P_n(z_i{}^{(\nu)})}{P_n'(z_i{}^{(\nu)})} \displaystyle\sum_{\substack{j=1 \\ j\neq i}}^{n} \dfrac{1}{z_i{}^{(\nu)} - z_j{}^{(\nu)}}} \quad \text{for} \quad (i = 1, \cdots, n) \end{cases}$$

$$P_n'(z_i^{(\nu)}) = b_{n-1}^{(2)}$$

収束判定は, $P_n(z)$ の値がホーナー法で計算するときに含まれる誤差内になれば収束したものとみなす. すなわち, ε を誤差判定のための単位として,

$$\begin{cases} b_0{}^{(1)} &= a_0 \\ b_0{}^{(2)} &= a_0 \\ b_k{}^{(1)} &= z_i{}^{(\nu)} b_{k-1}^{(1)} + a_k \\ b_k{}^{(2)} &= z_i^{(\nu)} b_{k-1}^{(2)} + b_k^{(1)} \end{cases} \quad \text{for} \quad (k = 1, \cdots, n-1)$$

$$P_n'(z_i{}^{(\nu)}) = b_{n-1}{}^{(2)}$$

よって, $P_n(z_i^{(\nu)})$ の最大誤差の指定値 δ_n は,

$$\begin{cases} \delta_0 &= 0.0 \\ \delta_k &= \delta_{k-1} \mid z_i^{(\nu)} \mid + \varepsilon \{ \mid b_k' \mid + \max(\mid a_k \mid, \mid b_k^{'} \mid, \mid b_k \mid) \} \end{cases} \quad (k = 1, \cdots, n)$$

であり, $\mid P_n(z_i^{(\nu)}) \mid \leq \delta_n$ ならば, 収束したとみなす.
反復は, 未収束の $z_i^{(\nu)}$ に対してのみ続けるようにし, すべて収束したとみなされたとき, 計算は終了する.
反復の初期値は次の 3 段階で決める.

(i) アッバースの初期値の決定

$$\beta = \frac{\alpha_1 + \alpha_2 + \cdots + \alpha_n}{n} = -\frac{a_1}{n a_0}$$

を中心に, 全根を内部に含む半径 r を求める. 組立除法により,
$P_n(Z) = P_n(\beta + \zeta) = c_0 \zeta^n + c_1 \zeta^{n-1} + \cdots + c_{n-1}\zeta + c_n$
とした係数 $c_0, \cdots, c_n$ を求める. $c_1, \cdots, c_n$ の中で 0 でないものの個数を m として,

$r^+ = \max_{k=1,\cdots,n} (m \mid c_k \mid / \mid c_0 \mid)^{1/k}$
r は r^+ を初期値として
$Q_n(x) = \mid c_0 \mid x^n - \mid c_1 \mid x^{n-1} - \cdots - \mid c_{n-1} \mid x - \mid c_n \mid = 0$
となる x を求め, これを r にする.

(ii) 全根を含む円の半径を小さくする.
$z = \beta + rw$ として

$$\begin{aligned} P_n(\beta + rw) &= (c_0 r^n)w^n + (c_1 r^{n-1})w^{n-1} + \cdots + (c_{n-1}r)w + c_n \\ &= d_0^{(0)} w^n + d_1^{(0)} w^{n-1} + \cdots + d_{n-1}^{(0)} w + d_n^{(0)} \\ &= \varphi_0(w) \\ \varphi_\ell(w) &= d_0^{(\ell)} w^{n\ell} + d_1^{(\ell)} w^{n\ell-1} + \cdots + d_{n\ell-1}^{(\ell)} w + d_{n\ell}^{(\ell)} \quad (d_0^{(\ell)} \neq 0) \end{aligned}$$

- $\mid a_0^{(\ell)} \mid < \mid a_{n\ell}^{(\ell)} \mid$ のとき
  $\varphi_{\ell+1}(w) = \varphi_\ell(w) - \frac{d_0^{(\ell)}}{d_{n\ell}^{(\ell)}} \tilde{\varphi}\ell(w)$
- $\mid a_0^{(\ell)} \mid \geq \mid a_{n\ell}^{(\ell)} \mid$ のとき
  $\varphi_{\ell+1}(w) = \left\{ \varphi_\ell(w) - \frac{d_{n\ell}^{(\ell)}}{d_0^{(\ell)}} \tilde{\varphi}_\ell(w) \right\} / w$

ここで,
$\tilde{\varphi}_\ell(w) = \overline{d_{n\ell}^{(\ell)}} w^{n\ell} + \overline{d_{n\ell-1}} w^{n\ell-1} + \cdots + \overline{d_1^{(\ell)}} w + \overline{d_0^{(\ell)}}$
として $\varphi_n =$ 定数まで定義できる. 半径 r の円外の根の個数は, φ_n まで求めたときの $\mid a_0^{(\ell)} \mid < \mid a_{n\ell}^{(\ell)} \mid$ になった回数であり, 円外はその残りである. 全根を含む最小円の半径は, すべて $\mid a_0^{(\ell)} \mid > \mid a_{n\ell}^{(\ell)} \mid$ になる条件でアッパースの初期値 r から 2 分法で求める.

(iii) 各根からの距離の 2 乗の総和が最小となる半径にする.
(ii) で求めた半径 r を 2 分しながら各半径の円の内外の根の個数を (ii) で示した方法で求める. j 回の反復で幅 $r \times 2^{-j}$ の円環 D_i が求まる. 円環 D_i の内径と外径の平均を r_i^*, D_i に含まれる根の個数を N_i とすると, 求める半径 r は,

$$r = (\sum_{i=1}^{m} r_i^* N_i)/n$$

で得られる. 初期値はこの r から次式で求める.

$$z_i^{(0)} = \beta + r \ \exp[i(\frac{2\pi(j-1)}{n} + \frac{3}{2n})] \quad (\ j\ = 1, \cdots, n,\ i\ = \sqrt{-1})$$

4.1.2.3 実関数の根 (初期値指定) (導関数定義必要)

与えられた非線形方程式を $f(x) = 0, f(x)$ の導関数を $f'(x)$ として

$$x^{(\nu+1)} = x^{(\nu)} - f(x^{(\nu)})/f'(x^{(\nu)})$$

のニュートン法を基本とする. ただし, この方法は関数が振動していたり, 根の更新量が大きすぎると根が求まらないので, 次のように改良する.

(i) ニュートン反復 $x^{(\nu+1)} = x^{(\nu)} - f(x^{(\nu)})/f'(x^{(\nu)})$ を行う.

(ii) $f'(x^{(\nu)}) = 0$ または $\mid f(x^{(\nu+1)}) \mid \geq \mid f(x^{(\nu)}) \mid$ のとき (iii) に移る. それ以外は $x^{(\nu)} = x^{(\nu+1)}$ として (i) にもどる.

(iii) $\nu = 1$ または $f'(x^{(\nu-1)}) \geq 0$ のとき IS $= -1$, それ以外は IS $= 1$ にする.
R $= 1$, P $= 0$, IOLD $=$ SIGN$\{1, \mathrm{IS}, \cdot f(x^{(\nu)})\}$, I1 $= -$IOLD とする.

(iv) I2 $=$ SIGN$\{1, \mathrm{IS} \cdot f(x^{(\nu)})\}$ とする.
I2 $\cdot$ I1 > 0 のとき P $=$ P $+ 1$ で加速, それ以外は R $=$ R $+ 1$ で減速されるようにする.
(I1 : 旧更新方向, I2 : 新更新方向で, $+$ は正, $-$ は負方向の更新)
次の反復で $x^{(\nu)}$ の更新をする.
ε : 誤差判定のための単位として
$x^{(\nu+1)} = x^{(\nu)} + \mathrm{IS}\ \sinh^{-1}(f(x^{(\nu)}))2^{\{(p-3)/3-\mathrm{R}\}} + |\, x^{(\nu)} + 1 \,| \mathrm{I2}\varepsilon$
(IS : 大域的な関数の傾き推定で, $+$ のとき右下がり, $-$ のとき右上がり)
この更新は, 少なくとも 3 回は連続して行う.

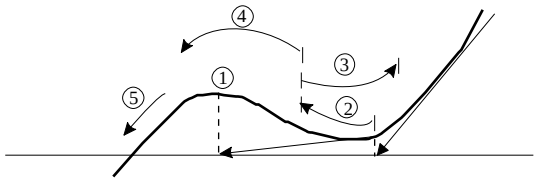
(v) $f(x^{(\nu-1)}) \gg f(x^{(\nu)}) \gg f(x^{(\nu+1)})$ で $|\, f(x^{(\nu+1)}) \,|$ が十分小さくなれば (i) のニュートン法にもどる.

(vi) $|\, f(x^{(\nu+1)}) \,| > |\, f(x^{(\nu)}) \,|$ で, $f(x^{(\nu+1)})$ と $f(x^{(\nu)})$ が同符号であり I2 $=$ IOLD のとき, IS $=$ $-$IS, IOLD $=$ IOLD, I1 $=$ IOLD として関数の傾きと探索方向を修正する.
初めてこの条件に入ったとき,
探索済区間左端 XP $= x^{(\nu+1)}$, FP $= f(x^{(\nu+1)})$
探索済区間右端 XQ $=$ XP, FQ $=$ FP
2 回目以後のとき
I2 > 0 のとき (更新方向が正)
XQ $= x^{(\nu+1)}$, FQ $= f(x^{(\nu+1)})$ で探索済区間右端更新
$x^{(\nu)} =$ XP, $f(x^{(\nu)}) =$ FP で探索済区間左端から始める.
I2 ≤ 0 のとき (更新方向が負)
XP $= x^{(\nu+1)}$, FP $= f(x^{(\nu+1)})$ で探索済区間左端更新
$x^{(\nu)} =$ XQ, $f(x^{(\nu)}) =$ FQ で探索済区間右端から始める.

P $= 0$ にし, R は谷から抜け出すように, 何度も連続して (vi) の条件に入るときは加速させるため R $= 2$, それ以外は R $= 0$ と設定し (iv) にもどる.

(vii) (vi) の条件に入らなかったら, I1 $=$ I2, $x^{(\nu)} = x^{(\nu+1)}$ として (iv) にもどる. 以上 (i) から (vii) の動作例を図示すると図 4−1 のようになる.

図 4−1



① ニュートン反復で｜f(x$^{(\nu+1)}$)｜＞｜f(x$^{(\nu)}$)｜になる.
② (iv)の変更式でx の更新をする.
③ ｜f(x$^{(\nu+1)}$)｜＞｜f(x$^{(\nu)}$)｜で更新方向に変更がなかったので(iv)により探索済区間左右端を決め探索方向を逆転する.
④ ③と同様, 探索開始と探索方向の変更を繰り返す.
⑤ ｜f(x$^{(\nu+1)}$)｜＜｜f(x$^{(\nu)}$)｜になるので(iv)または(i)のニュートン法で根が求まる.

収束判定は, e_r を要求精度とし,

- $\mid x^{(\nu+1)} - x^{(\nu)} \mid < e_r \max(1, \mid x^{(\nu+1)} \mid)$ かつ $\mid f(x^{(\nu+1)}) \mid < e_r + 64\varepsilon \mid x^{(\nu+1)} \mid$
- $f(x^{(\nu+1)}) = 0$

のいずれかを満たしたとき収束したものと見なすようにしている.

4.1.2.4 実関数の根 (初期値指定) (導関数定義不要)

基本的にはセカント法

$$x^{(\nu+1)} = x^{(\nu)} - f(x^{(\nu)}) \frac{x^{(\nu)} - x^{(\nu-1)}}{f(x^{(\nu)}) - f(x^{(\nu-1)})}$$

を採用し, (3) と同様の改良を行うが, $f(x) = 0$ に向かい谷になっている所は極値探索, 根の符号の反転が見つかれば 2 分法を採用する. 具体的にアルゴリズムを次に示す.

(i) セカント法で解を更新する.

(ii) 関数値, 更新量ともに大きいとき, $x^{(\nu)}$ が根から遠いので安全のため,
$\left.\begin{array}{l} \mathrm{XP} = \mathrm{XQ} = x^{(\nu)} \\ \mathrm{FP} = \mathrm{FQ} = f(x^{(\nu)}) \end{array}\right\}$ (探索済区間左右端の設定, XP は左端, XQ は右端)
$x^{(\nu+1)} = x^{(\nu)}, f^{(\nu+1)} = f^{(\nu)}$ として (iii) に移る.
$\mid f(x^{(\nu+1)}) \mid \geq \mid f(x^{(\nu)}) \mid$ のとき
$\mid (x^{(\nu)} - x^{(\nu-1)})/f(x^{(\nu-1)}) \mid > 0.125$ のとき ($x^{(\nu)}$ に近いところに根があると予想される).
$\mathrm{XP} = \mathrm{XQ} = x^{(\nu-1)}$
$\mathrm{FP} = \mathrm{FQ} = f(x^{(\nu-1)})$
$x^{(\nu+1)} = x^{(\nu-1)}, f(x^{(\nu+1)}) = f(x^{(\nu-1)})$
として (iii) に行く.
$\mid (x^{(\nu)} - x^{(\nu-1)})/f(x^{(\nu-1)}) \mid \leq 0.125$ のとき
正方向に更新されていたなら
$\left.\begin{array}{l} \mathrm{XP} = x^{(\nu-1)}, \mathrm{XQ} = x^{(\nu+1)} \\ \mathrm{FP} = f(x^{(\nu-1)}), \mathrm{FQ} = f(x^{(\nu+1)}) \end{array}\right\}$ (探索済区間の設定)
として, $x^{(\nu-1)} \sim x^{(\nu+1)}$ 間で極値探索を行う.
負方向に更新されていたなら
$\left.\begin{array}{l} \mathrm{XP} = x^{(\nu+1)}, \mathrm{XQ} = x^{(\nu-1)} \\ \mathrm{FP} = f(x^{(\nu+1)}), \mathrm{FQ} = f(x^{(\nu-1)}) \end{array}\right\}$ (探索済区間の設定)
として, $x^{(\nu+1)} \sim x^{(\nu-1)}$ 間で極値探索を行う.
極値探索で根が見つからなければ, (iii) に移る.
以上のような状況がなければ, $x^{(\nu)} = x^{(\nu+1)}$ として (i) にもどる.

(iii) $x^{(\nu-1)} \sim x^{(\nu)}$ で関数が右上りなら $\mathrm{IS} = -1$, 左上りなら $\mathrm{IS} = 1$ にする.
$\mathrm{R} = 1, \mathrm{P} = 0, \mathrm{IOLD} = \mathrm{SIGN}\{1, \mathrm{IS} \cdot f(x^{(\nu+1)})\}, \mathrm{I1} = -\mathrm{IOLD}$ とする.

(iv) $\mathrm{I2} = \mathrm{SIGN}\{1, \mathrm{IS} \cdot f(x^{(\nu+1)})\}$ とする.
$\mathrm{I2} \cdot \mathrm{I1} > 0$ のとき $\mathrm{P} = \mathrm{P} + 1$ で加速, それ以外は $\mathrm{R} = \mathrm{R} + 1$ で減速されるようにする.
(I1 : 旧更新方向, I2 : 新更新方向で + は正, − は負方向の更新)

$$\begin{array}{llll} x^{(\nu)} & = & x^{(\nu+1)}, & f(x^{(\nu)}) & = & f(x^{(\nu+1)}) \\ x^{(\nu-1)} & = & x^{(\nu)}, & f(x^{(\nu-1)}) & = & f(x^{(\nu)}) \end{array}$$

とする.
次の反復で $x^{(\nu)}$ の更新をする.
ε : 誤差判定のための単位として
$x^{(\nu+1)} = x^{(\nu)} + \mathrm{IS}\ \sinh^{-1}(f(x^{(\nu)}))2^{\{(P-3)/3-R\}} + |\, x^{(\nu)} + 1\,|\, \mathrm{I2}_{\varepsilon}$
(IS : 大域的な関数の傾きの推定で, + のとき右下がり, − のとき右上がり)
この更新は, 少なくとも 3 回は連続して行う.

(v) $f(x^{(\nu-1)}) \gg f(x^{(\nu)}) \gg f(x^{(\nu+1)})$ で $|\, f(x^{(\nu+1)})\,|$ が十分小さくなければ (i) のセカント法にもどる.

(vi) $|\, f(x^{(\nu-1)})\,| > |\, f(x^{(\nu)})\,| < |\, f(x^{(\nu+1)})\,|$ で $f(x^{(\nu-1)}) \sim f(x^{(\nu+1)})$ の符号が等しいとき,

- $\mathrm{I2} > 0$ (更新方向が正) のとき
  $\mathrm{XQ} = x^{(\nu+1)},\ \mathrm{FQ} = f(x^{(\nu+1)})$ で探索済区間右端更新.
- $\mathrm{I2} \leq 0$ (更新方向が負) のとき
  $\mathrm{XP} = x^{(\nu+1)},\ \mathrm{FP} = f(x^{(\nu+1)})$ で探索済区間左端更新.

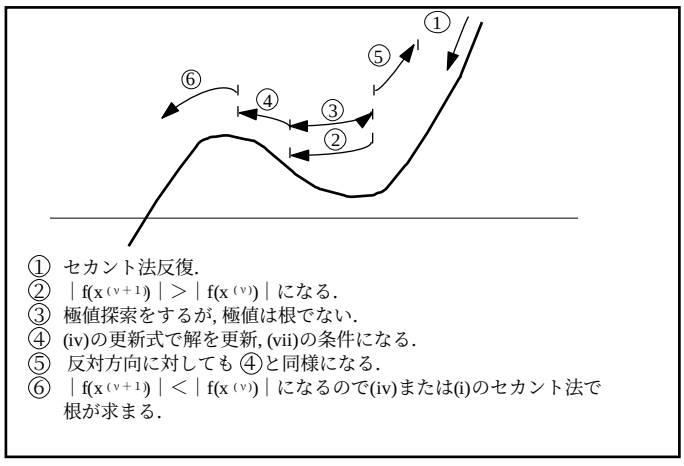
そして, $x^{(\nu+1)} \sim x^{(\nu-1)}$ 間で極値探索を行う.
極値探索で根が見つからなければ, 解を更新してきた方向と反対方向の探索済区間端に
$x^{(\nu+1)}, f(x^{(\nu+1)})$ をセットし,
$\mathrm{IS} = -\mathrm{IS}, \mathrm{IOLD} = -\mathrm{IOLD}, \mathrm{I2} = \mathrm{IOLD}, \mathrm{P} = 0, \mathrm{R} = 1$
として反対方向に更新させるようにして (iv) にもどる.

(vii) $|\, f(x^{(\nu-1)})\,| < |\, f(x^{(\nu)})\,| < |\, f(x^{(\nu+1)})\,|$ で $f(x^{(\nu-1)}) \sim f(x^{(\nu+1)})$ の符号が等しいとき, 解を更新してきた方向と反対方向の探索区間端に $x^{(\nu+1)}, f(x^{(\nu+1)})$ をセットし,
$\mathrm{IS} = -\mathrm{IS}, \mathrm{IOLD} = -\mathrm{IOLD}, \mathrm{I2} = \mathrm{IOLD}, \mathrm{P} = 0, \mathrm{R} = 1$
として反対方向に更新されるようにして (iv) にもどる.

(viii) (v) から (vii) の条件に入らなかったら $\mathrm{I1} = \mathrm{I2}$, として (iv) にもどる.

極値探索部は, 黄金分割探索と逐次放物線補間を組み合わせて最小点を求めるもので, その詳細は第 5 章「極値問題」5.1.2「使用しているアルゴリズム」に述べている.
なお, 極値探索中に方程式の値の符号の逆転が見つかれば, その点の初期値に近い方の根に対し 2 分法探索を行う.
以上 (i) から (viii) の動作例を図示すると, 図 4−2 のようになる.

図 4−2



収束判定は, e_r を要求精度とし,

- ($\mid x^{(\nu+1)} - x^{(\nu)} \mid < e_r \ \max(1, \mid x^{(\nu+1)} \mid)$ かつ $\mid f(x^{(\nu+1)}) \mid < e_r + 64\varepsilon \mid x^{(\nu+1)} \mid$
- $f(x^{(\nu+1)}) = 0$

のいずれかを満たしたとき, 収束したものとする. また極値探索や 2 分法では,
(探索縮小区間)$< \ e_r \max(1, \mid x \mid)$ and $\mid f(x) \mid < \ e_r + 64\varepsilon \mid x \mid$
のとき収束したものとする.

4.1.2.5 実関数の根 (区間指定) (導関数定義不要)

$f(x)$ を $[a, b]$ で連続かつ $f(a)$ と $f(b)$ で符号が逆転するものとする.

(i) $c = a, d = e = b - a$ にする.

(ii) $\mid f(c) \mid < \mid f(b) \mid$ ならば, b と c を入れ替え, a を入れ替え後の c にする.
こうして常に $\mid f(b) \mid \leq \mid f(c) \mid$ になるようにしておき, $[b, c]$ 間で根をさがす.

(iii) $m = (c - b)/2$ とする. また,
ε : 誤差判定のための単位, $e_r =$ 要求精度
として,
$\delta = (\varepsilon \mid b \mid + e_r)/2$
とする.

(iv) $\mid e \mid \geq \delta$ かつ $\mid f(b) \mid < \mid f(a) \mid$ のとき,
$a = c$ のとき c と b の間で線形補間法を適用する.

$P = (c - b)f(b)/f(c)$
$Q = 1 - f(b)/f(c)$

$a = c$ ($\mid f(a) \mid \leq \mid f(c) \mid$) のとき a, b, c 間で逆 2 次補間法を適用する.

$r_1 = f(a)/f(c), r_2 = f(b)/f(c), r_3 = f(b)/f(a)$ として
$\mathrm{P} = r_3\{(c - b)r_1(r_1 - r_2) - (b - a)(r_2 - 1)\}$
$Q = (r_1 - 1)(r_2 - 1)(r_3 - 1)$

$\mid P/Q \mid > \frac{3}{4} \mid c - b \mid - \mid \delta \mid$ または $\mid P/Q \mid > \mid e/2 \mid$ のとき (v) に移る.
これ以外のときは,
$e = d$
$d = -P/Q$ とする.
$a = b$ にしておいて
$b = b + \max(d, \mathrm{sign}(\delta, c - b))$

(v) (iv) の条件にならないときは, $a = b$ にし b を $[b, c]$ の中点とする 2 分法を採用し,
$d = e = m$ とする.

(vi) $f(b)$ と $f(c)$ の符号が等しいときは, $c = a$ にし, $d = e = b - a$ にする.
以上の操作の後, (ii) にもどる.
収束判定は, $f(b) = 0$, または $\mid c - b \mid \leq e_r + 2\varepsilon \mid b \mid$ のとき収束したものとみなす.

4.1.2.6 実関数の全根 (区間指定) (導関数定義不要)

基本は (4.1.2.1.2) 「導関数定義不要とした非線形方程式の 1 根」と同じアルゴリズムを利用するが, ここで (iv) で示した式が IS を使って探索方向を一定に保つことができることを利用し, 常に区間の両端から内側に向かい根を求めていくようにしている. すなわち, 次の点を変更している.

(1) 解の探索除外区間を区間の両側からとっていく.

(2) まず区間右端から根を探索し, 根が見つかればそこを区間右端に変更する. 次に区間左端より根を探索する. 根が見つかれば次々にそこを区間左端にしていく.

(3) 解の更新値が区間外に出たときは, 区間内にもどして処理を続ける.

(4) 極値探索中に方程式の値の符号が反転したときには, その左右双方の根を 2 分法で求める.

(5) 区間 $\leq \delta$ になれば処理を終了する.

なお, 根が見つかったとき区間縮小するが, その場合はその根より区間内側へ, 次の δ だけずらすものとする.

$$\delta = \max(2e_r, (\mid \mathrm{A} \mid + \mid \mathrm{B} \mid)\varepsilon, \sqrt[3]{\varepsilon})$$

ここで e_r : 要求精度
ε : 誤差判定のための単位
A, B : 最初に設定した区間の左右端

4.1.2.7 複素関数の根 (初期値指定) (導関数定義不要)

マラー (Muller) 法によって解く.
出発値を, 初期値 $z = 0$ のとき

$$\begin{cases} z_1 = -1.0 \\ z_2 = 1.0 \\ z_3 = 0.0 \end{cases}$$

初期値 $z = 0$ のとき

$$\begin{cases} z_1 = 0.9z \\ z_2 = 1.1z \\ z_3 = z \end{cases}$$

とする.
$f_i = f(z_i)$ として
$q = (z_3 - z_2)/(z_2 - z_1)$
$q' = q + 1$
$a = qf_3 - qq'f_2 + q^2 f_1$
$b = (q + q')f_3 - q^{'2} f_2 + q^2 f_1$
$c = q' f_3$
$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$ として絶対値の小さい方の r を採用する (r の分母が 0 のときで $f_1 = f_2 = f_3$ のときは, $r = 1$ としておく).

$$z = z_3 + r(z_3 - z_2)$$

$$\begin{cases} z_3 = z \\ z_2 = z_3 \\ z_1 = z_2 \end{cases}$$

として反復をくり返す.
ここで収束判定は, e_r を要求精度, ε を誤差判定のための単位として,
$(\mid z - z_3 \mid < e_r \ \max(1, \mid z \mid)$ and $\mid f(z) \mid < e_r + 64\varepsilon \mid z \mid)$ or $f(z) = 0$
のとき収束したものとする.

4.1.2.8 連立非線形方程式の根 (ヤコビ行列定義任意)

マルカール (Marquardt) 法によって解く. その方法を以下に示す.

(i) $\lambda = 0.1$ とする.

(ii) $\boldsymbol{f}(\boldsymbol{x})$ のヤコビ行列を A とする.
$\boldsymbol{f}(\boldsymbol{x}) = (f_1, f_2, \cdots, f_n)^T$
$\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^T, n$: 元数として

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \cdots\cdots\cdots \frac{\partial f_1}{\partial x_n} \\ \vdots \qquad\qquad\qquad \vdots \\ \vdots \qquad\qquad\qquad \vdots \\ \frac{\partial f_n}{\partial x_1} \cdots\cdots\cdots \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_{11} \cdots\cdots\cdots a_{1n} \\ \vdots \qquad\qquad\qquad \vdots \\ \vdots \qquad\qquad\qquad \vdots \\ a_{n1} \cdots\cdots\cdots a_{nn} \end{bmatrix}$$

D を $A^T A$ の対角要素以外を 0 にした行列とする.

(iii) $A^T A + \lambda D \Delta \boldsymbol{x} \ = -A^T \boldsymbol{f}(\boldsymbol{x})$
の連立 1 次方程式を解き $\Delta\ \boldsymbol{x}$ を得る ($\boldsymbol{x}$ を $D^{\frac{1}{2}}$ でスケール変換したうえで付加項 λI のマルカール法を適用したのと同等の式).

(iv) $\boldsymbol{y} = \boldsymbol{x} + \Delta\ \boldsymbol{x}$ として
$\| \boldsymbol{f}(\boldsymbol{y}) \| \geq \| \boldsymbol{f}(\boldsymbol{x}) \|$ ならば　　($\| \cdots \|$ 記号は, 2 乗ノルムとする)
$\lambda = 0$ のとき $\lambda = 0.001$,
$\lambda = 0$ のとき $\lambda = 10\lambda$ として (iii) にもどる.
$\| \boldsymbol{f}(\boldsymbol{y}) \| < \| \boldsymbol{f}(\boldsymbol{x}) \|$ ならば
前回も $\| \boldsymbol{f}(\boldsymbol{y}) \| < \| \boldsymbol{f}(\boldsymbol{x}) \|$ なら $\lambda = \lambda/10$ にする.
$\boldsymbol{x} = \boldsymbol{y}$ にして (ii) にもどる.
以上のマルカール法は正規方程式系を陽に形成するため, 誤差に対して弱いという欠点がある. そこで
e_r : 要求精度
ε : 誤差判定のための単位
として
($\|\Delta\ \boldsymbol{x}\ \|_\infty \leq e'\ \max(1, \| \boldsymbol{y} \|_\infty)$ and $\| \boldsymbol{f}(\boldsymbol{y}) \|_\infty \leq e'$) or $\| \boldsymbol{f}(\boldsymbol{y}) \|_\infty = 0$
$e' = e_r^{0.2}, \| \cdots \|_\infty$ は各要素の絶対値最大の値
に達したとき, または $\| \boldsymbol{f}(\boldsymbol{y}) \|$ が $4n$ 回反復前の $\| \boldsymbol{f}(\boldsymbol{y}) \|$ に対して 0.75 倍にしかならないときは, 次のスケーリング付ニュートン法に処理移行する.
スケーリング付ニュートン法
$g_i = \max(\varepsilon, a_{i1}, a_{i2}, \cdots, a_{in}) \quad (i = 1, \cdots, n)$
これより
$(a_{i1}, a_{i2}, \cdots, a_{in}) = (a_{i1}, a_{i2}, \cdots, a_{in})/g_i$
$f_i = f_i/g_i$
このようにスケーリングされたヤコビ行列 A および関数値 $\boldsymbol{f}(\boldsymbol{x})$ を使って連立 1 次方程式 $A\Delta\ \boldsymbol{x} = -\boldsymbol{f}(\boldsymbol{x})$ を解き, $\Delta\ \boldsymbol{x}$ を修正ベクトルとして解を更新する.
収束判定は次の式が成立したとき収束するものとする.
($\|\Delta\ \boldsymbol{x}\ \|_\infty < e_r \max(1, \| \boldsymbol{y} \|_\infty)$ and $\| \boldsymbol{f}(\boldsymbol{y}) \|_\infty < e_r + 64\varepsilon \| \boldsymbol{y} \|_\infty$) or $\| \boldsymbol{f}(\boldsymbol{y}) \|_\infty = 0$

4.1.2.9 連立非線形方程式の根 (ヤコビ行列定義不要)

(1) 修正ベクトル $\Delta \boldsymbol{x}$ の計算

$\boldsymbol{f}(\boldsymbol{x})$: 変数値 $\boldsymbol{x}$ での関数値 $\boldsymbol{f}(\boldsymbol{x})$

A : ヤコビ行列 $\partial\ \boldsymbol{f}/\ \partial\ \boldsymbol{x}$

G : ヤコビ行列の逆行列

とする. まずガウスニュートン解を $\Delta\ \boldsymbol{x_G}$, 最急降下解を $\Delta\ \boldsymbol{x_S}$ として

$\Delta \boldsymbol{x_G} = -G\boldsymbol{f}(\boldsymbol{x})$

$\Delta \boldsymbol{x_S} = (\|\boldsymbol{b}\|^2/\|\boldsymbol{A}\ \boldsymbol{b}\ \|^2)\boldsymbol{b}$

ただし, $\boldsymbol{b} = -A^T \boldsymbol{f}(\boldsymbol{x})$

最初は, $\Delta \boldsymbol{x} = \Delta \boldsymbol{x_S}$ とし, $d = \|\Delta \boldsymbol{x_S}\|$ とする.

2 回目以後は, d をステップ幅とし,

(a) $d \leq \|\Delta \boldsymbol{x_S}\|$ の場合

$\Delta \boldsymbol{x} = d\ \Delta\ \boldsymbol{x_S}\ /\ \|\Delta\ \boldsymbol{x_S}\ \|$

(b) $\|\Delta\ \boldsymbol{x_S}\ \| < d < \|\Delta\ \boldsymbol{x_G}\ \|$ の場合

$\Delta\ \boldsymbol{x}= \alpha\Delta\ \boldsymbol{x_S}+\beta\Delta\ \boldsymbol{x_G} \quad (\alpha > 0, \beta > 0, \|\Delta\ \boldsymbol{x}\ \| = d)$

(c) $\|\Delta\ \boldsymbol{x_G}\ \| \leq d$ の場合

$\Delta\ \boldsymbol{x}= \Delta\ \boldsymbol{x_G}$

(2) ステップサイズ d の変更

線形化した模型の関数値 2 乗和の差を Δs, 実際の関数値 2 乗和の差を ΔT として, その比 r を用いて非線形性の程度を測る.

$\Delta s = \|\boldsymbol{f}(\boldsymbol{x}) - A\Delta \boldsymbol{x}\|^2 - \|\boldsymbol{f}(\boldsymbol{x})\|^2$

$\Delta T = \|\boldsymbol{f}(\boldsymbol{x}+\Delta \boldsymbol{x})\|^2 - \|\ \boldsymbol{f}(\boldsymbol{x})\|^2$

$r = \Delta T/\Delta s$

(a) $r < 0.1$ のとき d を半減させ, $\tau = 1.0$ とする.

(b) $r \geq 0.1$ のとき d の増加率 λ を以下のように計算する.

$\lambda = \sqrt{1 - \frac{(r-0.1)\Delta S}{s_p+\sqrt{s_p^2-(r-0.1)S_s\Delta S}}}$

ただし, $\delta \boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{\Delta x}) - \{\boldsymbol{f}(\boldsymbol{x}) + A\Delta x\}$ として

$s_p = \sum_{i=1}^{n} \|f_i(\boldsymbol{x}+\Delta \boldsymbol{x})\delta f_i\|$

$s_s = \|\delta\ \boldsymbol{f}\ \|^2$

実際には, d の振動を防ぐために 2 回続けて増加が要求されたときにだけ d を増加させる. また増加率は 2 以下に抑える. 実際の増加率 μ は以下のように計算される.

$\mu = \min(2, \lambda, \tau)$

$\tau = \lambda/\mu$

ここで, τ は初期値を 1 とする.

また d には上限 $d_{\max}$ と下限 $d_{\min}$ が設けられており, その間に入るように制御されている.

(3) ヤコビ行列 A およびその逆行列 G の計算

ヤコビ行列は, 最初の 1 回だけは差分によって求め, 後は逐次更新していく. 同様に, その逆行列も最初の 1 回だけはヤコビ行列から求め, 後は逐次更新していく. 逐次更新の方法は以下の式によって計算する.

$A = A + \delta \boldsymbol{f}\Delta \boldsymbol{x^T}/\|\Delta \boldsymbol{x}\|^2$

$G = G + (\Delta \boldsymbol{x} - G\Delta \boldsymbol{f})\Delta \boldsymbol{x}^{\boldsymbol{T}} G / \Delta \boldsymbol{x}^{\boldsymbol{T}} G \Delta \boldsymbol{f}$
ただし, $\Delta \boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}+\Delta \boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})$

(4) 修正ベクトルの独立性検査
ヤコビ行列の修正が効率よく行われるためには, 順次取られる修正ベクトルが互いに直交に近いことが必要である. そのために, ハイブリッド法では, 独自の独立性概念を定義して, 修正ベクトルができるだけ独立な方向に取られるように制御している. ベクトル $\boldsymbol{a}$ が j 個のベクトル ($\boldsymbol{a_1}$, $\boldsymbol{a_2}$, $\cdots$, $\boldsymbol{a_i}$) と独立であるとは, $\boldsymbol{a}$ がこれらのベクトルで張られた空間の任意のベクトルと 30 度以上の角度を成していることである. パウエルによって考案された独立性検査の算法を以下に示す.
過去 2 n 回のヤコビ行列の修正で用いられたベクトルのうち互いに独立な n 個のベクトルを直交化して, $\Omega = (\omega_1, \omega_2, \cdots, \omega_n)$ に保持しておく. 大きさ n の配列 $\boldsymbol{\ell}$ を用いて, ω_i が何回前の修正ベクトルであったかの情報を保持しておく. すなわち, ω_i は ℓ_i 回前に取られたベクトルであることを意味する. Ω は単位行列で初期化し, $\boldsymbol{\ell}$ は $\boldsymbol{\ell_i}$= $n - i + 1 (i = 1, \cdots, n)$ で初期化する.
解の修正を行うときには, 以下のようにする.

(a) $\Delta \boldsymbol{x}$= $\Delta \boldsymbol{x_G}$ の場合
独立性のいかんにかかわらず, $\Delta \boldsymbol{x}$ を修正ベクトルとして採用する.

(b) $\Delta \boldsymbol{x}$= $\Delta \boldsymbol{x_G}$ でない場合
$\boldsymbol{j_1}$< $2n$ であるか, または Δ $\boldsymbol{x}$ が $(\omega_2, \omega_3, \cdots, \omega_n)$ と独立, すなわち $\|\omega_1^T \Delta \boldsymbol{x}\| > \|\Delta$ $\boldsymbol{x}\|/2$ なら, Δ $\boldsymbol{x}$ を修正ベクトルとして採用する. そうでないなら, 解の修正を行わない.
ヤコビ行列の修正を行うときには以下のようにする.
$\|\Delta \boldsymbol{x}\| < d_{\min}$ であるか, または, $j_1 = 2n$ で $\Delta \boldsymbol{x}$ が $(\omega_2, \omega_3, \cdots, \omega_n)$ と独立でないなら, $\Delta \boldsymbol{x}$= $d_{\min}$ ω_1 とする. そうでないなら, Δ $\boldsymbol{x}$ を採用する.
つぎに, Ω と $\boldsymbol{j}$ の改訂を行う. Δ $\boldsymbol{x}$= $d_{\min}\omega_1$ としたときには,
$\omega_i = \omega_{i+1} (i = 1, \cdots, n-1)$
$\omega_n = \omega_1$
$j_i = j_{i+1} + 1 (i = 1, \cdots, n-1)$
$j_n = 1$
とすればよい. そうでないときには, 以下のようにする.
$(\omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta \boldsymbol{x})$ が互いに独立になる最小の k を求める.
$(\omega_1, \cdots, \omega_{k-1}, \omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta$ $\boldsymbol{x})$ を直交化して, それを改めて
$(\omega_1, \omega_2, \cdots, \omega_n)$ とする.
$j_i = j_i + 1 (i = 1, \cdots, k-1)$
$j_i = j_{i+1} + 1 (i = k, \cdots, n-1)$
$j_n = 1$
このようにして, 常に, $j_1 \leq 2n$ となるようにする.

(5) スケーリング付ニュートン法への処理移行
与えられた問題の各方程式がスケーリングされていないと, 解の変更過程に異常がおこり, 本アルゴリズムでは収束しないので, 下記の条件で解の変更過程の異常を検出し, スケーリング付ニュートン法へ処理を移行する.
$(\|\boldsymbol{f}(\boldsymbol{x}+\Delta \boldsymbol{x})\|^2 - \|\boldsymbol{f}(\boldsymbol{x})\|^2)$ / $d > 10^{10}$
なお, スケーリング付ニュートン法のアルゴリズムを以下に示す.

(a) ヤコビ行列の計算
ヤコビ行列は毎回差分によって求める.

(b) 修正ベクトルの計算
ヤコビ行列の (i, j) 成分を a_{ij} とする. まず次を求める.

$g_i = \max$ (誤差判定のための単位, $a_{i1}, a_{i2}, \cdots, a_{iN}$)
$(i = 1, 2, \cdots, N)$
これより
$(a_{i1}, a_{i2}, \cdots, a_{iN}) \leftarrow (a_{i1}, a_{i2}, \cdots, a_{iN})/g_i$
$f_i \leftarrow f_i/g_i$
$(i = 1, 2, \cdots, N)$
このようにスケールされたヤコビ行列 A および関数値 $\boldsymbol{f(x)}$ を使って連立 1 次方程式 $A\Delta\boldsymbol{x} = -\boldsymbol{f(x)}$ をクラウト法によって求める. この $\Delta\ \boldsymbol{x}$ が修正ベクトルとなる.

(6) 収束判定
収束判定は以下の式によって行う.
$(\|\Delta\boldsymbol{x}\|_\infty < e_r \max(1, \|\boldsymbol{x}+\Delta\boldsymbol{x}\|_\infty)$ and$\|\boldsymbol{f(x}+\Delta\boldsymbol{x)}\|_\infty < e_r + 64\varepsilon\|\boldsymbol{x}+\Delta\boldsymbol{x}\|)$
or $\boldsymbol{f(x}+\Delta\boldsymbol{x)} = 0$
ここで, e_r は要求相対精度であり,
$\|\boldsymbol{x}\|_\infty = \max_i | x_i |$
$\|\Delta\boldsymbol{x}\|_\infty = \max_i | \Delta x_i |$
$\|\boldsymbol{f(x}+\Delta\boldsymbol{x)}\|_\infty = \max_i | f_i |$ である.

4.1.3 参考文献

(1) 伊理正夫, “数値計算”, 朝倉書店, (1981).

(2) 渡部力, 名取亮, 小国力編, “第 3 版数値解析と FORTRAN”, 丸善, (1983).

(3) 森正武, “FORTRAN77 数値計算プログラミング”, 岩波コンピュータサイエンス, 岩波書店, (1986).

(4) 中田多美, 川本則行, 名取亮, “代数方程式に対する Durand-Kerner 法の初期値の改良”, 情報処理学会論文誌, Vol. 29, No. 12, pp. 1200-1207, (1988).

(5) 伊理正夫, 藤野和建, “数値計算の常識”, 共立出版, (1985).

(6) Kantaris, N. , Howden, P. F. , 三井斌友訳, “方程式の数値解法”, 啓学出版, (1987).

(7) Forsythe, G. E. , Malcolm, M. A. , and Moler, C. B. , 森正武訳, “計算機のための数値計算法”, 日本コンピュータ協会, (1978).

(8) 一松信, 戸川隼人編, “数値計算における誤差”, 共立出版, (1975).

(9) Ralston, A. , Rabinowitz, P. , 戸田英雄, 小野令美訳, “電子計算機のための数値解析の理論と応用 (下)”, ブレイン図書 (株), (1986).

(10) 田辺國士, “非線形方程式”, コンピュートロール, コロナ社, No12, pp. 36-39, (1985).

(11) 中川徹, 小柳義夫, “最小二乗法による実験データ解析”, 東京大学出版会, (1982).

4.2 代数方程式

4.2.1 ASL_dlarha, ASL_rlarha
実係数代数方程式の根

(1) **機 能**

実数係数の代数方程式を解く (根は実数型配列).

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlarha  (a, n, xr, xi, wk);
```

単精度関数:

```
ierr = ASL_rlarha  (a, n, xr, xi, 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∗}	n+1	入 力	代数方程式の係数 (高次項から a [0] , a [1] , ⋯, a [n])
2	n	I	1	入 力	代数方程式の次数
3	xr	{D∗ / R∗}	n	出 力	代数方程式の根の実部
4	xi	{D∗ / R∗}	n	出 力	代数方程式の根の虚部
5	wk	{D∗ / R∗}	$4\times(n+1)$	ワーク	作業領域 n≤4 のときは使用しない
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) $a[i-1],\ i=1,2,\cdots,n$ のうち少なくとも 1 つが $|a[i-1]| >$ (誤差判定の単位) を満たす.

(b) $n \geq 1$

(5) エラーインディケータ (戻り値)

戻り値	意　　味	処　理　内　容
0	正常終了.	
2000+i	i 番目までの根は正しく求められているが, それ以外は精度が悪い.	i + 1 番目以降の根の精度が悪いままで処理を終了する.
2500+j	$\|a[i-1]\| \leq$ 誤差判定の単位, $i = 1, 2, \cdots, j$ であった.	$a[i-1] = 0.0, i = 1, 2, \cdots, j$ と見なして処理する.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000+i	i 番目までの根は求まったが, それ以後の根は求められない.	i 番目までの根を計算して, 処理を打ち切る.

(6) 注意事項

(a) 根が 4 根以上あるときは, 最後の 4 根を除き 0.0 に近い根から求まる傾向にある.

(b) 重根のときの相対精度は根の多重度を n として, $\sqrt[n]{\text{誤差判定のための単位}}$ 程度になる.

(7) 使用例

(a) 問 題

$x^{10} - 55x^8 + 1023x^6 - 7645x^4 + 21076x^2 - 14400 = 0$ を解く.

(b) 入力データ

代数方程式の係数を格納した配列 a: a [0] =1.0, a [1] =0.0, a [2] = −55.0, a [3] =0.0, a [4] =1023.0, a [5] = 0.0, a [6] =−7645.0, a [7] =0.0, a [8] =21076.0, a [9] =0.0, a [10] =14400.0,

n=10

(c) 主プログラム

```
/*      C interface example for ASL_dlarha */

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

int main()
{
    double *ar;
    int nn;
    double *zr;
    double *zi;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dlarha ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &nn );

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

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

    zi = ( double * )malloc((size_t)( sizeof(double) * nn ));
```

```
    if( zi == NULL )
    {
        printf( "no enough memory for array zi\n" );
        return -1;
    }

    wk = ( double * )malloc((size_t)( sizeof(double) * (4*(nn+1)) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    printf( "\tDegree of Algebraic Equation = %6d\n", nn );
    printf( "\n\tCoefficients of Algebraic Equation \n\n" );
    for( i=0 ; i<nn+1 ; i++ )
    {
        fscanf( fp, "%lf", &ar[i] );
        printf( "\t%8.3g\n", ar[i] );
    }

    fclose( fp );

    ierr = ASL_dlarha(ar, nn, zr, zi, wk);

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

    printf( "\n\tRoots\n" );
    printf( "\t  Real Part          Imaginary Part\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "       %8.3g           %8.3g\n", zr[i],zi[i] );
    }

    free( ar );
    free( zr );
    free( zi );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dlarha ***

 ** Input **

Degree of Algebraic Equation =     10

Coefficients of Algebraic Equation

       1
       0
     -55
       0
1.02e+03
       0
-7.65e+03
       0
2.11e+04
       0
-1.44e+04

 ** Output **

ierr =      0

Roots
  Real Part          Imaginary Part
          1                  0
         -1                  0
          2                  0
         -2                  0
          3                  0
         -3                  0
          4                  0
          5                  0
         -5                  0
         -4                  0
```

4.2.2 ASL_zlacha, ASL_clacha
複素係数代数方程式の根

(1) **機 能**

複素数係数の代数方程式を解く (係数, 根は複素数型配列).

(2) **使用法**

倍精度関数:

```
ierr = ASL_zlacha  (ca, n, & nev, cx, wk, cwk);
```

単精度関数:

```
ierr = ASL_clacha  (ca, n, & nev, cx, wk, cwk);
```

(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	ca	{Z∗ / C∗}	n+1	入 力	代数方程式の係数 (高次項から ca [0] , ca [1] , ⋯, ca [n])
2	n	I	1	入 力	代数方程式の次数
3	nev	I*	1	入 力	最大関数評価回数 (既定値:100)
				出 力	実際の関数評価回数
4	cx	{Z∗ / C∗}	n	出 力	代数方程式の根
5	wk	{D∗ / R∗}	n + 1	ワーク	作業領域
6	cwk	{Z∗ / C∗}	4 × (n + 1)	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) $ca[0] \neq (0, 0)$

(b) $n \geq 1$

(c) $nev > 0$ (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (c) を満足しなかった.	既定値にセットして処理を続ける.
2000+i	i + 1 番目以降は最大関数評価回数で収束しなかった.	i + 1 番目以降の根が収束しないままで処理を終了する.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.

(6) **注意事項**

(a) 引数 nev は, 0 を入力すれば既定値がセットされる.

(b) 重根のときの相対精度は, 根の多重度を n として, $\sqrt[n]{\text{誤差判定のための単位}}$ 程度になる.

(7) **使用例**

(a) 問 題

$x^{10} - 100\sqrt{-1}x^9 - 17x^6 + 1700\sqrt{-1}x^5 + 16x^2 - 1600\sqrt{-1}x = 0$ を解く.

(b) 入力データ

代数方程式の係数を格納した配列 ca:

ca [0] =(1.0, 0.0), ca [1] =(0.0, −100.0), ca [2] =(0.0, 0.0), ca [3] =(0.0, 0.0), ca [4] =(−17.0, 0.0), ca [5] =(0.0, 1700.0), ca [6] =(0.0, 0.0), ca [7] =(0.0, 0.0), ca [8] =(16.0, 0.0), ca [9] =(0.0, −1600.0), ca [10] =(0.0, 0.0)

n=10, nev=0

(c) 主プログラム

```
/*      C interface example for ASL_zlacha */

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

int main()
{
    double _Complex *ca;
    int n;
    int nev;
    double _Complex *cx;
    double *wk;
    double _Complex *cwk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_zlacha ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &nev );

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


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


    cwk = ( double _Complex * )malloc((size_t)( sizeof(double _Complex) * (4*(n+1)) ));
    if( cwk == NULL )
    {
        printf( "no enough memory for array cwk\n" );
        return -1;
    }

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

    printf( "\tDegree of Algebraic Equation = %6d\n", n );
    printf( "\n\tMaximum Number of Function Evaluation = %6d\n", nev );
    printf( "\n\tCoefficients of Algebraic Equation\n\n" );
```

```
    printf( "\t      Real Part  Imaginary Part\n" );
    for( i=0 ; i<n+1 ; i++ )
    {
        double tmp_re, tmp_im;
        fscanf( fp, "%lf,%lf", &tmp_re ,&tmp_im);
        ca[i] = tmp_re + tmp_im * _Complex_I;
        printf( "\t%15.3g %15.3g\n", creal(ca[i]),cimag(ca[i]) );
    }


    fclose( fp );

    ierr = ASL_zlacha(ca, n, &nev, cx, wk, cwk);

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

    printf( "\n\tPractical Number of Function Evaluations =%6d\n",nev );
    printf( "\n\tRoots\n\n" );
    printf( "\t      Real Part  Imaginary Part\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t%15.3g %15.3g\n", creal(cx[i]),cimag(cx[i]) );
    }

    free( ca );
    free( cx );
    free( wk );
    free( cwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zlacha ***

 ** Input **

Degree of Algebraic Equation =      5

Maximum Number of Function Evaluation =      0

Coefficients of Algebraic Equation

      Real Part  Imaginary Part
              1              1
             11             -1
            123            493
      -3.07e+03       1.45e+03
      -1.52e+04      -8.49e+03
       2.47e+04      -4.29e+04

 ** Output **

ierr =      0

Practical Number of Function Evaluations =     6

Roots

      Real Part  Imaginary Part
              3             -4
             -7             24
             -5              1
              2            -10
              2             -5
```

4.3 非線形方程式

4.3.1 ASL_dlnrds, ASL_rlnrds
実関数の根 (初期値指定) (導関数定義必要)

(1) **機 能**

導関数を定義して, 非線形方程式の 1 根を初期値から求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlnrds  (f, df, & x, er, nev);
```

単精度関数:

```
ierr = ASL_rlnrds  (f, df, & x, er, nev);
```

(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	f	{D / R}	–	入 力	方程式を定義する関数 f(x) の関数名
2	df	{D / R}	–	入 力	導関数を定義する関数 df(x) の関数名
3	x	{D∗ / R∗}	1	入 力	根の初期値
				出 力	根
4	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位) × 64)
5	nev	I*	2	入 力	nev [0] :最大関数評価回数 (既定値:100) nev [1] :最大導関数評価回数 (既定値:100)
				出 力	nev [0] :実際の関数評価回数 nev [1] :実際の導関数評価回数
6	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nev[0] > 0 (既定値にするため, 0 を入力する場合は除く)

(b) nev[1] > 0 (既定値にするため, 0 を入力する場合は除く)

(c) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(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.	
1500	制限条件 (a), (b) または (c) を満足しなかった.	既定値にセットして処理を続ける.
4000	根が求められない.	処理を打ち切る.
5000	最大関数評価回数または最大導関数評価回数に到達しても根が求められない.	その時点での x を返す.

(6) **注意事項**

(a) 関数 f, df の作り方は以下のようにする

```
double FORTRAN f(double * x)
{
    return(f(*x));
}
double FORTRAN df(double * x)
{
    return (f'(*x));
}
```

(b) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(c) 初期値が根から離れていても大域的収束性を持ち, 問題を解くことができるようになっている.

(d) 収束判定は {| 根の更新量 |<er× max(1,| 根の値 |)} および {| 関数値 |<er+64×(誤差判定のための単位) × | 根の値 |} が成立したとき収束したものとする.

(7) **使用例**

(a) 問 題

$$\begin{cases} f = x^7 + 28x^4 - 483.809683 = 0 \\ f' = 7x^6 + 112x^3 \end{cases}$$

の 1 根を求める.

(b) 入力データ

関数 $f(x)$ に対応する関数名: f1

導関数 $f'(x)$ に対応する関数名: f2

x = −1.0, er=0.0, nev [0] =0, nev [1] =0

(c) 主プログラム

```
/*      C interface example for ASL_dlnrds */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double    f1(double *x)
#else
double    f1(x)
double *x;
```

```
#endif
{
    return (pow((*x),7.0)+28.0*pow((*x),4.0)-483.809683);
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double    f2(double *x)
#else
double    f2(x)
double *x;
#endif
{
    return (7.0*pow((*x),6.0)+112.0*pow((*x),3.0));
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double er;
    int m[2];
    int ierr;
    FILE *fp;

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

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

    fscanf( fp, "%d", &m[0] );
    fscanf( fp, "%d", &m[1] );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &x );
    printf( "\tMaximum Number of Function Evaluations =%6d\n", m[0] );
    printf( "\tMaximum Number of Derivative Evaluations =%6d\n", m[1] );
    printf( "\tRequired Accuracy =%8.3g\n", er );
    printf( "\tInitial Value of Root =%8.3g\n", x );

    fclose( fp );

    ierr = ASL_dlnrds(f1, f2, &x, er, m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Function Evaluations =%6d\n", m[0] );
    printf( "\tPractical Number of Derivative Evaluations =%6d\n", m[1] );
    printf( "\tRoot =%8.3g\n", x );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dlnrds ***

 ** Input **

Maximum Number of Function Evaluations =     0
Maximum Number of Derivative Evaluations =     0
Required Accuracy =       0
Initial Value of Root =      -1

 ** Output **

ierr =      0

Practical Number of Function Evaluations =    36
Practical Number of Derivative Evaluations =     1
Root =    1.93
```

4.3.2 ASL_dlnris, ASL_rlnris
実関数の根 (初期値指定) (導関数定義不要)

(1) **機 能**

導関数が与えられないものとして非線形方程式の 1 根を初期値から求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlnris  (f, & x, er, & nev);
```

単精度関数:

```
ierr = ASL_rlnris  (f, & x, er, & nev);
```

(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	f	{D / R}	–	入 力	方程式を定義する関数 f(x) の関数名
2	x	{D∗ / R∗}	1	入 力	根の初期値
				出 力	根
3	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位) × 64)
4	nev	I*	1	入 力	最大関数評価回数 (既定値:100)
				出 力	実際の関数評価回数
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nev $>$ 0 (既定値にするため, 0 を入力する場合は除く)

(b) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(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.	
1500	制限条件 (a) または (b) を満足しなかった.	既定値にセットして処理を続ける.
4000	根が求められない.	処理を打ち切る.
5000	最大関数評価回数に到達しても根が求められない.	その時点での x を返す.

(6) **注意事項**

(a) 関数 f の作り方は以下のようにする.

double FORTRAN f(double ∗ x)
{
return(f(∗x));
}

(b) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(c) 初期値が根から離れていても大域性収束性を持ち, 問題を解くことができるようになっている.

(d) 収束判定は {| 根の更新量 |<er× max(1,| 根の値 |)} および {| 関数値 |<er+64×(誤差判定のための単位) × | 根の値 |} が成立したとき収束したものとする.

(7) **使用例**

(a) 問 題

$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$ の 1 根を求める.

(b) 入力データ

関数 $f(x)$ に対応する関数名: f

x=0.0, er=0.0, nev=0

(c) 主プログラム

```
/*      C interface example for ASL_dlnris */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
#else
double f(x)
double *x;
#endif
{
    return exp(0.01*(*x))+(3.0)-((*x)-231.0)*((*x)-597.0);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double er;
    int m;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dlnris ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &x );
    printf( "\tMaximum Number of Function Evaluations =%6d\n", m );
    printf( "\tRequired Accuracy =%8.3g\n", er );
    printf( "\tInitial Value of Root =%8.3g\n", x );

    fclose( fp );

    ierr = ASL_dlnris(f, &x, er, &m);
```

```
        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tPractical Number of Function Evaluations =%6d\n", m );
        printf( "\tRoot =%8.3g\n", x );

        return 0;
    }
```

(d) 出力結果

```
        *** ASL_dlnris ***

         ** Input **

        Maximum Number of Function Evaluations =     0
        Required Accuracy =       0
        Initial Value of Root =       0

         ** Output **

        ierr =      0

        Practical Number of Function Evaluations =    10
        Root =     231
```

4.3.3 ASL_dlnrss, ASL_rlnrss
実関数の根 (区間指定) (導関数定義不要)

(1) **機 能**

非線形方程式の関数値符号が変わる区間内の 1 根を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlnrss  (f, ax, bx, er, & x);
```

単精度関数:

```
ierr = ASL_rlnrss  (f, ax, bx, er, & x);
```

(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	f	{D / R}	–	入 力	方程式を定義する関数 f(x) の関数名
2	ax	{D / R}	1	入 力	根をはさむ区間の左端
3	bx	{D / R}	1	入 力	根をはさむ区間の右端
4	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位) × 64)
5	x	{D∗ / R∗}	1	出 力	根
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) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(b) ax $\neq$ bx

(c) f(ax) $\times$ f(bx) $\leq$ 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.	
1500	制限条件 (a) を満足しなかった.	既定値にセットして処理を続ける.
3000	制限条件 (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) 関数 f の作り方は以下のようにする.

```
double FORTRAN f(double *x)
{
    return(f(*x));
}
```

(b) 引数 er は, 0.0 を入力すれば既定値がセットされる.

(c) ax が区間の右端, bx が区間の左端のときでも, 問題が解ける.

(d) 収束判定は,
{| 根をはさむ区間 |$\leq$ er + 2$\times$ (誤差判定のための単位) $\times$ | 根の値 | が成立したとき収束したものとする.

(7) **使用例**

(a) 問 題
$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$ の 1 根を求める.

(b) 入力データ
関数 $f(x)$ に対応する関数名: f
ax $= -200.0$, bx=240.0, er=0.0

(c) 主プログラム

```
/*      C interface example for ASL_dlnrss */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
#else
double f(x)
double *x;
#endif
{
    return exp(0.01*(*x))+(3.0)-((*x)-231.0)*((*x)-597.0);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
    double bx;
    double er;
    double x;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dlnrss ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    printf( "\tRequired Accuracy =%8.3g\n", er );
    printf( "\n\tInterval in Which Roots Exist\n" );
    printf( "\t  ax=%8.3g\n", ax );
    printf( "\t  bx=%8.3g\n", bx );

    fclose( fp );

    ierr = ASL_dlnrss(f, ax, bx, er, &x);
```

```
        printf( "\n     ** Output **\n\n" );
        printf( "\tierr = %6d\n", ierr );
        printf( "\n\tRoot =%8.3g\n", x );

        return 0;
    }
```

(d) 出力結果

```
*** ASL_dlnrss ***

 ** Input **

Required Accuracy =       0

Interval in Which Roots Exist
  ax=    -200
  bx=     240

 ** Output **

ierr =      0

Root =     231
```

4.3.4 ASL_dlnrsa, ASL_rlnrsa
実関数の全根 (区間指定) (導関数定義不要)

(1) **機 能**

非線形方程式の区間内の全根を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dlnrsa (f, ax, bx, er, & nev, x, & m);

単精度関数:

ierr = ASL_rlnrsa (f, ax, bx, er, & nev, x, & 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	f	{D / R}	–	入 力	方程式を定義する関数 f(x) の関数名
2	ax	{D / R}	1	入 力	根をはさむ区間の左端
3	bx	{D / R}	1	入 力	根をはさむ区間の右端
4	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位) × 64)
5	nev	I*	1	入 力	最大関数評価回数 (既定値:100)
				出 力	実際の関数評価回数
6	x	{D* / R*}	m	出 力	根
7	m	I*	1	入 力	求める根の最大数
				出 力	求められた根の数
8	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(b) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(c) m > 0

(d) ax $\neq$ bx

(e) ax $<$ bx

(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.	
1200	制限条件 (e) を満足しなかった.	ax と bx を交換して処理を続ける.
1500	制限条件 (a) または (b) を満足しなかった.	既定値にセットして処理を続ける.
2000	関数評価が最大関数評価回数に達した.	求まった m 個の根までを出力し, 処理を打ち切る.
2500	根の数が根の最大数 m をこえる.	
3000	制限条件 (c) または (d) を満足しなかった.	処理を打ち切る.
4000	区間内に根が存在しない.	

(6) **注意事項**

(a) 関数 f の作り方は以下のようにする.

```
double FORTRAN f(double *x)
{
    return(f(*x));
}
```

(b) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(c) 収束判定は
{| 根の更新量 |<er× max(1,| 根の値 |)} および {| 関数値 |<er+64× (誤差判定のための単位)× | 根の値 |} が成立したとき収束したものとする.

(d) max(2× (要求精度), (| A | + | B |)× (誤差判定のための単位), $\sqrt[3]{\text{誤差判定のための単位}}$) よりも近接した 2 根は, 根の分離ができないのでどちらか一方の根が求められないことになる.

(7) **使用例**

(a) 問 題
$f = \exp(0.01x) + 3 - (x - 231)(x - 597) = 0$ の全根を求める.

(b) 入力データ
関数 $f(x)$ に対応する関数名: f
ax = −200.0, bx=800.0, er=0.0, nev=0, m=15

(c) 主プログラム

```
/*      C interface example for ASL_dlnrsa */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
#else
double f(x)
double *x;
#endif
{
    return exp(0.01*(*x))+(3.0)-((*x)-231.0)*((*x)-597.0);
}
#ifdef __cplusplus
}
```

```
#endif


int main()
{
    double ax;
    double bx;
    double er;
    int nev;
    double *x;
    int m;
    int ierr;
    int i;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );
    x  = ( double * )malloc((size_t)( sizeof(double) * m ));
    if( x  == NULL )
    {
        printf( "no enough memory for array  x\n" );
        return -1;
    }

    printf( "\tMaximum Number of Function Evaluations =%6d\n", nev );
    printf( "\tRequired Accuracy =%8.3g\n", er );
    printf( "\n\tInterval in Which Roots Exist\n" );
    printf( "\t  ax=%8.3g\n", ax );
    printf( "\t  bx=%8.3g\n", bx );

    fclose( fp );

    ierr = ASL_dlnrsa(f, ax, bx, er, &nev, x, &m);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Function Evaluations =%6d\n", nev );
    printf( "\n\tRoots\n");
    for( i=0 ; i<m ; i++ )
    {
        printf( "\t%8.3g\n", x[i] );
    }

    free( x );

    return 0;
}
```

(d) 出力結果

```
    *** ASL_dlnrsa ***

     ** Input **

    Maximum Number of Function Evaluations =     0
    Required Accuracy =       0

    Interval in Which Roots Exist
      ax=    -200
      bx=     800

     ** Output **

    ierr =      0

    Practical Number of Function Evaluations =    98

    Roots
         231
         598
```

4.3.5 ASL_zlncis, ASL_clncis
複素関数の根 (初期値指定) (導関数定義不要)

(1) **機 能**

導関数が与えられないものとして複素型の非線形方程式の 1 根を初期値から求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_zlncis  (f, & cx, er, & nev);
```

単精度関数:

```
ierr = ASL_clncis  (f, & cx, er, & nev);
```

(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	f	–	–	入 力	方程式を定義する関数 f(x, y) の関数名
2	cx	{ Z∗ / C∗ }	1	入 力	根の初期値
				出 力	根
3	er	{ D / R }	1	入 力	要求精度 (既定値:誤差判定のための単位) × 64)
4	nev	I*	1	入 力	最大関数評価回数 (既定値:100)
				出 力	実際の関数評価回数
5	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) nev $>$ 0 (既定値にするため, 0 を入力する場合は除く)

(b) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(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.	
1500	制限条件 (a) または (b) を満足しなかった.	既定値にセットして処理を続ける.
4000	ゼロ除算エラーが発生した.	処理を打ち切る.
5000	最大関数評価回数に到達しても根が求められない.	その時点での x を返す.

(6) **注意事項**

(a) 関数 f の作り方は以下のようにする.

```
/ * C interface example for ASL_zincis * /
#include < stdio.h >
#include < complex.h >
#include < asl.h >

void FORTRAN f (double_Complex * x, double_Complex * y)
{
    *y = ∼
}
int main()
{
      ≀
    ierr = ASL_zincis(f,&cx, er, &nev);
      ≀
}
```

(b) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(c) 収束判定は,
{| 根の更新量 |<er× max(1,| 根の値 |)} および {| 関数値 |<er+64×(誤差判定のための単位) × | 根の値 |} が成立したとき収束したものとする.

(7) **使用例**

(a) 問 題
$f = x^{10} - 1 = 0$ の 1 根を求める.

(b) 入力データ
関数 $f(x)$ に対応する関数名: f
cx $= (1.0, -1.0)$, er $= 1.0\mathrm{e} - 10$, nev=100

(c) 主プログラム

```
/*      C interface example for ASL_zlncis */

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

#ifdef __cplusplus
extern "C"
{
#endif
void      f(double _Complex *x, double _Complex *y)
{
    double _Complex temp;
    int i;
    temp = (*x) * (*x);
    for( i=3 ; i<11 ; i++ )
        temp = temp * (*x);
    *y = temp-1.0;
    return;
}
#ifdef __cplusplus
}
#endif

int main()
{
    double _Complex cx;
    double er;
```

```
    int nev;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_zlncis ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    double tmp_re, tmp_im;
    fscanf( fp, "%lf", &tmp_re );
    fscanf( fp, "%lf", &tmp_im );
    cx = tmp_re + tmp_im * _Complex_I;
    printf( "\tMaximum Number of Function Evaluations =%6d\n", nev );
    printf( "\n\tRequired Accuracy =%8.3g\n", er );
    printf( "\n\tInitial Value of Root\n" );
    printf( "\tReal Part            Imaginary Part\n" );
    printf( "\t%8.3g\t    %8.3g\n", creal(cx) , cimag(cx) );

    fclose( fp );

    ierr = ASL_zlncis(f, &cx, er, &nev);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Function Evaluations =%6d\n", nev );
    printf( "\n\tRoot\n");
    printf( "\tReal Part            Imaginary Part\n" );
    printf( "\t%8.3g\t    %8.3g\n", creal(cx) , cimag(cx) );

    return 0;
}
```

(d) 出力結果

```
*** ASL_zlncis ***

 ** Input **

Maximum Number of Function Evaluations =   100

Required Accuracy =   1e-10

Initial Value of Root
Real Part            Imaginary Part
       1               1

 ** Output **

ierr =      0

Practical Number of Function Evaluations =    11

Root
Real Part            Imaginary Part
   0.809           0.588
```

4.4 連立非線形方程式

4.4.1 ASL_dlsrds, ASL_rlsrds
連立非線形方程式の根 (ヤコビ行列定義任意)

(1) **機 能**

ヤコビ行列を定義する場合および定義しない場合のどちらにも対応して連立非線形方程式の 1 根を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlsrds  (sub, subj, x, n, er, nev, isw, iwk, wk, dwk);
```

単精度関数:

```
ierr = ASL_rlsrds  (sub, subj, x, n, er, nev, isw, iwk, wk, dwk);
```

(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	sub	−	−	入 力	n 個の非線形方程式を定義する 関数 sub(x, n, f) の関数名
2	subj	−	−	入 力	ヤコビ行列を定義する 関数 subj(x, n, a) の関数名
3	x	{D∗ / R∗}	n	入 力	根の初期値
				出 力	根
4	n	I	1	入 力	非線形方程式の連立数
5	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位)×64
6	nev	I*	2	入 力	nev [0] :最大関数評価回数 (既定値: 100 × n) nev [1] :最大ヤコビ行列評価回数 (既定値: 100 × n)
				出 力	nev [0] :実際の関数評価回数 nev [1] :実際のヤコビ行列評価回数
7	isw	I	1	入 力	入力スイッチ isw = 0:ヤコビ行列自動計算 isw ≠ 0:使用者側の定義したヤコビ行列作成用関数を使う. (注意事項 (b) 参照)
8	iwk	I*	n	ワーク	作業領域
9	wk	{D∗ / R∗}	n × (n + 3)	ワーク	作業領域
10	dwk	D*	内容参照	ワーク	作業領域. この変数の型は, 倍精度. **大きさ:** n × (2 × n + 2)
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) nev [0] >0 (既定値にセットするため, 0 を入力する場合は除く)

(b) nev [1] >0 (既定値にセットするため, 0 を入力する場合は除く)

(c) er≥ 誤差判定のための単位 (既定値にセットするため, 0.0 を入力する場合は除く)

(d) 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.	
1500	制限条件 (a) または (b) または (c) を満足しなかった.	既定値にセットして処理を続ける.
3000	制限条件 (d) を満足しなかった.	処理を打ち切る.
4000	連立 1 次方程式を解くときにエラーが発生した.	
4500	非線形性が強すぎての解の更新ができなくなった.	
5000	最大関数評価回数または最大ヤコビ行列評価回数に到達したが, 解が得られなかった.	その時点での x を返す.

(6) **注意事項**

(a) 関数の作り方は次に示すとおりである.

連立非線形方程式を

$$\begin{cases} f_1(x_1,\cdots,x_N)=0 \\ \vdots \\ f_N(x_1,\cdots,x_N)=0 \end{cases}$$

とすると, 関数 sub は

```
void FORTRAN sub(double *x, int *n, double *f)
{
        f[0]= f1(x[0],···,x[*n − 1]);
  ⋮
        f[(*n) − 1] =f(*n)(x[0],···,x[*n − 1]);
}
```

ヤコビ行列 A は

$$A=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix} = \begin{bmatrix} a[0], & \cdots, & a[n*(n-1)] \\ \vdots & & \vdots \\ \vdots & & \vdots \\ a[n-1]), & \cdots, & a[n-1+n*(n-1)] \end{bmatrix}$$

よって関数 subj は,

```
void FORTRAN subj(double *x, int *n, double *a)
{
        a[0] =∂f1/∂x1
        a[(*n) − 1] = ∂fn/∂x1
```

$$\vdots$$
$$\mathrm{a}[(*\mathrm{n})*((*\mathrm{n})-1)] = \partial f_1/\partial x_n$$
$$\vdots$$
$$\mathrm{a}[(*\mathrm{n})*(*\mathrm{n})-1)] = \partial f_n/\partial x_n$$

}

として記述される.

例

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1x_2 + x_2x_3 + x_3x_1 = 0 \\ x_1x_2x_3 = 0 \end{cases}$$

$$a = \begin{bmatrix} 1, & 1, & 1 \\ x_2 + x_3, & x_1 + x_3, & x_1 + x_2 \\ x_2x_3, & x_1x_3, & x_1x_2 \end{bmatrix}$$

したがって各関数 sub, subj は次のように記述される.

```
void FORTRAN sub(double *x, int *n, double *f)
{
        f[0]= x[0]+x[1]+x[2];
        f[1]= x[0]*x[1]+x[1]*x[2]+x[2]*x[0];
        f[2]= x[0]*x[1]*x[2];
}
void FORTRAN subj(double *x, int *n, double *a)
{
        a[0]= 1.0;
        a[1]= x[1]+x[2];
        a[2]= x[1]*x[2];
        a[3]= 1.0;
        a[4]= x[0]+x[2];
        a[5]= x[0]*x[2];
        a[6]= 1.0;
        a[7]= x[0]+x[1];
        a[8]= x[0]*x[1];
}
```

(b) isw=0 のときは, subj という引数はダミーであり, 関数を作る必要はない. また, nev [1] も入力不要である. isw $\neq 0$ のときは, subj の実際の名前の関数を作る必要がある. また, nev [1] も入力が必要である.

(c) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) 収束判定はすべての根またはすべての関数値に対し, {$\|$ 根の更新量 $\|_\infty <$ er $\times \max(1, \|$ 根の値 $\|_\infty)$} および {$\|$ 関数値 $\|_\infty <$ er $+ 64 \times \varepsilon \times \|$ 根の値 $\|_\infty$} が成立したとき収束したものする.

(7) 使用例

(a) 問 題

$\begin{cases} 10(\mathrm{x}_2 - \mathrm{x}_1^2) = 0 \\ 1 - \mathrm{x}_1 = 0 \end{cases}$ を解く

(b) 入力データ

非線形方程式に対応した関数 $\boldsymbol{f}(x)$ を定義する関数名: f

x[0] $= -1.2$, x [1] $= 1.0$, n $= 2$, er $= 0.0$, nev [0] $= 200$, nev [1] $= 200$, isw $= 0$

(c) 主プログラム

```
/*      C interface example for ASL_dlsrds */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void   f1(double *x,int *n,double *f)
#else
void   f1(x,n,f)
double *x;
double *f;
int  *n;
#endif
{
    f[0]=10.0*(x[1]-x[0]*x[0]);
    f[1]=1.0-x[0];
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void   f2(double *x,int *n,double *a)
#else
void   f2(x,n,a)
double *x;
double *a;
int  *n;
#endif
{
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x;
    int n;
    double er;
    int nev[2];
    int isw;
    int *iwk;
    double *wk;
    double *dwk;
    int ierr;
    int i;
    FILE *fp;

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

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

    fscanf( fp, "%d", &nev[0] );
    fscanf( fp, "%d", &nev[1] );
    fscanf( fp, "%d", &n );
    iwk = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*(n+3)) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
```

```
        return -1;
    }

    dwk  = ( double * )malloc((size_t)( sizeof(double) * (n*(2*n+2)) ));
    if( dwk  == NULL )
    {
        printf( "no enough memory for array dwk\n" );
        return -1;
    }
    fscanf( fp, "%lf", &er );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    fscanf( fp, "%d", &isw );
    printf( "\tMaximum Number of Functions Evaluations =%6d\n", nev[0] );
    printf( "\tMaximum Number of Jacobian Evaluations =%6d\n", nev[1] );
    printf( "\tNumber of Nonlinear Equations =%6d\n", n );
    printf( "\tRequired Accuracy =%8.3g\n", er );
    printf( "\n\tInitial Value of Root \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t%8.3g\n", x[i] );
    }

    fclose( fp );

    ierr = ASL_dlsrds(f1, f2, x, n, er, nev, isw, iwk, wk, dwk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tPractical Number of Functions Evaluations =%6d\n", nev[0] );
    printf( "\tPractical Number of Jacobian Evaluations =%6d\n", nev[1] );
    printf( "\n\tRoot \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t%8.3g\n", x[i] );
    }

    free( x );
    free( iwk );
    free( wk );
    free( dwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dlsrds ***

 ** Input **

Maximum Number of Functions Evaluations =   200
Maximum Number of Jacobian Evaluations =   200
Number of Nonlinear Equations =     2
Required Accuracy =       0

Initial Value of Root
    -1.2
       1

 ** Output **

ierr =      0

Practical Number of Functions Evaluations =    66
Practical Number of Jacobian Evaluations =     0

Root
       1
       1
```

4.4.2 ASL_dlsris, ASL_rlsris
連立非線形方程式の根 (ヤコビ行列定義不要)

(1) **機 能**

ヤコビ行列を定義しないものとして連立非線形方程式の 1 根を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dlsris  (sub, x, n, er, & nev, iwk, wk);
```

単精度関数:

```
ierr = ASL_rlsris  (sub, x, n, er, & nev, iwk, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	sub	−	−	入 力	n 個の非線形方程式を定義する関数 sub(x, n, f) の関数名
2	x	{D∗ / R∗}	n	入 力	根の初期値
				出 力	根
3	n	I	1	入 力	連立非線形方程式の連立数
4	er	{D / R}	1	入 力	要求精度 (既定値:(誤差判定のための単位) × 64)
5	nev	I*	1	入 力	最大関数評価回数 (既定値: 100 × n)
				出 力	実際の関数評価回数
6	iwk	I*	2 × n	ワーク	作業領域
7	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** n × (3 × n + 7)
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) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(b) er ≥ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(c) 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.	
1500	制限条件 (a) または (b) を満足しなかった.	既定値にセットして処理を続ける.
3000	制限条件 (c) を満足しなかった.	処理を打ち切る.
4000+i	ヤコビ行列の行列式が 0 である. すなわち, i は最初に 0 となるヤコビ行列の (i, i) 成分である.	
4500	ヤコビ行列の逆行列またはステップサイズの計算時にゼロ除算エラーが発生した.	
5000	最大関数評価回数に到達したが解が得られなかった.	その時点での x を返す.

(6) **注意事項**

(a) 関数の作り方は, 次に示すとおりである.
連立非線形方程式を
$$\begin{cases} \mathrm{f_1}(\mathrm{x_1}, \cdots, \mathrm{x_N}) = 0 \\ \mathrm{f_N}(\mathrm{x_1}, \cdots, \mathrm{x_N}) = 0 \end{cases}$$
とすると, 関数 sub は

```
void FORTRAN sub(double *x, int *n, double *f)
{
        f[0] =f1(x[0], ..., x[*n - 1]);
   ⋮
        f[(*n) - 1] = f(*n)(x[0], ..., x[*n - 1]);
}
```

(b) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(c) 連立非線形方程式の各式をスケーリングしてなくても解けるが, 効率を考えるとスケーリングした方がよい.

(d) 収束判定はすべての根またはすべての関数値に対し, $\{\| \text{根の更新量} \|_\infty < \mathrm{er} \times \max(1, \| \text{根の値} \|_\infty)\}$ および $\{\| \text{関数値} \|_\infty < \mathrm{er} + 64 \times \varepsilon \times \| \text{根の値} \|_\infty\}$ が成立したとき収束したものとする.

(7) **使用例**

(a) 問 題
$$\begin{cases} 10(\mathrm{x_2} - \mathrm{x_1^2}) = 0 \\ 1 - \mathrm{x_1} = 0 \end{cases}$$
を解く

(b) 入力データ
非線形方程式に対応した関数 $\boldsymbol{f}(x)$ を定義する関数名: f
x[0] = −1.2, x [1] =1.0, n=2, er=0.0, nev=200

(c) 主プログラム

```
/*      C interface example for ASL_dlsris */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void   f(double *x,int *n,double *f)
#else
void   f(x,n,f)
double *x;
double *f;
int  *n;
#endif
{
    f[0]=10.0*(x[1]-x[0]*x[0]);
    f[1]=1.0-x[0];
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x;
    int nn;
    double ddps;
    int m;
    int *iwk;
    double *wk;
    int ierr;
    int i,nwk;
    FILE *fp;

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

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

    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &nn );
    fscanf( fp, "%lf", &ddps );
    iwk = ( int * )malloc((size_t)( sizeof(int) * (2*nn) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

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

    nwk=nn*(3*nn+7);
    wk  = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    for( i=0 ; i<nn ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    printf( "\tMaximum Number of Functions Evaluations =%6d\n", m );
    printf( "\tNumber of NOnlinear Evaluations =%6d\n", nn );
    printf( "\tRequired Accuracy =%8.3g\n", ddps );
    printf( "\n\tInitial Value of Root \n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf(  "\t%8.3g\n", x[i] );
    }

    fclose( fp );

    ierr = ASL_dlsris(f, x, nn, ddps, &m, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\tPractical Number of Functions Evaluations =%6d\n", m );
    printf( "\n\tRoot \n" );
```

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

    free( x );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dlsris ***

 ** Input **

Maximum Number of Functions Evaluations =   200
Number of NOnlinear Evaluations =     2
Required Accuracy =       0

Initial Value of Root
    -1.2
       1

 ** Output **

ierr =      0
Practical Number of Functions Evaluations =    34

Root
       1
       1
```

第 5 章　極値問題・最適化

5.1　概 要

本章では, 制約なし 1 変数関数の極小化, 制約なし多変数関数の極小化, 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, 制約付き多変数線形関数の最小化 (線形計画, 混合 0-1 計画, 最小費用流問題, 日程計画問題, 輸送問題), 多変数 2 次関数の最小化 (2 次計画), 制約付き多変数関数の最小化 (非線形計画), ネットワーク上の距離の最短化 (最短路問題) を行う, 関数について説明する.

制約なし多変数関数の極小化では, 以下の関数が用意されている.

(i) 1 変数関数の極小化

この関数は, 与えられた初期点から出発して, 1 変数関数の極小値探索を行う.
多変数関数の極小化では, 以下の関数が用意されている.

(i) 多変数関数の極小化 (導関数定義不要)

(ii) 多変数関数の極小化 (導関数定義必要)

これらの関数は, 与えられた初期点から出発して, 多変数関数の極小値探索を行う. 利用者が関数の勾配ベクトルを計算する関数を用意できないときには, 導関数定義不要の関数を利用することができる. 関数の勾配ベクトルを計算する関数を用意できるときには, 導関数定義必要の関数を利用した方がよい結果が得られる.

制約なし関数二乗和の極小化では, 以下の関数が用意されている.

(i) 非線形最小二乗法 (導関数定義不要)

この関数は, 与えられた初期点から出発して, m 個の多変数関数の二乗和の極小値探索を行う. 利用者が関数のヤコビ行列を計算する関数を用意する必要はない.

制約付き 1 変数関数の極小化では, 以下の関数が用意されている.

(i) 1 変数関数の極小化 (区間指定)

この関数は, 与えられた区間の中で, 1 変数関数の極小値探索を行う.

制約付き多変数線形関数の最小化では, 以下の関数が用意されている.

(i) 制約付き多変数線形関数の最小化 (線形制約)

(ii) 制約付き多変数線形関数の最小化 (実不規則スパース行列で与えられる線形制約)

(iii) 0-1 変数を含む制約付き多変数線形関数の最小化 (線形制約)

(iv) ネットワーク上の流れに対する費用の最小化

(v) プロジェクトの日程計画に対する費用の最小化

(vi) 供給地から需要地への輸送費用の最小化

これらの関数は与えられた線形制約の中で, 多変数線形関数の最小値探索を行う. (線形計画, 混合 0-1 計画, 最小費用流問題)

多変数 2 次関数の最小化では, 以下の関数が用意されている.

(i) 制約付き多変数凸型 2 次関数の最小化 (線形制約)

(ii) 制約付き多変数広義凸型 2 次関数の最小化 (線形制約)

(iii) 制約無し 0-1 多変数 2 次関数の最小化

これらの関数は与えられた制約の中で, 多変数 2 次関数の最小値探索を行う. (2 次計画問題)
制約付き多変数関数の最小化では, 以下の関数が用意されている.

(i) 制約付き多変数線形関数の最小化 (非線形制約)

この関数は与えられた非線形制約の中で, 多変数関数の最小値探索を行う. (非線形計画)
ネットワーク上の距離の最小化では, 以下の関数が用意されている.

(i) ネットワーク上のある節点から他のすべての節点までの距離の最小化

(ii) ネットワーク上の全 2 節点間の距離の最小化

(iii) ネットワーク上の 2 節点間の距離の最小化

これらの関数はネットワーク上の節点間の距離の最短化を行う. (最短路問題)

5.1.1 使用上の注意

(1) 探索の出発点はできるかぎり真解に近いところにとるのが望ましい.

(2) 要求精度は, $\sqrt{(\text{誤差判定のための単位})}$ 程度にとるのが適当である.

(3) 多変数関数の極小化を行う場合, 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい. たとえば, $f(\boldsymbol{x}) = 10000{x_1}^2 + {x_2}^3$ のような問題の場合, $y_1 = 100x_1$, $y_2 = x_2$ と変数変換を行って, $h(\boldsymbol{y}) = {y_1}^2 + {y_2}^3$ としたほうがよい結果が得られる. なお, 制約条件がある場合は, 制約条件も変数変換によって変更する必要がある.

5.1.2 使用しているアルゴリズム

5.1.2.1 1 変数関数の極小化

黄金分割探索と逐次放物線補間を組み合わせた方法を用いる．最初, 黄金分割探索で探索を開始し, 可能になれば逐次放物線補間に切り替える．これは, ブレント (1973) にもとづくものである．

(1) 黄金分割探索
黄金分割による区間縮小のアルゴリズムを以下に述べる．
縮小されるべき区間を $[a, b]$ とし, このなかの 1 点 $x(a < x < b)$ で関数値 $f(x)$ が計算されているとする．区間の中点 $c = (a+b)/2$ に対して, $x < c$ であるとき, $u = x + r(b - x)$ で関数値 $f(u)$ を計算し, $f(x) < f(u)$ ならば, $b = u$ とする．そうでなければ, $a = x$, $x = u$ とする．$x \geq c$ のときも, 同様である．このようにすると, 前に計算した関数値を利用できるので, 1 回区間縮小するのに関数計算を 1 回行えばよい．ここで, $r = (3 - \sqrt{5})/2$ とし, 最初に $x = a + r(b - a)$ ととっておくと, 常に一定の比率 $1 - r$ で区間縮小されていく．
収束の次数は 1 である．

(2) 逐次放物線補間
逐次放物線補間による区間縮小のアルゴリズムを以下に述べる．
縮小されるべき区間を $[a, b]$ とする．いま, 3 点 v, w, x で関数値が計算されているとき, $(v, f(v))$, $(w, f(w))$, $(x, f(x))$ の 3 点を通る放物線を作り, その頂点の x 座標を u とする．$x < u$ のとき, $f(x) < f(u)$ ならば, $b = u$ とし, そうでなければ, $a = x$, $x = u$ とする．$x \geq u$ のときも, 同様である．
極小点で $f''(x) > 0$ であり, 十分よい近似から出発すれば, 収束の次数は $1.324\cdots$ であることが証明される．

(3) 黄金分割探索から逐次放物線補間への切り替え
新たに関数値が計算されると, いままで計算された関数値のうち, もっとも小さな関数値の 3 点を与えるように v, w, x が更新される．可能ならば上記の放物線の頂点の x 座標 u が計算され, 次の条件が満たされたとき, 黄金分割探索から逐次放物線補間へ切り替えられる．

$$a < u < b$$

$$| x - u | < (b - a)/2$$

(4) 収束判定
要求精度を e_r とするとき, 次の条件により収束判定を行う．

$$\max(b - x, x - a) \leq 2e_r \max(1, | x |)$$

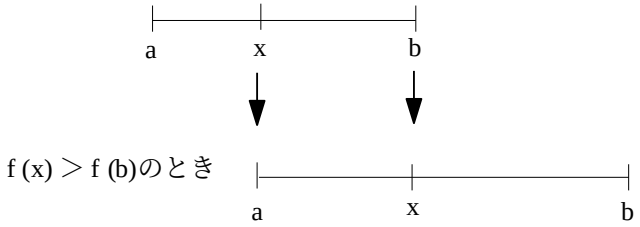
(5) 囲い込み
制約なしの極小化では, 任意の出発点から極小点を含む区間を求める囲い込みの操作を行っている．
このアルゴリズムは, 次のようになっている．

(i) 出発点 a の傾斜方向を求める．

(ii) 傾斜方向に $D = \max(\sqrt{\text{(誤差判定のための単位)}} \mid x \mid, 1.0, 2e_r \mid x \mid)$ の幅で囲い込み b 点とする．

(iii) b 点が a 点よりも関数値が小さい場合は極小点がないと判断し, b 点を x 点にして b 点を a 点から $(3 - r)(x - a)$ の幅に広げ囲い込む．こうすれば, $b - a : x - a = 1 : r$ の黄金分割比になる．b 点が a 点よりも関数値が大きい場合は区間 $[a, b]$ に極小点があるので, この間で探索を開始する．

(iv) b 点が x 点よりも関数値が小さい場合は, x 点を a 点に, b 点を x 点にし, 新しい b 点は a 点から $(3-r)(x-a)$ のところにとる. このようにして (iv) の作業を繰り返す. b 点が x 点よりも関数値が大きい場合は区間 $[a,b]$ で探索を開始する. このとき x 点は黄金分割探索の関数値計算点になっており, 囲い込みで得られた関数値を利用することができる.



5.1.2.2 多変数関数の極小化

問題は, n 変数の関数 $f(\boldsymbol{x})$ が与えられたとき, $f(\boldsymbol{x})$ を極小にする $\boldsymbol{x}$ を求めることである.
これは, 方程式 $\nabla f(\boldsymbol{x}) = 0$ の解になっている. この方程式を解くために近似解を $\boldsymbol{x}_0$ を出発点として逐次修正していく.
いま, $\boldsymbol{x}$ まで探索が進んで次の修正解 $\boldsymbol{x}'$ を求めようとしているとする. ニュートン法の修正ベクトル $\boldsymbol{d}$ は, 次式で与えられる.

$$\boldsymbol{d} = -H\boldsymbol{g}$$

ただし, $\boldsymbol{g}$ は勾配ベクトル $\boldsymbol{g} = \nabla^T f(\boldsymbol{x})$ であり, H はヘッセ行列 $B = \nabla^2 f(\boldsymbol{x})$ の逆行列である
(^{T} は転置を意味する). 実際には, $\boldsymbol{d}$ を方向ベクトルとして直線探索を行い次式により解を修正する.

$$\boldsymbol{x}' = \boldsymbol{x} + \alpha\boldsymbol{d}$$

ここで, H を直接に計算せず, 逐次更新していく方法を準ニュートン法という. α は (b) 直線探索で決める.

(1) ヘッセ行列の逆行列 H の計算
H の更新公式は種々のものが提案されているが, 本ライブラリでは BFGS 公式を用いる. これは, Broyden, Fletcher, Goldfarb, Shanno によって提案されたものである.
H の初期値は単位行列 E であり, 以後, 以下の式で更新する.

$$H' = \{E - \frac{\delta_\gamma^T}{\delta_\gamma^T}\}H\{E - \frac{\gamma\delta^T}{\delta_\gamma^T}\} + \frac{\delta\delta^T}{\delta_\gamma^T}\}$$

ただし,

$$\begin{aligned} \delta &= \boldsymbol{x}' - \boldsymbol{x} \\ \gamma &= \nabla^T f(\boldsymbol{x}') - \nabla^T f(\boldsymbol{x}) \end{aligned}$$

である.

(2) 直線探索
正確な直線探索は効率が良くないため, Armijo の方法を用いる.

$$f(\boldsymbol{x} + \beta^m\alpha_0\boldsymbol{d}) - f(\boldsymbol{x}) \leq \beta^m\alpha_0\mu\nabla f(\boldsymbol{x})\boldsymbol{d}$$

を満たす最小の非負整数 m を求めて, $\alpha = \beta^m\alpha_0$ とする.
ただし, $\alpha_0 > 0$, $0 < \beta < 1$, $0 < \mu < 1$ である.

(3) 収束判定

収束判定は以下の式によって行い, $\boldsymbol{x}'$ を解とする.

$\|\nabla^T f(\boldsymbol{x}')\|_\infty \leq \varepsilon (\|\boldsymbol{x}' - \boldsymbol{x}\|_\infty \leq \varepsilon$でかつ前回の修正に異常が発生した)

または,

$\|\boldsymbol{x}' - x\|_\infty \leq e_r \max(1, \|\boldsymbol{x}'\|_\infty)$ かつ $\|\nabla^T f(\boldsymbol{x}')\|_\infty \leq 2e_r$

ここで, ε は誤差判定のための単位, e_r は要求精度であり, $\|\boldsymbol{x}\|_\infty = \max_i \|\boldsymbol{x}_i\|$ である.

5.1.2.3 非線形最小二乗法

問題は, m 個の n 変数関数 $f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})$ が与えられたとき, 関数の二乗和,

$$S(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(\boldsymbol{x})^2 = \|\boldsymbol{f}(\boldsymbol{x})\|^2$$

を極小とする $\boldsymbol{x}$ を求めることである. ただし, $\boldsymbol{f}(\boldsymbol{x})$ は, $f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})$ を成分とするベクトル値関数であり, $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T \boldsymbol{x}}$ である (^{T} は転置を意味する).
それには方程式 $\partial S/\partial \boldsymbol{x} = \boldsymbol{0}$ を解く必要がある. 解を求めるために, 初期値 $\boldsymbol{x}_0$ から探索を開始し, パウエルのハイブリッド法を用いて, 逐次修正していく.
ハイブリッド法では, 関数の非線形性に応じて線形化したモデルのガウスニュートン解と最急降下解とを折衷する. 修正ベクトルは常に独立性の検査が行われ, 部分空間の中に閉じ込められてしまわないように配慮されている. また, ヤコビ行列の計算を直接行わず, 関数情報を用いて修正する.

(1) 修正ベクトル $\Delta\boldsymbol{x}$ の計算

$\boldsymbol{f}$ を線形化したモデルは,

$$S_L(\boldsymbol{x} + \Delta\boldsymbol{x}) = \|\boldsymbol{f}(\boldsymbol{x}) + A\Delta\boldsymbol{x}\|^2$$

である. ここで, A は$\boldsymbol{f}$ のヤコビ行列 $\partial\boldsymbol{f}/\partial\boldsymbol{x}$ である.
このモデルの最急降下解 $\Delta\boldsymbol{x}_S$ は, $\boldsymbol{b} = -A^T\boldsymbol{f}(\boldsymbol{x})$ として,

$$\Delta\boldsymbol{x}_S = (\|\boldsymbol{b}\|^2/\|A\boldsymbol{b}\|^2)\boldsymbol{b}$$

によって与えられる.
一方, ガウスニュートン解 $\Delta\boldsymbol{x}_G$ は, 正規方程式

$$A^T A\Delta\boldsymbol{x} = \boldsymbol{b}$$

を解いて得られる. 本ライブラリでは, これを解くために線形最小二乗法の QR 分解法を用いる.
一般に, $\|\Delta\boldsymbol{x}_S\| \leq \|\Delta\boldsymbol{x}_G\|$ であることが証明される.
この 2 つの解を関数の非線形性に応じて決定されるステップサイズ d によって以下のように折衷する.

(i) $d \leq \|\Delta\boldsymbol{x}_S\|$ の場合

$$\Delta\boldsymbol{x} = d\Delta\boldsymbol{x}_S/\|\Delta\boldsymbol{x}_S\|$$

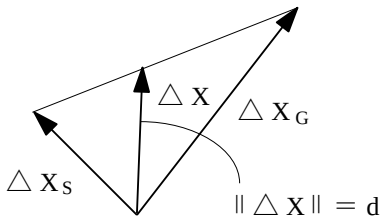
(ii) $\|\Delta\boldsymbol{x}_S\| < d < \|\Delta\boldsymbol{x}_G\|$ の場合 (図 5−1 参照)

$$\Delta\boldsymbol{x} = \alpha\Delta\boldsymbol{x}_S + \beta\Delta\boldsymbol{x}_G (\alpha > 0, \beta > 0, \|\Delta\boldsymbol{x}\| = d)$$

(iii) $\|\Delta\boldsymbol{x}_G\| \leq d$ の場合

$$\Delta\boldsymbol{x} = \Delta\boldsymbol{x}_G$$

図 5−1



(2) ステップサイズ d の変更

ステップサイズは, 初期値 $d = \|\Delta \boldsymbol{x}_S\|$ として, 以後, 関数の非線形性が強いときには減少させ, 線形に近いときには増加させる.

非線形性の程度を測るために線形のモデルの変化量 $\Delta S_L = \Delta S_L(\boldsymbol{x} + \Delta \boldsymbol{x}) - S_L(\boldsymbol{x})$ と実際の変化量 $\Delta S = S(\boldsymbol{x} + \Delta \boldsymbol{x}) - S(\boldsymbol{x})$ との比 $r = \Delta S / \Delta S_L$ を用いる.

a) $r < 0.1$ のとき非線形性が強いと判断し, d を半分にする.

b) $r \geq 0.1$ のとき, d の増加率 λ を以下のように計算する.

$$\lambda^2 = 1.0 - (r - 0.1)\Delta S_L / (S_P + \sqrt{(S_P^2 - S_S(r - 0.1)\Delta S_L)})$$

ただし, $\delta \boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x} + \Delta \boldsymbol{x}) - (\boldsymbol{f}(\boldsymbol{x}) + A\Delta \boldsymbol{x})$ として

$$S_P = \sum_{i=1}^{n} \|f_i(\boldsymbol{x} + \Delta \boldsymbol{x})\delta f_i\|$$

$$S_S = \|\delta \boldsymbol{f}\|^2$$

である.

実際には, d の振動を防ぐために 2 回続けて増加が要求されたときにだけ d を増加させる. また, 増加率は 2 以下におさえる. 実際の増加率 μ は以下のように計算される.

$$\begin{aligned} \mu &= \min(2, \lambda, \tau) \\ \tau &= \lambda / \mu \end{aligned}$$

ここで, τ は初期値が 1 であり, 縮小が要求されたとき 1 にリセットされる.

また, d には上限 $d_{\max}$ と下限 $d_{\min}$ が設けられており, その間にはいるように制御される.

(3) 修正ベクトルの独立性検査

ヤコビ行列の修正が効率よく行われるためには, 順次取られる修正ベクトルが互いに直交に近いことが必要である. そのために, ハイブリッド法では, 独自の独立性概念を定義して, 修正ベクトルができるだけ独立な方向に取られるように制御している. ベクトル $\boldsymbol{p}$ が i 個のベクトル $(\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_i)$ と独立であるとは, $\boldsymbol{p}$ がこれらのベクトルで張られた空間の任意のベクトルと 30 度以上の角度をなしていることである. パウエルによって考案された独立性検査の算法を以下に示す.

過去 $2n$ 回のヤコビ行列の修正で用いられたベクトルのうち互いに独立な n 個のベクトルを直交化して, $\Omega = (\omega_1, \omega_2, \cdots, \omega_n)$ に保持しておく. 大きさ n の配列 $\boldsymbol{j}$ を用いて, ω_i が何回前の修正ベクトルであったかという情報を保持しておく. すなわち, ω_i は j_i 回前に取られたベクトルであることを意味する. Ω は単位行列で初期化し, $\boldsymbol{j}$ は $j_i = n - i + 1$ $(i = 1, \cdots, n)$ で初期化する.

解の修正を行うときには, 以下のようにする.

(i) $\Delta\boldsymbol{x} = \Delta\boldsymbol{x}_G$ の場合
独立性のいかんにかかわらず, $\Delta\boldsymbol{x}$ を修正ベクトルとして採用する.

(ii) $\Delta\boldsymbol{x} = \Delta\boldsymbol{x}_G$ でない場合
$j_1 < 2n$ であるか, または $\Delta\boldsymbol{x}$ が $(\omega_2, \omega_3, \cdots, \omega_n)$ と独立なら, $\Delta\boldsymbol{x}$ を修正ベクトルとして採用する. そうでないなら, 解の修正を行わない.

(4) ヤコビ行列 A の計算
ヤコビ行列は, 最初の 1 回だけは差分によって求め, 後は逐次更新していく. この方法は, ブロイデンによるもので, 以下の式によって計算する.

$$A' = A + \delta\boldsymbol{f}\Delta\boldsymbol{x}^T/\|\Delta\boldsymbol{x}\|^2$$

ただし, $\|\Delta\boldsymbol{x}\| < d_{\min}$ であるか, または, $j_1 = 2n$ で $\Delta\boldsymbol{x}$ が $(\omega_2, \omega_3, \cdots, \omega_n)$ と独立でないなら, $\Delta\boldsymbol{x} = d_{\min}\omega_1$ とする.

(5) Ω と $\boldsymbol{j}$ の改訂
Ω と $\boldsymbol{j}$ の改訂は以下のように行う.
$\Delta\boldsymbol{x} = d_{\min}\omega_1$ としたときには,

$$\begin{aligned}
\omega_i &= \omega_{i+1} \quad (i = 1, \cdots, n-1) \\
\omega_n &= \omega_1 \\
j_i &= j_{i+1} + 1 \quad (i = 1, \cdots, n-1) \\
j_n &= 1
\end{aligned}$$

とすればよい. そうでないときには, 以下のようにする.
$(\omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta\boldsymbol{x})$ が互いに独立になる最小の k を求める.
$(\omega_1, \cdots, \omega_{k-1}, \omega_{k+1}, \omega_{k+2}, \cdots, \omega_n, \Delta\boldsymbol{x})$ を直交化して, それを改めて $(\omega_1, \omega_2, \cdots, \omega_n)$ とする.

$$\begin{aligned}
j_i &= j_i + 1 \quad (i = 1, \cdots, k-1) \\
j_i &= j_{i+1} + 1 \quad (i = k, \cdots, n-1) \\
j_n &= 1
\end{aligned}$$

このようにして, 常に, $j_1 \leq 2n$ となるようにする.

(6) 収束判定
収束判定は以下の式によって行い, $\boldsymbol{x} + \Delta\boldsymbol{x}$ を解とする.

$$\|\Delta\boldsymbol{x}\|_\infty \leq e_r \max(1, \|\boldsymbol{x} + \Delta\boldsymbol{x}\|_\infty)$$

ここで, e_r は要求精度であり, $\|\boldsymbol{x}\|_\infty = \max_i |\, x_i \,|$ である.

5.1.2.4 制約付き多変数線形関数の最小化 (線形制約)

ここでは, 次のように, n 次ベクトル $\boldsymbol{x}$ に関する多変数線形関数 $f(\boldsymbol{x})$ と, m 個の線形制約条件, 並びにベクトル $\boldsymbol{x}$ の定義域が与えられたとき, 線形制約条件を満たし, かつ $f(\boldsymbol{x})$ を最小にする n 次のベクトル $\boldsymbol{x}$ を求める問題 (線形計画問題) を扱う.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} \to \min$$

$$
\begin{array}{lrcll}
\text{線形制約条件：} & \boldsymbol{a}_i^T\boldsymbol{x} & = & b_i & (i=1,2,\cdots,m_e) \\
 & \boldsymbol{a}_i^T\boldsymbol{x} & \leq & b_i & (i=m_e+1,m_e+2,\cdots,m_{ne}) \\
 & \boldsymbol{a}_i^T\boldsymbol{x} & \geq & b_i & (i=m_{ne}+1,m_{ne}+2,\cdots,m) \quad 0\leq m_e\leq m_{ne}\leq m \\
\boldsymbol{x}\text{の定義域：} & \boldsymbol{d}\leq\boldsymbol{x}\leq\boldsymbol{u} & & &
\end{array}
$$

ここで, $\boldsymbol{c}, \boldsymbol{a}_i, \boldsymbol{d}, \boldsymbol{u}$ はそれぞれ n 次の定数ベクトルであり, (b_i) は m 次の定数ベクトルである. これらは問題によって定まる.

なお, $\boldsymbol{d}\leq\boldsymbol{x}$ というように, ベクトルに対して不等式を用いているが, これらは, ベクトルの各要素について, この不等式が成立するという意味で用いている.

なお, 変数 X_i を新たに追加することによって, 不等式制約条件 $\boldsymbol{a}_i^T\boldsymbol{x}\leq b_i$ は $\boldsymbol{a}_i^T\boldsymbol{x}+X_i=b_i$ に $\boldsymbol{a}_i^T\boldsymbol{x}\geq b_i$ は $\boldsymbol{a}_i^T\boldsymbol{x}-X_i=b_i$ にそれぞれ変換できる. この変数 X_i をスラック変数と呼ぶ. スラック変数の導入により, 制約条件は次のように全て等式で表現できるようになる.

$$
\begin{array}{ll}
\text{線形制約条件：} & A\boldsymbol{x}'=\boldsymbol{b} \\
\boldsymbol{x}\text{の定義域：} & \boldsymbol{d}'\leq\boldsymbol{x}'\leq\boldsymbol{u}'
\end{array}
$$

ここで, A は n 行 $n+m-m_e$ 列の行列, $\boldsymbol{x}'$, $\boldsymbol{b}$, $\boldsymbol{d}'$, $\boldsymbol{u}'$ はそれぞれ $n+m-m_e$ 次のベクトルである. 従って, 以降では等式制約条件のみを考える. なお, 表記を簡単にするため, $\boldsymbol{x}'$, $\boldsymbol{d}'$, $\boldsymbol{u}'$ をそれぞれ$\boldsymbol{x}$, $\boldsymbol{d}$, $\boldsymbol{u}$と表すことにする. 本ライブラリでは, 線形計画問題を解くために, 制約条件を与える行列 $A=(a_{ij})$ が密行列である場合を扱うための改訂シンプレックス法を用いた関数と, A がスパース行列である場合を扱うための内点法を用いた関数を用意している.

まず, 改訂シンプレックス法について説明する.

(1) 実行可能基底解

最適解は, 実行可能基底解の中に存在する. シンプレックス法は, 有限集合である実行可能基底解の中から最適解を求めるものである. 以下, 実行可能基底解について説明する.

行列 A の列から m 個の 1 次独立な列を選び, このように選んだ列を要素とする m 行 m 列の正則 B を考える. 今 $n+m-m_e$ 次の変換行列 P を

$$
\begin{array}{rcl}
AP & = & [B|L|U] \\
\left(P^{-1}\boldsymbol{x}\right)^T & = & \left[\boldsymbol{x}_B^T \mid \boldsymbol{x}_L^T \mid \boldsymbol{x}_U^T\right], \\
\left(P^{-1}\boldsymbol{d}\right)^T & = & \left[\boldsymbol{d}_B^T \mid \boldsymbol{d}_L^T \mid \boldsymbol{d}_U^T\right], \\
\left(P^{-1}\boldsymbol{u}\right)^T & = & \left[\boldsymbol{u}_B^T \mid \boldsymbol{u}_L^T \mid \boldsymbol{u}_U^T\right]
\end{array}
$$

を満たすように定めると, 制約条件は次の形に書き表せる.

$$
\begin{array}{rcl}
B\boldsymbol{x}_B+L\boldsymbol{x}_L+U\boldsymbol{x}_U & = & \boldsymbol{b} \\
\boldsymbol{d}_B\leq\boldsymbol{x}_B\leq\boldsymbol{u}_B & & \\
\boldsymbol{d}_L\leq\boldsymbol{x}_L\leq\boldsymbol{u}_L & & \\
\boldsymbol{d}_U\leq\boldsymbol{x}_U\leq\boldsymbol{u}_U & &
\end{array}
$$

ここでは, $\boldsymbol{x}_B, \boldsymbol{d}_B, \boldsymbol{u}_B$ は m 次のベクトル, L は m 行 ℓ 列の行列, $\boldsymbol{x}_L, \boldsymbol{d}_L, \boldsymbol{u}_L$ は ℓ 次のベクトル, U は m 行 $n-m_e-1$ 列の行列, $\boldsymbol{x}_U, \boldsymbol{d}_U, \boldsymbol{u}_U$ は $n-m_e-1$ 次のベクトルをあらわす. ただし, ℓ は $0\leq\ell\leq n-m_e$ を満たす整数である.

今, $\boldsymbol{x}_B, \boldsymbol{x}_L, \boldsymbol{x}_U$ が次の条件を満たしている場合, $\boldsymbol{x}=P\begin{bmatrix}\boldsymbol{x}_B\\ \boldsymbol{x}_L\\ \boldsymbol{x}_U\end{bmatrix}$ を実行可能解と呼ぶ.

$$
\boldsymbol{d}_B\leq\quad\boldsymbol{x}_B\quad\leq\boldsymbol{u}_B
$$

$$\begin{aligned}\boldsymbol{x}_L &= \boldsymbol{d}_L \\ \boldsymbol{x}_U &= \boldsymbol{u}_U\end{aligned}$$

なお, B を基底行列, $\boldsymbol{x}_B$ を基底変数, そして L, U を非基底行列, $\boldsymbol{x}_L, \boldsymbol{x}_U$ を非基底変数と呼ぶ.

(2) シンプレックス法

シンプレックス法の手順を示す.

(a) 初期実行可能基底解を求める.

(b) 他の非基底変数を固定した状態で, ある 1 つの非基底変数 X_{IN} ($\boldsymbol{x}_L$ または $\boldsymbol{x}_U$ の要素) を, ある変数 x($\boldsymbol{x}_B$ の要素または X_{IN}) が上限値または下限値になるまで, 変化させる.

(c) $\boldsymbol{x}_B$ が上限値または下限値になったときには, 基底と非基底を入れ替える.

(d) この操作を, 関数の値を小さくできる限り, いろいろな非基底変数 x を X_{IN} として選び, 繰り返し行う.

以下に具体的に説明する.

まず, 初期実行可能基底解を求める (参考文献 (7) 参照).

次に, 変化させる非基底変数 X_{IN} を求める. 基底変数, 非基底変数を用い, 制約条件を含めた形で関数値 $f(\boldsymbol{x})$ を表すと次式になる.

$$\begin{aligned}f(\boldsymbol{x}) &= \boldsymbol{c}_B^T B^{-1}\boldsymbol{b} + \boldsymbol{g}_L^T \boldsymbol{x}_L + \boldsymbol{g}_U^T \boldsymbol{x}_U \\ \boldsymbol{g}_L^T &= \boldsymbol{c}_L^T - \boldsymbol{c}_B^T B^{-1} L \\ \boldsymbol{g}_U^T &= \boldsymbol{c}_U^T - \boldsymbol{c}_B^T B^{-1} U\end{aligned}$$

ただし, $(P\boldsymbol{c})^T = (\boldsymbol{c}_B^T \ \ \boldsymbol{c}_L^T \ \ \boldsymbol{c}_U^T)$ で $\boldsymbol{c}_B, \boldsymbol{c}_L, \boldsymbol{c}_U$ はそれぞれ m 次, ℓ 次, $n - m_e - 1$ 次のベクトルである.

$\boldsymbol{x}_L = \boldsymbol{d}_L$ および $\boldsymbol{x}_U = \boldsymbol{u}_U$ であるので, $\boldsymbol{x}_L$ の要素は $+$ 方向, $\boldsymbol{x}_U$ の要素は $-$ 方向にしか変化させることができない. そのため, ある $\boldsymbol{g}_L$ の要素が負, またはある $\boldsymbol{g}_U$ の要素が正のときのみ関数値 $f(\boldsymbol{x})$ を減少させることができる. このような要素がないときは現在の値 $\boldsymbol{x}$ が最適解である. この条件を満たす $\boldsymbol{g}_U$ または $\boldsymbol{g}_L$ の要素で絶対値が最大のものが, 変化させる要素 IN となる.

このとき, 上限値または下限値に変化する x($\boldsymbol{x}_B$ の要素または X_{IN}) は次のものになる. 非基底変数 X_{IN} を変化させたとき, X_{IN} が $\boldsymbol{x}_L$ の要素と仮定すると, 制約条件を満たしている間は次式が成立する. ここで, $\boldsymbol{a}_{IN}$ は A の第 IN 列 (前述の $\boldsymbol{a}$ とは異なる), Δ は X_{IN} の変化量 ($\Delta \geq 0$) である.

$$\begin{aligned}\boldsymbol{d}_B \leq \ & \boldsymbol{x}_B - B^{-1}\boldsymbol{a}_{IN}\Delta \ \leq \boldsymbol{u}_B \\ \boldsymbol{d}_{IN} \leq \ & X_{IN} + \Delta \ \leq \boldsymbol{u}_{IN}\end{aligned}$$

Δ を変化させたとき, 最初に上式を満たさなくなる x($\boldsymbol{x}_B$ の要素または X_{IN}) が上限値または下限値に変化する変数になる. なお, $\boldsymbol{x}_{IN}$ が $\boldsymbol{x}_U$ の要素のときには, 上式ではなく次式になる.

$$\begin{aligned}\boldsymbol{d}_B \leq \ & \boldsymbol{x}_B + B^{-1}\boldsymbol{a}_{IN}\Delta \ \leq \boldsymbol{u}_B \\ d_{IN} \leq \ & X_{IN} - \Delta \ \leq \boldsymbol{u}_{IN}\end{aligned}$$

このときの $f(\boldsymbol{x})$ の減少量は次のようになる. なお更新後の $f(\boldsymbol{x})$ を $f(\boldsymbol{x}')$ とする.

$$\begin{aligned}f(\boldsymbol{x}') = \ & f(\boldsymbol{x}) + g_{IN}\Delta \quad (X_{IN}\text{が}\boldsymbol{x}_L\text{の要素のとき}, \ g_{IN} < 0) \\ f(\boldsymbol{x}') = \ & f(\boldsymbol{x}) - g_{IN}\Delta \quad (X_{IN}\text{が}\boldsymbol{x}_U\text{の要素のとき}, \ g_{IN} > 0)\end{aligned}$$

最後に, 変数の値の変化に合わせて, B, L, U の要素を入れ替える.

以上の操作を, 選択できる X_{IN} がなくなるまで繰り返すと, 最適解が求められる.

(3) 改訂シンプレックス法

改訂シンプレックス法は, 基底行列の取扱いや, 基底に入る変数の選び方等を改良し, 効率的に計算できるようにしたものである.

改訂シンプレックス法の大きな特徴は, 基底行列 B_{k-1} を B_k に更新するときに次のような形で定義されるエータ行列 E_k を用いることである.

$$E_k = \begin{bmatrix} 1 & & & B_{k-1}^{-1} & \vdots & \boldsymbol{a}_{IN} & & \\ & \ddots & & & \vdots & & 0 & \\ & & 1 & & \vdots & & & \\ & & & \ddots & \vdots & & & \\ & & & B_{k-1}^{-1} & & \boldsymbol{a}_{IN}(OUT) & & \\ & & & & \vdots & & \ddots & \\ & & & & \vdots & & & 1 \\ & 0 & & & \vdots & & & \ddots \\ & & & B_{k-1}^{-1} & \vdots & \boldsymbol{a}_{IN} & & 1 \end{bmatrix}$$

(OUT: 下限値または上限値に変化する基底変数の添字)

($\boldsymbol{a}_{IN}(OUT)$: $\boldsymbol{a}_{IN}$ の第 OUT 要素)

なお, 更新は $B_k = B_{k-1}E_k$ で行う.

次に内点法について説明する.

(1) 内点法で扱う問題の形式

内点法では下記のように$\boldsymbol{x}$の下限値が 0 になるように定式化された問題を扱う.

$$\begin{aligned} &f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} \to \min \\ &\text{線形制約条件}: \quad A\boldsymbol{x} = \boldsymbol{b} \\ &\boldsymbol{x}\text{の定義域}: \quad \boldsymbol{0} \le \boldsymbol{x} \le \boldsymbol{u} \end{aligned} \tag{5.1}$$

ただし, A は n 次元ベクトル $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ に対する m 個の制約条件を与える $m \times n$ 行列である. 改訂シンプレックス法で扱った一般の線形計画問題は $\boldsymbol{x}$ の空間での平行移動とスラック変数の導入により, (5.1) の形に定式化することができる. なお, 内点法に分類されるアルゴリズムには多くの種類があるが, 本ライブラリではアフィン変換を用いた方法を利用している.

(2) 内点法における最適解の探索

シンプレックス法が実行可能領域 S の頂点に当たる, 実行可能基底解の中から最適解を探索するのに対し, 内点法では実行可能領域の内部から出発して, 目的関数を減少させる方向に解を変化させながら最適解を探索する.

いま $\boldsymbol{x}^{(k)}$ が実行可能解であるとき, $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}$ が実行可能かつ $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k+1)} < \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{(k)}$ を満たすベクトル $\boldsymbol{d}^{(k)}$ のうちで, 目的関数の減少率ががもっとも大きくなる方向ベクトル $\boldsymbol{d}^{(k)}$ の方向を探索する. ここで t をステップサイズと呼ぶ. もし, 制約条件を考慮しなければ, 目的関数の値 $\boldsymbol{c}\cdot\boldsymbol{x}$ をもっとも減少させる方向は $-\boldsymbol{c}$ の方向であるである. 実際は, 制約条件 $A\boldsymbol{x}^{(k+1)} = A(\boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}) = \boldsymbol{b}$ を満たす必要があるから, $\boldsymbol{d}^{(k)}$ の方向としては $-\boldsymbol{c}$ を S の部分空間 $\{x|A\boldsymbol{x} = \boldsymbol{0}\}$ に射影したものを取れば良い.

(3) ビッグ M 法

内点法により最適解を探索するためには, 探索の出発点として実行可能領域の内点すなわち, 実行可能領域の内部にあって, 境界上にない初期解 $\boldsymbol{x}^{(0)}$ が必要である. しかしながら, このような初期解を求めるのは自明なことではない. そこで, 本ライブラリでは, ビッグ M 法と呼ばれる方法を用いて実行可能解の計算と最適解の計算を同時に行うようにしている. まず人為変数

$$\boldsymbol{x'} = (x_{n+1}, x_{n+2}, \ldots, x_{n+m})^{\mathrm{T}}$$

を導入して, 以下のような $n+m$ 変数の線形計画問題を定式化する.

$$\begin{array}{ll} \bar{f}(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{c}^T\boldsymbol{x} + M\boldsymbol{x'} \rightarrow \min & \\ \text{線形制約条件}: & A\boldsymbol{x} + \bar{A}\boldsymbol{x'} = \boldsymbol{b} \\ \boldsymbol{x}\text{の定義域}: & \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \boldsymbol{x'}\text{の定義域}: & \boldsymbol{0} \leq \boldsymbol{x'} \end{array} \tag{5.2}$$

任意の $\boldsymbol{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^{\mathrm{T}}$ に対して $m \times m$ 行列 $\bar{A} = (\bar{a}_{ij})$ $(1 \leq i, j \leq m)$ を

$$\bar{a}_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik} x_k^{(0)} & (i = j \text{ のとき}) \\ 0 & (i \neq j \text{ の場合}) \end{cases}$$

で定義しておけば,

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{bmatrix} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}, 1, 1, \ldots, 1)^{\mathrm{T}}$$

は, あきらかに (5.2) の実行可能解である. パラメータ M を十分大きな正数に取っておけば, 人為変数 $x' = (x'_1, x'_2, \ldots, x_m)^{\mathrm{T}}$ (5.2) の目的関数 $\bar{f}(\boldsymbol{x}, \boldsymbol{x'})$ を内点法の手続きによって減少させて行くとき, まず, 人為変数の項 $M\boldsymbol{x'}$ が速やかに 0 に近づく. このことは, 各人為変数が急速に 0 に近づくことを意味する. 各人為変数が十分に 0 に近ければ, (5.2) の実行可能解の人為変数を除いた部分は (5.1) の実行可能解と見なせる. さらに目的関数 $\bar{f}(\boldsymbol{x}, \boldsymbol{x'})$ を減少させて行くと, 今度は $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$ の項が減少して行くので, 最終的に (5.2) の最適解の人為変数を除いた部分は, (5.1) の実行可能な最適解になる. なお, (5.2) の最適解において人為変数が十分に 0 に近づいていないとき, すなわち与えられた小さな正数 ϵ_A に対して

$$\max_{1 \leq j \leq m} \{|x'_{n+j}|\} > \epsilon_A$$

であるとき問題 (5.1) は実行不可能であったと判定する.

以下の説明では (5.1) にはすでに人為変数を組み入れてあるものとする.

(4) アフィン変換による探索方向の決定

上記の最適解の探索方向の決定方法 $\boldsymbol{x}^{(k)}$ が S の中央付近にある場合はステップサイズ t を十分大きく取れるため, 目的関数の値を大きく減少させることができるが, $\boldsymbol{x}^{(k)}$ が S の境界付近にある場合, ステップサイズを大きく取ることができないため, 目的関数を十分に減少させられない. そこで変数 $\boldsymbol{x}$ に対する変換を行って現在の解 $\boldsymbol{x}^{(k)}$ が変換先の空間における実行可能領域 S' の境界から十分離れるようにしてから, 目的関数を大きく減少させるように解を更新し, そうして得られた変換先の空間での新しい解に対して逆変換を行って $\boldsymbol{x}^{(k+1)}$ とする. この目的のために, 現在の解 $\boldsymbol{x}^{(k)}$ に対して定義される対角行列

$$D^{(k)} = \begin{bmatrix} x_1^{(k)} & & & 0 \\ & x_2^{(k)} & & \\ & & \ddots & \\ 0 & & & x_n^{(k)} \end{bmatrix}$$

によりアフィン変換

$$y = (D^{(k)})^{-1}x$$

を行う. 変換先の空間において $\boldsymbol{x}^{(k)}$ がつねに $(1, 1, \ldots, 1)^{\mathrm{T}}$ に移されることは明らかである. $\boldsymbol{y}$ の空間においてもとの線形計画問題は次のように表される.

$$\begin{aligned} &f(\boldsymbol{y}) = \hat{\boldsymbol{c}}^{(k)\mathrm{T}}\boldsymbol{y} \to \min \\ &\text{線形制約条件}: \quad \hat{A}^{(k)}\boldsymbol{y} = \boldsymbol{b} \\ &\boldsymbol{y}\text{の定義域}: \quad \boldsymbol{0} \leq \boldsymbol{y} \leq \hat{\boldsymbol{u}}^{(k)} \end{aligned}$$

ここで

$$\hat{A}^{(k)} = AD^{(k)}$$

$$\hat{\boldsymbol{c}}^{(k)} = D^{(k)}\boldsymbol{c}$$

である. この問題における目的関数の減少率を最大にする方向 $\hat{\boldsymbol{d}}^{(k)}$ は $-\hat{\boldsymbol{c}}^{(k)}$ を部分空間 $\{\boldsymbol{y}|\hat{A}^{(k)}\boldsymbol{y} = \boldsymbol{0}\}$ に射影したものである. この射影は射影行列

$$\hat{P}^{(k)} = I - (\hat{A}^{(k)})^{\mathrm{T}}(\hat{A}^{(k)}(\hat{A}^{(k)})^{\mathrm{T}})^{-1}\hat{A}^{(k)}$$

で与えられるので, 探索方向は

$$\hat{\boldsymbol{d}}^{(k)} = -\hat{P}^{(k)}\hat{\boldsymbol{c}}^{(k)}$$

となるが, これはもとの変数での探索方向 $\boldsymbol{d}^{(k)}$ とは,

$$\boldsymbol{d}^{(k)} = D^{(k)}\hat{\boldsymbol{d}}^{(k)}$$

という関係を持っているので, 結局, もとの空間における探索方向は

$$\boldsymbol{d}^{(k)} = -D^{(k)}\hat{P}^{(k)}\hat{\boldsymbol{c}}^{(k)} = -(D^{(k)})^2(I - A^{\mathrm{T}}(A(D^{(k)})^2A^{\mathrm{T}})^{-1}A(D^{(k)})^2)\boldsymbol{c}$$

で与えられる. この式には逆行列の計算が含まれているが, 実際の計算では, まず連立 1 次方程式

$$(A(D^{(k)})^2A^{\mathrm{T}})\boldsymbol{w}^{(k)} = A(D^{(k)})^2\boldsymbol{c}$$

を解いて, $\boldsymbol{w}^{(k)}$ を求め

$$\boldsymbol{d}^{(k)} = -(D^{(k)})^2(\boldsymbol{c} - A^{\mathrm{T}}\boldsymbol{w}^{(k)})$$

として探索方向 $\boldsymbol{d}^{(k)}$ を定めることができる.

(5) ステップサイズの決定

上記のように定められた探索方向 $\boldsymbol{d}^{(k)}$ により与えられる直線 $\boldsymbol{x}^{(k)} + t\boldsymbol{d}^{(k)}$ $(t \geq 0)$ 上の点は制約条件

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

を満たし, かつ目的関数はステップサイズ t の増加に対して単調に減少する. しかし, 変数の定義域 $\mathbf{0} \leq \boldsymbol{x} \leq \boldsymbol{u}$ の外に出ることは許されないので, ステップサイズ t の取りうる最大値は

$$t_{\max} = \min\left\{ \min\left\{ \left. \frac{x_j^{(k)}}{-d_j^{(k)}} \right| d_j^{(k)} < 0 \right\}, \min\left\{ \left. \frac{u_j - x_j^{(k)}}{d_j^{(k)}} \right| d_j^{(k)} > 0 \right\} \right\}$$

となる. 実際には, 実行可能領域の境界に達してしまうと, これ以上探索を続けることができないので, $0 < \alpha < 1$ を満たすパラメータ α を用いて,

$$t^{(k)} = \alpha t_{\max}$$

が $\boldsymbol{x}^{(k)}$ に対するステップサイズとなり, 新しい解は,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + t^{(k)} \boldsymbol{d}^{(k)}$$

で与えられる.

(6) 変数の上限と下限の入れ替え

最適解の探索の途中で, いずれかの変数が下限に近づいた場合に前述のアフィン変換は有効であるが, 変数が上限に近づいた場合は同じ方法は用いることができない. そこで, 本関数では変数 x_j がその上限 u_j に近づくと以下のような変換を行って変数の上限と下限を入れ替える.

$$\begin{array}{rcll} x_j & \leftarrow & u_j - x_j & \\ u_j & \leftarrow & u_j & \\ c_j & \leftarrow & -c_j & \\ b_i & \leftarrow & b_i - a_{ij} u_j & (i = 1, 2, \ldots, m) \\ a_{ij} & \leftarrow & -a_{ij} & (i = 1, 2, \ldots, m) \end{array}$$

(7) 制約条件に対する残差のチェック

最適解の探索のための反復計算を繰り返すうちに, 計算誤差のため制約条件に対する残差 $||A\boldsymbol{x} - \boldsymbol{b}||$ が大きくなってしまうことがある. いったん残差が大きくなってしまうとそれ以降の計算は意味をなさなくなる. これを防ぐため, 反復回数の ℓ 回ごとに与えられた正数 ϵ_r に対して

$$||A\boldsymbol{x}^{(k)} - \boldsymbol{b}|| \leq \epsilon_r$$

が満たされているかどうかをチェックし, 満たされていなければ

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{bmatrix} = (x_1^{(k-\ell)}, x_2^{(k-\ell)}, \ldots, x_n^{(k-\ell)}, 1, 1, \ldots, 1)^{\mathrm{T}}$$

を初期解として問題 (5.2) の行列 $\bar{A}$ を再計算して最適解の探索を始めからやり直す.

(8) 収束の判定

$$\frac{\boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^{(k-1)} - \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^{(k)}}{1 + ||\boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^{(k)}||} \leq \epsilon_C$$

を満たした時, $\boldsymbol{x}^{(k)}$ は最適解に収束したものとみなす. ϵ_C は収束判定のためのパラメータである.

5.1.2.5 0-1 変数を含む線形制約付き多変数線形関数の最小化

ここでは線形計画問題に, 変数のうちのいくつかが 0-1 変数すなわち 0 または 1 のみを値として取るという条件を付加した問題 (混合 0-1 計画問題) を扱う.

$$
\begin{array}{lcll}
\text{目的関数} & : & f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} & \rightarrow min \\
\text{線形制約条件} & : & a_{ij}x_j = b_i & (i = 1, \cdots, m; j = 1, \cdots, n) \\
\boldsymbol{x}\text{の定義域} & : & d_j \leq x_j \leq u_j & (j = n_{01} + 1, \cdots, n) \\
\text{0-1 変数条件} & : & x_j = 0 \ or \ 1 & (j = 1, \cdots, n_{01}) \\
 & & & (1 \leq n_{01} \leq n)
\end{array}
$$

ただしここであらかじめ必要な個数のスラック変数を導入するなどして, 制約条件としては等式制約条件だけを含むようにしてあるものとして説明を進める.

本ライブラリでは, 混合 0-1 計画問題を解くために, 分枝限定法を用いる.

分枝限定法とは元の問題をいくつかの部分問題に分解し, それらを全て解くことにより元の問題の解を得る方法である. 混合 0-1 計画問題での部分問題とは, 0-1 変数のいくつかの値を 0 または 1 に固定したものであって, 以下のように表される.

$$
\begin{array}{lcll}
\text{目的関数} & : & f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} & \rightarrow min \\
\text{線形制約条件} & : & a_{ij}x_j = b_i & (i = 1, \cdots, m; j = 1, \cdots, n) \\
\boldsymbol{x}\text{の定義域} & : & d_j \leq x_j \leq u_j & (j \notin S^0 \cup S^1 \cup F) \\
\text{0-1 変数条件} & : & x_j = 0 & (j \in S^0) \\
 & & x_j = 1 & (j \in S^1) \\
 & & x_j = 0 \ or \ 1 & (j \in F)
\end{array}
$$

ここで S^0, S^1, F はそれぞれ部分問題 P_k において, 0 に固定された 0-1 変数の添字の集合, 1 に固定された 0-1 変数の添字の集合, ならびに値が固定されていない 0-1 変数の添字の集合である. なお, 0 または 1 に固定された 0-1 変数を固定変数, それ以外の 0-1 変数を自由変数と呼ぶ. 部分問題は, それ自体混合 0-1 計画問題である. とくに $S^0 \cup S^1 = \phi$ すなわち固定変数を持たない部分問題は元の混合 0-1 計画問題 (以後, P_1 と呼ぶ.) そのものである. また, 部分問題に対して次のように表される線形計画問題を元の部分問題に対する緩和問題と呼ぶ.

$$
\begin{array}{lcll}
\text{目的関数} & : & f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} & \rightarrow min \\
\text{線形制約条件} & : & a_{ij}x_j = b_i & (i = 1, \cdots, m; j = 1, \cdots, n) \\
\boldsymbol{x}\text{の定義域} & : & d_j \leq x_j \leq u_j & (j \notin S^0 \cup S^1 \cup F) \\
\text{0-1 変数条件} & : & x_j = 0 & (j \in S^0) \\
 & & x_j = 1 & (j \in S^1) \\
 & & 0 \leq x_j \leq 1 & (j \in F)
\end{array}
$$

以下に分枝限定法により P_1 の解を求める方法について説明する.

(1) 初期設定

(a) 部分問題のリスト
分枝限定法では部分問題のリスト PLIST を使用する. 初期状態では PLIST のメンバは P_1 だけで, PLIST のメンバ数 L は 1 である. L は分枝限定法の計算が進むにしたがって, 増減する.

(b) 暫定解
初期状態では P_1 の実行可能解は 1 つも見つかっていないので, 暫定解 $\boldsymbol{x}^*$ は未定で暫定解 $\boldsymbol{x}^*$ に対する目的関数値 z^* としては十分大きな正の値を設定しておく.

(c) 擬コスト

PLIST の更新に使用する擬コスト $h_{up}(j), h_{down}(j)$ $(j \in F)$ の初期値を以下のように設定する. まず P_1 に対する緩和問題を改訂シンプレックス法を用いて解く. このとき求められた最適解を $\bar{\boldsymbol{x}}$ とし, そのうち基底変数の添字の集合を B, 定義域の上限値を取る非基底変数の添字の集合を N_{up}, 定義域の下限値を取る非基底変数の添字の集合を N_{down} とすると各基底変数 $\bar{x}_{B_i}$ と目的関数値 $f(\bar{\boldsymbol{x}})$ は非基底変数を用いて以下のように表わされる.

$$\begin{aligned} f(\bar{\boldsymbol{x}}) &= \bar{z}_0 + \sum_{j \in N_{down}} \bar{c}_j \bar{x}_j - \sum_{j \in N_{up}} \bar{c}_j \bar{x}_j \\ \bar{x}_{B_i} &= \bar{b}_i - \sum_{j \in N_{down}} y_{ij} \bar{x}_j - \sum_{j \in N_{up}} y_{ij} \bar{x}_j \quad (B_i \in B) \end{aligned}$$

$\bar{z}_0, \bar{b}_i, \bar{c}_j$ および y_{ij} は P_1 の緩和問題の最適解を求める際に改訂シンプレックス法の計算の過程で求められる量であって,

$$\bar{c}_j \geq 0 \quad (j \in N_{up} \cup N_{down})$$

を満たす. これらの量を用いて $h_{up}(p), h_{down}(p)$ の初期値を以下のように設定する.

$p \in N_{up}$ の場合:

$$\begin{aligned} h_{up}(p) &= 0 \\ h_{down}(p) &= -\bar{c}_p \end{aligned}$$

$p \in N_{down}$ の場合:

$$\begin{aligned} h_{up}(p) &= \bar{c}_p \\ h_{down}(p) &= 0 \end{aligned}$$

$p = B_i \in B$ の場合:

$$\begin{aligned} h_{up}(p) &= \min\{-\bar{c}_j / y_{ij} | y_{ij} < 0, j \in N_{down} - F \text{ または } y_{ij} > 0, j \in N_{up} - F\} \\ h_{down}(p) &= \min\{\bar{c}_j / y_{ij} | y_{ij} > 0, j \in N_{down} - F \text{ または } y_{ij} < 0, j \in N_{up} - F\} \end{aligned}$$

ただし $p = B_i \in B$ の場合で $h_{up}(p)$ または $h_{down}(p)$ の一方について右辺の条件を満たす j が存在しない場合には, 値として十分大きな正の数を設定する. もしも $h_{up}(p)$ と $h_{down}(p)$ の両方で右辺の条件を満たす j が存在しない場合には

$$\begin{aligned} h_{up}(p) &= \frac{\sum_{j \in N_{up}} -\bar{c}_j / y_{ij}}{|M_{up}|} \\ h_{down}(p) &= \frac{\sum_{j \in N_{up}} \bar{c}_j / y_{ij}}{|M_{down}|} \end{aligned}$$

ここで

$$\begin{aligned} M_{up}(p) &= \{j | y_{ij} < 0, j \in N_{down} \cap F \text{ または } y_{ij} > 0, j \in N_{up} \cap F\} \\ M_{down}(p) &= \{j | y_{ij} > 0, j \in N_{down} \cap F \text{ または } y_{ij} < 0, j \in N_{up} \cap F\} \end{aligned}$$

(2) 緩和問題

PLIST の含まれる部分問題について対応する緩和問題が解かれていないものがあれば, それを改訂シンプレックス法を用いてを解く. 部分問題 P_i に対応する緩和問題の解に対する最適値 $g(P_i)$ について $g(P_i) > z^*$ であった場合には, P_i を PLIST から除き, $L \leftarrow L - 1$ とする. また最適解を $\bar{\boldsymbol{x}}$ とすると. すべての $j \in F$ について $\bar{\boldsymbol{x}}_j$ の値が 0 または 1 であった場合には, 緩和問題の解が元の部分問題の解になり, 部分問題の最適値は緩和問題の最適値 $g(P_i)$ に等しい. このとき $f(P_i) < z^*$ ならば $\boldsymbol{x}^*$ と z^* を $\bar{\boldsymbol{x}}$ と $f(P_i)$ で置き換える.

なお, 緩和問題を新たに解く必要があるのは, 上記の初期設定のときと後述する分枝操作によって L が増加したときである. 後者の場合, PLIST の最後の 2 つの部分問題は分枝変数 x_{j_0} を 0 に固定した部分問題 P_{i0} と 1 に固定した部分問題 P_{i1} の組になっている. このとき上で求めた緩和問題の最適値 $g(P_{i0}), g(P_{i1})$ および分枝操作前の部分問題 P_i の最適値 $g(P_i)$ と最適解の値 $\bar{x}_{j_0}$ を用いて, 擬コストを次のように更新する.

$$\begin{aligned} h_{\rm up}(j_0) &= (g(P_{i1}) - g(P_i))/(1 - \bar{x}_{j_0}) \\ h_{\rm down}(j_0) &= (g(P_{i0}) - g(P_i))/\bar{x}_{j_0} \end{aligned}$$

(3) 最適値の推定

もし後述する分枝操作の後であれば $K' = \min\{K+1, L\}$, そうでない場合には $K' = \min\{K, L\}$ とする. ここで, K は分枝限定法における探索の深さと呼ばれ, 前もって与えておく正の整数のパラメータである. PLIST の最後の K' 個の部分問題 P_i について, 緩和問題の最適値 $g(P_i)$, 最適解 $\bar{\boldsymbol{x}}$ および擬コスト $h_{up}(j), h_{down}(j)$ を用いて, 部分問題の最適値 $f(P_k)$ の推定値

$$h(P_i) = g(P_i) + \sum_{j \in F} \min\{h_{up}(j)(1 - \bar{x}_j), h_{down}(j)\bar{x}_j\}$$

を計算を計算し, $h(P_i)$ の降順に PLIST の最後の K' 個を並べ代える.

(4) 部分問題のテスト

PLIST の最後の部分問題 P_k についてその緩和問題の最適解の基底変数 $\bar{x}_{B_i}$ $(B_i \in B)$ を非基底変数 $\bar{x}_j$ $(j \in N_{up} \cup N_{down})$ で表現すると, 先に擬コストの初期値を計算する際に現われた $\bar{z}_0, y_{ij}, \bar{c}_j, \bar{b}_i$ といった量に相当するものが現われる. これらの量を用いて, $Z_{max}(i), Z_{min}(i)$ を次のように定義する.

$$\begin{aligned} Z_{max}(i) &= \bar{b}_i - \sum_{j \in N_{down} - S^0} \min\{0, y_{ij}\}(u_j - d_j) + \sum_{j \in N_{up} - S^1} \max\{0, y_{ij}\}(u_j - d_j) \\ Z_{min}(i) &= \bar{b}_i - \sum_{j \in N_{down} - S^0} \max\{0, y_{ij}\}(u_j - d_j) + \sum_{j \in N_{up} - S^1} \min\{0, y_{ij}\}(u_j - d_j) \end{aligned}$$

そして以下の 8 つの規則により自由変数 $x_j (j \in F)$ のうちのいくつかを 0 または 1 に固定することができる.

- $Z_{max}(i) < 1$ かつ $B_i \in F$ ならば $x_{B_i} = 0$
- $Z_{min}(i) > 0$ かつ $B_i \in F$ ならば $x_{B_i} = 1$
- $Z_{max}(i) - \max\{0, y_{ij}\} < d_{B_i}$ かつ $j \in N_{down} \cap F$ ならば, $x_j = 0$
- $Z_{max}(i) + \min\{0, y_{ij}\} < d_{B_i}$ かつ $j \in N_{up} \cap F$ ならば, $x_j = 1$
- $Z_{min}(i) - \min\{0, y_{ij}\} > u_{B_i}$ かつ $j \in N_{down} \cap F$ ならば, $x_j = 0$
- $Z_{min}(i) + \max\{0, y_{ij}\} > u_{B_i}$ かつ $j \in N_{up} \cap F$ ならば, $x_j = 1$
- $|\bar{c}_j| \geq z^* - g(P_k)$ かつ $j \in N_{down} \cap F$ ならば, $x_j = 0$
- $|\bar{c}_j| \geq z^* - g(P_k)$ かつ $j \in N_{up} \cap F$ ならば, $x_j = 1$

なお, 変数が固定されるごとに F, S^0, S^1 などは更新していく. 計算の結果, すべての自由変数が固定された場合には, 部分問題 P_k が解けたことになり, P_k を PLIST から除き, $L \leftarrow L - 1$ とする. このとき $f(P_k) < z^*$ ならば, P_k の解と $f(P_k)$ で $\boldsymbol{x}^*$ と z^* を置き換える. それから更新された PLIST により, 処理 (6) に移る. もし, 固定されない自由変数が残っている場合には, 次の分枝操作に移る.

(5) 分枝操作

PLIST から P_k を除く. そして上記の部分問題のテストで固定されなかった自由変数から分枝変数 x_{j_0} を 1 つ選び, j_0 を S^0 に追加した部分問題 P_{k0} と j_0 を S^1 に追加した部分問題 P_{k1} を PLIST の最後に追加して $L \leftarrow L+1$ とする. ここで分枝変数 x_{j_0} は以下のようにして決定する.

暫定解 $\boldsymbol{x}^*$ が求まっていない場合:

$|h_{up}(j)(1-\bar{x}_j) - h_{down}(j)\bar{x}_j|$ を最大にする $j \in F \cap B$ を j_0 とする. ここで $\bar{\boldsymbol{x}}$ は部分問題 P_k の緩和問題の最適解である.

暫定解 $\boldsymbol{x}^*$ が求まっている場合:

$\min\{h_{up}(j)(1-\bar{x}_j), h_{down}(j)\bar{x}_j\}$ を最大にする $j \in F \cap B$ を j_0 とする.

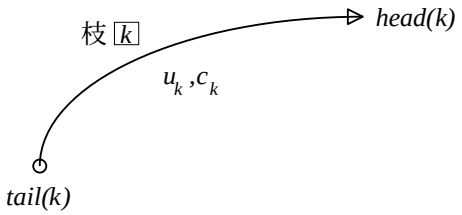
(6) 終了条件

もし, $L = 0$ になっていれば, 分枝限定法の処理は終了し, そのときの暫定解 x^* が P_1 すなわち元の混合 0-1 計画問題の解になる. そうでない場合には, ふたたび (2) 以降の処理を繰り返す.

5.1.2.6 ネットワーク上の流れに対する費用の最小化

n 個の節点と m 本の枝をもつネットワークを考える. ただしループ (始点と終点が同じ節点である枝) は存在しないと仮定する. 有向枝 k はその始点 $tail(k)$ と終点 $head(k)$ で表され, 非負の容量 u_k と単位流量あたりの費用係数 c_k をもつ. 最小費用流問題はこのようなネットワーク上で, 節点 i の流入出量 b_i および枝 k の容量 u_k の制約を満たし,

図 5–2　枝 k とその容量 u_k および費用係数 c_k



かつ全枝の費用の和を最小にする非負の流れ $x_k \ \ (k = 1, \cdots, m)$ を求める問題である.

$$\text{目的関数} \quad : \quad \sum_{k=1}^{m} c_k x_k \longrightarrow \text{最小}$$

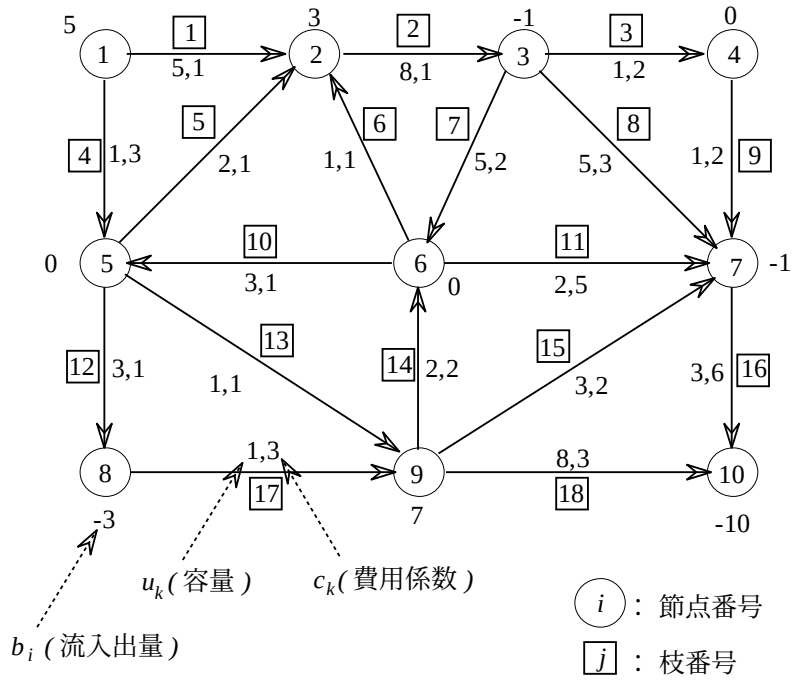
$$\text{制約条件} \quad : \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k = b_i, \quad (i = 1, \cdots, n)$$

$$0 \le x_k \le u_k, \quad (k = 1, \cdots, m)$$

$$\sum_{i=1}^{n} b_i = 0$$

これは線形計画問題でありシンプレックス法を適用できるが, 制約条件の特殊性を生かしてタブローをグラフ的に表現するデータ構造が可能でありピボット演算にともなう計算を効率よく実行できる.

図 5–3　ネットワークの例



(1) **初期実行可能解**

ネットワークに節点 $n+1$ を追加し, $b_i \geq 0$ なる節点 i からは枝 $k=(i,n+1)$ を, $b_i < 0$ なる節点 i へは枝 $k=(n+1,i)$ を作る. ただし, 追加枝の費用係数は十分大きな正の値 L, 容量は ∞(具体的には十分大きな正の値 U), $b_{n+1}=0$ と定める.

そのとき修正後の最小費用流問題は以下のようになる.

$$
\begin{aligned}
&\text{目的関数} \quad : \quad \sum_{k=1}^{m'} c_k x_k \longrightarrow \text{最小} \\
&\text{制約条件} \quad : \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k + sign(b_i)\cdot x_{m+i} = b_i, \quad (i=1,\cdots,n) \\
&\qquad\qquad\quad 0 \leq x_k \leq u_k, \quad (k=1,\cdots,m'(=m+n)) \\
&\qquad\qquad\quad \sum_{i=1}^{n} b_i = 0
\end{aligned}
$$

ただし,

$$
\begin{aligned}
c_k &= L \\
u_k &= U
\end{aligned}
\quad , k=m+1,\cdots,m'
$$

$$
sign(b_i) = \begin{cases} 1, & b_i \geq 0 \\ -1, & b_i < 0 \end{cases}
$$

これより, 初期実行可能解は

$$
x_k = \begin{cases} 0 & (k=1,\cdots,m) \\ sign(b_{k-m})\cdot b_{k-m} & (k=m+1,\cdots,m') \end{cases}
$$

とすればよい.

(2) **ピボット列の選択とピボット演算**

A_k は制約条件の係数行列の第 k 列, B は係数行列から適当な $n \times n$ 正則部分行列を選んで定まった基底行列とすると, 非基底変数 x_k に対して

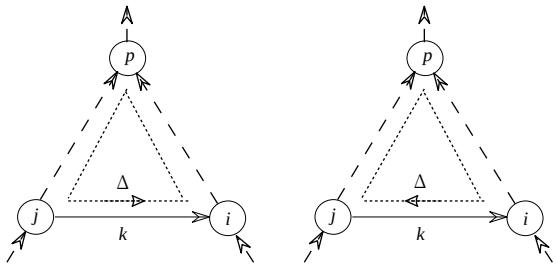
$$\bar{c_k} \quad = \quad c_k - c_B^T B^{-1} A_k$$

を求め,

$$\begin{array}{l} \bar{c_k} < 0, \quad x_k = 0 \text{ の場合} \\ \bar{c_k} > 0, \quad x_k = u_k \text{の場合} \end{array} \tag{5.3}$$

を満たすものを一つピボット列に選択する. ピボット列 k が定まったとき, x_k の変化の上限 Δ は, 枝 k の端点 i と j の共通の祖先のうち深さ最大のもの p を通る閉路 $j \longrightarrow i \longrightarrow p \longrightarrow j$ に沿って環流させるとき, 実現可能な流量の最大値に等しい.

図 5−4　流量 Δ の環流



(a) $x_k = 0$ の場合　　(b) $x_k = u_k$ の場合

この閉路中の各枝 ℓ について次のように定まる Δ_ℓ のうちの最小値を Δ とすればよい.

- 枝 k では $\Delta_k = u_k$
- i から p への路上の枝 ℓ の方向が下から上ならば $\Delta_\ell = u_\ell - x_\ell$, 上から下ならば $\Delta_\ell = x_\ell$
- j から p への路上の枝 ℓ の方向が下から上ならば $\Delta_\ell = x_\ell$, 上から下ならば $\Delta_\ell = u_\ell - x_\ell$

次にこの閉路に沿って流量 Δ を環流させる. すなわち閉路中の各枝 ℓ について

$$x_\ell \quad := \quad \begin{cases} x_\ell + \Delta, & \text{枝 } \ell \text{ と閉路の向きが一致} \\ x_\ell - \Delta, & \text{枝 } \ell \text{ と閉路の向きが逆} \end{cases}$$

とし閉路以外の枝の流量は変化しない. これがピボット演算である. また, 閉路中 $\Delta_\ell = \Delta$ を満たしていた枝 ℓ の新しい流量 $x_\ell = 0$ あるいは u_ℓ を新しい非基底変数 x_r とする (複数個存在すればその一つを任意に選ぶ). 以上の処理を繰り返し, 式 (5.3) を満たすものがなくなったとき処理を終了する.

5.1.2.7　プロジェクトの日程計画に対する費用の最小化

複数の作業からなるプロジェクトを考える. 各作業はその先行作業をいくつかもち, それらがすべて完了していなければ開始できない. プロジェクトネットワークは, 各作業を有向枝で表し, 節点を介することで先行関係を示す.

各作業 k は通常作業時間 $t_N(k)$ と特急作業時間 $t_C(k)$ をもち

$$t_N(k) \geq t_C(k)$$

の関係がある. また, 作業 k の遂行にかかる通常作業費用 $c_N(k)$ と特急作業費用 $c_C(k)$ には

$$c_N(k) \leq c_C(k)$$

の関係がある. $t_N(k) \geq t_C(k)$ のとき, 作業 k を中間の作業時間 $t(k)$ で遂行する場合の作業費用 $c(k)$ は,

$$c(k) = a(k) - b(k) \times t(k)$$

ただし,

$$a(k) = \frac{c_C(k) \times t_N(k) - c_N(k) \times t_C(k)}{t_N(k) - t_C(k)}, \quad b(k) = \frac{c_C(k) - c_N(k)}{t_N(k) - t_C(k)}$$

で与えられると考える.

ここで, 各作業 k を作業時間 $t(k)$ で行った場合, 各節点 i に対し, i に入る作業が全て完了し, i から出る枝の作業にいつでもかかれるという時刻 (最早節点時刻) $E(i)$ は,

$$E(1) = 0$$
$$E(i) = \max\{E(tail(k)) + t(k) \mid head(k) = i\}, \quad i = 2, 3, \cdots, n$$

で計算できる. ここで, n は節点数, $tail(k)$ は作業 k の始点の節点, $head(k)$ は作業 k の終点の節点を表す. またプロジェクトの完了時刻 T は

$$T = E(n)$$

で与えられる.

また, このプロジェクトを時刻 T に完了するために, 各節点 i に入る作業を遅くとも完了しなければならない時刻 (最遅節点時刻) $L(i)$ は,

$$L(n) = T$$
$$L(i) = \min\{L(head(k)) - t(k) \mid tail(k) = i\}, \quad i = n-1, n-2, \cdots, 1$$

で計算できる.

具体的な日程計画を立案するには, まず予定完了時刻 T_S を設定する. 通常

$$T_C \leq T_S \leq T_N$$

の範囲におく. 予定完了時刻 T_S までにプロジェクトを最小費用で完了する問題は, 作業 k の所要時間 $t(k)$ と節点 i の節点時刻 $\tau(i)$ を変数として, 次の線形計画問題に定式化できる.

$$\begin{aligned}
&\text{目的関数} \quad : \quad \sum_{k=1}^{n} (a(k) - b(k) \times t(k)) \rightarrow \text{最小} \\
&\text{制約条件} \quad : \quad t(k) + \tau(tail(k)) - \tau(head(k)) \leq 0, \quad k = 1, 2, \cdots, n \\
&\qquad\qquad\quad \tau(1) = 0 \\
&\qquad\qquad\quad \tau(n) \leq T_S \\
&\qquad\qquad\quad t(k) \leq t_N(k), \quad k = 1, 2, \cdots, n \\
&\qquad\qquad\quad -t(k) \leq -t_C(k), \quad k = 1, 2, \cdots, n
\end{aligned}$$

節点時刻 $\tau(i)$ が求まると各作業 k の所要時間 $t(k)$ を求めることができる.

$$t(k) = \min\{t_N(k), \tau(head(k)) - \tau(tail(k))\}$$

とすればよい. さらに, 各節点 i の最早節点時刻 $E(i)$ と最遅節点時刻 $L(i)$ を求めることができる.

このとき, 作業 k にとりかかれる最も早い時刻 (作業 k の最早開始時刻) $ES(k)$ およびプロジェクトを T_S までに完了するために遅くともとりかからなければならない時刻 (作業 k の最遅開始時刻) $LS(k)$ はそれぞれ

$$ES(k) = E(tail(k))$$
$$LS(k) = L(head(k)) - t(k)$$

で求められる. また, 作業 k の開始を遅らせても残りの作業の日程を調整すればプロジェクトを T_S 以内に完了できる範囲 (総余裕時間) $TF(k)$ および総余裕時間のうち作業 k の開始を遅らせてもその後の作業計画に影響を与えない範囲 (自由余裕時間) $FF(k)$ はそれぞれ

$$TF(k) = LS(k) - ES(k)$$
$$FF(k) = E(head(k)) - E(tail(k)) - t(k)$$

で求められる.

5.1.2.8 供給地から需要地への輸送費用の最小化

輸送問題は線形計画問題の特殊な問題である. これは, 物の移動に伴う問題である品物を, いくつかの供給地よりいくつかの需要地に最小の輸送費用で運搬するには, どんなルートがあるのか, またその時の費用はどのくらいかを見いだす問題である. この輸送問題を解くには, 第 1 次近似解で最初の計画を定め, それを改善して最終計画 (最適解) に行き着く. 本ライブラリでは, 第 1 次近似解を求める方法として北西隅の規則, ハウサッカー法, 改善の方法には, 改訂シンプレックス法 (使用しているアルゴリズム (5.1.2.4) 参照) を用いる. ここでは, 供給地 i $(i = 1, \cdots, m)$ での供給量を a_i, 需要地 j $(j = 1, \cdots, n)$ での需要量を b_j とし, 供給地 i から需要地 j への輸送量を x_{ij} とすると, 制約条件

$$\begin{aligned}
\sum_{j=1}^{n} x_{ij} &= a_i \quad (i = 1, \cdots, m) \\
\sum_{i=1}^{m} x_{ij} &= b_i \quad (j = 1, \cdots, n) \\
x_{ij} &\geq 0 \quad (\text{すべての } i, j \text{ に対して})
\end{aligned}$$

に従って目的関数である総輸送費用は, 次の関数を最小にする x_{ij} を求めることで得られる.

$$Z = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij}$$

(1) 第 1 次近似解法 (北西隅の規則, ハウサッカー法)
北西隅の規則は与えられた単位量の輸送費と供給量および需要量の行列の左上から, 需要量と供給量の資源を比較し, その量の最小をつぎつぎと配分していく方法である. ハウサッカー法は, 各供給地から各需要地への単位輸送費用の最小に資源を多量に配分して, つぎつぎと配分していく方法である.

(2) 不釣合いな輸送問題
ある輸送問題において, 総供給量 $\sum a_i$ が総需要量 $\sum b_i$ より少ない場合, 全体の需要量を満足できないにしても総出荷費用を最小にする方法で, 需要地へ供給地よりある量を配分することができる. この場合は, $\sum b_i - \sum a_i$ の総量を取り扱う架空の供給地を仮定する. この架空の供給地と需要地との間の 1 単位出荷する費用は零とする. もし, 原形の問題が ns × nd の問題ならば, (ns+1) × nd の問題を取りあげる輸送問題と同様にこの問題を解くことになる. もし問題が, 総供給量が総需要量より多い場合, これと同様に架空の問題を組みたてる. この場合は $\sum a_i - \sum b_i$ を満たす架空の需要地を仮定し, この架空の需要地の出荷費を零とする. もし, 原形の問題が ns × nd の問題ならば, ns × (nd+1) の問題を取りあげる輸送問題と同様にこの問題を解くことになる.

5.1.2.9 制約付き多変数凸型 2 次関数の最小化 (線形制約)

ここでは n 変数の目的関数

$$M(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T G \boldsymbol{x}$$

を制約条件

$$\boldsymbol{a}_i^T\boldsymbol{x} = b_i \quad i = 1, 2, \cdots, m_e$$
$$\boldsymbol{a}_i^T\boldsymbol{x} \geq b_i \quad i = m_e + 1, \cdots, m$$

の下で最小にする問題 (2 次計画問題) を扱う. ここで $\boldsymbol{a}_i$ と $\boldsymbol{c}$ は n 次元列ベクトル, b_i は 実数, G は $n \times n$ 正定値対称行列である. この問題を解くために 本ライブラリでは GI 法を用いる. これは D. Goldfarb と A. Idnani によって提案されたものである.

制約条件の添字の内のいくつかからなる集合を J とし, J の要素に対応するすべての制約条件の下で目的関数 $M(\boldsymbol{x})$ を最小にする $\boldsymbol{x}$ を $J-$ 最適解と呼び, 元の問題の解を単に最適解と呼ぶことにする. また, J の要素の個数を $\mid J \mid$, $\boldsymbol{a}_i \ \ (i \in J)$ を列に持つ $n \times \mid J \mid$ 行列を A_J と表記する. さらに K をすべての不等号制約条件の添字の集合, E をすべての等号制約条件の添字の集合とする. したがって, $J = E \cup K$ のとき $J-$ 最適解は最適解そのものである. 最適解を求める手順としてまず最初に $J = E$ とし, このときの $J-$ 最適解を初期解として, J に満たすべき不等号制約条件を 1 つずつ付け加えていき, 付け加えた制約条件を満たすように $J-$ 最適解を補正していく. 最終的に $J = E \cup K$ となるまで補正を繰り返す.

(1) 初期解の計算

$J = E$ のときの $J-$ 最適解を求める. 目的関数が凸型であるとき, 最適解を求める問題は Kuhn-Tucker 条件

$$\begin{aligned} G\boldsymbol{x} - A_J\boldsymbol{v}_J &= -\boldsymbol{c} \\ A_J^T\boldsymbol{x} &= \boldsymbol{b}_J \\ \boldsymbol{v}_i &\geq 0 \quad (i \in J) \end{aligned}$$

を満たす $\boldsymbol{x}$ と $\boldsymbol{v}_J$ を求める事と等価である. ここで $\boldsymbol{v}_J$ と $\boldsymbol{b}_J$ はそれぞれ $v_i \ \ (i \in J)$ と $b_i \ \ (i \in J)$ を成分とする $\mid J \mid$ 次元のベクトルである. いま

$$\begin{aligned} A_J^\dagger &= (A_J^TG^{-1}A_J)^{-1}A_J^TG^{-1} \\ H_J &= G^{-1}(I - A_JA_J^\dagger) \end{aligned}$$

と置くと, $\boldsymbol{x}$ と $\boldsymbol{v}_J$ は

$$\begin{aligned} \boldsymbol{x} &= (A_J^\dagger)^Tb_J - H_J\boldsymbol{c} \\ \boldsymbol{v}_J &= (A_J^TG^{-1}A_J)^{-1}\boldsymbol{b}_J + A_J^\dagger\boldsymbol{c} \end{aligned}$$

として, 計算される.

(2) J の更新

ある $J \subset K$ について $J-$ 最適解が求められており, しかもそれが最適解でないとき, 次のようにして J に付け加える制約条件を決定する. まず

$$c_i(x) = \boldsymbol{a}_i^T\boldsymbol{x} - \boldsymbol{b}_i$$

とすると, $\boldsymbol{c}_s(x) < 0$ を満たす $s \notin J$ が少なくとも 1 つ存在する. そこで

$$c_s(x) = \min\{c_i(x) | i \notin J\} < 0$$

により s を決定する. そして, $\boldsymbol{a}_s$ が A_J の列を成す $\boldsymbol{a}_i \ \ (i \in J)$ と独立ならば $J \cup \{s\}$ を新たな J とする. また, そうでないときは J から, 要素を 1 つ取り除いて $J-$ 最適解の計算をやり直す.

(3) $J-$ 最適解の更新

J に (b) で決定した不等号制約条件

$$\boldsymbol{a}_s^T\boldsymbol{x} - \boldsymbol{b}_s \geq 0$$

を付け加えて J を更新した後の $J-$ 最適解を次のようにして求める. $\boldsymbol{x}$ が J を更新する前の $J-$ 最適解であり, かつ $c_i = 0 \quad (i \in J)$ であるとする. このとき $\bar{\boldsymbol{x}} = \boldsymbol{x} + tH_J\boldsymbol{a}_s$ と置くと (b) で定義した $c_i \quad (i \in J)$ と c_s および $v_i \quad (i \in J)$ について

$$\begin{aligned} c_i(\bar{\boldsymbol{x}}) &= c_i(\boldsymbol{x}) = 0 \quad (i \in J) \\ c_s(\bar{\boldsymbol{x}}) &= c_s(\boldsymbol{x}) + t\boldsymbol{a}_s^T H_J \boldsymbol{a}_s \\ v_i(\bar{\boldsymbol{x}}) &= v_i(\boldsymbol{x}) - tr_i \end{aligned}$$

が成り立つ. ただし, $r_i \quad (i \in J)$ は $\mid J \mid$ 次元ベクトル $\boldsymbol{r} = A_J^{\dagger}\boldsymbol{a}_s$ の第 i 成分である. いま

$$\begin{aligned} t_1 &= \min\left\{ \frac{v_i(\boldsymbol{x})}{r_i} \middle| r_i > 0, \quad i \in J \cap K \right\} \\ t_2 &= -\frac{c_s(\boldsymbol{x})}{\boldsymbol{a}_s^T H_J \boldsymbol{a}_s} \end{aligned}$$

とすると, $0 \leq t \leq t_1$ の範囲では

$$v_i(\bar{\boldsymbol{x}}) \geq 0 \quad (i \in \bar{J} \cup K)$$

であり, また $t = t_2$ のとき

$$c_s(\bar{\boldsymbol{x}}) = 0$$

である. そこで, もし $t_1 \geq t_2$ ならば $t = t_2$ のときの $\bar{\boldsymbol{x}}$ は $J \cup \{s\}$ に対応する制約条件について (a) で使用した Kuhn-Tucker 条件を満たしており, これが J 更新後の $J-$ 最適解となる. 一方, $t_1 < t_2$ ならば J から要素を 1 つ取り除いて $J-$ 最適解の計算をやり直す.

(4) 終了条件

$J-$ 最適解 $\boldsymbol{x}$ がすべての $i \in \bar{J} \cap K$ について

$$c_i(\boldsymbol{x}) \geq 0$$

を満たしているとき, 最適解は $\boldsymbol{x}$ であるものとして, 計算を終了する.

5.1.2.10 多変数広義凸型 2 次関数の最小化 (線形制約)

ここでは, 制約条件

$$\begin{aligned} \sum_{j=1}^{n} a_{ij}x_j &= b_i \quad (i = 1, \cdots, m_e) \\ \sum_{j=1}^{n} a_{ij}x_j &\geq b_i \quad (i = m_e + 1, \cdots, m) \\ x_i &\geq 0 \quad (i = 1, \cdots, n) \end{aligned}$$

の下で目的関数である n 変数の広義凸型 2 次関数

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad (G : \text{半正定値対称行列})$$

を最小にする $\boldsymbol{x}^*$ とそのときの関数値 $f(\boldsymbol{x}^*)$ を求める問題を扱う. この問題を解くために 本ライブラリでは与えられた問題をそれと等価な線形相補性問題に変換し, それをレムケ法を用いて解く.
以下に具体的な手順を示す.

(1) 線形相補性問題の作成
まず, 等式制約条件を用いて m_e 個の変数を消去し, 元の 2 次計画問題を残りの変数についての 2 次計画問題に変換する. そして新しい 2 次計画問題に対して, 等価な線形相補性問題を作成する. $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$ から等式制約条件を用いて, $x_1, x_2, \cdots, x_{m_e}$ を $x_{m_e+1}, \cdots, x_n$ で

$$x_i = \sum_{j=m_e+1}^{n} p_{ij}x_j + r_i \quad (i = 1, \cdots, m_e)$$

のように表わす. この式を用いて, 与えられた 2 次計画問題を $n - m_e$ 個の変数 $\boldsymbol{x}' = [x_{m_e+1}, \cdots, x_n] = [x'_1, \cdots, x'_{n-m_e}]$ についての 2 次計画問題に変換する.

$$\begin{aligned}
\text{制約条件} : \sum_{j=1}^{n-m_e} a'_{ij}x'_j &\geq b'_i \quad (i = 1, \cdots, n - m_e) \\
x'_i &\geq 0 \quad (i = 1, \cdots, n - m_e) \\
\text{目的関数} : f(\boldsymbol{x}') &= \frac{1}{2}(\boldsymbol{x}')^T G'\boldsymbol{x}' + \boldsymbol{c}'^T\boldsymbol{x}'
\end{aligned}$$

ここで

$$\begin{aligned}
(G')_{i-m_e,j-m_e} &= (G)_{i,j} + \sum_{k=1}^{m_e} G_{i,k}p_{k,j} + \sum_{k=1}^{m_e} G_{k,j}p_{k,i} + \sum_{s=1}^{m_e}\sum_{t=1}^{m_e} p_{s,i}G_{s,t}p_{t,j} \\
c'_{i-m_e} &= c_i + \sum_{j=1}^{m_e} c_j p_{j,i} + \sum_{k=1}^{m_e} G_{i,k}r_k + \sum_{s=1}^{m_e}\sum_{t=1}^{m_e} G_{s,t}p_{s,i}r_t \\
a'_{i,j-\mathrm{ME}} &= p_{i,j} \quad (i = 1, \cdots, m_e; j = m_e + 1, \cdots, n) \\
a'_{i,j-m_e} &= a_{i,j} + \sum_{k=1}^{m_e} a_{i,k}p_{k,j} \quad (i = m_e + 1, \cdots, m; j = m_e + 1, \cdots, n) \\
b'_i &= -r_i \quad (i = 1, \cdots, m_e) \\
b'_i &= b_i - \sum_{k=1}^{m_e} a_{ik}r_k \quad (i = m_e + 1, \cdots, m)
\end{aligned}$$

とし, $A' = (a'_{i,j})$, $G' = (G'_{i,j})$, $(\boldsymbol{c}')^T = [c'_1, \cdots, c'_{n-m_e}]$, $(\boldsymbol{b}')^T = [b'_1, \cdots, b'_{n-m_e}]$ と置くことで以下の線形相補性問題を作成する. ここで $\boldsymbol{v}$ は不等式制約条件に対するラグランジュ乗数ベクトルである.

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{w} \end{bmatrix} &= \begin{bmatrix} G' & -A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix} + \begin{bmatrix} \boldsymbol{c}' \\ -\boldsymbol{b}' \end{bmatrix} \\
y_i &\geq 0 \ \text{ for } \ (i = 1, \cdots, n - m_e) \\
w_i &\geq 0 \ \text{ for } \ (i = 1, \cdots, m) \\
x_i &\geq 0 \ \text{ for } \ (i = 1, \cdots, n - m_e) \\
v_i &\geq 0 \ \text{ for } \ (i = 1, \cdots, m) \\
[\boldsymbol{y}^T \ \boldsymbol{w}^T]\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix} &= 0
\end{aligned}$$

(2) 線形相補性問題

線形相補性問題を解いて, 等式制約条件を消去して得られた新しい 2 次計画問題の解を求める.

与えられた $n \times n$ 行列

$$T = \begin{bmatrix} G' & -A^T \\ A & 0 \end{bmatrix}$$

に対して,

$$\boldsymbol{y} = T\boldsymbol{x} + \boldsymbol{q} \tag{5.4}$$

$$x_i \geq 0 \quad (i = 1, \cdots, n) \tag{5.5}$$

$$y_i \geq 0 \quad (i = 1, \cdots, n) \tag{5.6}$$

を満たす, n 次元ベクトル $\boldsymbol{x}$, $\boldsymbol{y}$ を求める. まず線形相補性問題 (5.4), (5.5), (5.6) に対し, $2n + 1$ 個の変数 $(\boldsymbol{y}, \boldsymbol{x}, \xi)$ についての方程式

$$\boldsymbol{y} - T\boldsymbol{x} - \boldsymbol{d}\xi = \boldsymbol{q}$$

を定義する. ここで, $\boldsymbol{d}$ は全成分が正である n 次元定数ベクトルである. $(\boldsymbol{y}, \boldsymbol{x}, \xi)$ のうち, 0 でない成分を基底変数, それ以外を非基底変数と呼ぶ.

次に (5.4) の方程式に対して初期解を $(\boldsymbol{y}, \boldsymbol{x}, \xi) = (\boldsymbol{q} + \boldsymbol{d}\xi_0, \boldsymbol{0}, \xi_0)$ とする. ここで

$$\xi_0 = \max\{-q_i/d_i | i = 1, 2, \cdots, n\}$$
$$\eta = \max\{-q_i/d_i | i = 1, 2, \cdots, n\}$$

とする. 式 (5.5), (5.6) を満たしたまま η の値を無限に大きくできるならば計算を終了する. ある基底変数の値が 0 になった時, この時の基底変数が y_s ならば, y_s を基底から出して, η を基底に入れ, $\eta = x_s$ とする. 基底変数が x_s ならば, x_s を基底から出して, η を基底に入れ, $\eta = y_s$ とする. この処理を繰り返し, その基底変数が ξ になったところで計算を終了する. また, 式 (5.5), (5.6) を満たしたまま η の値を無限に大きくできる場合には解が存在しないものとして計算を終了する.

(3) 解の変換

求めた解を等式制約条件を消去する前の元の 2 次計画問題の解に変換する.

5.1.2.11 制約無し 0-1 多変数 2 次関数の最小化

すべての変数が 0-1 変数すなわち 0 または 1 のみを値として取るという条件の下で 2 次関数を最小化する問題 (0-1 無制約 2 次計画問題) を扱う.

$$\begin{array}{lll} \text{目的関数} & : f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} & \rightarrow min \\ \text{0-1 変数条件} & : x_j = 0 \ or \ 1 \quad (j = 1, \cdots, n) & \end{array}$$

本ライブラリでは, 0-1 無制約 2 次計画問題を解くために, 分枝限定法を用いる.

0-1 無制約 2 次計画問題では以下のような部分問題を用いる.

$$\begin{array}{lll} \text{目的関数} & : f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} & \rightarrow min \\ \text{0-1 変数条件} & : x_j = 0 \quad (j \in S^0) & \\ & \quad x_j = 1 \quad (j \in S^1) & \\ & \quad x_j = 0 \ or \ 1 \quad (j \in F) & \end{array}$$

ここで S^0, S^1, F はそれぞれ部分問題 P_k において, 0 に固定された 0-1 変数の添字の集合, 1 に固定された 0-1 変数の添字の集合, ならびに自由変数の添字の集合である. 部分問題は, それ自体 0-1 無制約 2 次計画問題である. とくに $S^0 \cup S^1 = \phi$ すなわち固定変数を持たない部分問題は元の 0-1 無制約 2 次計画問題 (以後, P_1 と呼ぶ.) そのものである.

また, 部分問題に対する緩和問題を以下のように定義する.

$$
\begin{array}{lcll}
\text{目的関数} & : & f(\boldsymbol{x}) = & \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \rightarrow min \\
\text{0-1 変数条件} & : & x_j = 0 & (j \in S^0) \\
 & & x_j = 1 & (j \in S^1) \\
 & & 0 \leq x_j \leq 1 & (j \in F)
\end{array}
$$

以下に分枝限定法により P_1 の解を求める方法について説明する.

(1) 初期設定

(a) 係数行列の前処理

後述する緩和問題を解くためには, 目的関数の係数行列が正定値行列である必要がある. 0-1 無制約 2 次計画問題では 0-1 変数が

$$x_i^2 = x_i \qquad (i = 1, \ldots, n)$$

という性質を持つことを利用して, 目的関数を対称行列を係数に持つ以下のような 2 次関数に書き換えることが出来る.

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G'\boldsymbol{x}$$

ただし,

$$G' = \frac{1}{2}(G + G^T) + 2\mathrm{diag}(c_1, \ldots, c_n)$$

ここで, G' が半正定値であれば, P_1 は自明な最適解

$$x^* = (0, 0, \ldots, 0)^T$$

を持つ. G' が半正定値でない, すなわち負の固有値を持つ場合には λ_{min} を G' の最小の固有値, $\beta(> 0)$ を正数のパラメータとして

$$G'' = G' + (|\lambda_{min}| + \beta)\mathrm{diag}(1, 1, \ldots, 1)$$

$$\boldsymbol{c}'' = -\frac{1}{2}(|\lambda_{min}| + \beta)(1, 1, \ldots, 1)^T$$

と定義することにより, 目的関数を正値対称行列 G'' を係数に持つ 2 次関数

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G''\boldsymbol{x} + (\boldsymbol{c}'')^T\boldsymbol{x}$$

に書き換えることが出来る. ここで β は行列 G'' の最小の固有値となっている. なお, 以降の説明では係数行列 G はあらかじめ, このような前処理を施して, 正値対称行列に変換されているものとする.

(b) 部分問題のリスト

分枝限定法では部分問題のリスト PLIST を使用する. 初期状態では PLIST のメンバは P_1 だけで, PLIST のメンバ数 L は 1 である. L は分枝限定法の計算が進むにしたがって, 増減する.

(c) 暫定解

混合 0-1 計画問題の場合と異なり, 実行可能解の存在は自明であるが, なるべく目的関数の値の小さな解を暫定解として用いる方が分枝限定法による最適解の探索の効率が良い. 本ライブラリでは模擬焼なまし法およびタブー探索という 2 つの発見的探索の方法により得た解を初期状態の暫定解として用いる.

(d) 模擬焼なまし法

① 初期解 $\boldsymbol{x} = \boldsymbol{x}_0$ をランダムに選ぶ.

② 初期温度を $T = T_0$ に設定する.

③ $z \leftarrow f(\boldsymbol{x}_0)$ とする.

④ $y \leftarrow f(\boldsymbol{x})$ とする.

⑤ 温度パラメータを $T \leftarrow \alpha \times T$ とする. $(0 < \alpha \leq 1)$

⑥ 添字 $i(1 \leq i \leq n)$ をランダムに一つ選ぶ.

⑦ $x_i \leftarrow 1 - x_i$ とする.

⑧ $y < f(\boldsymbol{x})$ ならば, 実数 $r \in [0, 1]$ をランダムに選び $r > \exp(-1/T)$ の場合には $x_i \leftarrow 1 - x_i$ とする.

⑨ $z \leftarrow \min\{y, z\}$ とする.

⑩ 所定の回数だけ, 処理④から⑨を繰り返し, 最終的に得られた z を分枝限定法における初期暫定解の目的関数値とする.

(e) タブー探索

① 初期解 $\boldsymbol{x} = \boldsymbol{x}_0$ をランダムに選ぶ.

② タブーリストを $K = \phi$ に設定する.

③ $z \leftarrow f(\boldsymbol{x}_0)$ とする.

④ $U(\boldsymbol{x}) = \{\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(n)}\}$ $\boldsymbol{u}^{(i)} = (x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n)^T$ とする.

⑤ $\boldsymbol{x} \leftarrow \min_{U \setminus K} \boldsymbol{u}_i$ とする.

⑥ $K \leftarrow K \cup \{\boldsymbol{x}\}$ とする. K の要素数が所定の数を越えた場合は最も以前に K に加えられた要素を K から取り除く.

⑦ $z \leftarrow \min\{f(\boldsymbol{x}), z\}$ とする.

⑧ 所定の回数だけ, 処理④から⑦を繰り返し, 最終的に得られた z を分枝限定法における初期暫定解の目的関数値とする.

タブー探索は計算時間がかかるが良好な解が得られ, 模擬焼なまし法は計算時間が短い. これらの探索方法に加えて, 本ライブラリでは模擬焼なまし法により第 1 次の解を計算したあとで, それをタブー探索により改良する方法もサポートしている. なお, 変数の数 n が大きい場合, 後述する分枝限定法で厳密な最適解を実用的な計算時間で求めるのは困難である. しかし, 多くの場合, 発見的探索で得られた解は十分な近似精度を持っている. そこで, 発見的探索で得られた解を近似解としてそのまま利用するため分枝限定法の計算を省略する方法も本ライブラリではサポートしている.

(2) 下界値の計算

各部分問題について対応する緩和問題を用いて最適値の下界値を以下のように計算する.

(a) 係数行列 G の固有値を $\lambda_1, \ldots, \lambda_n$ 対応する正規直交化された固有ベクトルを $\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(n)}$ とする.

(b) 緩和問題である凸型 2 次計画問題を G-I 法を用いて解き, その最適解を $\bar{\boldsymbol{x}}$ とする.

(c) 自由変数 $x_i(i \in F)$ が

$$f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{\bar{x}_i^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right| \leq z^* < f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{(1 - \bar{x}_i)^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right|$$

を満たせば, x_i は 0 に固定される. このとき, $S_0 \leftarrow S_0 \cup \{i\}, F \leftarrow F - \{i\}$ とする.

また, 自由変数 $x_i(i \in F)$ が

$$f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{(1 - \bar{x}_i)^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right| \leq z^* < f(\bar{\boldsymbol{x}}) + \frac{1}{2}\left|\frac{\bar{x}_i^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}\right|$$

を満たせば, x_i は 1 に固定される. このとき, $S_1 \leftarrow S_1 \cup \{i\}, F \leftarrow F - \{i\}$ とする.

このような操作により, 自由変数のいくつかが固定された場合には, 緩和問題を解き直し, それを新たに固定される自由変数が無くなるまで繰り返す.

(d) 座標変換

$$x_i - \bar{x}_i \leftarrow \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j$$

で定義される変数 $\hat{\boldsymbol{x}}$ の空間における点の集合 U_i^0 および U_i^1 を以下のように定義する.

$$U_i^0 = \{\hat{\boldsymbol{x}} | \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j = 0\}$$

$$U_i^1 = \{\hat{\boldsymbol{x}} | \sum_{j=1}^{n} \frac{u_i^{(j)}}{\sqrt{\lambda_j}} \hat{x}_j = 1\}$$

任意の自由変数の添字の集合 $\{i_1, i_2, \ldots, i_r\}$ について

$$D = \min\{|\hat{\boldsymbol{x}}| | \boldsymbol{x} \in \cap_{k=1}^{r} (U_{i_k}^0 \cup U_{i_k}^1)\}$$

とすると

$$g(\bar{\boldsymbol{x}}) = f(\bar{\boldsymbol{x}}) + \frac{D^2}{2}$$

は部分問題の下界値を与える. 本ライブラリでは, 自由変数のうちで

$$\frac{\min\{\bar{x}_i, 1 - \bar{x}_i\}^2}{\sum_{k=1}^{n} \frac{(u_i^{(k)})^2}{\lambda_k}}$$

の大きい方から r 個を D の計算に用いている.

部分問題 P_i に対応する最適値の下界値 $g(P_i)$ について $g(P_i) > z^*$ であった場合には, P_i を PLIST から除き, $L \leftarrow L - 1$ とする. また最適解を $\bar{\boldsymbol{x}}$ とすると. すべての $j \in F$ について $\bar{\boldsymbol{x}}_j$ の値が 0 または 1 であった場合には, 緩和問題の解が元の部分問題の解になり, 部分問題の最適値は緩和問題の最適値 $g(P_i)$ に等しい. このとき $f(P_i) < z^*$ ならば $\boldsymbol{x}^*$ と z^* を $\bar{\boldsymbol{x}}$ と $f(P_i)$ で置き換える.

なお, 緩和問題を新たに解く必要があるのは, 上記の初期設定のときと後述する分枝操作によって L が増加したときである. 後者の場合, PLIST の最後の 2 つの部分問題は分枝変数 x_{j_0} を 0 に固定した部分問題 P_{i0} と 1 に固定した部分問題 P_{i1} の組になっている.

(3) 分枝操作

PLIST から最後尾の部分問題 P_k を除く. そして自由変数から分枝変数 x_{j_0} を 1 つ選び, j_0 を S^0 に追加した部分問題 P_{k0} と j_0 を S^1 に追加した部分問題 P_{k1} を PLIST の最後に追加して $L \leftarrow L + 1$ とする. ここで分枝変数 x_{j_0} は以下のようにして決定する.

$$j_0 = \mathrm{argmin}_{j \in F} \left| \bar{x}_j - \frac{1}{2} \right|$$

分枝操作の後, 部分問題 P_{k0} および P_{k1} のそれぞれについて, 最適値の下界値を計算する.

(4) PLIST のソート

$K' = \min\{K, L\}$ とする. ここで, K は分枝限定法における探索の深さと呼ばれ, 前もって与えておく正の整数のパラメータである. PLIST の最後の K' 個の部分問題 P_i について, 下界値 $g(P_i)$ の降順に PLIST の最後の K' 個を並べ代える.

(5) 部分問題のテスト
PLIST の最後の部分問題 P_k についてそのすべての自由変数が固定された場合には, 部分問題 P_k が解けたことになり, P_k を PLIST から除き, $L \leftarrow L-1$ とする. このとき $f(P_k) < z^*$ ならば, P_k の解と $f(P_k)$ で $\boldsymbol{x}^*$ と z^* を置き換える. それから更新された PLIST により, 処理 (6) に移る. もし, 固定されない自由変数が残っている場合には, 次の分枝操作に移る.

(6) 終了条件
もし, $L=0$ になっていれば, 分枝限定法の処理は終了し, そのときの暫定解 $\boldsymbol{x}^*$ が P_1 すなわち元の 0-1 無制約 2 次計画問題の解になる. そうでない場合には, ふたたび (2) 以降の処理を繰り返す.

5.1.2.12 制約付き多変数関数の最小化

ここでは n 変数の関数 $f(\boldsymbol{x})$ が与えられた時, これを与えられた $m+\ell$ 個の制約条件

$$\begin{cases} g_i(\boldsymbol{x}) \leq 0 & (i=1,\cdots,m) \\ h_i(\boldsymbol{x}) = 0 & (i=1,\cdots,\ell) \end{cases}$$

のもとで最小化する点 $\boldsymbol{x}^*$(最適解) を求める問題を扱う. 大域的な最小化は一般に困難であり, 本ライブラリでは局所的な最小化のみを扱う. したがって, 以下の説明において, 最小化という用語は局所的な意味でのみ用いる. 同様に局所的な最適解 $\boldsymbol{x}^*$ を単に最適解と称する. いま, ペナルティ関数 $\theta_\delta(\boldsymbol{x})$ を

$$\theta_\delta(\boldsymbol{x}) = f(\boldsymbol{x}) + \delta \max(0, g_1(\boldsymbol{x}), \cdots, g_m(\boldsymbol{x}), \|h_1(\boldsymbol{x})\|, \cdots, \|h_\ell(\boldsymbol{x})\|) \quad (0<\delta)$$

で定義すると,

$$\theta_\delta(\boldsymbol{x}) \geq f(\boldsymbol{x})$$

が任意の $\boldsymbol{x}$ について成り立ち, 特に $\boldsymbol{x}$ が上記の制約条件を満たす点である時には

$$\theta_\delta(\boldsymbol{x}) = f(\boldsymbol{x})$$

が成り立つ. したがって制約条件 (1), (2) の下で $f(\boldsymbol{x})$ を最小化する問題は, 制約なしで $\theta_\delta(\boldsymbol{x})$ を最小化する問題に帰着される. $\theta_\delta(\boldsymbol{x})$ は厳密なペナルティ関数であり, これに準ニュートン法を適用すれば, 原理的には最適解を求めることができる. しかし, $\theta_\delta(\boldsymbol{x})$ は数値的に不安定な性質を持っているため, 本ライブラリでは逐次 2 次計画法と呼ばれるアルゴリズムを採用し, Lagrange 関数

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m-m_e} \lambda_i g_i(\boldsymbol{x}) + \sum_{i=1}^{m_e} \mu_i h_i(\boldsymbol{x})$$

に対して準ニュートン法に類似した解の反復改良を行う. $\theta_\delta(\boldsymbol{x})$ は, 直線探索の評価関数として使用する. 計算の手順は以下の通りである.

(1) 部分 2 次計画問題
最適解 $\boldsymbol{x}^*$ の第 k 近似 $\boldsymbol{x}_k$ が求められている時, 目的関数と制約条件を $\boldsymbol{x}_k$ のまわりで展開して得られる 2 次計画問題

目的関数 : $\frac{1}{2}\boldsymbol{d}^T B_k \boldsymbol{d} + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}$

不等式制約条件 : $g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{d} \leq 0 \quad (i=1,\cdots,m)$

等式制約条件 : $h_i(\boldsymbol{x}_k) + \nabla h_i(\boldsymbol{x}_k)^T \boldsymbol{d} \leq 0 \quad (i=1,\cdots,\ell)$

を解き, その解を $\boldsymbol{d}_k$ とする. ただし, ヘッセ行列 $\nabla^2 f(\boldsymbol{x}_k)$ そのものを計算するのは数値的に困難であるため, 準ニュートン法の場合と同じく, その近似行列である B_k で置き換えている. この 2 次計画問題を解くためのアルゴリズムとしては, Goldfarb-Idnani 法を用いる. もし $\boldsymbol{d}_k = 0$ であれば, $\boldsymbol{x}^* = \boldsymbol{x}_k$ として計算を終了する. なおここで求められた Lagrange 乗数ベクトルを $\boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}$ とする.

(2) 直線探索
$\boldsymbol{d}_k$ を探索方向として $\boldsymbol{x}_{k+1}$ を探索する. $\alpha_k = 1$ から出発して, 条件

$$\theta_\delta(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \leq \theta_\delta(\boldsymbol{x}_k) - \omega \alpha_k \boldsymbol{d}_k^T B_k \boldsymbol{d}_k$$

を満たすならば (c) の処理に移り, もし満たさなければ $\alpha_k \to \tau \alpha_k$ なる置き換えを, 条件が満たされるまで繰り返す.

(3) $\boldsymbol{x}_k$ の更新

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

(4) B_k の更新
オリジナルの BFGS 公式では B_k の正定値性が維持されないので, 以下に示す修正 BFGS 公式により B_k を更新する.

$$B_{k+1} = B_k - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k} + \frac{\boldsymbol{\eta}_k \boldsymbol{\eta}_k^T}{\boldsymbol{s}k^T \eta_k}$$

ここで

$$\begin{aligned}
\boldsymbol{s}_k &= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \\
\boldsymbol{y}_k &= \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{mu}_{k+1}) - \nabla_{\boldsymbol{x}} L(\boldsymbol{x}_k, \boldsymbol{\lambda}_{k+1}, \boldsymbol{mu}_{k+1}) \\
\phi &= \begin{cases} 1 & : \boldsymbol{s}_k^T \boldsymbol{y}_k \geq 0.2 \boldsymbol{s}_k^T \\ \frac{0.8 \boldsymbol{s}_k^T B_k \boldsymbol{s}_k}{\boldsymbol{s}_k^T (B_k \boldsymbol{s}_k - \boldsymbol{y}_k)} & : \text{その他の場合} \end{cases} \\
\boldsymbol{\eta}_k &= \phi \boldsymbol{y}_k + (1 - \phi) B_k \boldsymbol{s}_k
\end{aligned}$$

(5) 更新の終了
$\boldsymbol{x}_k$ の更新は Karush-Kuhn-Tucker 条件

$$\begin{aligned}
\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla g(\boldsymbol{x}) + \sum_{j=1}^{\ell} \mu_j \nabla h_j(\boldsymbol{x}) &= \boldsymbol{0} \\
g_i(\boldsymbol{x}) &= 0 \quad (i = 1, \cdots, m) \\
h_j(\boldsymbol{x}) &= 0 \quad (j = 1, \cdots, \ell) \\
\sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) &= 0 \\
\lambda_i &\geq 0 \quad (i = 1, \cdots, m)
\end{aligned}$$

が満たされるまで繰り返される.

5.1.2.13 ネットワーク上の 2 節点間の距離の最小化

(1) ある節点から他のすべての節点への最短路の計算
Dijkstra の方法を用いてグラフ $G = (V, E)$ におけるある指定された節点 $init$ から他のすべての節点までの最短路とその距離を求める. ただし, すべての枝の重みは非負であると仮定する.

(i) 各節点 $i \in V$ に対して $Distance(i) = \infty$, $Path(i) = init$ および $P = \phi$ とする. 出発点 $init$ に対して $Distance(init) = 0$ とする. また, $next = init$ とする.

(ii) $P = P \cup \{next\}$ とする. 節点 $next$ に接続している各節点 j に対して, もし $Distance(j) > Distance(next) + Weight(next, j)$ ならば, $Distance(j) = Distance(next) + Weight(next, j)$ および $Path(j) = next$ と更新する.

(iii) 各節点 $i \in P$ に対して $Distance(i)$ が最小である節点 v を選択する. それを $next = v$ とする.

(iv) $P = V$ となるまで, (ii), (iii) を繰り返す.

(2) 全 2 節点間の最短路の計算

Floyd の方法を用いてグラフ $G = (V, E)$ におけるすべての 2 節点間の最短路とその距離を求める.
まず, 無向グラフの場合について説明する. 負の重みの枝を含まないと仮定する.

(i) 各節点 $i, j \in V$ に対して, $Distance(i, j) = \infty$ および $Path(i, j) = 0$ とする.

(ii) 各節点 $i, j \in V$ に対して, もし $Distance(i, j) > Distance(i, k) + Distance(k, j)$ ならば, $Distance(i, j) = Distance(i, k) + Distance(k, j)$ および $Path(i, j) = k$ と更新する.

(iii) $k = 1, \ldots, n$ まで, (ii) を繰り返す.

次に, 有向グラフの場合について説明する. 負の重みの枝を含むが, 負の長さのサイクルを含まないと仮定する.

(i) 各節点 $i, j \in V$ に対して $Distance(i, j) = \infty$ および $Path(i, j) = i$ とする.

(ii) 各節点 $i, j \in V$ に対して $Distance(i, j) > Distance(i, k) + Distance(k, j)$ ならば, $Distance(i, j) = Distance(i, k) + Distance(k, j)$ および $Path(i, j) = k$ と更新する.

(iii) もし $Distance(i, i)$ が負であるならば解が存在しないので, 処理を打ち切る. そうでなければ, $k = 1, \ldots, n$ まで (ii) を繰り返す.

(3) 2 節点間の最短路の計算

前述した Dijkstra の方法を応用してグラフ $G = (V, E)$ における 2 節点間の最短路とその距離を求める. ただし, すべての枝の重みは非負であると仮定する.

(i) 各節点 $i \in V$ に対して $Distance(i) = \infty$, $Path(i) = i$ および $P = \phi$ とする. 出発点 $init$ に対し, $Distance(init) = 0$ とする. また, $next = init$ とする.

(ii) $P = P \cup \{next\}$ とする. 節点 $next$ に接続している各節点 j に対して, もし $Distance(j) > D(next) + Weight(next, j)$ ならば, $Distance(j) = D(next) + Weight(next, j)$ および $Path(j) = next$ と更新する.

(iii) 各節点 $i \in P$ に対して $Distance(i)$ が最小である節点 v を選択する. それを $next = v$ とする.

(iv) $next = iend$ となるまで, (ii),(iii) を繰り返す.

5.1.3 参考文献

(1) G. E. Forsythe, M. A. Malcolm, C. B. Moler, 森正武訳, “計算機のための数値計算法”, 日本コンピュータ協会 (1978).

(2) R. P. Brent, “Algorithms for minimization without derivatives”, Englewood Cliffs, N. J., Prent–Hall (1973).

(3) 今野浩, 山下浩, “非線形計画法”, 日科技連 (1978).

(4) 中川徹, 小柳義夫, “最小二乗法による実験データ解析”, 東京大学出版会 (1982).

(5) M. J. D. Powell, “A Hybrid Method for Nonlinear Equations”, Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowits, ed. , Gordon and Breach, pp. 87–161 (1970).

(6) 茨木俊秀, 福島雅夫, “最適化プログラミング”, 岩波書店 (1991).

(7) V. フバータル, 阪田省二郎, 藤野和建訳, “線形計画法 (上)”, 啓学出版 (1986).

(8) 玄光男, 井田憲一, “BASIC による線形計画”, 共立出版 (1988).

(9) ANSOP 研究会編, “パソコン FORTRAN 版非線形最適化プログラミング”, 日刊工業新聞社 (1991).

(10) 伊理正夫, 白川功, 梶谷洋司, 篠田庄司, “演習グラフ理論”, コロナ社,(1983).

(11) 茨木俊秀, “アルゴリズムとデータ構造”, 照光堂,(1989).

(12) 茨木俊秀, “離散最適化とアルゴリズム”, 岩波書店,(1993).

(13) 茨木俊秀, 福島雅夫, “最適化の手法”, 共立出版,(1993).

(14) 棒沢芳雄, “オペレーションズ・リサーチ”, コロナ社,(1997).

(15) 今野浩, “線形計画法”, 日科技連,(1997).

5.2 制約なし 1 変数関数の極小化

5.2.1 ASL_dmuusn, ASL_rmuusn 1 変数関数の極小化

(1) **機 能**

1 変数関数 $f(x)$ の極小値探索を行う.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmuusn  (f, & x, er, & nev, & y);
```

単精度関数:

```
ierr = ASL_rmuusn  (f, & x, er, & nev, & y);
```

(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	f	{D / R}	–	入 力	関数 $f(x)$ を定義する関数名 (注意事項 (a) 参照)
2	x	{D∗ / R∗}	1	入 力	探索の出発 x_0
				出 力	最終到達点 x^*
3	er	{D / R}	1	入 力	要求精度 (既定値 : $2 \times \sqrt{\text{(誤差判定のための単位)}}$)
4	nev	I*	1	入 力	関数 $f(x)$ の最大評価回数 (既定値:100)
				出 力	実際の関数評価回数
5	y	{D∗ / R∗}	1	出 力	最終到達点 x^* での関数値 $y = f(x^*)$
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) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(b) nev > 3 (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (a) または (b) を満足しなかった.	既定値にセットして処理する.
4000	囲い込みに失敗した.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点の x, y の値を出力して, 処理を打ち切る.

(6) **注意事項**

(a) 関数 f の作り方は, 次に示すとおりである.

```
double FORTRAN f(double *x)
{
    return  (f(*x));
}
```

(b) 区間縮小により探索区間が $[a, b]$ となったとき, 次の条件により収束判定を行う.

$$\max(b - x, x - a) \leq 2 \times \mathrm{er} \times \max(1, \mid x \mid)$$

er としては, 既定値程度にとるのが望ましい.

(c) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) 極小値が複数あるときは, いずれの極小値に収束するか保証されない.

(e) 関数は 1 階連続微分可能である必要がある.

(f) 極大値探索を行いたい場合は, f(x) の関数値が正負逆になるように設定すればよい. このとき, 引数第 5 項 y の値は, 正負逆に出力される.

(7) **使用例**

(a) 問 題

$f(x) = x(x^2 - 2) - 5$

の極小値探索を行う.

(b) 入力データ

関数 $f(x)$ に対応する関数名: f

x = 1.0, er=0.0, nev=0

(c) 主プログラム

```
/*      C interface example for ASL_dmuusn */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
#else
double f(x)
double *x;
#endif
{
    return (*x)*((*x)*(*x)-2.0)-5.0;
}
#ifdef __cplusplus
}
#endif

int main()
{
    double x;
    double er;
    int nev;
    double y;
    int ierr;
    FILE *fp;

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

```
    printf( "    *** ASL_dmuusn ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &x );
    printf( "\tnev =  %6d\n", nev );
    printf( "\ter  =%8.3g\n", er );
    printf( "\n\tInitial Value \n" );
    printf( "\t  x =%8.3g\n", x );

    fclose( fp );

    ierr = ASL_dmuusn(f, &x, er, &nev, &y);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tnev  = %6d\n", nev );
    printf( "\n\tSolution\n" );
    printf( "\t  x =%8.3g\n", x );
    printf( "\n\tFunction Value\n" );
    printf( "\t  y =%8.3g\n", y );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmuusn ***

 ** Input **

nev =       0
er  =       0

Initial Value
  x =       1

 ** Output **

ierr =      0

nev  =     23

Solution
  x =   0.809

Function Value
  y =   -6.09
```

5.3 制約なし多変数関数の極小化

5.3.1 ASL_dmumqn, ASL_rmumqn
多変数関数の極小化 (導関数定義不要)

(1) **機 能**

n 変数関数 $f(\boldsymbol{x})$ の極小値探索を行う.

(2) **使用法**

倍精度関数:

ierr = ASL_dmumqn (f, x, n, er, & nev, & y, wk);

単精度関数:

ierr = ASL_rmumqn (f, x, n, er, & nev, & y, 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	f	{D / R}	–	入 力	関数 $f(\boldsymbol{x})$ を定義する関数 f(x) の関数名 (注意事項 (a) 参照)
2	x	{D∗ / R∗}	n	入 力	探索点の初期値 $\boldsymbol{x}_0$
				出 力	探索点の最終到達点 $\boldsymbol{x}^*$
3	n	I	1	入 力	独立変数 $\boldsymbol{x}$の成分の数 n
4	er	{D / R}	1	入 力	要求精度 (既定値 : $2 \times \sqrt{(\text{誤差判定のための単位})}$)
5	nev	I*	1	入 力	関数 $f(\boldsymbol{x})$ の最大評価回数 (既定値: 400 ×n)
				出 力	実際の関数評価回数
6	y	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}^*$ での関数値 $y = f(\boldsymbol{x}^*)$
7	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** n × (3 × n + 7)
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) er ≥ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(c) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (b) または (c) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点の x, y の値を出力して, 処理を打ち切る.

(6) **注意事項**

(a) 関数 f の作り方は, 次に示すとおりである.

```
double FORTRAN f(double *x)
{
    return (f(*x));
}
```

(b) 収束判定は次式によって行い, $\boldsymbol{x}$+$\triangle\boldsymbol{x}$を解とする.

$\|\triangle\boldsymbol{x}\| \leq \mathrm{er} \times \max(1, \|\boldsymbol{x} + \triangle\boldsymbol{x}\|)$ かつ $\|\nabla f(\boldsymbol{x})\| \leq 2 \times \mathrm{er}$

または $\|\nabla f(\boldsymbol{x})\| \leq$ 誤差判定のための単位

ここで, $\triangle\boldsymbol{x}$は$\boldsymbol{x}$に対する修正ベクトルであり, $\|\boldsymbol{x}\| = \max_i | x_i |$ である.

また $\nabla f(\boldsymbol{x})$ は $f(\boldsymbol{x})$ の勾配ベクトルで $\partial f(\boldsymbol{x})/\partial x_i$ を成分とする.

er としては, 既定値程度にとるのが望ましい.

(c) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) 勾配ベクトルが解析的に計算できるなら, 5.3.2 $\left\{\begin{matrix}\mathrm{ASL_dmumqg}\\ \mathrm{ASL_rmumqg}\end{matrix}\right\}$を利用したほうが効率がよい.

(e) 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい (詳細は 5.1.1 参照).

(f) 初期点で勾配ベクトルが 0 のときは, その点が解として出力される.

(g) 極小値がないときは, 最大関数評価回数まで計算し, ierr=5000 となる.

(h) 極小値が複数あるときは, いずれの極小値に収束するか保証されない.

(i) 関数は 2 階連続微分可能である必要がある.

(j) 極大値探索を行いたい場合は, $f(\boldsymbol{x})$ の関数値が正負逆になるように設定すればよい. このとき, 引数第 6 項 y の値は, 正負逆に出力される.

(7) 使用例

(a) 問 題

$f(\boldsymbol{x}) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2$

を初期値$\boldsymbol{x}$= $[-1.2, 1.0]^T$ として, 極小値探索する.

(b) 入力データ

関数 $f(\boldsymbol{x})$ に対応する関数名: f

x[0] = −1.2, x [1] =1.0, er=0.0, nev=0

(c) 主プログラム

```
/*      C interface example for ASL_dmumqn */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
#else
double f(x)
double *x;
#endif
{
    double y[2];

    y[0]=10.0*(x[1]-x[0]*x[0]);
    y[1]=1.0-x[0];
    return (y[0]*y[0]+y[1]*y[1]);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x;
    int n;
    double er;
    int nev;
    double y;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dmumqn ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*(3*n+7)) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    printf( "\tn   =  %6d\n", n );
    printf( "\tnev =  %6d\n", nev );
    printf( "\ter  =%8.3g\n", er );
    printf( "\n\tInitial Value \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  x[%6d]=%8.3g\n", i,x[i] );
    }
```

```
    fclose( fp );

    ierr = ASL_dmumqn(f, x, n, er, &nev, &y, wk);

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

    free( x );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmumqn ***

 ** Input **

n   =       2
nev =       0
er  =       0

Initial Value
  x[     0]=    -1.2
  x[     1]=       1

 ** Output **

ierr =      0

nev  =    140

Solution
  x[     0]=       1
  x[     1]=       1

Function Value
  y =5.45e-24
```

5.3.2 ASL_dmumqg, ASL_rmumqg
多変数関数の極小化 (導関数定義必要)

(1) **機 能**

n 変数関数 $f(\boldsymbol{x})$ の極小値探索を行う.

(2) **使用法**

倍精度関数:

ierr = ASL_dmumqg (f, subg, x, n, er, nev, & y, wk);

単精度関数:

ierr = ASL_rmumqg (f, subg, x, n, er, nev, & y, 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	f	{D / R}	−	入 力	関数 $f(\boldsymbol{x})$ を定義する関数 f(x) の関数名 (注意事項 (a) 参照)
2	subg	−	−	入 力	勾配ベクトル $\nabla f(\boldsymbol{x})$ を計算する関数 subg(x, g) の関数名 (注意事項 (b) 参照)
3	x	{D∗ / R∗}	n	入 力	探索点の初期値 $\boldsymbol{x}_0$
				出 力	探索点の最終到達点 $\boldsymbol{x}^*$
4	n	I	1	入 力	独立変数 $\boldsymbol{x}$ の成分の数 n
5	er	{D / R}	1	入 力	要求精度 (既定値: $2 \times \sqrt{\text{(誤差判定のための単位)}}$)
6	nev	I*	2	入 力	nev [0] は, 関数 f の最大評価回数 (既定値: 100 × n) nev [1] は, 関数 subg の最大評価回数 (既定値: 100 × n)
				出 力	実際の関数評価回数
7	y	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}^*$ での関数値 $y = f(\boldsymbol{x}^*)$
8	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** n × (3 × n + 7)
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) er ≥ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(c) nev[i − 1] > 0 (i = 1, 2) (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (b) または (c) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点の x, y の値を出力して, 処理を打ち切る.

(6) **注意事項**

(a) 関数 f の作り方は, 次に示すとおりである.

```
double FORTRAN f(double *x)
{
    return  (f(*x));
}
```

(b) 関数 subg の作り方は, 次に示すとおりである.

```
void FORTRAN subg(double *x, double *g)
{
    g[0] = ∇ f(x) の第 1 成分 = ∂f(x)/∂x_i
      ⋮
    g[n − 1] = ∇ f(x) の第 n 成分 = ∂f(x)/∂x_n
}
```

(c) 収束判定は次式によって行い, $\boldsymbol{x}$ +△$\boldsymbol{x}$を解とする.

$\|\triangle \boldsymbol{x}\| \leq \mathrm{er} \times \max(1, \|\boldsymbol{x} + \triangle \boldsymbol{x}\|)$ かつ $\|\nabla f(\boldsymbol{x})\| \leq 2 \times \mathrm{er}$

または $\|\nabla f(\boldsymbol{x})\| \leq$ 誤差判定のための単位

ここで, △$\boldsymbol{x}$ は $\boldsymbol{x}$に対する修正ベクトルであり, $\|\boldsymbol{x}\| = \max_i \mid x_i \mid$ である.

er としては, 既定値程度にとるのが望ましい.

(d) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(e) 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい (詳細は 5.1.1 参照).

(f) 初期点で勾配ベクトルが 0 のときは, その点が解として出力される.

(g) 極小値がないときは, 最大関数評価回数まで計算し, ierr=5000 となる.

(h) 極小値が複数あるときは, いずれの極小値に収束するか保証されない.

(i) 関数は 2 階連続微分可能である必要がある.

(j) 極大値探索を行いたい場合は, 関数値 $f(\boldsymbol{x})$ と勾配ベクトル $\nabla f(\boldsymbol{x})$ が正負逆になるように設定すればよい. このとき, 引数第 7 項 y の値は, 正負逆に出力される.

(7) **使用例**

(a) 問 題

$f(\boldsymbol{x}) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2$

を初期値$\boldsymbol{x}$= $[-1.2, 1.0]^T$ として, 極小値探索する.

(b) 入力データ

関数 $f(\boldsymbol{x})$ に対応する関数名: f1

勾配ベクトル $\nabla f(\boldsymbol{x})$ を計算する関数名: f2

x[0] = −1.2, x [1] =1.0, n=2, er=0.0, nev [0] =0, nev [1] =0

(c) 主プログラム

```
/*      C interface example for ASL_dmumqg */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f1(double *x)
#else
double f1(x)
double *x;
#endif
{
    double y[2];

    y[0]=10.0*(x[1]-x[0]*x[0]);
    y[1]=1.0-x[0];
    return (y[0]*y[0]+y[1]*y[1]);
}
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    f2(double *x,double *g)
#else
void    f2(x,g)
double *x;
double *g;
#endif
{
    double y[2];

    y[0] = 10.0*(x[1]-x[0]*x[0]);
    y[1] = 1.0-x[0];
    g[0] = -2.0*(20.0*x[0]*y[0]+y[1]);
    g[1] = 20.0*y[0];
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x;
    int n;
    double er;
    int nev[2];
    double y;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

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

    wk  = ( double * )malloc((size_t)( sizeof(double) * (n*(3*n+7)) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    fscanf( fp, "%d", &nev[0] );
```

```
    fscanf( fp, "%d", &nev[1] );
    fscanf( fp, "%lf", &er );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    printf( "\tn      =  %6d\n", n );
    printf( "\tnev[0] =  %6d\n", nev[0] );
    printf( "\tnev[1] =  %6d\n", nev[1] );
    printf( "\ter     =%8.3g\n", er );
    printf( "\n\tInitial Value\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  x[%6d] = %8.3g\n", i,x[i] );
    }

    fclose( fp );

    ierr = ASL_dmumqg(f1, f2, x, n, er, nev, &y, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tnev[0] = %6d\n", nev[0] );
    printf( "\tnev[1] = %6d\n", nev[1] );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  x[%6d] = %8.3g\n", i,x[i] );
    }
    printf( "\n\tFunction Value\n\n" );
    printf( "\t  y = %8.3g\n", y );

    free( x );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmumqg ***

 ** Input **

n      =       2
nev[0] =       0
nev[1] =       0
er     =       0

Initial Value

  x[     0] =     -1.2
  x[     1] =        1

 ** Output **

ierr =      0

nev[0] =     55
nev[1] =     36

Solution

  x[     0] =        1
  x[     1] =        1

Function Value

  y = 7.66e-25
```

5.4 制約なし関数二乗和の極小化

5.4.1 ASL_dmussn, ASL_rmussn
非線形最小二乗法 (導関数定義不要)

(1) **機 能**

n 変数関数の二乗和 $s = \sum_{i=1}^{m} f_i(\boldsymbol{x})^2$ の極小値探索を行う.

ここで, $f_i(\boldsymbol{x})$ は, n 変数ベクトル値関数 $\boldsymbol{f}(\boldsymbol{x})$ の第 i 成分である $(i = 1, \cdots, m)$.

(2) **使用法**

倍精度関数:

ierr = ASL_dmussn (sub, x, n, er, & nev, y, m, & s, iwk, wk);

単精度関数:

ierr = ASL_rmussn (sub, x, n, er, & nev, y, m, & s, iwk, 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	sub	−	−	入 力	関数値 $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ を計算する関数 sub(x, y) の関数名 (注意事項 (a) 参照)
2	x	{D∗ / R∗}	n	入 力	探索点の初期値 $\boldsymbol{x}_0$
				出 力	探索点の最終到達点 $\boldsymbol{x}^*$
3	n	I	1	入 力	独立変数$\boldsymbol{x}$の成分の数 n
4	er	{D / R}	1	入 力	要求精度 (既定値: $2 \times \sqrt{(\text{誤差判定のための単位})}$)
5	nev	I*	1	入 力	関数$\boldsymbol{f}(\boldsymbol{x})$ の最大評価回数 (既定値: 100 ×n)
				出 力	実際の関数評価回数
6	y	{D∗ / R∗}	m	出 力	最終到達点 $\boldsymbol{x}^*$ での関数の値 $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}^*)$
7	m	I	1	入 力	従属変数$\boldsymbol{y}$の成分の数 m
8	s	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}^*$ での関数二乗和の値 $\text{s}=\sum_{i=1}^{m} f_i(\boldsymbol{x}^*)^2$
9	iwk	I*	4 × n	ワーク	作業領域
10	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: m × (2 × n + 1) + n × (n + 4)
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) $0 < \mathrm{n} \leq \mathrm{m}$

(b) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(c) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (b) または (c) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	線形最小二乗法が解けなかった.	その時点の x, y, s の値を出力して処理を打ち切る.
4100	最急降下を計算できなかった.	
4200	2n 回連続して解の修正ができなかった.	
5000	与えられた最大評価回数に達しても収束しなかった.	

(6) **注意事項**

(a) 関数 sub の作り方は, 次に示すとおりである.

```
void FORTRAN sub(double *x, double *y)
{
    y[0] = f_1(x)
      ⋮
    y[m − 1] = f_m(x)
}
```

(b) 収束判定は次式によって行い, $\boldsymbol{x} + \triangle\boldsymbol{x}$ を解とする.

$\|\triangle\boldsymbol{x}\| \leq \mathrm{er} \times \max(1, \|\boldsymbol{x}+\triangle\boldsymbol{x}\|)$

ここで, $\triangle\boldsymbol{x}$ は $\boldsymbol{x}$ に対する修正ベクトルであり, $\|\boldsymbol{x}\| = \max_i |x_i|$ である.

er としては, 既定値程度にとるのが望ましい.

(c) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい (詳細は 5.1.1 参照).

(e) 極小値が複数あるときは, いずれの極小値に収束するか保証されない.

(f) 関数は 1 階連続微分可能である必要がある.

(7) 使用例

(a) 問 題

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} 10(x_2 - x_1^2) \\ 1 - x_2 \end{bmatrix}$$

のとき, 初期値 $\boldsymbol{x}$=$[-1.2, 1.0]^T$ として,
$s = f_1(\boldsymbol{x})^2 + f_2(\boldsymbol{x})^2$ を極小にする.

(b) 入力データ

関数値$\boldsymbol{f}(\boldsymbol{x})$ を計算する関数名: f

x[0] = −1.2, x [1] =1.0, n=2, er=0.0, nev=0, m=2

(c) 主プログラム

```
/*      C interface example for ASL_dmussn */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void   f(double *x,double *y)
#else
void   f(x,y)
double *x;
double *y;
#endif
{
    y[0]=10.0*(x[1]-x[0]*x[0]);
    y[1]=1.0-x[0];
}
#ifdef __cplusplus
}
#endif

int main()
{
    double *x;
    int n;
    double er;
    int nev;
    double *y;
    int m;
    double s;
    int *iwk;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

    printf( "    *** ASL_dmussn ***\n" );
    printf( "\n     ** Input **\n\n" );
    n=2;
    m=2;

    iwk  = ( int * )malloc((size_t)( sizeof(int) * (3*n) ));
    if( iwk  == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
    x  = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( x  == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    y  = ( double * )malloc((size_t)( sizeof(double) * m ));
    if( y  == NULL )
    {
        printf( "no enough memory for array y\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * (m*(2*n+1)+n*(n+4)) ));
    if( wk  == NULL )
```

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

    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
    }
    printf( "\tn   =  %6d\n", n );
    printf( "\tm   =  %6d\n", m );
    printf( "\tnev =  %6d\n", nev );
    printf( "\ter  =%8.3g\n", er );
    printf( "\n\tInitial Value \n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  x[%6d]=%8.3g\n", i,x[i] );
    }

    fclose( fp );

    ierr = ASL_dmussn(f, x, n, er, &nev, y, m, &s, iwk, wk);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tnev  = %6d\n", nev );
    printf( "\n\tSolution\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf(  "\t  x[%6d]=%8.3g\n", i,x[i] );
    }
    printf( "\n\tLeast Squares\n" );
    printf( "\t  s    =%8.3g\n", s    );
    printf( "\n\tFunction Value\n" );
    printf( "\t  y[0] =%8.3g\n", y[0] );
    printf( "\t  y[1] =%8.3g\n", y[1] );

    free( x );
    free( y );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmussn ***

 ** Input **

n   =       2
m   =       2
nev =       0
er  =       0

Initial Value
  x[     0]=    -1.2
  x[     1]=       1

 ** Output **

ierr =      0

nev  =     30

Solution
  x[     0]=       1
  x[     1]=       1

Least Squares
  s    =       0

Function Value
  y[0] =       0
  y[1] =       0
```

5.5 制約付き 1 変数関数の極小化

5.5.1 ASL_dmcusn, ASL_rmcusn
1 変数関数の極小化 (区間指定)

(1) **機 能**

区間 $[a, b]$ の中で, 1 変数関数 $f(x)$ の極小値探索を行う.

(2) **使用法**

倍精度関数:

ierr = ASL_dmcusn (f, ax, bx, er, & nev, & x, & y);

単精度関数:

ierr = ASL_rmcusn (f, ax, bx, er, & nev, & x, & y);

(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	f	{D / R}	–	入 力	関数 $f(x)$ を定義する関数名 (注意事項 (a) 参照)
2	ax	{D / R}	1	入 力	探索区間の左端の初期値 a
3	bx	{D / R}	1	入 力	探索区間の右端の初期値 b
4	er	{D / R}	1	入 力	要求精度 (既定値: $2 \times \sqrt{(\text{誤差判定のための単位})}$)
5	nev	I*	1	入 力	関数 $f(x)$ の最大評価回数 (既定値:100)
				出 力	実際の関数評価回数
6	x	{D*, R*}	1	出 力	最終到達点 x^*
7	y	{D*, R*}	1	出 力	最終到達点 x^* での関数値 $y = f(x^*)$
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) ax $<$ bx

(b) ax $\neq$ bx

(c) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

(d) nev $>$ 0 (既定値にするため, 0 を入力する場合は除く)

(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.	
1200	制限条件 (a) を満足しなかった.	ax と bx を入れ換えて処理する.
1500	制限条件 (c) または (d) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (b) を満足しなかった.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点の x, y の値を出力して, 処理を打ち切る.

(6) **注意事項**

(a) 関数 f の作り方は, 次に示すとおりである.

```
double FORTRAN f(double *x)
{
    return  (f(*x));
}
```

(b) 区間縮小により探索区間が (a, b) となったとき, 次の条件により収束判定を行う.

$$\max(b - x, x - a) \leq 2 \times \mathrm{er} \times \max(1, \mid x \mid)$$

er としては, 既定値程度にとるのが望ましい.

(c) 引数の内容の欄に既定値が記されている場合は, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) 極小値がない場合, 端点を解とし, ierr は 0 になる.

(e) 区間内に極小値が複数あるときは, いずれの極小値に収束するか保証されない.

(f) 関数は 1 階連続微分可能である必要がある.

(g) 極大値探索を行いたい場合は, $f(x)$ の関数値が正負逆になるように設定すればよい. このとき, 引数第 7 項 y の値は, 正負逆に出力される.

(7) **使用例**

(a) 問 題

区間 $[0, 1]$ において,

$f(x) = x(x^2 - 2) - 5$

の極小値探索を行う.

(b) 入力データ

関数 $f(x)$ に対応する関数名: f

ax=0.0, bx=1.0, er=0.0, nev=0

(c) 主プログラム

```
/*      C interface example for ASL_dmcusn */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double f(double *x)
```

```
#else
double f(x)
double *x;
#endif
{
    return ((*x)*((*x)*(*x)-2.0)-5.0);
}
#ifdef __cplusplus
}
#endif

int main()
{
    double ax;
    double bx;
    double er;
    int nev;
    double x;
    double y;
    int ierr;
    FILE *fp;

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

    printf( "    *** ASL_dmcusn ***\n" );
    printf( "\n     ** Input **\n\n" );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%lf", &er );
    fscanf( fp, "%lf", &ax );
    fscanf( fp, "%lf", &bx );

    printf( "\tnev =  %6d\n", nev );
    printf( "\ter  =%8.3g\n", er );
    printf( "\n\tSearch Section\n" );
    printf( "\t  ax =%8.3g\n", ax );
    printf( "\t  bx =%8.3g\n", bx );

    fclose( fp );

    ierr = ASL_dmcusn(f, ax, bx, er, &nev, &x, &y);

    printf( "\n     ** Output **\n\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\n\tnev  = %6d\n", nev );
    printf( "\n\tSolution\n" );
    printf( "\t  x  =%8.3g\n", x );
    printf( "\n\tFunction Value\n" );
    printf( "\t  y  =%8.3g\n", y );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmcusn ***

 ** Input **

nev =       0
er  =       0

Search Section
  ax =       0
  bx =       1

 ** Output **

ierr =      0

nev  =     11

Solution
  x  =   0.816

Function Value
  y  =   -6.09
```

5.6 制約付き多変数線形関数の最小化 (線形計画)

5.6.1 ASL_dmclsn, ASL_rmclsn
多変数線形関数の最小化 (線形制約)

(1) **機 能**

線形制約の中で, n 次のベクトル $\boldsymbol{x} = [x_1, \cdots, x_n]^T$ の多変数線形関数 $f(\boldsymbol{x})$ を最小にする $\boldsymbol{x}$ を求める.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$$

制約条件は m 個で, $i = 1, 2, \cdots, m$ について次のどれかである.

- $\boldsymbol{a}_i^T\boldsymbol{x} = b_i$
- $\boldsymbol{a}_i^T\boldsymbol{x} \leq b_i$
- $\boldsymbol{a}_i^T\boldsymbol{x} \geq b_i$

また, $\boldsymbol{x}$ の定義域は

$$d_j \leq x_j \leq u_j \quad (j = 1, 2, \cdots, n)$$

ここで, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$, $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ は, n 次ベクトル, b_i, dj, u_j は定数 $(i = 1, 2, \cdots, m;\ \ j = 1, 2, \cdots, n)$ とする.

(2) **使用法**

倍精度関数:

ierr = ASL_dmclsn (a, lma, & nm, b, m, xup, xlow, c, itype, er, & nev, x, & y, isw, iwk, wk);

単精度関数:

ierr = ASL_rmclsn (a, lma, & nm, b, m, xup, xlow, c, itype, er, & nev, x, & y, isw, iwk, wk);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D*, R*}	lma×nm	入 力	isw=0 のとき制約条件の係数に対応する行列 $A = (a_{i,j})$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ (注意事項 (a) (f) 参照)
				出 力	スラック変数を用いて変更した制約条件の係数に対応する行列
2	lma	I	1	入 力	配列 a の整合寸法
3	nm	I*	1	入 力	isw=0 のとき $n + 2 \times m$ の値. ここで n は変数の数, m は制約条件の数 (注意事項 (f) 参照)
				出 力	作業変数

項番	引数と戻り値	型	大きさ	入出力	内　　容
4	b	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	m	入 力	isw=0 のとき制約条件の右辺 $\boldsymbol{b} = (b_i)$ $(i = 1, 2, \cdots, m)$ (注意事項 (f) 参照)
				出 力	スラック変数を用いて変更した制約条件の右辺
5	m	I	1	入 力	制約条件の数 m
6	xup	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nm	入 力	isw=0 のとき変数の上限値 $\boldsymbol{u} = (u_j)$ $(j = 1, 2, \cdots, n)$ (注意事項 (c)(f) 参照)
				出 力	スラック変数を用いて変更した変数の上限値
7	xlow	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nm	入 力	isw=0 のとき変数の下限値 $\boldsymbol{d} = (d_j)$ $(j = 1, 2, \cdots, n)$ (注意事項 (c)(f) 参照)
				出 力	スラック変数を用いて変更した変数の下限値
8	c	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nm	入 力	isw=0 のとき関数の係数 $\boldsymbol{c} = (c_j)$ $(j = 1, 2, \cdots, n)$ (注意事項 (f) 参照)
				出 力	スラック変数を用いて変更した関数の係数
9	itype	I*	m	入 力	制約条件の等号, 不等号の区別 0: $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$ 1: $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$ -1: $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$
10	er	$\left\{\begin{matrix}\mathrm{D} \\ \mathrm{R}\end{matrix}\right\}$	1	入 力	要求精度 (既定値 : $2 \times \sqrt{(\text{誤差判定のための単位})}$)
11	nev	I*	1	入 力	関数 $f(\boldsymbol{x})$ の最大評価回数 (既定値: $10 \times$ m)
				出 力	実際の関数評価回数
12	x	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	nm	出 力	最終到達点 $\boldsymbol{x}$ (注意事項 (b) 参照)
13	y	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	1	出 力	最終到達点 $\boldsymbol{x}$ での関数値 $f(\boldsymbol{x})$
14	isw	I	1	入 力	処理スイッチ (注意事項 (i) 参照) 0: 最初の処理 1: 継続の処理
15	iwk	I*	nm + m	ワーク	作業領域
16	wk	$\left\{\begin{matrix}\mathrm{D}* \\ \mathrm{R}*\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**: $2 \times \mathrm{m}^2 + \mathrm{nm} \times (7 + \mathrm{m})$
17	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $\mathrm{isw} = 0$ または $\mathrm{isw} = 1$

(c) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

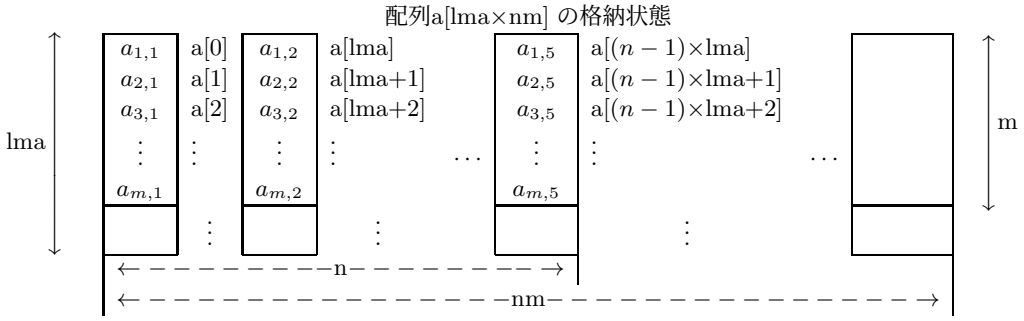
(d) $\mathrm{nev} > 0$ (既定値にするため, 0 を入力する場合は除く)

(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.	
1500	制限条件 (c) または (d) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
4000	実行可能基底解を求められなかった.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点での x, y の値を出力して, 処理を打ち切る. (注意事項 (h)(i) 参照)

(6) **注意事項**

(a) isw=0 の時, 制約条件に対応する係数 $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ を以下の様に配列 a に格納する. ここで, m は制約条件の数, n は変数の数をそれぞれ表す.



備　考
a.　nm $= n + 2 \times m$ を満たさなければならない.

(b) 最終到達点$\boldsymbol{x}$の値は x [0] ∼x $[n-1]$ に設定される. 配列 x の残りはスラック変数等の値である.

(c) 変数の上限値または下限値が特に定められていないときには, 適当に絶対値の大きな正数または負数をそれぞれ上限値または下限値とする. なお, 最終到達点での変数値がここで定めた上限値または下限値と一致するときには, 最適解が求められていない可能性があるので, さらに絶対値の大きな数を上限値または下限値に設定して再計算を実行する必要がある.

(d) 引数の内容の欄に既定値が記されている場合には, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(e) 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい. 例えば, $f(\boldsymbol{x}) = 100x_1 + x_2$ 制約条件：$200x_1 + 5x_2 = 3$ のような場合, $y_1 = 100x_1, y_2 = x_2$ と変数変換を行って, $h(\boldsymbol{y}) = y_1 + y_2$ 制約条件：$2y_1 + 5y_2 = 3$ としたほうがよい結果が得られる.

(f) a, b, xup, xlow, c, nm の出力には制約条件等を変換した後の値が入る.

(g) 最大値探索を行いたい場合は, $-f(\boldsymbol{x})$ の最小値探索を行えば良い. このとき, y の値を正負逆にしたものが最大値である.

(h) ierr = 5000 のときの$\boldsymbol{x}$の値は, 最適解ではないが, 制約条件を満たしている.

(i) 指定した収束回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて, 引き続き計算することができる. この処理を行うときには, isw の値を 1, nev の値を十分な値とし, それ以外の入力値には前回の出力値をそのまま用いる. また, 前回の作業領域の情報も用いる ((7) 使用例参照) .

(7) 使用例

(a) 問 題

$$\begin{aligned} x_1 + 2x_2 + 3x_3 - 3x_4 - 2x_5 - x_6 &\leq 20 \\ 2x_1 - 3x_2 - x_3 + 2x_4 + 4x_5 + x_6 &\geq 28 \\ -3x_1 + 2x_2 + 2x_3 + x_4 + 5x_5 + 2x_6 &= 40 \\ 5x_1 - x_2 + 3x_3 + 3x_4 - 2x_5 + 4x_6 &= 50 \end{aligned}$$

$$0.0 \leq x_i \leq 1000.0 \ (i = 1, 2, 3, 4, 5, 6)$$

のもとで

$$f(\boldsymbol{x}) = -x_1 - 3x_2 + 2x_3 + 3x_4 - 4x_5 - 2x_6$$

を最小にする.

なお, この例では, 注意事項 (i) で述べた継続処理を行うために, 最大評価回数 nev=6 と小さく設定し, ierr=5000 が出力されるようにしている.

(b) 入力データ

(1 回目): nm=14, m=4, er=0.0, nev=6, isw=0, lma=11, 配列 a, b, xup, xlow, c, itype

(2 回目以降): nev=6, isw=1

(他は, 前回の計算後の値を, そのまま入力値とする)

(c) 主プログラム

```
/*      C interface example for ASL_dmclsn */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *xup, *xlow, *c, er, *x, y, *wk;
    int *itype, nm, m, lma, nev, isw, *iwk, ierr;
    int i,j,nn,once=0,nwk;
    FILE *fp;

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

    lma = 11;
    fscanf( fp, "%d %d %lf %d %d", &nm, &m, &er, &nev, &isw );
    printf( "\tnm = %6d\tm = %6d\n", nm, m );
    nn = nm-2*m;

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

```
    printf( "no enough memory for array c\n" );
    return -1;
}
itype = ( int * )malloc((size_t)( sizeof(int) * m ));
if( itype == NULL )
{
    printf( "no enough memory for array itype\n" );
    return -1;
}
x = ( double * )malloc((size_t)( sizeof(double) * nm ));
if( x == NULL )
{
    printf( "no enough memory for array x\n" );
    return -1;
}
nwk=2*m*m+nm*(7+m);
wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * (nm+m) ));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}

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

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

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

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

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

printf( "\n      ** Vector itype **\n\n" );
printf( "\t" );
for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &itype[i] );
    printf( "  %6d ", itype[i] );
}
printf( "\n" );


ierr = ASL_dmclsn(a, lma, &nm, b, m, xup, xlow, c, itype, er, &nev, x, &y, isw, iwk, wk);
printf( "\n     ** Output **\n" );

printf( "\n     ** First Result (isw == 0) **\n" );
printf( "\n\tierr = %6d\n", ierr );

printf( "\n      ** Vector x **\n\n" );
```

```
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\tx[ %6d ] = %8.3g\n", i, x[i] );
    }
    printf( "\ty = %8.3g\n", y );

loop:
    fscanf( fp, "%d %d", &nev, &isw );

    ierr = ASL_dmclsn(a, lma, &nm, b, m, xup, xlow, c, itype, er, &nev, x, &y, isw, iwk, wk);


    printf( "\n\n     ** Improved Result (isw != 0) **\n" );
    printf( "\n\tierr = %6d\n", ierr );

    printf( "\n      ** Vector x **\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\tx[ %6d ] = %8.3g\n", i, x[i] );
    }
    printf( "\ty = %8.3g\n", y );

    if (ierr == 5000 && once==0)
    {
        once=1;
        goto loop;
    }

    fclose( fp );
    free( a );
    free( b );
    free( xup );
    free( xlow );
    free( c );
    free( itype );
    free( x );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmclsn ***

 ** Input **

nm =     14 m =      4

  ** Matrix a **

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

  ** Vector b **

      20       28       40       50

  ** Vector xup **

   1e+03    1e+03    1e+03    1e+03    1e+03    1e+03

  ** Vector xlow **

       0        0        0        0        0        0

  ** Vector c **

      -1       -3        2        3       -4       -2

  ** Vector itype **

       1       -1        0        0

 ** Output **

 ** First Result (isw == 0) **

ierr =   5000

  ** Vector x **

x[      0 ] =     5.16
x[      1 ] =        0
x[      2 ] =     9.84
x[      3 ] =        0
x[      4 ] =     6.41
x[      5 ] =     1.87
y =    -14.9


 ** Improved Result (isw != 0) **

ierr =      0
```

```
  ** Vector x **

x[      0 ] =     18.1
x[      1 ] =     14.2
x[      2 ] =        0
x[      3 ] =        0
x[      4 ] =     13.2
x[      5 ] =        0
y =     -113
```

5.6.2 ASL_dmclaf, ASL_rmclaf
多変数線形関数の最小化 (実不規則スパース行列で与えられる線形制約)

(1) **機 能**

線形制約の中で, n 次のベクトル $\boldsymbol{x} = [x_1, \cdots, x_n]^T$ の多変数線形関数 $f(\boldsymbol{x})$ を最小にする $\boldsymbol{x}$ を求める.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x}$$

制約条件は m 個で, $i = 1, 2, \cdots, m$ について次のどれかである.

- $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$
- $\boldsymbol{a}_i^T \boldsymbol{x} \geq b_i$

また, $\boldsymbol{x}$ の定義域は

$$d_j \leq x_j \leq u_j \quad (j = 1, 2, \cdots, n)$$

ここで, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$, $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ は, n 次ベクトル, b_i, dj, u_j は定数 $(i = 1, 2, \cdots, m;\ \ j = 1, 2, \cdots, n)$ とする.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmclaf  (aval, na, jcn, ia, nm, b, m, xup, xlow, c, itype, etb, ap, bm, nck, & nev, x,
                    & y, & isw1, isw2, iw, w, nw);
```

単精度関数:

```
ierr = ASL_rmclaf  (aval, na, jcn, ia, nm, b, m, xup, xlow, c, itype, etb, ap, bm, nck, & nev, x,
                    & y, & isw1, isw2, iw, w, nw);
```

(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	aval	{D∗ / R∗}	na＋2×m	入 力	isw1=0 のとき制約条件の係数に対応する行列 $A=(a_{i,j})$ $(i=1,2,\cdots,m;\ j=1,2,\cdots,n)$ (注意事項 (e) 参照)
				出 力	スラック変数を用いて変更した制約条件の係数に対応する行列
2	na	I	1	入 力	行列 A の非零要素の個数
3	jcn	I*	na＋2×m	入 力	行列 A の非零要素の列番号 (注意事項 (e) (f) 参照)
4	ia	I*	m+1	入 力	行列 A の非零要素の情報 (注意事項 (e) (f) 参照)
5	nm	I	1	入 力	isw2=0 のとき $n+2\times m$ の値. ここで n は変数の数, m は制約条件の数 (注意事項 (f) 参照)
6	b	{D∗ / R∗}	m	入 力	isw2=0 のとき制約条件の右辺 $\boldsymbol{b}=(b_i)$ $(i=1,2,\cdots,m)$ (注意事項 (f) 参照)
				出 力	スラック変数を用いて変更した制約条件の右辺
7	m	I	1	入 力	制約条件の数 m
8	xup	{D∗ / R∗}	nm	入 力	isw2=0 のとき変数の上限値 $\boldsymbol{u}=(u_j)$ $(j=1,2,\cdots,n)$ (注意事項 (b)(f) 参照)
				出 力	スラック変数を用いて変更した変数の上限値
9	xlow	{D∗ / R∗}	nm	入 力	isw2=0 のとき変数の下限値 $\boldsymbol{d}=(d_j)$ $(j=1,2,\cdots,n)$ (注意事項 (b)(f) 参照)
				出 力	スラック変数を用いて変更した変数の下限値
10	c	{D∗ / R∗}	nm	入 力	isw2=0 のとき関数の係数 $\boldsymbol{c}=(c_j)$ $(j=1,2,\cdots,n)$ (注意事項 (f) 参照)
				出 力	スラック変数を用いて変更した関数の係数
11	itype	I*	m	入 力	制約条件の等号, 不等号の区別 0: $\boldsymbol{a}_i^T\boldsymbol{x}=b_i$ 1: $\boldsymbol{a}_i^T\boldsymbol{x}\le b_i$ −1: $\boldsymbol{a}_i^T\boldsymbol{x}\ge b_i$
12	etb	{D∗ / R∗}	3	入 力	etb[0]：収束判定の単位 ϵ_C (既定値：$10\times\sqrt{(\text{誤差判定のための単位})}$) etb[1]：残差の検定の単位 ϵ_r (既定値：$\sqrt{(\text{誤差判定のための単位})}$) etb[2]：実行可能性の検定の単位 ϵ_A (既定値：$\sqrt{(\text{誤差判定のための単位})}$) (注意事項 (i) 参照)

項番	引数と戻り値	型	大きさ	入出力	内　　容
13	ap	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	入 力	ステップサイズのパラメータ α (既定値：0.99)(注意事項 (h) 参照)
14	bm	$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{R}\end{matrix}\right\}$	1	入 力	ビッグ M 法のパラメータ M (既定値：$10 \times \max_i \lvert c_i \rvert$)
15	nck	I	1	入 力	制約条件に対する残差のチェックを行う間隔 (既定値: 10) (注意事項 (i) 参照)
16	nev	I*	1	入 力	関数 $f(\boldsymbol{x})$ の最大評価回数 (既定値: $10 \times$ m)
				出 力	実際の関数評価回数
17	x	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	nm	出 力	最終到達点 $\boldsymbol{x}$ (注意事項 (a) 参照)
18	y	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	1	出 力	最終到達点 $\boldsymbol{x}$ での関数値 $f(\boldsymbol{x})$
19	isw1	I*	1	入 力	処理スイッチ (既定値:0) (注意事項 (j) 参照) 0: 自動的に選択 1: 連立 1 次方程式の係数行列を密行列として扱う. 2: 連立 1 次方程式の係数行列を疎行列として扱う.
				出 力	isw1=0 のとき選択された処理スイッチ
20	isw2	I	1	入 力	処理スイッチ (既定値:0) (注意事項 (k) 参照) 0: 最初の処理 1: 継続の処理
21	iw	I*	内容参照	ワーク	作業領域 **大きさ**: nw + na + 14 × m + 4 × nm + 30
22	w	$\left\{\begin{matrix}\mathrm{D}*\\ \mathrm{R}*\end{matrix}\right\}$	nw	ワーク	作業領域
23	nw	I	1	入 力	配列 w の大きさ (注意事項 (l) 参照)
24	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) xlow[j − 1] ≤ xup[j − 1] j = 1, 2, . . . , nm − 2 × m

(b) isw1 = 0 または isw1 = 1

(c) isw2 = 0 または isw2 = 1

(d) etb[i − 1] ≥ 誤差判定のための単位 (i = 1, 2, 3) (既定値にするため, 0.0 を入力する場合は除く)

(e) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(f) nck > 0 (既定値にするため, 0 を入力する場合は除く)

(g) ap > 0 (既定値にするため, 0.0 を入力する場合は除く)

(h) bm > 0 (既定値にするため, 0.0 を入力する場合は除く)

(i) 0 < nm, 0 < m

(j) ia [0] =1
0 < ia[i] − ia[i − 1] ≤ n (i = 1, 2, · · · , n − 1)
0 < na − ia[n − 1] + 1 ≤ n

(k) m ≤ na ≤ m × n

(l) 0< 行列 A の非零要素の列番号 ≤n
配列 jcn に格納されている行列 A の非零要素の列番号は, 各行ごとに昇順でなければならない. また, 各列に非零要素が少なくとも 1 個存在しなければならない.

(m) nw ≥ na + 17 × m + 13 + ($A(D^{(k)})^2A^{\mathrm{T}}$ を LU 分解するのに必要な領域の大きさ).

なお, ここで n = nm − 2 × m である. また, $D^{(k)}$ は, k 回目の反復の時点での解 $\boldsymbol{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})^{\mathrm{T}}$ に対して $x_i^{(k)}$ を (i, i) 要素に持つ対角行列である.

(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	ある j (j = 1, 2, . . . ,nm−2×m) について, xlow[$j-1$] と xup[j] が, 制限条件 (a) を満足しなかった.	xlow[$j-1$] と xup[$j-1$] の値を入れ替えて処理する.
1500	制限条件 (b)-(h) のいずれかを満足しなかった.	既定値にセットして処理する.
3000	制限条件 (i) を満足しなかった.	処理を打ち切る.
3010	制限条件 (j)-(l) のいずれかを満足しなかった.	
3100	制限条件 (m) を満足しなかった.	
4000	実行可能基底解を求められなかった.	
4100	制約条件を与える行列 A が $rank(A) < m$ であった. (m は制約条件の個数)	
4200	制約条件に対する残差が要求精度を満たさなくなった. (注意事項 (i) 参照)	
5000	与えられた最大評価回数に達しても収束しなかった.	その時点での x, y の値を出力して, 処理を打ち切る. (注意事項 (h)(i) 参照)

(6) **注意事項**

(a) 最終到達点$\boldsymbol{x}$の値は x [0] ∼x [$n-1$] に設定される. 配列 x の残りはスラック変数等の値である.

(b) 変数の上限値または下限値が特に定められていないときには, 適当に絶対値の大きな正数または負数をそれぞれ上限値または下限値とする. なお, 最終到達点での変数値がここで定めた上限値または下限値と一致するときには, 最適解が求められていない可能性があるので, さらに絶対値の大きな数を上限値または下限値に設定して再計算を実行する必要がある.

(c) 引数の内容の欄に既定値が記されている場合には, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

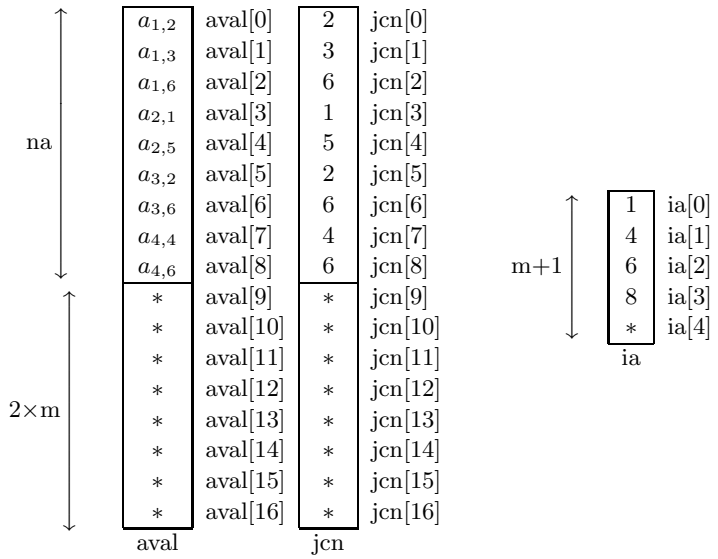
(d) 各変数から関数値への寄与が同程度になるようにスケーリングするのが望ましい. 例えば, $f(\boldsymbol{x}) = 100x_1 + x_2$ 制約条件 : $200x_1 + 5x_2 = 3$ のような場合, $y_1 = 100x_1, y_2 = x_2$ と変数変換を行って, $h(\boldsymbol{y}) = y_1 + y_2$ 制約条件 : $2y_1 + 5y_2 = 3$ としたほうがよい結果が得られる.

(e) isw=0 の時, 制約条件に対応する係数 $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ のうち 0 でないものだけを配列 aval に格納する. ここで, m は制約条件の数, n は変数の数をそれぞれ表す. 例えば, 制約条件を表す行列 $A = (a_{ij})$ が

$$A = \begin{bmatrix} 0.0 & a_{12} & a_{13} & 0.0 & 0.0 & a_{16} \\ a_{21} & 0.0 & 0.0 & 0.0 & a_{25} & 0.0 \\ 0.0 & a_{32} & 0.0 & 0.0 & 0.0 & a_{36} \\ 0.0 & 0.0 & 0.0 & a_{44} & 0.0 & a_{46} \end{bmatrix}$$

で与えられるときに配列 aval, jcn および ia の格納状態は次のようになる.

配列 aval, ja, ia の格納状態



備考 ∗ の部分は入力値として設定されていなくてもよい.

(f) a, b, xup, xlow, c, nm の出力には制約条件等を変換した後の値が入る.

(g) 最大値探索を行いたい場合は, $-f(\boldsymbol{x})$ の最小値探索を行えば良い. このとき, y の値を正負逆にしたものが最大値である.

(h) ステップサイズのパラメータ α が 1 に近いほど, 理論上は最適解への収束に要する反復回数は少なくなるが, α が 1 に近すぎると, 数値計算上の誤差が大きくなる場合がある.

(i) この関数では, 計算誤差により制約条件が十分な精度で満たされなくなることを防止するため, 反復回数 k が

i. k が 1 のとき

ii. k が引数 nck の値で割り切れるとき

iii. k が引数 nev の値に等しい時

のいずれかの条件を満たす時には, その時点での解 $\boldsymbol{x}^{(k)}$ の制約条件に対する残差が条件

$$||A\boldsymbol{x}^{(k)} - b|| \leq \epsilon_r$$

を満たしているかどうかをチェックする. ここで ϵ_r は引数 etb[1] に設定されている値である. もしこの条件が満たされないときには, それ以前にこの条件を満たした解をもとに初期解を再計算して反復を繰り返す. 初期解の再計算から nck 回後の反復で再び上記の条件を満たさなかったときは, nck $\leftarrow \max(\mathrm{nck}/2, 1)$ としてさらに反復を繰り返す. nck=1 になっても上記の条件を満たさなかった場合は ierr=4200 を出力して処理を打ち切る.

(j) 本関数では, 最適解の探索方向を定めるために, 連立 1 次方程式を解くが, その方法として連立 1 次方程式の係数行列を密行列として扱うか, 疎行列として扱うかを引数 isw1 の値により, 選択することができる. 係数行列を密行列として扱う場合には, < 基本機能第 2 分冊 > 2.2.2 $\left\{\begin{matrix}\text{ASL\_dbgmsl}\\ \text{ASL\_rbgmsl}\end{matrix}\right\}$ が使用され, 疎行列として扱う場合には < 基本機能第 2 分冊 > 2.21.1 $\left\{\begin{matrix}\text{ASL_dbmfsl}\\ \text{ASL_rbmfsl}\end{matrix}\right\}$が使用される. 制約条件の係数行列 A が疎行列で, なおかつ AA^{T} もまた疎行列である場合には isw1=2 に設定して, 連立 1 次方程式の係数行列を疎行列として扱った方が計算時間は少なくて済む. それ以外の場合は isw1=0 に設定した方がよい.

(k) 指定した収束回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて, 引き続き計算することができる. この処理を行うときには, isw2 の値を 1, nev の値を十分な値とし, それ以外の入力値には前回の出力値をそのまま用いる. また, 前回の作業領域の情報も用いる (例題参照) .

(l) 反復計算が終了した時点で, 人為変数の絶対値の最大値が etb[2] に設定された値より大きい時には, 与えられた問題が実行不可能であると判定し, ierr=4000 が出力される.

(m) 制約条件を与える行列 A について $rank(A) < m$ の場合, 本関数では最適解を計算することができず, ierr=4100 が出力される. このような場合には 5.6.1 $\left\{\begin{matrix}\text{ASL\_dmclsn}\\ \text{ASL\_rmclsn}\end{matrix}\right\}$ を利用されたい.

(n) 配列 w の大きさ nw は前もって見積もる必要がある. nw が十分大きくないために ierr=3100 が出力されることを防ぐには, nw = na + 11 × nm − 4 × m + 2 × m × m + 10 に設定すれば良い.

(7) 使用例

(a) 問 題

$$\begin{aligned} x_1 + 2x_4 + 3x_5 &\geq -7 \\ 2x_2 - x_3 &= 0 \\ x_1 + 2x_5 &\leq 8 \end{aligned}$$

$$0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 1 \leq x_3 \leq 2, -3 \leq x_4 \leq 0, 2 \leq x_5 \leq 5$$

のもとで

$$f(\boldsymbol{x}) = 2x_1 + x_2 + x_3 - 3x_4 + x_5$$

を最小にする.

なお, この例では, 注意事項 (k) で述べた継続処理を行うために, 最大評価回数 nev=5 と小さく設定し, ierr=5000 が出力されるようにしている.

(b) 入力データ

(1 回目): nm=14, m=4, ap=0.0, bm=0.0, nev=5, isw1=0, isw2=0, lma=11, 配列 a, jcn, ia, b, xup, xlow, c, etb, itype

(2 回目以降): nev=20, isw2=1

(他は, 前回の計算後の値を, そのまま入力値とする)

(c) 主プログラム

```
/*      C interface example for ASL_dmclaf */

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

int main()
{
    double *aval;
    int na;
    int *jcn;
    int *ia;
    int nm;
    double *b;
    int m;
    double *xup;
    double *xlow;
    double *c;
    int *itype;
    double er[3];
    double ap;
    double bm;
    int nck;
    int nev;
    double *x;
    double y;
    int isw1;
    int isw2;
    int *iw;
    double *w;
    int nw;
    int ierr;
    int i,n;
    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &nm );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%lf %lf %lf", &er[0], &er[1], &er[2] );
    fscanf( fp, "%lf", &ap );
    fscanf( fp, "%lf", &bm );
    fscanf( fp, "%d", &nck );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw1 );
    fscanf( fp, "%d", &isw2 );
    fscanf( fp, "%d", &nw );

    n = nm - 2*m;

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

    jcn = ( int * )malloc((size_t)( sizeof(int) * (na+2*m) ));
    if( jcn == NULL )
    {
        printf( "no enough memory for array jcn\n" );
        return -1;
    }

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

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

```
    return -1;
}

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

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

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

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

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

iw = ( int * )malloc((size_t)( sizeof(int) * (nw+na+14*m+4*nm+30) ));
if( iw == NULL )
{
    printf( "no enough memory for array iw\n" );
    return -1;
}

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

printf( "\tna   = %6d nm  = %6d m    = %6d\n", na, nm, m );
printf( "\tnck  = %6d nev = %6d isw1 = %6d\n", nck, nev, isw1 );
printf( "\tisw2 = %6d nw  = %6d\n\n", isw2, nw );
printf( "\ter[0] = %8.3g er[1] = %8.3g er[2] = %8.3g\n\n", er[0], er[1], er[2] );
printf( "\tap  = %8.3g bm = %8.3g\n", ap, bm );

printf( "\n\n\tMatrix A\n" );
printf("\n\t");
for( i=0 ; i<na ; i++ )
{
    fscanf( fp, "%lf", &aval[i] );
    printf( "%8.3g ", aval[i] );
}
printf("\n");

for( i=0 ; i<na ; i++ )
{
    fscanf( fp, "%d", &jcn[i] );
}

for( i=0 ; i<m ; i++ )
{
    fscanf( fp, "%d", &ia[i] );
}

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

printf( "\n\n\tUpper bound of each variable xup\n" );
printf("\n\t");
for( i=0 ; i<n ; i++ )
{
    fscanf( fp, "%lf", &xup[i] );
    printf( "%8.3g ", xup[i] );
}
printf("\n");

printf( "\n\n\tLower bound of each variable xlow\n" );
```

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

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

    printf( "\n\n\tType of each constraint itype\n" );
    printf("\n\t");
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%d", &itype[i] );
        printf( "%6d ", itype[i] );
    }
    printf("\n");

    fclose( fp );

    ierr = ASL_dmclaf(aval, na, jcn, ia, nm, b, m, xup, xlow, c, itype,
                      er, ap, bm, nck, &nev, x, &y, &isw1, isw2, iw, w, nw);

    printf( "\n     ** Output **\n\n" );
    printf( "\n     ** First result **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\tSelected isw1 = %6d \n", isw1 );

    isw2 = 1;
    nev = 100;
    ierr = ASL_dmclaf(aval, na, jcn, ia, nm, b, m, xup, xlow, c, itype,
                      er, ap, bm, nck, &nev, x, &y, &isw1, isw2, iw, w, nw);

    printf( "\n     ** Output **\n\n" );
    printf( "\n     ** Improved result **\n\n" );
    printf( "\tierr = %6d\n\n", ierr );
    printf( "\tSelected isw1 = %6d \n", isw1 );
    printf( "\tIteration number = %6d\n\n", nev );
    printf( "\n\tSolution\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t   x[%6d] = %8.3g\n", i, x[i] );
    }
    printf( "\t   y = %8.3g\n", y );

    free( aval );
    free( ia );
    free( jcn );
    free( b );
    free( xup );
    free( xlow );
    free( c );
    free( itype );
    free( x );
    free( iw );
    free( w );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmclaf ***

 ** Input **

na   =      7 nm  =     11 m    =      3
nck  =      0 nev =      5 isw1 =      0
isw2 =      0 nw  =    200

er[0] =        0 er[1] =        0 er[2] =        0

ap  =        0 bm =        0


Matrix A

       1        2        3        2       -1        1        2


Constant vector b

      -7        0        8


Upper bound of each variable xup

       1        1        2        0        5


Lower bound of each variable xlow

       0        0        1       -3        2


Coefficient vector c

       2        1        1       -3        1


Type of each constraint itype

    -1      0      1

 ** Output **


 ** First result **

ierr =   5000

Selected isw1 =      1

 ** Output **


 ** Improved result **

ierr =   1500

Selected isw1 =      1
Iteration number =     20


Solution

   x[     0] = 2.19e-09
   x[     1] =      0.5
   x[     2] =        1
   x[     3] = -1.32e-10
   x[     4] =        2
   y =      3.5
```

5.6.3 ASL_dmclmz, ASL_rmclmz
0-1 変数を含む線形制約付き多変数線形関数の最小化 (混合 0–1 計画)

(1) **機 能**

制約条件

$$\begin{array}{rcl} \sum_{j=1}^{n} a_{i,j}x_j & = & b_i \qquad (i=1,\cdots,m_e; j=1,\cdots,n) \\ \sum_{j=1}^{n} a_{i,j}x_j & \leq & b_i \qquad (i=m_e+1,\cdots,m; j=1,\cdots,n) \\ d_j & \leq & x_j \leq u_j \quad (j=1,\cdots,n) \\ x_j & = & 0,1 \qquad (j \in N_{01}) \end{array}$$

のもとで目的関数

$$f(\boldsymbol{x}) = \sum_{j=1}^{n} c_{jx_j}$$

を最小にする $\boldsymbol{x} = (x_1, \ldots, x_n)$ と, それに対する目的関数 $f(\boldsymbol{x})$ の値を求める. N_{01} は 0-1 変数の添え字の集合である.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmclmz  (a, ma, n, b, m, me, xup, xlow, lz, ln, c, mp, np, er, nev, x, & y, isw, iwk,
                    wk);
```

単精度関数:

```
ierr = ASL_rmclmz  (a, ma, n, b, m, me, xup, xlow, lz, ln, c, mp, np, er, nev, x, & y, isw, iwk,
                    wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	a	{D∗ / R∗}	ma×n	入 力	制約条件の左辺の係数 $a_{i,j}$(注意事項 (a) 参照)
2	ma	I	1	入 力	配列 a の整合寸数
3	n	I	1	入 力	変数の数 n
4	b	{D∗ / R∗}	m	入 力	制約条件の右辺の定数 b_i を成分に持つ m 次元ベクトル
5	m	I	1	入 力	制約条件の数 m
6	me	I	1	入 力	等式制約条件の数 m_e
7	xup	I*	n	入 力	変数 x_j の上限値 u_j (注意事項 (b) 参照)
8	xlow	I*	n	入 力	変数 x_j の下限値 d_j (注意事項 (b) 参照)
9	lz	I*	ln	入 力	0-1 変数の添え字の集合 N_{01} (注意事項 (i) 参照)
10	ln	I	1	入 力	0-1 変数の個数 (注意事項 (h) 参照)
11	c	{D∗ / R∗}	n	入 力	目的関数の係数ベクトル c_j

項番	引数と戻り値	型	大きさ	入出力	内　　容
12	mp	I	1	入 力	分枝限定法における探索の深さ p $p=1$ のとき深さ優先探索 p が大きくなるにしたがって発見的探索に近づく (注意事項 (f) 参照)
13	np	I	1	入 力	分枝限定法における部分問題のリストの長さの最大値
14	er	{D R}	1	入 力	要求精度 (注意事項 (c) 参照)
15	nev	I	1	入 力	緩和問題を解くときの最大反復回数 (注意事項 (d) 参照)
16	x	{D∗ R∗}	n	出 力	最適解 $\boldsymbol{x}$
17	y	{D∗ R∗}	1	出 力	最適解 $\boldsymbol{x}$ に対する目的関数の値 $f(\boldsymbol{x})$
18	isw	I	1	入 力	処理スイッチ (注意事項 (g) および (h) 参照) isw=0:最初の処理 isw=1:継続処理
19	iwk	I*	内容参照	ワーク	作業領域 **大きさ**:$\mathrm{mp}+(\mathrm{mp}+3)\times(\mathrm{n}+\mathrm{m}-\mathrm{me})+\mathrm{m}+(\mathrm{ln}+2)\times\mathrm{np}+7$
20	wk	{D∗ R∗}	内容参照	ワーク	作業領域 **大きさ**: $\mathrm{m}\times4+(\mathrm{m}+4)\times(\mathrm{n}+2\times\mathrm{m}-\mathrm{me})+(\mathrm{mp}+1)\times(\mathrm{m}+1)\times(\mathrm{n}-\mathrm{me}+1)+(\mathrm{m}+1)\times(\mathrm{n}+\mathrm{m}-\mathrm{me}+1)+2\times\mathrm{ln}+(\mathrm{n}+\mathrm{m}-\mathrm{me}+2)\times(\mathrm{np}+1)+2$
21	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) $\mathrm{ma} \geq \mathrm{m}$

(c) $0 \leq \mathrm{me} \leq \mathrm{n}, \mathrm{m}$

(d) $\mathrm{n} \geq \mathrm{ln}$

(e) $\mathrm{np} \geq 2$

(f) $0 \leq \mathrm{lz}[0] < \mathrm{lz}[1] < \cdots < \mathrm{lz}[\mathrm{ln}-1] \leq \mathrm{n}$

(g) er > 0.0 (既定値を使用するため 0.0 または負値を入力する場合を除く)

(h) nev > 0 (既定値を使用するため 0 または負値を入力する場合を除く)

(i) isw=0 または isw=1

(j) $\mathrm{xup}[\mathrm{i}-1] \geq \mathrm{xlow}[\mathrm{i}-1] \quad (\mathrm{i}=1,2,\cdots,\mathrm{n})$

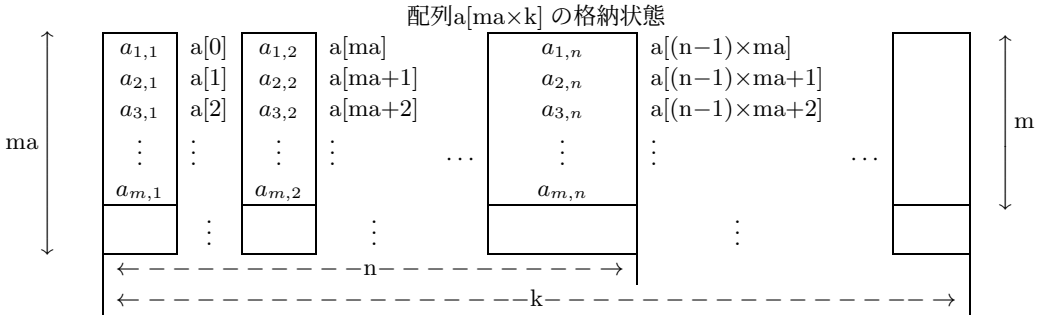
(k) ierr=5600 が出力された後に isw=1 を設定して継続処理を行うときには, np の値は前回の処理よりも大きく設定されていなければならない.

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (g) または (h) を満たさなかった.	既定値を使用して処理を続ける.
1100	制限条件 (i) を満たさなかった.	isw=0 と見なして処理を続ける.
1200	制限条件 (j) を満たさなかった.	上限値と下限値を入れ換えて処理を続ける.
3000	制限条件 (a) を満たさなかった.	処理を打ち切る.
3010	制限条件 (b) を満たさなかった.	
3020	制限条件 (c) を満たさなかった.	
3030	制限条件 (d) を満たさなかった.	
3040	制限条件 (e) を満たさなかった.	
3100	制限条件 (f) を満たさなかった.	
3500	制限条件 (k) を満たさなかった.	
4000	実行可能解が存在しない.	
5000	与えられた反復回数内で緩和問題の解が求められなかった. 暫定解は求められている.	その時点での暫定解を出力して処理を打ち切る.
5100	与えられた反復回数内で緩和問題の解が求められなかった. 暫定解は求められていない.	処理を打ち切る.
5500	分枝限定法の計算において, 終端されずにメモリ上に保持されている部分問題の個数が, np に達した. 暫定解は求められている.	その時点での暫定解を出力して処理を打ち切る.
5600	分枝限定法の計算において, 終端されずにメモリ上に保持されている部分問題の個数が, np に達した. 暫定解は求められていない.	処理を打ち切る.

(6) **注意事項**

(a) 制約条件の左辺の係数 $a_{i,j}$ は配列 a に以下のように格納されている.



備　考
a.　ma ≥ m, k ≥ n を満たさなければならない.

(b) x_j が 0-1 変数である場合については, xup $[j-1]$ と xlow $[j-1]$ の値は設定しておかなくてもよい.

(c) 引数 er に正でない値が入力された場合, 既定値の $2 \times \sqrt{\text{誤差判定のための単位}}$ が要求精度として用いられる.

(d) 混合 0-1 問題の 0-1 変数の一部を 0 または 1 に固定した問題を部分問題と呼ぶ. 部分問題の固定されていない 0-1 変数についての条件を弱めてそれらが 0 以上 1 以下の実数値を取り得るようにしてできる線形計画問題をその部分問題の緩和問題と呼ぶ. 混合 0-1 計画問題の解を探索する過程で, 生成される部分問題

の緩和問題をシンプレックス法で解く際に, 引数 nev の値が反復回数の上限として用いられる. 引数 nev に正でない値が入力された場合, 既定値の $10 \times$ m が反復回数の上限として用いられる.

(e) 分枝限定法による最適解の探索の途中で, その時点までに得られている制約条件を満たす解の中で最も目的関数の値を小さくする解をその時点での暫定解と呼ぶ. ierr=5000 または 5500 が返されたときには, 処理が打ち切られた時点での暫定解が x に出力される. このときには, 得られた解は制約条件は満たすが, 真の最適解とは必ずしも一致しない.

(f) 探索の深さ mp は分岐限定法における解の探索方法を設定するために利用者があたえるパラメータである. mp を大きくするほど, 早い時点で良い暫定解が得られる傾向が強い. しかし mp が大きくなるほど, 終端されずにメモリ上に保持される部分問題の個数は急激に増える傾向にあるため, np の値を十分大きく設定しておく必要がある. mp=1 のとき, 計算に必要な領域は最小となる. np の大きさが十分でないために ierr=5500 または 5600 が返されることを防ぐためには, mp の値を 1 に, np の値を ln+1 に設定すれば良い.

(g) ierr=5000 または 5100 を出力して処理が打ち切られた場合に, isw=1 を設定し, nev をより大きい値に設定し直して, 前回の計算結果を利用して計算を継続することができる. この場合, nev 以外の引数については前回の処理で使用した引数の内容を保存しておく必要がある.

(h) ierr=5500 または 5600 を出力して処理が打ち切られた場合に, isw=1 を設定し, np をより大きい値に設定し直して, 前回の計算結果を利用して計算を継続することができる. この場合, np, iwk および wk 以外の引数については前回の処理で使用した引数の内容を保存しておく必要がある. さらに作業用の配列である iwk と wk については, np の値に合わせて, 適切な大きさの領域を確保し, その先頭の領域に前回の処理の終了時の iwk と wk の内容を格納しておく必要がある.

(i) 配列 lz には変数 $x_1, x_2, \cdots, x_n$ のうち 0-1 変数である変数の添字を昇順に格納し, 変数 ln に 0-1 変数の個数を格納する. 例えば, 変数が $x_1, x_2, \cdots, x_8$ の 8 個あり, そのうち x_2, x_5, x_8 が 0-1 変数である混合 0-1 計画問題を扱う場合には以下のように設定する.

ln = 3;
lz[0] = 2;
lz[1] = 5;
lz[2] = 8;

(7) 使用例

(a) 問 題

制約条件

$$
\begin{aligned}
x_2 + x_3 + x_{10} &= 1 \\
6x_1 + 28x_2 + 6x_3 + 6x_4 + 3x_5 + 6x_6 & \\
+21x_7 + 8x_8 - 18x_9 + 12x_{10} + 20x_{11} + 23x_{12} &\leq 60 \\
7x_1 + 7x_2 + 5x_3 + 2x_4 + 2x_5 + 5x_6 & \\
+4x_7 + 2x_8 - 3x_9 + 3x_{10} + 8x_{11} + 3x_{12} &\leq 20 \\
-x_4 - x_{11} &\leq -1 \\
-x_6 - x_7 - x_{12} &\leq -1
\end{aligned}
$$

$$
0 \leq x_1 \leq 2\ ,\ 0 \leq x_5 \leq 3\ ,\ 0 \leq x_8 \leq 1\ ,\ -2 \leq x_9 \leq 0
$$

$$
x_j = 0, 1 \qquad (j = 2, 3, 4, 6, 7, 10, 11, 12)
$$

の下で目的関数

$$
\begin{aligned}
f(\boldsymbol{x}) \;=\; & -13x_1 - 18x_2 - 10x_3 - 8x_4 - 8x_5 - 12x_6 \\
& -10x_7 - 8x_8 + 11x_9 - 9x_{10} - 30x_{11} - 10x_{12}
\end{aligned}
$$

を最小にする.

(b) 入力データ

ma= 6, n= 12, m= 5, me= 1, mp= 4, np= 50, er= 0.0 (既定値にセット), nev= 0 (既定値にセット), isw= 0, 制約条件の係数を格納する配列 a と b, 目的関数の係数を格納する配列 c, 0-1 変数の添え字を格納する配列 lz

(c) 主プログラム

```
/*      C interface example for ASL_dmclmz */

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

int main()
{
    double *a;
    int ma;
    int n;
    double *b;
    int m;
    int me;
    double *xup;
    double *xlow;
    int *lz;
    int ln;
    double *c;
    int mp;
    int np;
    double er;
    int nev;
    double *x;
    double y;
    int isw;
    int *iw1;
    double *w1;
    int ierr;
    int i,j,flag,niw1,nw1;
    FILE *fp;

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

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

    fscanf( fp, "%d", &ma );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &me );
    fscanf( fp, "%d", &ln );
    fscanf( fp, "%d", &mp );
    fscanf( fp, "%d", &np );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );
    fscanf( fp, "%lf", &er );

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

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

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

    lz = ( int * )malloc((size_t)( sizeof(int) * ln ));
    if( lz == NULL )
    {
```

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

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

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

niw1 = mp+(mp+3)*(n+m-me)+m+(ln+2)*np+7;
iw1 = ( int * )malloc((size_t)( sizeof(int) * niw1 ));
if( iw1 == NULL )
{
    printf( "no enough memory for array iw1\n" );
    return -1;
}

nw1 = m*4+(m+4)*(n+2*m-me)+(mp+1)*(m+1)*(n-me+1)+(m+1)*(n+m-me+1)
    +2*ln+(n+m-me+2)*(np+1)+2;
w1 = ( double * )malloc((size_t)( sizeof(double) * nw1 ));
if( w1 == NULL )
{
    printf( "no enough memory for array w1\n" );
    return -1;
}

printf( "\tma = %6d n   = %6d m   = %6d\n", ma, n, m );
printf( "\tme = %6d ln  = %6d mp  = %6d\n", me, ln, mp );
printf( "\tnp = %6d nev = %6d isw = %6d\n", np, nev, isw );
printf( "\ter = %8.3g\n", er );

printf( "\n\t Matrix a\n\n");
for( i=0 ; i<m ; i++ )
{
    for( j=0 ; j<n ; j++ )
    {
        fscanf( fp, "%lf", &a[i+ma*j] );
        printf( "%7.2g", a[i+ma*j] );
    }
    fscanf( fp, "%lf", &b[i] );
    printf( "\t%7.2g", b[i] );
    printf( "\n" );
}

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

printf( "\n\tIndeces of 0-1 variables lz\n\n");
for( j=0 ; j<ln ; j++ )
{
    fscanf( fp, "%d", &lz[j] );
    printf( "\tlz[%6d] = %6d\n", j,lz[j] );
}

printf( "\n\tUpper and lower bounds\n\n");
for( j=0 ; j<n ; j++ )
{
    flag = 0;
    for( i=0 ; i<ln ; i++){
        if( lz[i]==j+1 ){
            flag = 1;
            continue;
        }
    }
    if(flag==1){
        continue;
    }
    fscanf( fp, "%lf %lf", &xup[j],&xlow[j] );
    printf( "\txup[%6d] = %8.3g xlow[%6d] = %8.3g\n", j,xup[j],j,xlow[j]);
}

fclose( fp );

ierr = ASL_dmclmz(a, ma, n, b, m, me, xup, xlow, lz, ln, c, mp, np, er, nev, x, &y, isw, iw1, w1);

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

printf( "\n\tVector x\n\n" );
for( i=0 ; i<n ; i++ )
{
    printf( "\t%8.3g\n", x[i] );
```

```
    }
    printf( "\n\ty = %8.3g\n", y );

    free( a );
    free( b );
    free( xup );
    free( xlow );
    free( lz );
    free( c );
    free( x );
    free( iw1 );
    free( w1 );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmclmz ***

 ** Input **

ma =      6 n   =     12 m   =      5
me =      1 ln  =      8 mp  =      4
np =     10 nev =     20 isw =      0
er =    1e-12

 Matrix a

  0      1      1      0      0      0      0      0      0      1      0      0          1
  6     28      6      6      3      6     21      8    -18     12     20     23         60
  7      7      5      2      2      5      4      2     -3      3      8      3         20
  0      0      0     -1      0      0      0      0      0      0     -1      0         -1
  0      0      0      0      0     -1     -1      0      0      0      0     -1         -1

 Vector c

     -13
     -18
     -10
      -8
      -8
     -12
     -10
      -8
      11
      -9
     -30
     -10

Indeces of 0-1 variables lz

lz[     0] =      2
lz[     1] =      3
lz[     2] =      4
lz[     3] =      6
lz[     4] =      7
lz[     5] =     10
lz[     6] =     11
lz[     7] =     12

Upper and lower bounds

xup[     0] =        2 xlow[     0] =        0
xup[     4] =        3 xlow[     4] =        0
xup[     7] =        1 xlow[     7] =        0
xup[     8] =        0 xlow[     8] =       -2

 ** Output **

ierr =      0

Vector x

       0
       0
       0
       1
       3
       1
       0
       1
  -0.667
       1
       0
       0

y =    -68.3
```

5.6.4 ASL_dmclmc, ASL_rmclmc
ネットワーク上の流れに対する費用の最小化 (最小費用流問題)

(1) **機 能**

n 個の節点と m 本の枝をもつネットワーク上で, 節点 i の流入出量 b_i および有向枝 k の非負の容量 u_k の制約を満たし, かつ全枝の費用の和を最小にする非負の流れ x_k $(k = 1, 2, \cdots, m)$ とそのときの最小値 $\sum_{k=1}^{m} c_k x_k$ を求める.

$$
\begin{aligned}
\text{目的関数} \quad &: \quad \sum_{k=1}^{m} c_k x_k \rightarrow \text{最小} \\
\text{制約条件} \quad &: \quad \sum_{tail(k)=i} x_k - \sum_{head(k)=i} x_k = b_i, \quad (i = 1, \cdots, n) \\
&\qquad 0 \leq x_k \leq u_k, \quad (k = 1, \cdots, m) \\
&\qquad \sum_{i=1}^{n} b_i = 0
\end{aligned}
$$

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmclmc  (n, m, b, itl, ihd, cap, cost, & nev, x, & y, isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rmclmc  (n, m, b, itl, ihd, cap, cost, & nev, x, & y, isw, iwk, wk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	節点の数 n
2	m	I	1	入 力	枝の数 m
3	b	{D∗ / R∗}	n	入 力	節点 i の流入出量 b_i
4	itl	I*	m+n	入 力	枝 k の始点の節点番号 $tail(k)$(注意事項 (a) 参照)
				出 力	グラフ変形後の始点の節点番号
5	ihd	I*	m+n	入 力	枝 k の終点の節点番号 $head(k)$(注意事項 (a) 参照)
				出 力	グラフ変形後の終点の節点番号
6	cap	{D∗ / R∗}	m+n	入 力	枝 k の容量 u_k (注意事項 (a), (b) 参照)
				出 力	グラフ変形後の容量
7	cost	{D∗ / R∗}	m+n	入 力	枝 k の単位流量あたりの費用係数 c_k (注意事項 (a) 参照)
				出 力	グラフ変形後の費用係数
8	nev	I*	1	入 力	基底木の最大更新回数 (既定値: n× 10) (注意事項 (c) 参照)
				出 力	実際の基底木の更新回数
9	x	{D∗ / R∗}	m	出 力	全枝の費用の和を最小にする流れ x_k
10	y	{D∗ / R∗}	1	出 力	最終到達点 x での費用の和 $\sum_{k=1}^{m} c_k x_k$
11	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) isw=0:最初の処理 isw=1:継続の処理
12	iwk	I*	内容参照	ワーク	作業領域 **大きさ**:$n^2 + 11 \times n + m + 3$
13	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ:** $2 \times n + m + 1$
14	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} \geq 2$

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

(c) $\mathrm{b}[0] + \mathrm{b}[1] + \cdots + \mathrm{b}[\mathrm{n}-1] = 0.0$

(d) $1 \leq \mathrm{itl}[k-1] \leq \mathrm{n} \quad (k = 1, \cdots, \mathrm{m})$

(e) $1 \leq \mathrm{ihd}[k-1] \leq \mathrm{n} \quad (k = 1, \cdots, \mathrm{m})$

(f) $\mathrm{itl}[k-1] \neq \mathrm{ihd}[k-1] \quad (k = 1, \cdots, \mathrm{m})$ (ループ (始点と終点が同じ枝) は存在しない)

(g) $\mathrm{cap}[k-1] \geq 0.0 \quad (k = 1, \cdots, \mathrm{m})$

(h) $\mathrm{nev} > 0$ (既定値にするため 0 または負値を入力する場合を除く)

(i) $\mathrm{isw} = 0$ または $\mathrm{isw} = 1$

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (h) を満足しなかった.	既定値にセットして処理する.
1100	制限条件 (i) を満足しなかった.	isw=0 にセットして処理する.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3100	制限条件 (c) を満足しなかった.	
3200	制限条件 (d) または (e) を満足しなかった.	
3300	制限条件 (f) を満足しなかった.	
3400	制限条件 (g) を満足しなかった.	
4000	実行可能基底解を求められなかった.	
5000	与えられた基底木の最大更新回数に達しても問題が解けなかった	

(6) **注意事項**

(a) itl, ihd, cap および cost には最初の m 個のみ値を入力すればよい. 後の n 個は作業領域として使用される.

(b) 容量が特に定められていない問題を解く場合は, 適当に大きな正数を容量として入力すればよい. なお, この時, 最終到達点での x の値が定めた容量と一致する場合は, 最適解が求められていない可能性があるのでさらに大きな値を容量として入力したほうがよい.

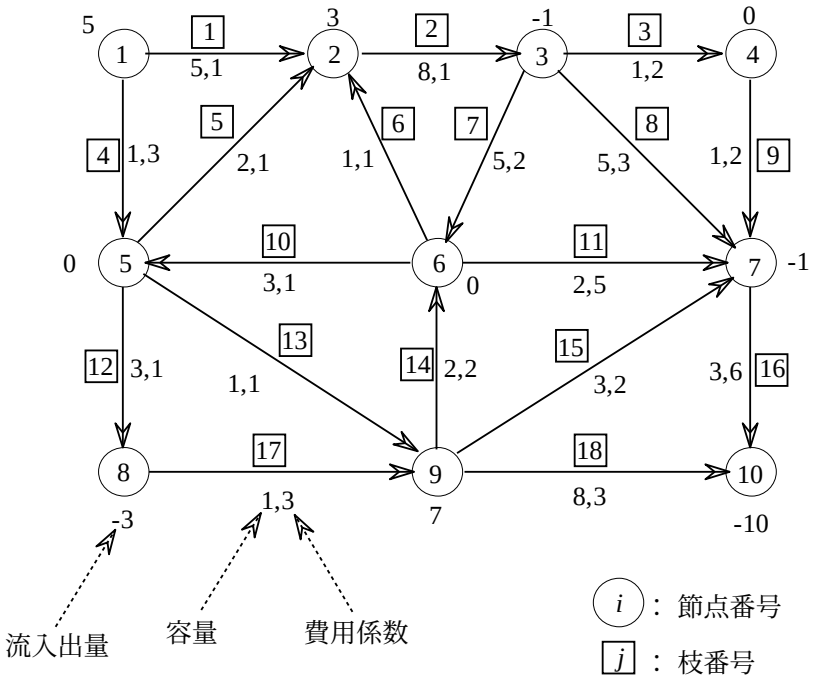
(c) nev に 0 または負の値を入力すれば既定値がセットされる.

(d) 指定した基底木の最大更新回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて引続き計算することができる. この処理を行うときには isw の値を 1, nev の値を十分に大きな値とし, それ以外の入力値には前回の値をそのまま用いる. また, 前回のワークの情報も用いる.

(7) **使用例**

(a) 問 題

以下のようなネットワーク上で各節点の流入出量および各枝の容量の制約を満たしかつ全枝の費用の和を最小にする流れを求める.



(b) 入力データ

n= 10, m= 18, 流入出量を格納する配列 b, 容量を格納する配列 cap, 枝の節点番号を格納する配列 itl と ihd, 費用係数を格納する配列 cost, nev =0 (既定値セット), isw =0

(c) 主プログラム

```
/*      C interface example for ASL_dmclmc */

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

int main()
{
    int n;
    int m;
    double *b;
    int *itail;
    int *ihead;
    double *cap;
    double *cost;
    int nev;
    double *x;
    double y;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i,nmax,mmax;
    FILE *fp;


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

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

    fscanf( fp, "%d", &nmax );
    fscanf( fp, "%d", &mmax );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw);

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

```
    {
        printf( "no enough memory for array itail\n" );
        return -1;
    }
    ihead = ( int * )malloc((size_t)( sizeof(int) * (mmax + nmax) ));
    if( ihead == NULL )
    {
        printf( "no enough memory for array ihead\n" );
        return -1;
    }
    cap = ( double * )malloc((size_t)( sizeof(double) * (mmax + nmax)));
    if( cap == NULL )
    {
        printf( "no enough memory for array cap\n" );
        return -1;
    }
    cost = ( double * )malloc((size_t)( sizeof(double) * (mmax + nmax)));
    if( cost == NULL )
    {
        printf( "no enough memory for array cost\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * mmax));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * (nmax*(nmax+11)+mmax+3) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * (2*nmax+mmax+1)));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

    printf( "\n\t       b\n" );
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &b[i] );
        printf( "\t%8.3g\n", b[i] );
    }
    printf( "\n\t itail  ihead      cap     cost\n" );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%d %d %lf %lf", &itail[i],&ihead[i],&cap[i],&cost[i] );
        printf( "\t%6d %6d %8.3g %8.3g\n", itail[i],ihead[i],cap[i],cost[i] );
    }

    fclose( fp );

    ierr = ASL_dmclmc(n, m, b, itail, ihead, cap, cost, &nev, x, &y, isw, iwk, wk);

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

    printf( "\ty = %8.3g\n", y );
    printf( "\n\tVector x\n" );
    for( i=0 ; i<m ; i++ )
    {
        printf( "\t%8.3g\n", x[i] );
    }

    free( b );
    free( itail );
    free( ihead );
    free( cap );
    free( cost );
    free( x );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmclmc ***

 ** Input **

n =     10 m =     18 nev =     0 isw =      0

       b
       5
       3
      -1
       0
       0
```

```
        0
       -1
       -3
        7
      -10

 itail  ihead       cap      cost
     1      2         5         1
     2      3         8         1
     3      4         1         2
     1      5         1         3
     5      2         2         1
     6      2         1         1
     3      6         5         2
     3      7         5         3
     4      7         1         2
     6      5         3         1
     6      7         2         5
     5      8         3         1
     5      9         1         1
     9      6         2         2
     9      7         3         2
     7     10         3         6
     8      9         1         3
     9     10         8         3

 ** Output **

ierr =   1000
nev =      13
y =        72

Vector x
        4
        7
        0
        1
        0
        0
        3
        3
        0
        3
        0
        3
        1
        0
        0
        2
        0
        8
```

5.6.5 ASL_dmclcp, ASL_rmclcp
プロジェクトの日程計画に対する費用の最小化 (日程計画問題)

(1) **機 能**

予定完了時刻までにプロジェクトを最小費用で完了する問題を解く.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmclcp  (n, m, itl, ihd, tn, tc, cn, cc, ts, & nev, & tmax, & tmin, time, & ttime, es,
                    ls, tf, ff, & cmax, & cmin, cost, & tcost, iwk, wk);
```

単精度関数:

```
ierr = ASL_rmclcp  (n, m, itl, ihd, tn, tc, cn, cc, ts, & nev, & tmax, & tmin, time, & ttime, es,
                    ls, tf, ff, & cmax, & cmin, cost, & tcost, iwk, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	節点の数
2	m	I	1	入 力	作業の数
3	itl	I*	m	入 力	作業 k の始点の節点番号
4	ihd	I*	m	入 力	作業 k の終点の節点番号
5	tn	{D*, R*}	m	入 力	作業 k の通常作業時間
6	tc	{D*, R*}	m	入 力	作業 k の特急作業時間
7	cn	{D*, R*}	m	入 力	作業 k の通常作業費用
8	cc	{D*, R*}	m	入 力	作業 k の特急作業費用
9	ts	{D, R}	1	入 力	予定完了時刻
10	nev	I*	1	入 力	最小費用流問題の基底木の更新回数 (既定値:n×10)
				出 力	実際の基底木の更新回数
11	tmax	{D*, R*}	1	出 力	通常作業時間によるプロジェクトの完了時刻
12	tmin	{D*, R*}	1	出 力	特急作業時間によるプロジェクトの完了時刻

項番	引数と戻り値	型	大きさ	入出力	内　　容
13	time	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の作業時間
14	ttime	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	1	出 力	総作業時間
15	es	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の最早開始時刻
16	ls	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の最遅開始時刻
17	tf	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の総余裕時間
18	ff	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の自由余裕時間
19	cmax	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	1	出 力	通常作業時間によるプロジェクトの総作業費用
20	cmin	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	1	出 力	特急作業時間によるプロジェクトの総作業費用
21	cost	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	m	出 力	作業 k の作業費用
22	tcost	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	1	出 力	総作業費用
23	iwk	I*	内容参照	ワーク	作業領域 **大きさ**:$\mathrm{n}^2 + 15 \times \mathrm{n} + 6 \times \mathrm{m} + 23$
24	wk	$\left\{\begin{matrix}\mathrm{D*}\\ \mathrm{R*}\end{matrix}\right\}$	内容参照	ワーク	作業領域 **大きさ**:$7 \times \mathrm{n} + 12 \times \mathrm{m} + 14$
25	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) $\mathrm{n} > 1$

(b) $\mathrm{m} \geq \mathrm{n} - 1$

(c) $1 \leq \mathrm{itl}[k-1] < \mathrm{n} \quad (k = 1, \cdots, \mathrm{m})$

(d) $1 < \mathrm{ihd}[k-1] \leq \mathrm{n} \quad (k = 1, \cdots, \mathrm{m})$

(e) $\mathrm{itl}[k-1] < \mathrm{ihd}[k-1] \quad (k = 1, \cdots, \mathrm{m})$

(f) $\mathrm{tn}[k-1] \geq 0.0 \quad (k = 1, \cdots, \mathrm{m})$

(g) $\mathrm{tc}[k-1] \geq 0.0 \quad (k = 1, \cdots, \mathrm{m})$

(h) $\mathrm{tn}[k-1] \geq \mathrm{tc}[k-1] \quad (k = 1, \cdots, \mathrm{m})$

(i) $\mathrm{cn}[k-1] \leq \mathrm{cc}[k-1] \quad (k = 1, \cdots, \mathrm{m})$

(j) $\mathrm{nev} > 0$ (既定値にするため 0 を入力する場合を除く)

(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	nev=0 であった.	既定値にセットして処理する.
1100	ts≥tmax であった.	ts=tmax として処理を続ける.
1200	ts≤tmin であった.	ts=tmin として処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3400	制限条件 (e) を満足しなかった.	
3500	制限条件 (f) を満足しなかった.	
3600	制限条件 (g) を満足しなかった.	
3700	制限条件 (h) を満足しなかった.	
3800	制限条件 (i) を満足しなかった.	
3900	制限条件 (j) を満足しなかった.	
4000	最小費用流問題において実行可能基底解を求められなかった.	
5000	最小費用流問題において与えられた基底木の最大更新回数に達しても問題が解けなかった.	

(6) **注意事項**

(a) 基底木の更新回数 nev に 0 の値を入力すれば既定値がセットされる.

(b) 予定完了時刻 ts に通常作業時間による完了時刻 tmax より大きい値を入力した場合, 通常作業時間による完了時刻を予定完了時刻として扱う. また, 予定完了時刻 ts に特急作業時間による完了時刻 tmin より小さい値を入力した場合, 特急作業時間による完了時刻を予定完了時刻として扱う.

(7) 使用例

(a) 問 題

以下のような作業リストで表されるプロジェクトの通常作業時間および特急作業時間による完了時刻, 最適日程の各作業の時間, 費用, 最早開始時刻, 最遅開始時刻, 総余裕時間および自由余裕時間, 総作業時間および総作業費用を求める.

作業	始点の節点	終点の節点	通常作業時間	特急作業時間	通常作業費用	特急作業費用
1	1	2	5	3	3	6
2	1	3	8	4	2	3
3	2	4	5	2	1	5
4	2	5	9	4	1	3
5	4	5	0	0	0	0
6	2	6	3	3	4	4
7	3	6	7	3	2	3
8	5	7	5	1	1	5
9	6	7	0	0	0	0
10	6	8	12	6	7	11
11	7	8	7	4	2	7
12	5	9	15	8	5	10
13	7	9	8	2	5	11
14	8	9	2	1	3	4
15	8	10	8	5	2	5
16	9	10	3	2	1	3
17	8	11	13	10	10	12
18	10	11	8	4	6	10

(b) 入力データ

n=11,m=18, 始点の節点 itl, 終点の節点 ihd, 通常作業時間 tn, 特急作業時間 tc, 通常作業費用 cn, 特急作業費用 cc, ts=36,nev=n×10.

(c) 主プログラム

```
/*  C interface example for ASL_dmclcp */

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

int main()
{
    int nmax=11,mmax=18;
    int n,m,*itail,*ihead,nev,*iwk,ierr;
    double *tn,*tc,*cn,*cc,ts;
    double tmin,tmax,*time,ttime;
    double *es,*ls,*tf,*ff;
    double cmin,cmax,*cost,tcost,*wk;
    int k;
    FILE *fp;

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

    itail = ( int * )malloc((size_t)( sizeof(int) * (mmax)));
    if( itail == NULL )
```

```
{
    printf( "no enough memory for array itail\n" );
    return -1;
}
ihead = ( int * )malloc((size_t)( sizeof(int) * (mmax)));
if( ihead == NULL )
{
    printf( "no enough memory for array ihead\n" );
    return -1;
}
iwk = ( int * )malloc((size_t)( sizeof(int) * (nmax*nmax+15*nmax+6*mmax+23)));
if( iwk == NULL )
{
    printf( "no enough memory for array iwk\n" );
    return -1;
}
tn = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( tn == NULL )
{
    printf( "no enough memory for array tn\n" );
    return -1;
}
tc = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( tc == NULL )
{
    printf( "no enough memory for array tc\n" );
    return -1;
}
cn = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( cn == NULL )
{
    printf( "no enough memory for array cn\n" );
    return -1;
}
cc = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( cc == NULL )
{
    printf( "no enough memory for array cc\n" );
    return -1;
}
time = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( time == NULL )
{
    printf( "no enough memory for array time\n" );
    return -1;
}
es = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( es == NULL )
{
    printf( "no enough memory for array es\n" );
    return -1;
}
ls = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( ls == NULL )
{
    printf( "no enough memory for array ls\n" );
    return -1;
}
tf = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( tf == NULL )
{
    printf( "no enough memory for array tf\n" );
    return -1;
}
ff = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( ff == NULL )
{
    printf( "no enough memory for array ff\n" );
    return -1;
}
cost = ( double * )malloc((size_t)( sizeof(double) * (mmax)));
if( cost == NULL )
{
    printf( "no enough memory for array cost\n" );
    return -1;
}
wk = ( double * )malloc((size_t)( sizeof(double) * (7*nmax+12*mmax+14)));
if( wk == NULL )
{
    printf( "no enough memory for array wk\n" );
    return -1;
}

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

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

printf( "\tn = %6d    m = %6d    nev = %6d\n", n,m,nev );
printf( "\n\n" );
printf( "\t                    normal    crash     normal    crash\n" );
printf( "\ttask   tail   head task time task time task cost task cost\n" );

for( k=0 ; k<m ; k++ )
{
    fscanf( fp, "%d %d %lf %lf %lf %lf",
```

```
            &itail[k],&ihead[k],&tn[k],&tc[k],&cn[k],&cc[k] );
        printf( "\t%4d %6d %6d  %8.3g  %8.3g  %8.3g  %8.3g\n",
            k,itail[k],ihead[k],tn[k],tc[k],cn[k],cc[k] );
    }
    printf( "\n\trequest task time = %8.3g\n", ts );

    fclose( fp );

    ierr = ASL_dmclcp
    (n,m,itail,ihead,tn,tc,cn,cc,ts,&nev,&tmax,&tmin,time,&ttime,es,ls,tf,ff,&cmax,&cmin,cost,&tcost,iwk,wk);

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

    printf( "\n" );
    printf( "\tnormal completion time = %8.3g\n", tmax );
    printf( "\tcrash  completion time = %8.3g\n", tmin );
    printf( "\n\n" );
    printf( "\t                              earliest    latest\n" );
    printf( "\t          task      task     start     start     total      free\n" );
    printf( "\ttask      time      cost      time      time     float     float\n" );
    for( k=0 ; k<m ; k++ )
    {
        printf( "\t%4d  %8.3g  %8.3g  %8.3g  %8.3g  %8.3g  %8.3g\n",
             k,time[k],cost[k],es[k],ls[k],tf[k],ff[k] );
    }
    printf( "\n" );
    printf( "\titeration count = %6d\n", nev );
    printf( "\n" );
    printf( "\ttotal task time = %8.3g\n", ttime );
    printf( "\ttotal task cost = %8.3g\n", tcost );

    free( itail );
    free( ihead );
    free( iwk );
    free( tn );
    free( tc );
    free( cn );
    free( cc );
    free( time );
    free( es );
    free( ls );
    free( tf );
    free( ff );
    free( cost );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmclcp ***

 ** Input **

n =     11    m =     18    nev =    110


                   normal    crash     normal    crash
task   tail   head task time task time task cost task cost
   0      1      2         5         3         3         6
   1      1      3         8         4         2         3
   2      2      4         5         2         1         5
   3      2      5         9         4         1         3
   4      4      5         0         0         0         0
   5      2      6         3         3         4         4
   6      3      6         7         3         2         3
   7      5      7         5         1         1         5
   8      6      7         0         0         0         0
   9      6      8        12         6         7        11
  10      7      8         7         4         2         7
  11      5      9        15         8         5        10
  12      7      9         8         2         5        11
  13      8      9         2         1         3         4
  14      8     10         8         5         2         5
  15      9     10         3         2         1         3
  16      8     11        13        10        10        12
  17     10     11         8         4         6        10

request task time =       36

 ** Output **

ierr =      0

normal completion time =       43
crash  completion time =       23


                          earliest    latest
          task      task     start     start     total      free
task      time      cost      time      time     float     float
   0         5         3         0         0         0         0
   1         7      2.25         0         0         0         0
   2         5         1         5         5         0         0
```

```
  3         5       2.6         5         5         0         0
  4         0         0        10        10         0         0
  5         3         4         5         7         2         2
  6         3         3         7         7         0         0
  7         5         1        10        10         0         0
  8         0         0        10        15         5         5
  9        12         7        10        10         0         0
 10         7         2        15        15         0         0
 11        15         5        10        10         0         0
 12         8         5        15        17         2         2
 13         2         3        22        23         1         1
 14         6         4        22        22         0         0
 15         3         1        25        25         0         0
 16        13        10        22        23         1         1
 17         8         6        28        28         0         0

iteration count =     19

total task time =       36
total task cost =     59.9
```

5.6.6 ASL_dmcltp, ASL_rmcltp
供給地から需要地への輸送費用の最小化 (輸送問題)

(1) **機 能**

供給地 i $(i = 1, \cdots, m)$ での供給量を a_i, 需要地 j $(j = 1, \cdots, n)$ での需要量を b_j, 供給地 i から需要地 j への輸送量を x_{ij} とすると, 制約条件

$$\begin{aligned}
\sum_{j=1}^{n} x_{ij} &= a_i \quad (i = 1, \cdots, m) \\
\sum_{i=1}^{m} x_{ij} &= b_i \quad (j = 1, \cdots, n) \\
x_{ij} &\geq 0 \quad (\text{すべての } i, j \text{ に対して})
\end{aligned}$$

に従って総輸送費用

$$Z = \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij}$$

を最小にする x_{ij} を求める.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmcltp  (c, lmc, ns, nd, s, d, & nev, x, & z, isw1, isw2, iwk, wk);
```

単精度関数:

```
ierr = ASL_rmcltp  (c, lmc, ns, nd, s, d, & nev, x, & z, isw1, isw2, iwk, wk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	c	{D∗ R∗}	内容参照	入 力	輸送費用 $c_{ij}(i=1,\cdots,m:j=1,\cdots,n)$ (注意事項 (a) 参照) **大きさ:** lmc×(nd + 1)
2	lmc	I	1	入 力	配列 c の整合寸法
3	ns	I	1	入 力	架空の供給地を含めた供給地の数 m (注意事項 (b) 参照)
4	nd	I	1	入 力	架空の需要地を含めた需要地の数 n (注意事項 (b) 参照)
5	s	{D∗ R∗}	ns+1	入 力	各供給地の供給量 $a_i(i=1,\cdots,m)$
6	d	{D∗ R∗}	nd+1	入 力	各需要地の需要量 $b_j(j=1,\cdots,n)$
7	nev	I*	1	入 力	輸送計画 x_{ij} の最大評価回数
				出 力	実際の評価回数
8	x	{D∗ R∗}	内容参照	出 力	改善された輸送計画 $x_{ij}(i=1,\cdots,m:j=1,\cdots,n)$ (注意事項 (c),(d) 参照) **大きさ:** lmc×(nd + 1)
9	z	{D∗ R∗}	1	出 力	改善された輸送計画の総輸送費用 Z
10	isw1	I	1	入 力	処理スイッチ (注意事項 (e) 参照) 0: 最初の処理 1: 継続の処理
11	isw2	I	1	入 力	第一次近似解法選択スイッチ (注意事項 (f) 参照) 0: 北西隅の規則 1: ハウサッカー法
12	iwk	I*	内容参照	ワーク	ワーク配列 **大きさ:** 12 × (ns + nd + 1) + 5
13	wk	{D∗ R∗}	内容参照	ワーク	ワーク配列 **大きさ:** 3 × (ns + nd + 2)
14	ierr	I	1	出 力	エラーインディケータ

(4) **制限条件**

(a) $\mathrm{c}[i + \mathrm{lmc} * j] \geq 0$ $(i = 1, \cdots, \mathrm{ns} : j = 1, \cdots, \mathrm{nd})$

(b) $\mathrm{s}[i-1] > 0$ ($i = 1, \cdots, \mathrm{ns}$)

(c) $\mathrm{d}[i-1] > 0$ ($i = 1, \cdots, \mathrm{nd}$)

(d) $\mathrm{isw1}, \mathrm{isw2} = 0$ または 1

(e) $\mathrm{ns} > 2$, $\mathrm{nd} > 2$

(f) $\mathrm{lmc} \geq \mathrm{ns} + 1$

(g) $\mathrm{nev} \geq 0$

(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	総需要量が総供給量よりも多い.	余分を扱う架空の供給地を設定し処理を続ける.
1100	総需要量が総供給量よりも少ない.	余分を扱う架空の需要地を設定し処理を続ける.
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
3100	制限条件 (b) を満足しなかった.	
3200	制限条件 (c) を満足しなかった.	
3300	制限条件 (d) を満足しなかった.	
3400	制限条件 (e) を満足しなかった.	
3500	制限条件 (f) を満足しなかった.	
3600	制限条件 (g) を満足しなかった.	
5000	与えられた最大評価回数に達しても収束しなかった.	その時点の x, z の値を出力して, 処理を打ち切る.

(6) **注意事項**

(a) 輸送費用は, 需要地と供給地に対応するよう以下の表の様に配列 c に格納する. 以下の表において, 例えば $c_{2,1}$ とは供給地 2 から 需要地 1 への単位輸送費用である.

	需要地 1	需要地 2	$\cdots$	需要地 n	供給量
供給地 1	c_{11}	c_{12}	$\cdots$	$c_{1,n}$	a_1
供給地 2	c_{21}	c_{22}	$\cdots$	$c_{2,n}$	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
供給地 m	$c_{m,1}$	$c_{m,2}$	$\cdots$	$c_{m,n}$	a_m
需要量	b_1	b_2	$\cdots$	b_n	

(b) 配列 c, x は $(\mathrm{nd}+1) \times (\mathrm{ns}+1)$ 個以上, 配列 s, d はそれぞれ, $(\mathrm{ns}+1)$ 個, $(\mathrm{nd}+1)$ 個以上確保しなければならない. 但し, 実際に入力する値は, 配列 c は $\mathrm{nd} \times \mathrm{ns}$ 個, 配列 s, d はそれぞれ ns 個, nd 個でよい.

(c) 改善された輸送計画は, 需要地と供給地に対応するよう以下の表のように配列 x に格納される. 以下の表において, 例えば $x_{1,2}$ とは供給地 1 から 需要地 2 への輸送量である.

	需要地 1	需要地 2	$\cdots$	需要地 n	(架空の需要地 $n+1$)
供給地 1	$x_{1,1}$	$x_{1,2}$	$\cdots$	$x_{1,n}$	$*$
供給地 2	$x_{2,1}$	$x_{2,2}$	$\cdots$	$x_{2,n}$	$*$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$*$
供給地 m	$x_{m,1}$	$x_{m,2}$	$\cdots$	$x_{m,n}$	$*$
(架空の供給地 $m+1$)	$*$	$*$	$*$	$*$	

(d) ierr = 5000 のときの$\boldsymbol{x}$の値は, 最適解ではないが, 制約条件を満たしている.

(e) 指定した反復回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて, 引き続き計算することができる. この処理を行うときには, isw の値を 1, nev の値を十分な値とし, それ以外の入力値には前回の出力値をそのまま用いる. また, 前回の作業領域の情報も用いる (例題参照) .

(f) この関数では, 第一次近似解法として北西隅の規則とハウサッカー法を採用している. ハウサッカー法は北西隅の規則より最適解に近い第一次近似解の値を求めることができ, その後の解の改善の時間を短縮することが可能である. しかし北西隅の規則に比べ, 近似解の計算にはより時間がかかる.

(7) **使用例**

(a) 問 題

輸送費用

$$C = \begin{bmatrix} 6.000 & 2.000 & 6.000 & 1.000 & 3.000 & 3.000 & 7.000 \\ 7.000 & 5.000 & 3.000 & 8.000 & 5.000 & 8.000 & 5.000 \\ 4.000 & 8.000 & 9.000 & 7.000 & 5.000 & 7.000 & 2.000 \\ 4.000 & 7.000 & 3.000 & 6.000 & 7.000 & 3.000 & 4.000 \\ 2.000 & 3.000 & 6.000 & 4.000 & 7.000 & 9.000 & 9.000 \\ 2.000 & 3.000 & 9.000 & 4.000 & 3.000 & 2.000 & 3.000 \end{bmatrix}$$

各需要地での需要量

$$D = \begin{bmatrix} 40.000 & 73.000 & 32.000 & 24.000 & 32.000 & 45.000 & 35.000 \end{bmatrix}$$

各供給地での供給量

$$S = \begin{bmatrix} 73.000 & 29.000 & 64.000 & 23.000 & 76.000 & 34.000 \end{bmatrix}$$

のもとで, 総輸送費用 z を最小にする輸送計画 x(i, j) を, ハウサッカー法を用いて求める. なおこの例題では, 注意事項 (e) で述べた継続処理を行うために, 最大評価回数を nev=2 回と小さく設定し, ierr=5000 が出力されるようにしている.

(b) 入力データ

(1 回目):配列 c, ns=7, lmc=7, nd=8, 配列 d, 配列 s, nev=2, isw1=0, isw2=1

(2 回目):nev=2, isw1=1(他は前回の計算後の値をそのまま入力値とする)

(c) 主プログラム

```
/*      C interface example for ASL_dmcltp */

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

int main()
{
    double *c;
    int lmc=7;
    int ns;
```

```
    int nd;
    double *s;
    double *d;
    int nev;
    double *x;
    double z;
    int isw1;
    int isw2;
    int *iwk;
    double *wk;
    int ierr;

    int i,j;
    FILE *fp;

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

    fscanf( fp, "%d", &isw1);
    fscanf( fp, "%d", &isw2);
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &ns  );
    fscanf( fp, "%d", &nd  );

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

    printf( "\tlmc     = %6d\n", lmc );
    printf( "\tns      = %6d\n", ns  );
    printf( "\tnd      = %6d\n", nd  );
    printf( "\tnev     = %6d\n", nev );
    printf( "\tisw1    = %6d\n", isw1);
    printf( "\tisw2    = %6d\n", isw2);

    c = ( double * )malloc((size_t)( sizeof(double) * (lmc*(nd+1)) ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }
    s = ( double * )malloc((size_t)( sizeof(double) * (ns+1) ));
    if( s == NULL )
    {
        printf( "no enough memory for array s\n" );
        return -1;
    }
    d = ( double * )malloc((size_t)( sizeof(double) * (nd+1) ));
    if( d == NULL )
    {
        printf( "no enough memory for array d\n" );
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * (lmc*(nd+1)) ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    for( i=0 ; i<ns  ; i++ )
    {
        for( j=0 ; j<nd ; j++ )
        {
            fscanf( fp, "%lf", &c[i + lmc*j] );
        }
    }

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

    for( j=0 ; j<nd ; j++ )
    {
        fscanf( fp, "%lf", &d[j] );
    }
    fclose( fp );

/*
  Output
*/
    printf( "\n\tTransportation Cost  = \n");
    for( i=0 ; i<ns  ; i++ )
    {
        printf( "\t%6d  ", i+1);
        for( j=0 ; j<nd ; j++ )
        {
            printf( "%8.3g", c[i + lmc*j] );
        }
        printf( "\n");
    }
    printf("\tSupply-Site\n"                          );
    printf("\t   Demand-Site 1       2       3       4");
```

```
    printf("       5       6       7\n"                );

    printf( "\n\tDemand  = \n");
    printf( "\t"              );
    for( j=0 ; j<nd ; j++ )
    {
        printf( "%8.3g", d[j] );
    }
    printf( "\n");

    printf( "\n\tSupply  = \n");

    printf( "\t");
    for( i=0 ; i<ns ; i++ )
    {
        printf( "%8.3g", s[i] );
    }
    printf( "\n");

    iwk = ( int * )malloc((size_t)( sizeof(int) * 12*(ns+nd+1)+5 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    wk  = ( double * )malloc((size_t)( sizeof(double) * 3*(ns+nd+2) ));
    if( wk  == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    ierr = ASL_dmcltp(c, lmc, ns, nd, s, d, &nev, x, &z, isw1, isw2, iwk, wk);

    printf( "\n\n\t     ** Output **\n"    );
    printf( "\n\t** First Result ** \n\n"  );
    printf( "\tierr        = %6d\n", ierr  );
    printf( "\tnev         = %6d\n", nev   );
    printf( "\tTotal Cost  = %8.3g\n\n", z );

    printf( "\tTransportation Plan  = \n");
    for( i=0 ; i<ns+1  ; i++ )
    {
        printf( "\t%6d  ", i+1);
        for( j=0 ; j<nd+1 ; j++ )
        {
            printf( "%8.3g", x[i + lmc*j] );
        }
        printf( "\n");
    }
    printf("\tSupply-Site\n"                          );
    printf("\t   Demand-Site 1       2       3       4");
    printf("       5       6       7       8\n"        );

    isw1 = 1;
    nev  = 1000;

    ierr = ASL_dmcltp(c, lmc, ns, nd, s, d, &nev, x, &z, isw1, isw2, iwk, wk);

    printf( "\n\t** Improved Result ** \n\n" );
    printf( "\tierr        = %6d\n", ierr    );
    printf( "\tnev         = %6d\n", nev     );
    printf( "\tTotal Cost  = %8.3g\n\n", z   );

    printf( "\tTransportation Plan  = \n");
    for( i=0 ; i<ns+1  ; i++ )
    {
        printf( "\t%6d  ", i+1);
        for( j=0 ; j<nd+1 ; j++ )
        {
            printf( "%8.3g", x[i + lmc*j] );
        }
        printf( "\n");
    }
    printf("\tSupply-Site\n"                          );
    printf("\t   Demand-Site 1       2       3       4");
    printf("       5       6       7       8\n"        );

    free(c  );
    free(s  );
    free(d  );
    free(x  );
    free(iwk);
    free(wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmcltp ***

 ** Input **
```

```
lmc     =      7
ns      =      6
nd      =      7
nev     =      2
isw1    =      0
isw2    =      1

Transportation Cost  =
     1         6       2       6       1       3       3       7
     2         7       5       3       8       5       8       5
     3         4       8       9       7       5       7       2
     4         4       7       3       6       7       3       4
     5         2       3       6       4       7       9       9
     6         2       3       9       4       3       2       3
Supply-Site
   Demand-Site 1       2       3       4       5       6       7

Demand  =
      40      73      32      24      32      45      35

Supply  =
      73      29      64      23      76      34


     ** Output **

** First Result **

ierr        =   5000
nev         =      2
Total Cost  =    707

Transportation Plan  =
     1         0      37       0      24       3       0       0       9
     2         0       0      20       0       0       0       0       9
     3         0       0       0       0      29       0      35       0
     4         0       0      12       0       0      11       0       0
     5        40      36       0       0       0       0       0       0
     6         0       0       0       0       0      34       0       0
     7         0       0       0       0       0       0       0       0
Supply-Site
   Demand-Site 1       2       3       4       5       6       7       8

** Improved Result **

ierr        =   1100
nev         =      3
Total Cost  =    680

Transportation Plan  =
     1         0      37       0      24      12       0       0       0
     2         0       0      29       0       0       0       0       0
     3         0       0       0       0      11       0      35      18
     4         0       0       3       0       0      20       0       0
     5        40      36       0       0       0       0       0       0
     6         0       0       0       0       9      25       0       0
     7         0       0       0       0       0       0       0       0
Supply-Site
   Demand-Site 1       2       3       4       5       6       7       8
```

5.7 多変数 2 次関数の最小化 (2 次計画)

5.7.1 ASL_dmcqsn, ASL_rmcqsn
多変数凸型 2 次関数の最小化 (線形制約)

(1) **機 能**

線形制約の中で, n 次のベクトル $\boldsymbol{x} = [x_1, \cdots, x_n]^T$ の多変数凸型 2 次関数 $f(\boldsymbol{x})$ を最小にする$\boldsymbol{x}$を求める.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + (1/2)\boldsymbol{x}^T G\boldsymbol{x}$$

制約条件 :

$$\boldsymbol{a}_i^T\boldsymbol{x} = b_i \quad (i = 1, 2, \cdots, m_e)$$

$$\boldsymbol{a}_i^T\boldsymbol{x} \geq b_i \quad (i = m_e + 1, m_e + 2, \cdots, m; \quad 0 \leq m_e \leq m)$$

ここで, G は $n \times n$ 正定値行列, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ と $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ は n 次ベクトル b_i は定数 $(i = 1, 2, \cdots, m$ とする.

(2) **使用法**

倍精度関数:

ierr = ASL_dmcqsn (a, lna, m, b, g, c, n, me, er, & nev, x, & y, isw, iwk, wk);

単精度関数:

ierr = ASL_rmcqsn (a, lna, m, b, g, c, n, me, er, & nev, x, & y, isw, iwk, wk);

(3) 引数と戻り値

D:倍精度実数型　Z:倍精度複素数型　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×m	入 力	isw=0 のとき制約条件の係数に対応する行列の転置行列 $\mathrm{A}^{\mathrm{T}} = (\mathrm{a_{j,i}})$ $(\mathrm{i} = 1, 2, \cdots, \mathrm{m};\ \mathrm{j} = 1, 2, \cdots, \mathrm{n})$ (注意事項 (a) (d) 参照)
				出 力	変換後の制約条件の係数に対応する行列の転置行列
2	lna	I	1	入 力	配列 a の整合寸法
3	m	I	1	入 力	制約条件の数 m
4	b	{D∗ / R∗}	m	入 力	isw=0 のとき制約条件の右辺 $\boldsymbol{b} = (b_i)$ $(i = 1, 2, \cdots, m)$ (注意事項 (d) 参照)
				出 力	変換後の制約条件の右辺
5	g	{D∗ / R∗}	lna×n	入 力	目的関数の 2 次の係数行列 G
6	c	{D∗ / R∗}	n	入 力	目的関数の 1 次の係数ベクトル$\boldsymbol{c}$
7	n	I	1	入 力	変数の数 n
8	me	I	1	入 力	等式の制約条件の数 m_e
9	er	{D / R}	1	入 力	要求精度 (既定値: $2 \times \sqrt{(\text{誤差判定のための単位})}$)
10	nev	I*	1	入 力	関数 $f(\boldsymbol{x})$ の最大評価回数 (既定値: $10 \times \mathrm{n} + \mathrm{m}$)
				出 力	実際の関数評価回数
11	x	{D∗ / R∗}	n	出 力	最終到達点 $\boldsymbol{x}$
12	y	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}$ での関数値 $f(\boldsymbol{x})$
13	isw	I	1	入 力	処理スイッチ (注意事項 (f) 参照) 0: 最初の処理 1: 継続の処理
14	iwk	I*	3	ワーク	作業領域
15	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $2 \times \mathrm{n}^2 + 17 \times \mathrm{n} + 16 \times \mathrm{m} + 16$
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

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

(b) isw $= 0$ または isw $= 1$

(c) er $\geq$ 誤差判定のための単位 (既定値にするため, 0.0 を入力する場合は除く)

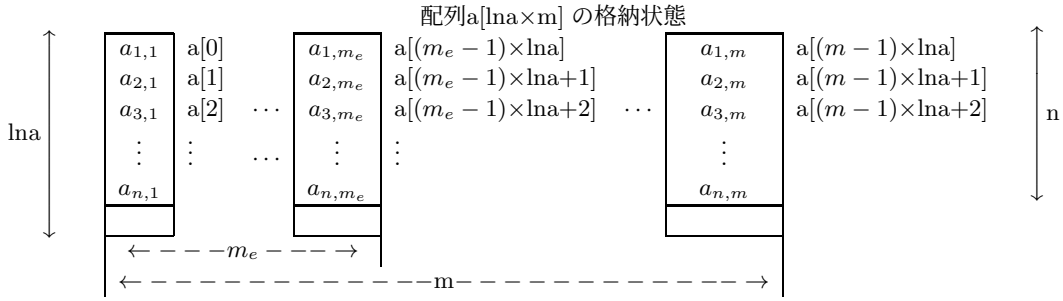
(d) nev > 0 (既定値にするため, 0 を入力する場合は除く)

(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.	処理を続ける.
1500	制限条件 (c) または (d) を満足しなかった.	既定値にセットして処理する.
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3500	等式制約条件が互いに独立でなかった.	処理を打ち切る.
4000	問題が実行不可能であった.	処理を打ち切る.
4000+i	QR 分解の i 列目の処理において対角要素が 0.0 となった. または LL^{T} 分解の i 列目の処理において, 対角要素が 0.0 以下となった.	処理を打ち切る.
5000	与えられた最大評価回数に達しても収束しなかった.	その時点での x, y の値を出力して, 処理を打ち切る. (注意事項 (e)(f) 参照)

(6) **注意事項**

(a) isw=0 の時, 等式制約条件に対応する係数 $a_{i,j}$ $(i = 1, 2, \cdots, m_e;\ j = 1, 2, \cdots, n)$ と不等式制約条件に対応する係数 $a_{i,j}$ $(i = m_e + 1, m_e + 2, \cdots, m;\ j = 1, 2, \cdots, n)$ は以下の様に配列 a に格納する. ここで, m は制約条件の数, n は変数の数をそれぞれ表す.



備　考
a.　lna $\geq n$ を満たさなければならない.

(b) 目的関数の 2 次の係数行列 G が正定値でないものは, 本関数では最適解を求められない.

(c) 引数の内容の欄に既定値が記されている場合には, 整数型のときは 0, 実数型のときは 0.0 を入力すれば既定値がセットされる.

(d) a, b の出力には変換した値が入る.

(e) 通常 ierr $= 5000$ が返されたとき $\boldsymbol{x}$ の値は制約条件も満たさない.

(f) 指定した収束回数が少なく ierr $= 5000$ が返されたとき, 途中まで計算した情報を用いて, 引き続き計算することができる. この処理を行うときには, isw の値を 1, nev の値を十分な値とし, それ以外の入力値には前回の出力値をそのまま用いる. また, 前回のワークの情報も用いる. (例題参照)

(7) 使用例

(a) 問 題

$$\begin{aligned} -x_1 - 2x_2 &\geq -10 \\ -3x_1 - x_2 &\geq -15 \\ -2x_1 - 3x_2 &\geq -30 \\ 15x_1 - 13x_2 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

のもとで

$$f(\boldsymbol{x}) = 1/2(5x_1^2 + 6x_1x_2 + 5x_2^2) - 95x_1 - 105x_2$$

を最小にする.

なお, この例では, 注意事項 (f) で述べた継続処理を行うために, 最大評価回数 nev=3 と小さく設定し, ierr=5000 が出力されるようにしている.

(b) 入力データ

(1 回目): n=2, m=6, me=0, lna=11, nev=3, isw=0, er=0.0

配列 a, b, g, c

(2 回目以降): nev=3, isw=1

(他は, 前回の計算後の値を, そのまま入力値とする)

(c) 主プログラム

```
/*      C interface example for ASL_dmcqsn */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *a, *b, *g, *c, eer, *x, y, *wk;
    int na, mm, nn, mme, nev, iisw, iwk[3], ierr;
    int i, j, once=0,nwk;
    FILE *fp;

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

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

    na = 11;
    fscanf( fp, "%d %d %d %lf %d %d", &nn, &mm, &mme, &eer, &nev, &iisw );
    printf( "\n\tn = %6d\tm = %6d\tme = %6d\n", nn, mm, mme );

    a = ( double * )malloc((size_t)( sizeof(double) * (na*mm) ));
    if( a == NULL )
    {
        printf( "no enough memory for array a\n" );
        return -1;
    }
    b = ( double * )malloc((size_t)( sizeof(double) * mm ));
    if( b == NULL )
    {
        printf( "no enough memory for array b\n" );
        return -1;
    }
    g = ( double * )malloc((size_t)( sizeof(double) * (na*nn) ));
    if( g == NULL )
    {
        printf( "no enough memory for array g\n" );
        return -1;
    }
    c = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
```

```
        return -1;
    }
    x = ( double * )malloc((size_t)( sizeof(double) * nn ));
    if( x == NULL )
    {
        printf( "no enough memory for array x\n" );
        return -1;
    }
    nwk=2*nn*nn+17*nn+16*mm+16;
    wk = ( double * )malloc((size_t)( sizeof(double) * nwk ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

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

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

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

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


    ierr = ASL_dmcqsn(a, na, mm, b, g, c, nn, mme, eer, &nev, x, &y, iisw, iwk, wk);

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

    printf( "\n     ** First Result (isw == 0) **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n      ** Vector x **\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\tx[ %6d ] = %8.3g\n", i, x[i] );
    }

loop:
    fscanf( fp, "%d %d", &nev, &iisw );

    ierr = ASL_dmcqsn(a, na, mm, b, g, c, nn, mme, eer, &nev, x, &y, iisw, iwk, wk);


    printf( "\n     ** Improved Result (isw != 0) **\n\n" );
    printf( "\tierr = %6d\n", ierr );

    printf( "\n      ** Vector x **\n\n" );
    for( i=0 ; i<nn ; i++ )
    {
        printf( "\tx[ %6d ] = %8.3g\n", i, x[i] );
    }
    printf( "\ty = %8.3g\n", y );

    if (ierr == 5000 && once==0)
    {
        once=1;
        goto loop;
    }

    fclose( fp );
```

```
    free( a );
    free( b );
    free( g );
    free( c );
    free( x );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmcqsn ***

 ** Input **

n =      2  m =      6  me =      0

  ** Matrix a **

      -1       -3       -2       15        1        0
      -2       -1       -3      -13        0        1

  ** Vector b **

     -10      -15      -30        0        0        0

  ** Matrix g **

       5        3
       3        5

  ** Vector c **

     -95     -105

 ** Output **

 ** First Result (isw == 0) **

ierr =   5000

  ** Vector x **

x[      0 ] =     2.14
x[      1 ] =     8.57

 ** Improved Result (isw != 0) **

ierr =      0

  ** Vector x **

x[      0 ] =        4
x[      1 ] =        3
y =     -597
```

5.7.2 ASL_dmcqlm, ASL_rmcqlm
多変数広義凸型 2 次関数の最小化 (線形制約)

(1) **機 能**

制約条件

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

$$\sum_{j=1}^{n} a_{i,j}x_j \geq b_i \quad (i = m_e + 1, \cdots, m)$$

$$x_i \geq 0 \quad (i = 1, \cdots, n)$$

の下で目的関数

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$$

を最小にする $\boldsymbol{x}$ とその時の目的関数の値 $f(\boldsymbol{x})$ を求める. ここで, G は $n \times n$ 半正定値行列, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$, $\boldsymbol{a}_i^T = [a_{i,1}, a_{i,2}, \cdots, a_{i,n}]$ は n 次のベクトル $(i = 1, \cdots, m)$.

(2) **使用法**

倍精度関数:

ierr = ASL_dmcqlm (a, na, n, b, m, me, g, ng, c, & nev, x, & y, isw, iw, w);

単精度関数:

ierr = ASL_rmcqlm (a, na, n, b, m, me, g, ng, c, & nev, x, & y, isw, iw, w);

(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∗}	na×n	入 力	isw=0 のとき制約条件の係数に対応する行列 $A=(a_{i,j})$ $(i=1,2,\cdots,m;\ j=1,2,\cdots,n)$
2	na	I	1	入 力	配列 a の整合寸法
3	n	I	1	入 力	変数の数 n
4	b	{D∗ / R∗}	m	入 力	制約条件の右辺の定数 b_i を成分に持つ m 次元ベクトル
5	m	I	1	入 力	制約条件の数 m
6	me	I	1	入 力	等式制約条件の数 m_e
7	g	{D∗ / R∗}	ng×n	入 力	目的関数の 2 次の係数行列 g
8	ng	I	1	入 力	配列 g の整合寸法
9	c	{D∗ / R∗}	n	入 力	目的関数の 1 次の係数ベクトル$\boldsymbol{c}$
10	nev	I*	1	入 力	最大反復回数 (注意事項 (c) 参照)
				出 力	計算終了に要した反復回数
11	x	{D∗ / R∗}	n	出 力	最適解 $\boldsymbol{x}^*$
12	y	{D∗ / R∗}	1	出 力	最適解 $\boldsymbol{x}$ に対する目的関数の値 $f(\boldsymbol{x})$
13	isw	I	1	入 力	処理スイッチ (注意事項 (d) 参照) 0:最初の処理 1:継続処理
14	iw	I*	内容参照	ワーク	作業領域 **大きさ**: $2\times \mathrm{n}-\mathrm{me}+\mathrm{m}+2$
15	w	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $(\mathrm{n}+\mathrm{m}-\mathrm{me}+2)\times(\mathrm{n}+\mathrm{m}-\mathrm{me})\times 2+(\mathrm{m}+\mathrm{me})\times(\mathrm{n}+1)+\mathrm{n}$
16	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

(a) $\mathrm{na}\geq \mathrm{m}, \mathrm{ng}\geq \mathrm{n}$

(b) $\mathrm{ng}, \mathrm{na}, \mathrm{n} > 0$

(c) $\mathrm{m}, \mathrm{me}\geq 0$

(d) $\mathrm{m}\geq \mathrm{me}$

(e) $\mathrm{n}\geq \mathrm{me}$

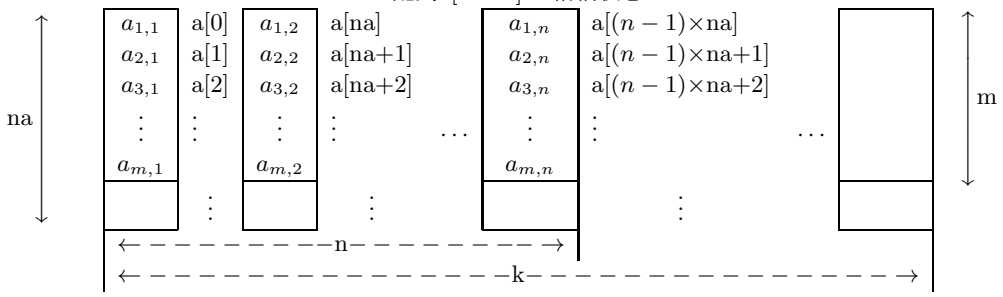
(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	n=1 であった.	処理を続ける.
1500	m=0 であった.	
3000	制限条件 (a)∼(e) のいずれかを満足しなかった.	処理を打ち切る.
3500	等式制約条件を満たす解が存在しない.	
4000	最適解が存在しないことがわかった.	
5000	与えられた反復回数内で解が求められなかった.	

(6) **注意事項**

(a) isw=0 の時, 制約条件に対応する係数 $a_{i,j}$ $(i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n)$ は以下の様に配列 a に格納する. ここで, m は制約条件の数, n は変数の数をそれぞれ表す.

配列a[na×k] の格納状態



備　考
a.　k ≥ n を満たさなければならない.

(b) 目的関数の 2 次の係数行列 G が半正定値でないものは, 本関数では最適解を求められない.

(c) 引数 nev に正でない値が入力された場合, 既定値として nev = n − m − me にセットし直す.

(d) 指定した反復回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて計算する. この処理を行うときは isw=1 として, 前回の出力値を入力値として用いる. このとき作業配列 w, iw の内容を保持しておかなければならない. なお, 反復回数がすでに n + m − me に達している場合には, さらに反復を繰り返しても解は求められない.

(7) 使用例

(a) 問 題

制約条件

$$\begin{aligned} -x_1 - 2x_2 &\geq -10 \\ -3x_1 - x_2 &\geq -15 \\ -2x_1 - 3x_2 &\geq -30 \\ 15x_1 - 13x_2 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

のもとで

$$f(\boldsymbol{x}) = \frac{1}{2}(5x_1^2 + 6x_1x_2 + 5x_2^2) - 95x_1 - 105x_2$$

を最小にする. なお, この例では, 注意事項 (d) で述べた継続処理を行うために, 最大評価回数 nev=2 と小さく設定し, ierr=5000 が出力されるようにしている.

(b) 入力データ

(1 回目): n=2, m=4, me=0, na=11, nev=0

配列 a, b, g, c

(2 回目以降): nev=0, isw=1

(他は前回の計算後の値を, そのまま入力値とする.)

(c) 主プログラム

```
/*      C interface example for ASL_dmcqlm */

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

int main()
{
    double *a;
    int na;
    int n;
    double *b;
    int m;
    int me;
    double *g;
    int ng;
    double *c;
    int nev;
    int isw;
    double *x;
    double y;
    int *iw;
    double *w;
    int ierr;
    int i,j;
    FILE *fp;

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

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

    fscanf( fp, "%d", &na );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &me );
    fscanf( fp, "%d", &ng );
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );

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

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

x = ( double * )malloc((size_t)( sizeof(double) * n ));
if( x == NULL )
{
    printf( "no enough memory for array x\n" );
    return -1;
}
c = ( double * )malloc((size_t)( sizeof(double) * n ));
if( c == NULL )
{
    printf( "no enough memory for array c\n" );
    return -1;
}
b = ( double * )malloc((size_t)( sizeof(double) * m ));
if( b == NULL )
{
    printf( "no enough memory for array b\n" );
    return -1;
}
w = ( double * )malloc((size_t)( sizeof(double) *
(2*(n+m-me+2)*(n+m-me)+2*m*(n+1)+n) ));
if( w == NULL )
{
    printf( "no enough memory for array w\n" );
    return -1;
}
iw = ( int * )malloc((size_t)( sizeof(int) *
(2*n+m-me+2) ));
if( iw == NULL )
{
    printf( "no enough memory for array iw\n" );
    return -1;
}

printf( "\tna = %6d ng = %6d\n", na, ng );
printf( "\tn = %6d m = %6d me = %6d nev = %6d isw = %6d\n",
n, m, me, nev, isw );

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

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

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

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



ierr = ASL_dmcqlm(a, na, n, b, m, me, g, ng, c, &nev, x, &y, isw, iw, w);

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

```
    fscanf( fp, "%d", &nev );
    fscanf( fp, "%d", &isw );
    fclose( fp );
    ierr = ASL_dmcqlm(a, na, n, b, m, me, g, ng, c, &nev, x, &y, isw, iw, w);

    printf( "\n     ** Improved Result **\n" );
    printf( "\tierr = %6d\n", ierr );
    printf( "\tnev  = %6d\n", nev );
    printf( "\tVector x\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\t x( %d ) = %8.3g\n", i, x[i] );
    }
    printf( "\t y = %8.3g\n", y );

    free( a );
    free( g );
    free( x );
    free( c );
    free( b );
    free( w );
    free( iw );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmcqlm ***

 ** Input **

na =     11 ng =      8
n =      2 m =      4 me =      0 nev =      2 isw =      0
Matrix a
 a( 0 , 0 ) =       -1 a( 0 , 1 ) =       -2
 a( 1 , 0 ) =       -3 a( 1 , 1 ) =       -1
 a( 2 , 0 ) =       -2 a( 2 , 1 ) =       -3
 a( 3 , 0 ) =       15 a( 3 , 1 ) =      -13
Vector b
 b( 0 ) =      -10
 b( 1 ) =      -15
 b( 2 ) =      -30
 b( 3 ) =        0
Matrix g
 g( 0 , 0 ) =        5 g( 0 , 1 ) =        3
 g( 1 , 0 ) =        3 g( 1 , 1 ) =        5
Vector c
 c( 0 ) =      -95
 c( 1 ) =     -105

 ** Output **


 ** First Result **
ierr =   5000
nev  =      2

 ** Improved Result **
ierr =      0
nev  =      2
Vector x
 x( 0 ) =        4
 x( 1 ) =        3
 y =     -597
```

5.7.3 ASL_dmcqaz, ASL_rmcqaz
制約無し 0-1 多変数 2 次関数の最小化 (0-1 無制約 2 次計画問題)

(1) **機 能**

n 次の 0-1 ベクトル $\boldsymbol{x} = [x_1, \cdots, x_n]^T$ $(x_i \in \{0, 1\})$ の多変数 2 次関数 $f(\boldsymbol{x})$ を最小にする$\boldsymbol{x}$を求める.

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + (1/2)\boldsymbol{x}^T G\boldsymbol{x}$$

ここで, G は $n \times n$ 行列, $\boldsymbol{c}^T = [c_1, c_2, \cdots, c_n]$ は n 次ベクトルとする.

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmcqaz  (g, na, n, c, & ir1, & ir2, ipm, rpm, ix, & y, iwk, wk);
```

単精度関数:

```
ierr = ASL_rmcqaz  (g, na, n, c, & ir1, & ir2, ipm, rpm, ix, & y, iwk, wk);
```

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	g	{D∗ / R∗}	na×n	入 力	目的関数の 2 次の係数行列 G
2	na	{D / R}	1	入 力	配列 g の整合寸法
3	n	I	1	入 力	変数の数 n
4	c	{D∗ / R∗}	n	入 力	目的関数の 1 次の係数ベクトル$\boldsymbol{c}$
5	ir1	I*	1	入 力	乱数の初期値 (注意事項 (a), (b) 参照)
				出 力	次の乱数の初期値
6	ir2	I*	1	入 力	乱数の初期値 (注意事項 (a), (b) 参照)
				出 力	次の乱数の初期値
7	ipm	I*	12	入 力	整数パラメータ
				出 力	ipm [1-10] ： 実際に使用された整数パラメータの値 ipm [11] ：分枝限定法における解の評価回数
8	rpm	I*	5	入 力	整数パラメータ
				出 力	rpm [1-4] ： 実際に使用された整数パラメータの値
9	ix	I*	n	入 力	初期解 $\boldsymbol{x}^{(0)}$ (ipm [1] ≠1 のときのみ)
				出 力	最終到達点 $\boldsymbol{x}$
10	y	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}$ での関数値 $f(\boldsymbol{x})$
11	iwk	I*	内容参照	ワーク	作業領域 **大きさ**: $\mathrm{n}\times(\max(\mathrm{ipm}[6],\mathrm{ipm}[9])+3)+\mathrm{ipm}[9]+13$
12	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $3\times\mathrm{n}^2+12\times\mathrm{n}+\max(\mathrm{ipm}[9],\mathrm{n})+(\mathrm{n}+1)\times\mathrm{ipm}[9]+9$
13	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) 制限条件

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

(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.	
3000	制限条件 (a) を満足しなかった.	処理を打ち切る.
4000	計算誤差により分枝限定法による最適解の探索が, 続行できなくなった	その時点での ix, y の値を出力して, 処理を打ち切る.
5000	緩和問題の解が与えられた最大評価回数に達しても得られなかった.	
5500	分枝限定法で終端されない部分問題が与えられた個数に達した,	
5600	与えられた評価回数で分枝限定法の計算が終了しなかった.	

(6) **注意事項**

(a) ir1, ir2 は奇数が望ましい.

(b) ir1, ir2 に前回の出力値を用いることによって, 前回発生した乱数列に続く乱数列を得ることができる.

(c) 最大値探索を行いたい場合は, $-f(\boldsymbol{x})$ の最小値探索を行えば良い. このとき, y の値を正負逆にしたものが最大値である.

(d) ierr = 4000, 5000, 5500 または 5600 が返されて, 計算をやり直す場合には, 最初の計算で得られた解$\boldsymbol{x}$を次の計算での初期解として用いれば効率的な探索が出来る. この処理を行うときには, ipm [0] と ipm [1] をともに 0 以外の値に設定し, それ以外の入力値には前回の出力値をそのまま用いる.

(e) n が大きい場合には分枝限定法による解の探索には多大な計算時間を要する. このような場合にはパラメータ ipm [5] を 0 以外の値に設定することにより, 最適解の発見的探索のみを行い, 得られた解を近似的な最適解として利用できる. ただし, 発見的探索で得られた解が真の最適解のどの程度良い近似となっているかは保証されない.

(f) パラメータ ipm [7] を大きく設定すると, 分枝限定法における下界値テストの精度が良くなり, 探索回数を減らせる場合がある. しかし, 1 回の下界値テストに要する計算時間がこのパラメータの値を増やすと指数的に増大するため, 通常はこのパラメータの値は 1 に設定しておけば良い.

(g) パラメータ rpm [1] を大きく設定すると, 計算時間は増大するが, 数値的には安定し, ierr = 4000 や 5000 が出力されにくくなる.

表 5–1　パラメータ表 (0-1 無制約 2 次計画問題)

引数名	既定値	内　　容
ipm [0]	–	既定値の設定 0:整数パラメータ ipm [1-11] を既定値に設定する. 0 以外:整数パラメータ ipm [1-11] の値を利用者が設定する.
ipm [1]	1	初期解設定スイッチ 0:初期解 $\boldsymbol{x}^{(0)}$ が自動的に設定される. 0 以外:初期解 $\boldsymbol{x}^{(0)}$ を利用者が設定する.
ipm [2]	0	発見的探索方法選択スイッチ 0:模擬焼なまし法で初期解を求め, それをタブー探索で改良する. 1:模擬焼なまし法を用いる. 2:タブー探索を用いる. それ以外:発見的探索を行わない.
ipm [3]	10000	模擬焼なまし法での探索回数. (0 以下に設定すると既定値を用いる.)
ipm [4]	100	タブー探索での探索回数. (0 以下に設定すると既定値を用いる.)
ipm [5]	0	分枝限定法による解の探索を行うかどうかの選択スイッチ. 0:分枝限定法による解の探索を行う. 0 以外:分枝限定法による解の探索を行わない.
ipm [6]	ipm [4]	タブーリストの長さ. (0 以下に設定すると既定値を用いる.)
ipm [7]	1	分枝限定法における部分問題の下界値計算に用いる自由変数の個数. (0∼10 以外に設定すると既定値を用いる.)
ipm [8]	1	緩和 2 次計画問題の計算における最大反復回数. (0 以下に設定すると既定値を用いる.)
ipm [9]	内容参照	分枝限定法における活性部分問題の個数の最大値. 既定値:max(n,ipm [6]) (0 以下に設定すると既定値を用いる.)
ipm [10]	1	分枝限定法における探索の深さ. (0 以下に設定すると既定値を用いる.)
ipm [11]	100×n	分枝限定法における最大評価回数. (0 以下に設定すると既定値を用いる.)

表 5–1　パラメータ表 (0-1 無制約 2 次計画問題) (続)

引数名	既定値	内　　容
rpm [0]	–	既定値の設定 0.0:実数パラメータ rpm [1-4] を既定値に設定する. 0.0 以外:実数パラメータ rpm [1-4] の値を利用者が設定する.
rpm [1]	内容参照	緩和 2 次計画問題の計算における要求精度. 既定値:$\sqrt{\text{誤算判定の単位}}$ (0.0 以下に設定すると既定値を用いる.)
rpm [2]	1.0	係数行列 G を正値対称行列に変換した後の最小固有値 β (0.0 以下に設定すると既定値を用いる.)
rpm [3]	50000.0	模擬焼なまし法における初期温度 T_0 (0.0 未満に設定すると既定値を用いる.)
rpm [4]	0.999	模擬焼なまし法における温度低下率 α (0.0 以下または 1.0 より大きい値に設定すると既定値を用いる.)

(7) 使用例

(a) 問 題

行列 G とベクトル$\boldsymbol{c}$がそれぞれ,

$$G = \begin{pmatrix} 1 & -2 & -1 & 3 & 2 & 0 \\ -1 & 4 & 2 & 1 & 5 & -4 \\ 0 & 1 & 3 & 0 & -3 & 2 \\ 2 & 5 & 0 & -3 & 1 & 3 \\ 1 & 1 & 5 & 3 & -3 & 2 \\ 4 & 5 & -1 & -1 & 3 & 1 \end{pmatrix}$$

$$\boldsymbol{c} = (1, 1, 1, -1, -1, -1)^T$$

で与えられるとき,

$$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x}$$

を最小にする 0-1 ベクトル $\boldsymbol{x}^*$ とそのときの関数値 $f(\boldsymbol{x}^*)$ を求める.

(b) 入力データ

na=7, n=6, ir1=1, ir2=1, ipm [0] =0, rpm [0] =0.0, 係数行列 G, 係数ベクトル $\boldsymbol{c}$

(c) 主プログラム

```
/*      C interface example for ASL_dmcqaz */
#include <stdio.h>
#include <stdlib.h>
#include <asl.h>

int main()
{
    double *g, *c, y, *wk, rpm[5];
    int na, n, ir1, ir2, ipm[12], *ix, *iwk, ierr;
    int i, j;
    FILE *fp;

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

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

```
    na = 7;
    ipm[0] = 0;
    rpm[0] = 0.0;

    fscanf( fp, "%d", &n );
    printf( "\n\tn = %6d\n", n );
    fscanf( fp, "%d%d", &ir1, &ir2 );
    printf( "\n\tir1 = %6d\tir2 = %6d\n", ir1, ir2 );

    g = ( double * )malloc((size_t)( sizeof(double) * (na*n) ));
    if( g == NULL )
    {
        printf( "no enough memory for array g\n" );
        return -1;
    }
    c = ( double * )malloc((size_t)( sizeof(double) * n ));
    if( c == NULL )
    {
        printf( "no enough memory for array c\n" );
        return -1;
    }
    ix = ( int * )malloc((size_t)( sizeof(int) * n ));
    if( ix == NULL )
    {
        printf( "no enough memory for array ix\n" );
        return -1;
    }
    wk = ( double * )malloc((size_t)( sizeof(double) * 989 ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }
    iwk = ( int * )malloc((size_t)( sizeof(int) * 731 ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

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

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


    ierr = ASL_dmcqaz(g, na, n, c, &ir1, &ir2, ipm, rpm, ix, &y, iwk, wk);

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

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

    printf( "\n      ** Vector ix **\n\n" );
    for( i=0 ; i<n ; i++ )
    {
        printf( "\tix[ %6d ] = %6d\n", i, ix[i] );
    }

    printf( "\n\ty = %8.3g\n", y );

    fclose( fp );
    free( g );
    free( c );
    free( ix );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmcqaz ***

 ** Input **

n =      6

ir1 =      1    ir2 =      1
```

```
  ** Matrix g **

       1        -2        -1         3         2         0
      -1         4         2         1         5        -4
       0         1         3         0        -3         2
       2         5         0        -3         1         3
       1         1         5         3        -3         2
       4         5        -1        -1         3         1

  ** Vector c **

       1         1         1        -1        -1        -1

 ** Output **

ierr =       0

  ** Vector ix **

ix[       0 ] =       0
ix[       1 ] =       0
ix[       2 ] =       0
ix[       3 ] =       1
ix[       4 ] =       1
ix[       5 ] =       0

y =        -3
```

5.8 制約付き多変数関数の最小化 (非線形計画)

5.8.1 ASL_dmsqpm, ASL_rmsqpm
制約付き多変数関数の最小化 (非線形制約)

(1) **機 能**

m 個の不等式制約条件

$$g_i(\boldsymbol{x}) \leq 0, \quad i = 1, \cdots, m$$

と, ℓ 個の等式制約条件

$$h_j(\boldsymbol{x}) = 0, \quad j = 1, \cdots, \ell$$

のもとで, 多変数関数 $f(\boldsymbol{x})$ を局所的に最小化する $\boldsymbol{x}^*$ とそのときの関数値 $f(\boldsymbol{x}^*)$ を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dmsqpm (f, gf, hf, m, ml, x, n, er, & nev, & y, iwk, wk);

単精度関数:

ierr = ASL_rmsqpm (f, gf, hf, m, ml, x, n, er, & nev, & y, iwk, wk);

(3) 引数と戻り値

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	f	{D / R}	–	入 力	関数 $f(\boldsymbol{x})$ を定義する関数 f(x) の関数名
2	gf	–	–	入 力	不等式制約条件を与える関数 gf(x, n, g, mm) の関数名 (mm=m−ml)
3	hf	–	–	入 力	等式制約条件を与える関数 hf(x, n, h, ml) の関数名
4	m	I	1	入 力	制約条件の数 $m+\ell$
5	ml	I	1	入 力	等式制約条件の数 ℓ
6	x	{D∗ / R∗}	n	入 力	探索点の初期値 $\boldsymbol{x}_0$
				出 力	最適解 $\boldsymbol{x}^*$
7	n	I	1	入 力	$\boldsymbol{x}$ の成分の数
8	er	{D / R}	1	入 力	0 判定のための単位 (既定値: 2 × $\sqrt{(\text{誤差判定のための単位})}$)
9	nev	I*	1	入 力	$\boldsymbol{x}$ の最大更新回数 (既定値: 100)
				出 力	実際の $\boldsymbol{x}$ の更新回数
10	y	{D∗ / R∗}	1	出 力	最終到達点 $\boldsymbol{x}^*$ での関数値 $f(\boldsymbol{x}^*)$
11	iwk	I*	m+3	ワーク	作業領域
12	wk	{D∗ / R∗}	内容参照	ワーク	作業領域 **大きさ**: $4\times n\times n+2\times m\times n+21\times n+108\times m$
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) m $>$ 0 (制約条件のない問題を解くため, 0 を入力する場合を除く)

(b) ml $\geq$ 0

(c) n $>$ 0

(d) n $\geq$ ml

(e) m $\geq$ ml

(f) er $>$ 0.0 (既定値にするため, 0.0 を入力する場合を除く)

(g) nev $\geq$ 0 (既定値にするため, 負値を入力する場合を除く)

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
1000	制限条件 (f), (g) を満足しなかった.	既定値にセットして処理する.
1500	m=0 であった.	処理を続ける.
3000	制限条件 (a) から (e) を満足しなかった.	処理を打ち切る.
3500	部分 2 次計画問題の等号制約条件が互いに独立でなかった.	その時点での x, y の値を出力して処理を打ち切る. 得られた x は制約条件を満たす.
3600	部分 2 次計画問題が実行不可能であった.	
3700	部分 2 次計画問題のための QR 分解の計算において分解される行列の対角要素が 0.0 となった (この行列はランク落ちしている). または部分 2 次計画問題のための LL^T 分解の計算において分解される行列の対角要素が 0.0 以下となった (この行列は特異に近い).	
3800	部分 2 次計画問題の解が収束しなかった.	
3900	直線探索で刻み幅が 0.0 以下になった.	
4000	部分 2 次計画問題の等号制約条件が互いに独立でなかった.	
4100	部分 2 次計画問題が実行不可能であった.	
4200	部分 2 次計画問題のための QR 分解の計算において分解される行列の対角要素が 0.0 となった (この行列はランク落ちしている). または部分 2 次計画問題のための LL^T 分解の計算において分解される行列の対角要素が 0.0 以下となった (この行列は特異に近い).	その時点での x, y の値を出力して処理を打ち切る. 得られた x は制約条件を満たさない.
4300	部分 2 次計画問題の解が収束しなかった.	
4400	直線探索で刻み幅が 0.0 以下になった.	
4500	与えられた x の最大更新回数に達しても問題が解けなかった	
5000	与えられた x の最大更新回数に達しても問題が解けなかった	

(6) **注意事項**

(a) 関数 f の作り方は, 次に示す通りである.

```
double FORTRAN f(double *x)
{
  return(f(*x));
}
```

(b) 関数 gf, hf の作り方は, 次に示す通りである.

```
void FORTRAN gf(double *x, int *n, double *g, int *mm)
{
  g[0] = g1(x);
  ⋮
  g[mm − 1] = gmm(x);
}

void FORTRAN hf(double *x, int *n, double *h, int *ml)
{
  h[0] = h1(x);
  ⋮
  h[ml − 1] = hml(x);
}
```

なお, ここで引数 mm の値は不等式制約条件の数である.

(c) 指定した $\boldsymbol{x}$ の最大更新回数が少なく ierr=5000 が返されたとき, 途中まで計算した情報を用いて計算する. この処理を行うときは前回の x の出力値を入力値として用いる.

(7) **使用例**

(a) 問 題
制約条件

$$\begin{aligned} -x_1 + x_2 &\leq 0 \\ {x_1}^2 + {x_2}^2 + {x_3}^2 - 1.0 &= 0 \end{aligned}$$

の下で

$$f(\boldsymbol{x}) = -x_3$$

を最小にする.

(b) 入力データ
目的関数の値を計算する関数名: f
不等式制約条件を与える関数名: gf
等式制約条件を与える関数名: hf
n=3, m=2, ml=1, er = 1.0e − 14, nev=100, x[0] = −0.1, x[1] = 1.0, x[2] = 0.1

(c) 主プログラム

```
/*      C interface example for ASL_dmsqpm */

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

#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
double  f(double *x)
#else
double  f(x)
double *x;
#endif
{
    return -1.0 * x[2];
}
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    gf(double *x,int *n,double *g,int *mm)
#else
void    gf(x,n,g,mm)
double *x;
double *g;
int *n;
int *mm;
#endif
{
    *g = -1.0*x[0]+1.0*x[1];
}
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef __STDC__
void    hf(double *x,int *n,double *h,int *ml)
#else
void    hf(x,n,h,ml)
double *x;
double *h;
int *n;
int *ml;
#endif
{
    *h = x[0]*x[0]+x[1]*x[1]+x[2]*x[2]-1.0;
}
#ifdef __cplusplus
}
#endif
int main()
{
    int m;
    int ml;
    double *x;
    int n;
    double er;
    int nev;
    double y;
    int *iwk;
    double *wk;
    int ierr;
    int i;
    FILE *fp;

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

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

    fscanf( fp, "%d", &m );
    fscanf( fp, "%d", &ml );
    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &nev );
    er=1.0e-12;

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

```
    }

    wk = ( double * )malloc((size_t)( sizeof(double) *
    (4*n*n+21*n+108*m+2*m*n) ));
    if( wk == NULL )
    {
        printf( "no enough memory for array wk\n" );
        return -1;
    }

    iwk = ( int * )malloc((size_t)( sizeof(int) * (m+3) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

    printf( "\tm = %6d ml = %6d n = %6d nev = %6d er = %8.3g \n",
    m, ml, n, nev, er );

    printf("\n        ** vector x **\n\n");
    for( i=0 ; i<n ; i++ )
    {
        fscanf( fp, "%lf", &x[i] );
        printf( "      x( %6d )= %8.3g\n", i, x[i] );
    }
    printf( "\n" );

    fclose( fp );

    ierr = ASL_dmsqpm(f, gf, hf, m, ml, x, n, er, &nev, &y, iwk, wk);

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

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

    printf( "       y = %8.3g\n", y );

    free( x );
    free( wk );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmsqpm ***

 ** Input **

m =      2 ml =      1 n =      3 nev =    100 er =    1e-12

    ** vector x **

  x(      0 )=     -0.1
  x(      1 )=        1
  x(      2 )=      0.1


 ** Output **

ierr =      0
nev =      13

    ** vector x **

  x(      0 )= -3.04e-13
  x(      1 )= -3.18e-13
  x(      2 )=        1
   y =       -1
```

5.9 ネットワーク上の距離の最短化 (最短路問題)

5.9.1 ASL_dmsp1m, ASL_rmsp1m
ネットワーク上のある節点から他のすべての節点までの距離の最小化

(1) **機 能**

n 個の節点と m 本の枝をもち, 全枝が非負の重みであるグラフ上で, ある節点 v_1 から他の節点 v_p への枝の重み $w(k_j)\{(v_j, v_{j+1})\}$ の和 $W(P)$ を最小にする経路 $P = (v_1, v_2, \cdots, v_p)$ とそのときの $W(P)$ の値 (最短距離) を求める.

$$\text{目的関数} \quad : \quad W(P) = \sum_{j=1}^{p-1} w(k_j) \rightarrow \text{最小}$$

(2) **使用法**

倍精度関数:

ierr = ASL_dmsp1m (n, m, itl, ihd, wght, init, d, ip, isw, iwk, wk);

単精度関数:

ierr = ASL_rmsp1m (n, m, itl, ihd, wght, init, d, ip, isw, iwk, wk);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	節点の数 n
2	m	I	1	入 力	枝の数 m
3	itl	I*	m	入 力	枝 k の始点の節点番号 $tail(k)$
4	ihd	I*	m	入 力	枝 k の終点の節点番号 $head(k)$
5	wght	{D*, R*}	m	入 力	枝 k の重み $weight(k)$
6	init	{D, R}	1	入 力	出発点の節点番号 v_1
7	d	{D*, R*}	n	出 力	出発点 v_1 から節点 v_p までの最短距離 $D(v_1, v_p)$ (注意事項 (a) 参照)
8	ip	I*	n	出 力	出発点 v_1 から節点 v_p までの最短経路 (注意事項 (b) 参照)
9	isw	I	1	入 力	処理スイッチ (注意事項 (c) 参照) isw=0:有向グラフ isw=1:無向グラフ
10	iwk	I*	内容参照	ワーク	作業領域 **大きさ:** $4 \times \mathrm{n} + 2 \times \mathrm{m}$
11	wk	{D*, R*}	$2 \times \mathrm{m}$	ワーク	作業領域
12	ierr	I	1	出 力	エラーインディケータ (戻り値)

(4) **制限条件**

(a) n ≥ 2

(b) m ≥ 1

(c) wght$[k-1]$ >0.0, $(k = 1, \cdots, \mathrm{m})$

(d) 1≤itl$[k-1]$≤n, $(k = 1, \cdots, \mathrm{m})$

(e) 1≤ihd$[k-1]$≤n, $(k = 1, \cdots, \mathrm{m})$

(f) 1≤init≤n

(g) isw=0 または isw=1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3100	制限条件 (c) を満足しなかった.	
3200	制限条件 (d) または (e) を満足しなかった.	
3300	制限条件 (f) を満足しなかった.	
3400	制限条件 (g) を満足しなかった.	

(6) **注意事項**

(a) 最短距離 $D(v_p)$ の値が負であるならば, 出発点 v_1 から節点 v_p への経路が存在しない.

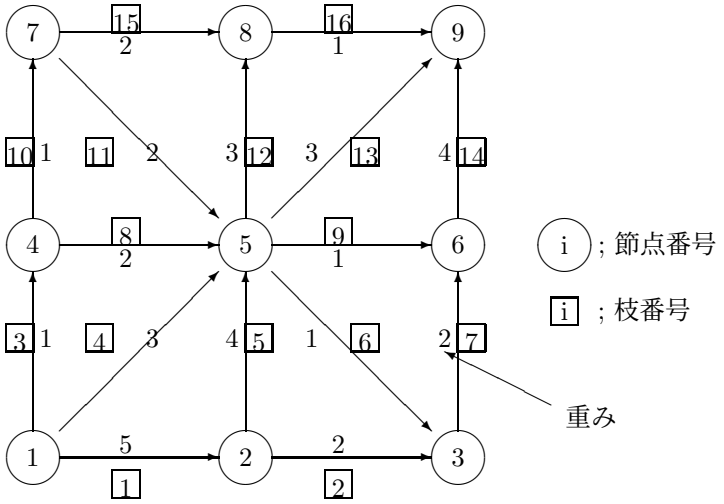
(b) 出発点 v_1 から節点 v_p への最短路上の節点の系列が $(v_1, \delta, \cdots, \beta, \alpha, v_p)$ であるとき, これらの節点は終点から始点へと順次 $\mathrm{ip}[v_p - 1] = \alpha, \mathrm{ip}[\alpha - 1] = \beta, \cdots, \mathrm{ip}[\delta - 1] = v_1$ のように求められる.

(c) 無向グラフの各枝は関数内で自動的に 2 本の有向枝に置き換えられるので, 入力データを 2 重化しておく必要はない.

(7) 使用例

(a) 問 題

以下のようなグラフ上で出発点 v_1 から各節点 v_p までの枝の重みの和を最小にする経路を求める．ただし，全枝の重みは非負とする．



(b) 入力データ

n= 9, m= 16, 枝の節点番号を格納する配列 itl と ihd, 枝の重みを格納する配列 wght, 出発点 init =1, isw =0.

(c) 主プログラム

```
/*      C interface example for ASL_dmsp1m */

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

int main()
{
    int n;
    int m;
    int *itl;
    int *ihd;
    double *wght;
    int init;
    double *d;
    int *ip;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i;

    FILE *fp;

    fp = fopen( "dmsp1m.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dmsp1m ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    init = 1;
    isw  = 0;

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

    ihd = ( int * )malloc((size_t) (sizeof(int) * m ));
    if( ihd == NULL )
    {
        printf( "no enough memory for array ihd\n" );
```

```
        return -1;
    }

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

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

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

    iwk = ( int * )malloc((size_t) (sizeof(int) * (4*n+2*m) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

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

    printf( "\tn    = %6d \n", n );
    printf( "\tm    = %6d \n", m );
    printf( "\tinit = %6d \n", init );
    printf( "\tisw  = %6d \n", isw );
    printf( "\n" );

    printf( "\t   itl    ihd     wght\n" );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%d %d %lf", &itl[i], &ihd[i], &wght[i] );
        printf( "\t%6d %6d %8.3g \n", itl[i], ihd[i], wght[i] );
    }

    fclose( fp );

    ierr = ASL_dmsp1m(n, m, itl, ihd, wght, init, d, ip, isw, iwk, wk);

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

    for( i=0 ; i<n ; i++ )
    {
        printf( "\t d( %2d ) = %8.3g   ip( %2d ) = %6d\n",
        i+1, d[i], i+1, ip[i] );
    }

    free( itl );
    free( ihd );
    free( wght );
    free( ip );
    free( d );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmsp1m ***

 ** Input **

n    =      9
m    =     16
init =      1
isw  =      0

   itl    ihd     wght
     1      2        5
     2      3        2
     1      4        1
     1      5        3
     2      5        4
     5      3        1
     3      6        2
     4      5        2
     5      6        1
```

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

 ** Output **

ierr =      0

 d(  1 ) =        0   ip(  1 ) =      1
 d(  2 ) =        5   ip(  2 ) =      1
 d(  3 ) =        4   ip(  3 ) =      5
 d(  4 ) =        1   ip(  4 ) =      1
 d(  5 ) =        3   ip(  5 ) =      1
 d(  6 ) =        4   ip(  6 ) =      5
 d(  7 ) =        2   ip(  7 ) =      4
 d(  8 ) =        4   ip(  8 ) =      7
 d(  9 ) =        5   ip(  9 ) =      8
```

5.9.2 ASL_dmspmm, ASL_rmspmm
ネットワーク上の全 2 節点間の距離の最小化

(1) **機 能**

n 個の節点と m 本の枝をもち, 無向グラフの場合は負の重みの枝を含まないという条件を満たし, 有向グラフの場合は負の長さのサイクルを含まないという条件を満たすグラフ上で,2 節点 $(v_1, v_p)(i = 1, 2, \cdots, n)$ 間の枝の重みの和 $W(P)$ を最小にする経路 $P = (v_1, v_2, \cdots, v_p)$ とそのとき $W(P)$ の値 (最短距離) を求める.

(2) **使用法**

倍精度関数:

ierr = ASL_dmspmm (n, m, itl, ihd, wght, d, ip, isw, iwk);

単精度関数:

ierr = ASL_rmspmm (n, m, itl, ihd, wght, d, ip, isw, iwk);

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	節点の数 n
2	m	I	1	入 力	枝の数 m
3	itl	I*	m	入 力	枝 k の始点の節点番号 $tail(k)$
4	ihd	I*	m	入 力	枝 k の終点の節点番号 $head(k)$
5	wght	{ D* / R* }	m	入 力	枝 k の重み $weight(k)$
6	d	{ D* / R* }	n×n	出 力	節点 v_i から節点 v_p までの最短距離 $D(v_i, v_p)$ (注意事項 (a) 参照)
7	ip	I*	n×n	出 力	節点 v_i から節点 v_p までの最短経路 (注意事項 (b) 参照)
8	isw	I	1	入 力	処理スイッチ (注意事項 (c) 参照) isw=0:有向グラフ isw=1:無向グラフ
9	iwk	I*	n×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) n $\geq$ 2

(b) m $\geq$ 1

(c) wght[$k-1$] >0.0, $(k = 1, \cdots,$ m) (ISW=1 のとき)

(d) 1≤itl[$k-1$]≤n, $(k = 1, \cdots,$ m)

(e) 1≤ihd[$k-1$]≤n, $(k = 1, \cdots,$ m)

(f) isw=0 または isw=1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3100	制限条件 (c) を満足しなかった.	
3200	制限条件 (d) または (e) を満足しなかった.	
3300	制限条件 (f) を満足しなかった.	
4000	グラフに負のサイクルが含まれていた.	

(6) **注意事項**

(a) 最短距離 $D(v_1, v_p)$ の値が負の値であるならば, 節点 v_1 から節点 v_p への経路は存在しない.

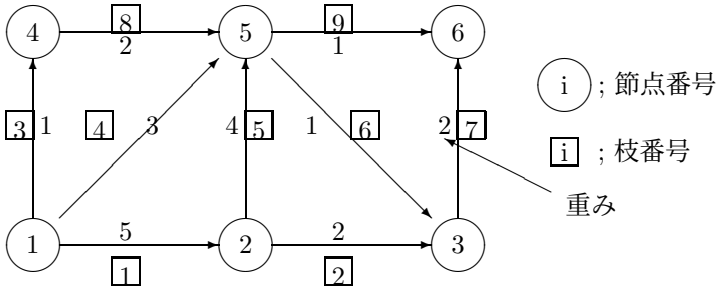
(b) 節点 v_1 から節点 v_p への最短路上の節点の系列が $(v_1, \delta, \cdots, \beta, \alpha, v_p)$ であるとき, これらの節点は終点から始点へと順次 $\mathrm{ip}[v_1 - 1 + v_p * n] = \alpha, \mathrm{ip}[v_1 - 1 + \alpha * n] = \beta, \cdots, \mathrm{ip}[v_1 - 1 + \delta * n] = v_1$ のように求められる.

(c) 無向グラフの各枝は関数内で自動的に 2 本の有向枝に置き換えられるので, 入力データを 2 重化しておく必要はない.

(7) **使用例**

(a) 問 題

以下のようなグラフ上で節点 v_1 から節点 v_p への枝の重みの和を最小にする経路を求める.



(b) 入力データ

n= 6, m= 9, 枝の節点番号を格納する配列 itl と ihd, 枝の重みを格納する配列 wght, isw =0.

(c) 主プログラム

```
/*      C interface example for ASL_dmspmm */

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

int main()
{
    int n;
    int m;
    int *itl;
    int *ihd;
    double *wght;
    double *d;
    int *ip;
    int isw;
    int *iwk;
    int ierr;
    int i,j;
```

```
    FILE *fp;

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

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

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    isw = 0;

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

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

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

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

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

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

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

    printf( "\t   itl    ihd     wght\n" );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%d %d %lf", &itl[i], &ihd[i], &wght[i] );
        printf( "\t%6d %6d %8.3g \n", itl[i], ihd[i], wght[i] );
    }
    printf( "\n" );

    fclose( fp );

    ierr = ASL_dmspmm(n, m, itl, ihd, wght, d, ip, isw, iwk);

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

    for( i=0 ; i<n ; i++ )
    {
        for( j=0 ; j<n ; j++ )
        {
          if( d[i+j*n]!=0 ) printf( "\td(%2d,%2d) =  %8.3g    ip(%2d,%2d) = %6d\n",
                        i+1,j+1, d[i+j*n], i+1,j+1, ip[i+j*n] );
        }
    }

    free( itl );
    free( ihd );
    free( wght );
    free( ip );
    free( d );
    free( iwk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmspmm ***

 ** Input **

n   =      6
m   =      9
isw =      0

   itl    ihd     wght
     1      2        5
     2      3        2
     1      4        1
     1      5        3
     2      5        4
     5      3        1
     3      6        2
     4      5        2
     5      6        1


 ** Output **

ierr =      0

d( 1, 2) =         5    ip( 1, 2) =      1
d( 1, 3) =         4    ip( 1, 3) =      5
d( 1, 4) =         1    ip( 1, 4) =      1
d( 1, 5) =         3    ip( 1, 5) =      1
d( 1, 6) =         4    ip( 1, 6) =      5
d( 2, 3) =         2    ip( 2, 3) =      2
d( 2, 5) =         4    ip( 2, 5) =      2
d( 2, 6) =         4    ip( 2, 6) =      3
d( 3, 6) =         2    ip( 3, 6) =      3
d( 4, 3) =         3    ip( 4, 3) =      5
d( 4, 5) =         2    ip( 4, 5) =      4
d( 4, 6) =         3    ip( 4, 6) =      5
d( 5, 3) =         1    ip( 5, 3) =      5
d( 5, 6) =         1    ip( 5, 6) =      5
```

5.9.3 ASL_dmsp11, ASL_rmsp11
ネットワーク上の 2 節点間の距離の最小化

(1) **機 能**

n 個の節点と m 本の枝をもち, 全枝が非負の重みであるグラフ上で, 2 節点間 (v_1, v_p) の枝の重み $w(k_j)\{(v_j, v_{j+1})\}$ の和 $W(P)$ を最小にする経路 $P = (v_1, v_2, \cdots, v_p)$ とそのときの $W(P)$ の値 (最短距離) を求める.

$$\text{目的関数} \quad : \quad W(P) = \sum_{j=1}^{p-1} w(k_j) \rightarrow \text{最小}$$

(2) **使用法**

倍精度関数:

```
ierr = ASL_dmsp11  (n, m, itl, ihd, wght, init, iend, & d, ip, isw, iwk, wk);
```

単精度関数:

```
ierr = ASL_rmsp11  (n, m, itl, ihd, wght, init, iend, & d, ip, isw, iwk, wk);
```

(3) **引数と戻り値**

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

項番	引数と戻り値	型	大きさ	入出力	内　　容
1	n	I	1	入 力	節点の数 n
2	m	I	1	入 力	枝の数 m
3	itl	I*	m	入 力	枝 k の始点の節点番号 $tail(k)$
4	ihd	I*	m	入 力	枝 k の終点の節点番号 $head(k)$
5	wght	{D*, R*}	m	入 力	枝 k の重み $weight(k)$
6	init	{D, R}	1	入 力	出発点 v_1
7	iend	{D, R}	1	入 力	終点 v_p
8	d	{D*, R*}	1	出 力	出発点 v_1 から終点 v_p までの最短距離 (注意事項 (a) 参照)
9	ip	I*	n	出 力	出発点 v_1 から終点 v_p までの最短経路 (注意事項 (b) 参照)
10	isw	I	1	入 力	処理スイッチ (注意事項 (c) 参照) isw=0:有向グラフ isw=1:無向グラフ
11	iwk	I*	内容参照	ワーク	作業領域 **大きさ:** $4 \times \mathrm{n} + 2 \times \mathrm{m}$
12	wk	{D*, R*}	$\mathrm{n} + 2 \times \mathrm{m}$	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) n $\geq$ 2

(b) m $\geq$ 1

(c) wght[$k-1$] >0.0, $(k = 1, \cdots,$ m)

(d) 1≤itl[$k-1$]≤n, $(k = 1, \cdots,$ m)

(e) 1≤ihd[$k-1$]≤n, $(k = 1, \cdots,$ m)

(f) 1≤init≤n

(g) 1≤iend≤n

(h) isw=0 または isw=1

(5) **エラーインディケータ (戻り値)**

戻り値	意　　味	処　理　内　容
0	正常終了.	
3000	制限条件 (a) または (b) を満足しなかった.	処理を打ち切る.
3100	制限条件 (c) を満足しなかった.	
3200	制限条件 (d) または (e) を満足しなかった.	
3300	制限条件 (f) または (g) を満足しなかった.	
3400	制限条件 (h) を満足しなかった.	

(6) **注意事項**

(a) 最短距離 D の値が負であるとき, 出発点 v_1 から終点 v_p への経路が存在しないことを表す.

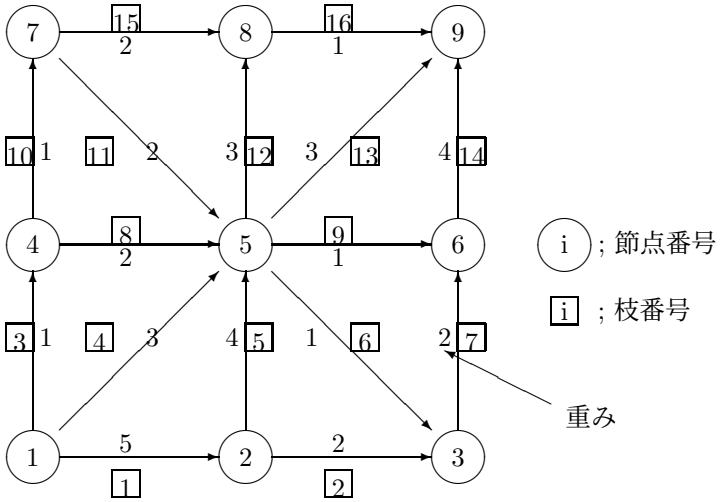
(b) 出発点 v_1 から節点 v_p への最短路上の節点の系列が $(v_1, \delta, \cdots, \beta, \alpha, v_p)$ であるとき, これらの節点は終点から始点へと順次 $\mathrm{ip}[v_p - 1] = \alpha, \mathrm{ip}[\alpha - 1] = \beta, \cdots, \mathrm{ip}[\delta - 1] = v_1$ のように求められる.

(c) 無向グラフの各枝は関数内で自動的に 2 本の有向枝に置き換えられるので, 入力データを 2 重化しておく必要はない.

(7) 使用例

(a) 問 題

以下のようなネットワーク上で出発点 v_1 から終点 v_p までの枝の重みの和を最小にする経路を求める. ただし, 全枝の重みは非負とする.



(b) 入力データ

n= 9, m= 16, 枝の節点番号を格納する配列 itl と ihd,
枝の重みを格納する配列 wght,
出発点 init =1, 終点 iend=8 ,isw= 0

(c) 主プログラム

```
/*      C interface example for ASL_dmsp11 */

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

int main()
{
    int n;
    int m;
    int *itl;
    int *ihd;
    double *wght;
    int init;
    int iend;
    double d;
    int *ip;
    int isw;
    int *iwk;
    double *wk;
    int ierr;
    int i;

    FILE *fp;

    fp = fopen( "dmsp11.dat", "r" );
    if( fp == NULL )
    {
        printf( "file open error\n" );
        return -1;
    }

    printf( "    *** ASL_dmsp11 ***\n" );
    printf( "\n     ** Input **\n\n" );

    fscanf( fp, "%d", &n );
    fscanf( fp, "%d", &m );
    init = 1;
    iend = 8;
    isw = 0;

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

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

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

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

    iwk = ( int * )malloc((size_t) (sizeof(int) * (4*n+2*m) ));
    if( iwk == NULL )
    {
        printf( "no enough memory for array iwk\n" );
        return -1;
    }

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

    printf( "\tn    = %6d \n", n );
    printf( "\tm    = %6d \n", m );
    printf( "\tinit = %6d \n", init );
    printf( "\tiend = %6d \n", iend );
    printf( "\tisw  = %6d \n", isw );
    printf( "\n" );

    printf( "\t   itl    ihd     wght\n" );
    for( i=0 ; i<m ; i++ )
    {
        fscanf( fp, "%d %d %lf", &itl[i], &ihd[i], &wght[i] );
        printf( "\t%6d %6d %8.3g \n", itl[i], ihd[i], wght[i] );
    }

    fclose( fp );

    ierr = ASL_dmsp11(n, m, itl, ihd, wght, init, iend, &d, ip, isw, iwk, wk);

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

    printf( "\td = %8.3g \n\n", d );

    for( i=0 ; i<n ; i++ )
    {
        printf( "\tip(%2d) = %6d\n", i+1, ip[i] );
    }

    free( itl );
    free( ihd );
    free( wght );
    free( ip );
    free( iwk );
    free( wk );

    return 0;
}
```

(d) 出力結果

```
*** ASL_dmsp11 ***

 ** Input **

n    =      9
m    =     16
init =      1
iend =      8
isw  =      0

   itl    ihd     wght
     1      2        5
     2      3        2
     1      4        1
     1      5        3
     2      5        4
     5      3        1
     3      6        2
     4      5        2
     5      6        1
     4      7        1
     7      5        2
     5      8        3
     5      9        3
     6      9        4
     7      8        2
     8      9        1

 ** Output **

ierr =      0

d =         4

ip( 1) =       0
ip( 2) =       1
ip( 3) =       5
ip( 4) =       1
ip( 5) =       1
ip( 6) =       5
ip( 7) =       4
ip( 8) =       7
ip( 9) =       5
```

付 録 A　用語説明

(1) **グラフ**
有限個の点の集合 $V = \{v_1, v_2, \cdots, v_n\}$ と V に属する点の対の集合 $V \times V = \{(v_i, v_j) | v_i \in V, v_j \in V\}$ に対し, $E \subseteq V \times V$ と V の組をグラフといい, $G = (V, E)$ と表す.

(2) **節点と枝**
与えられたグラフ $G = (V, E)$ に対して V の要素のことをグラフ G の節点, E の要素のことをグラフ G の枝と呼ぶ.

(3) **有向枝**
グラフ $G = (V, E)$ において E の要素 (v_i, v_j) $(v_i \in V, v_j \in V)$ と (v_j, v_i) を区別して扱う場合, E の各要素を有向枝と呼ぶ. なお有向枝 $e = (v_i, v_j)$ に対して $tail(e) = v_i$, $head(e) = v_j$ と表す.

(4) **閉路**
グラフ $G = (V, E)$ に対して, V の要素の組 $p = (v_{j_1}, v_{j_2}, \cdots, v_{j_m})$ が任意の $1 \leq k \leq m-1$ について $(v_{j_k}, v_{j_{k+1}}) \in E$ を満たすとき, p をグラフ G 上の路と呼ぶ. とくに $v_{j_1} = v_{j_m}$ であるような路を閉路と呼ぶ.

(5) **木 (tree)**
閉路を含まないグラフを木 (tree) と呼ぶ.

(6) **ネットワーク**
グラフ $G = (V, E)$ の各枝 $e \in E$ に対して, 費用係数 $c(e)$ と容量 $u(e)$ という 2 種類の実数と, 始点 s と終点 t という V の 2 つの要素が与えられているとき, G と c, u, s, t の組をネットワークと呼び, $N = (G, c, u, s, t)$ のように表す. なお, c, u, s, t の一部だけを扱う場合もやはり, ネットワークと呼ばれる. 例えば始点 s と終点 t を特に定めない問題を扱う場合には, $N = (G, c, u)$ のように表す.

(7) **流れ**
ネットワークの各枝 $e \in E$ に対して, $0 \leq x(e) \leq u(e)$ を満たすベクトル $x(e)$ をネットワーク N 上の流れ (flow) と呼ぶ.

(8) **最小費用流問題**
与えられたネットワーク N の各節点 $v \in V$ に対して実数 $b(v)$ が与えられ,

$$\sum_{tail(e)=v} x(e) - \sum_{head(e)=v} x(e) = b(v)$$

$$\sum_{v \in V} b(v) = 0$$

が成立しているときに, 全枝の費用の和

$$\sum_{e \in E} c(e)x(e)$$

を最小にする流れ $x(e)$ を求める問題を最小費用流問題と呼ぶ.

付 録 B　　ASL で使用している計算機依存定数

B.1　誤差判定のための単位

ASL では, 浮動小数点演算における誤差判定のための単位として次の値を設定している. 誤差判定のための単位は, 浮動小数点データの内部表現によって決まる数値であり, ASL C 言語インタフェース ではこの単位を収束判定, 零判定などに用いることがある.

表 B–1　誤差判定のための単位

単精度演算	倍精度演算
$2^{-23}(\simeq 1.19\times 10^{-7})$	$2^{-52}(\simeq 2.22\times 10^{-16})$

備考 誤差判定の単位 ε はマシン ε と呼ばれることもあり, 通常, 対応する浮動小数点形式で $1+\varepsilon$ の計算結果が 1 と異なるような最小の正の定数として定義される. したがって, 誤差判定の単位を見れば, その浮動小数点形式での (仮数部の) 演算の最大有効桁数がわかる.

B.2　浮動小数点データの値の最大値・最小値

ASL の内部で定義している浮動小数点データの値の最大値, 最小値を以下に示す.
なお, 以下の最大値, 最小値はハードウェアが実際に採用している浮動小数点形式のそれとは異なる場合があるので注意されたい.

表 B–2　浮動小数点データの値の最大値・最小値

	単精度演算	倍精度演算
最大値	$2^{127}(2-2^{-23})$ $(\simeq 3.40\times 10^{38})$	$2^{1023}(2-2^{-52})$ $(\simeq 1.80\times 10^{308})$
正の最小値	2^{-126} $(\simeq 1.17\times 10^{-38})$	2^{-1022} $(\simeq 2.23\times 10^{-308})$
負の最大値	-2^{-126} $(\simeq -1.17\times 10^{-38})$	-2^{-1022} $(\simeq -2.23\times 10^{-308})$
最小値	$-2^{127}(2-2^{-23})$ $(\simeq -3.40\times 10^{38})$	$-2^{1023}(2-2^{-52})$ $(\simeq -1.80\times 10^{308})$

索 引

アプリケーションシステム

科学技術計算ライブラリ
ASL C 言語インタフェース
ユーザーズガイド

〈 基本機能編 第 5 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